FEEDFORWARD NEURAL NETWORK FOR SOLVING PARTICULAR FRACTIONAL DIFFERENTIAL EQUATIONS

By

MOHD RASHID BIN ADMON

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfillment of the Requirements for the Degree of Doctor of Philosophy

January 2024

IPM 2024 4

DEDICATIONS

To all of my love
Father Admon Ahmad & Mother Jaliah Sulkiman
Elder Brother Mohammad Ali, Mohd Fairul Radzi & Mohd Ariff
Elder Sister Nor Delyliana & Siti Mastura
Brother-in-Law Shaiful Haili
Sister-in-Law Nik Hazura, Noor Aslinda, Faizah
Nephews Zahin Irfan, Muhammad Ariqq, Muhammad Ahnaf, Muhammad Anas, Muhammad Faeeq Luthfi & Rizqin
Niece Nur Farah Adila, Nur Fasha Amira, Nur Adelia Amani, Nur Aqilah Husna, Nur Aisyah Imaan & Aisyah
Their support and company made my Ph.D. journey went seamlessly

And most importantly;
To Myself
A guy who works so hard to realize his dream as a Ph.D. graduate.

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfillment of the requirement for the degree of Doctor of Philosophy

FEEDFORWARD NEURAL NETWORK FOR SOLVING PARTICULAR FRACTIONAL DIFFERENTIAL EQUATIONS

By

MOHD RASHID BIN ADMON

January 2024

Chairman : **Associate Professor Norazak bin Senu, PhD**
Institute : **Mathematical Research**

Fractional differential equations (FDEs) model real-world phenomena capturing memory effects. However, existing numerical methods are mostly traditional, prompting the need for innovative approaches. Artificial neural networks (ANNs), a machine learning tool, have exhibited promising capabilities in solving differential equations. This research aims to develop a scheme based on a feedforward neural network (FNN) with a vectorized algorithm (FNNVA) for solving FDEs in the Caputo sense (FDEsC) using selected first-order optimization techniques: simple gradient descent (GD), momentum method (MM), and adaptive moment estimation method (Adam). Then, a single hidden layer of FNN based on Chelyshkov polynomials with an extreme learning machine algorithm (SHLFNNCP-ELM) is constructed for solving FDEsC. Next, a scheme based on an extended single hidden layer of FNN using a second-order optimization technique known as the Broyden–Fletcher–Goldfarb–Shanno method (ESHLFNN-BFGS) is designed to solve FDEs in the Caputo-Fabrizio sense (FDEsCF). This study also focuses on solving fractal-fractional differential equations in the Caputo sense with a power-law kernel (FFDEsCP) using FNN in two hidden layers with a vectorized algorithm alongside Adam (FNN2HLVA-Adam). In the first scheme, a vectorized algorithm and automatic differentiation are implemented to minimize computational costs. Numerical results indicated that FNNVA with Adam in one or two hidden layers, 5 or 10 nodes, and an appropriate learning rate offers superior accuracy compared to FNNVA with GD and FNNVA with MM. The second approach relies on Chelyshkov basis functions for approximation and utilizes the extreme machine learning algorithm for weight determination, achieving high accuracy and low computational time. The third scheme employs the BFGS solver during the learning process, attained satisfactory numerical results with fewer iterations. The final scheme utilizes a two hidden layer FNNVA, with Adam optimization, using suitable number of nodes and value of learning rates to handle problems involving memory and fractal concepts. The numerical solutions obtained are consistent with reference solutions. In conclusion, all proposed schemes deliver more accurate results compared to existing methods while maintaining low computational costs.

SDG: Feedforward neural network, Fractal-fractional differential equations, Fractional differential equations, Hidden layers, Vectorized algorithm

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

RANGKAIAN NEURAL SUAP MAJU UNTUK MENYELESAIKAN PERSAMAAN PEMBEZAAN PECAHAN TERTENTU

Oleh

MOHD RASHID BIN ADMON

Januari 2024

Pengerusi	: **Profesor Madya Norazak bin Senu, PhD**
Institut	: **Penyelidikan Matematik**

Persamaan pembezaan pecahan (PPP) memodelkan fenomena dunia nyata dengan menangkap kesan memori. Walau bagaimanapun, kaedah berangka semasa secara kebanyakannya adalah konvensional, menggesa keperluan untuk pendekatan inovatif. Rangkaian neural buatan (RNB), sebuah alat pembelajaran mesin, telah menunjukkan keupayaan yang menjanjikan dalam menyelesaikan persamaan pembezaan. Kajian ini bertujuan untuk membangunkan satu skema berasaskan rangkaian neural suap maju (RNSM) dengan algoritma vektor (RNSMAV) untuk menyelesaikan PPP dalam pemahaman Caputo (PPPC) menggunakan teknik pengoptimuman peringkat pertama yang dipilih iaitu turun cerun mudah (TCM), kaedah momentum (KM) dan kaedah anggaran momen penyesuaian (KAMP). Kemudian, lapisan tersembunyi tunggal RNSM berdasarkan polinomial Chelyshkov dengan algoritma mesin pembelajaran ekstrim (LTTRNSMPC-MPE) dibina untuk menyelesaikan PPPC. Seterusnya, skema berdasarkan lapisan tersembunyi tunggal RNSM yang diperluaskan menggunakan teknik pengoptimuman peringkat kedua yang dikenali sebagai kaedah Broyden–Fletcher–Goldfarb–Shanno (LTTRNSMD-BFGS) direka untuk menyelesaikan PPP dalam pemahaman Caputo-Fabrizio (PPPCF). Kajian ini juga menumpukan kepada penyelesaian persamaan pembezaan pecahan fraktal dalam pemahaman Caputo dengan inti hukum kuasa (PPPFCIHK) menggunakan skema berdasarkan RNSM dalam dua lapisan tersembunyi dengan algoritma vektor bersama-sama KAMP (RNSM2LTAV-KAMP). Dalam skema pertama, algoritma vektor dan pembezaan automatik dilaksanakan untuk mengurangkan kos komputasi. Hasil berangka menunjukkan bahawa RNSMAV dengan KAMP dalam satu atau dua lapisan tersembunyi, 5 atau 10 nod, dan kadar pembelajaran yang sesuai menawarkan kejituan yang lebih unggul berbanding RNSMAV dengan TCM dan RNSMAV dengan KM. Skema kedua bergantung kepada fungsi asas Chelyshkov untuk penyelesaian dan menggunakan algoritma mesin pembelajaran ekstrim untuk penentuan berat rangkaian, telah menghasilkan kejituan yang tinggi dan masa pengiraan yang rendah. Skema ketiga menggunakan penyelesai BFGS semasa proses pembelajaran, mencapai keputusan berangka yang memuaskan dengan bilangan lelaran yang sedikit. Skema terakhir menggunakan dua lapisan tersembunyi RNSMAV, dengan pengoptimuman Adam, serta bilangan nod dan nilai kadar pembelajaran yang sesuai untuk menangani masalah berkaitan dengan memori dan konsep fraktal. Penyelesaian berangka yang diperolehi adalah konsisten dengan penyelesaian rujukan. Kesimpulannya, semua skema yang dicadangkan memberikan hasil yang lebih tepat berbanding dengan kaedah sedia ada, sambil mengekalkan kos pengiraan yang rendah.

SDG: Algoritma vektor, Lapisan tersembunyi, Persamaan pembezaan pecahan, Persamaan pembezaan pecahan fraktal, Rangkaian neural suap maju

ACKNOWLEDGEMENTS

In the name of Allah and with regards to Prophet Muhammad S.A.W for His blessing for giving me strength and health to complete this research.

First and foremost, a thousand thanks to my supervisor, Associate Professor Dr. Norazak Senu for his encouragement, patience, enthusiasm and immense knowledge. His guidance has helped me constantly in conducting this research and in writing this thesis. I am thankful to my co-supervisor, Dr. Ali Ahmadian for his absolute support and guidance through idea, facts and publications of my research. Also to other supervisory committee member, Professor Dr. Zanariah Abdul Majid and Dr. Mohamat Aidil Mohamat Johari for helping me during the completion of this thesis.

In preparing this thesis, I share the credit of my work with my beloved family because they continuously gave me strength and support to finish this project.

Last but not least, thanks to lecturers, friends and everyone, that have contributed directly and indirectly in the duration of conducting this research.

This thesis was submitted to the Senate of Universiti Putra Malaysia and has been accepted as fulfillment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Norazak bin Senu, PhD
Associate Professor
Faculty of Science
Universiti Putra Malaysia
(Chairman)

Zanariah binti Abdul Majid, PhD
Professor
Faculty of Science
Universiti Putra Malaysia
(Member)

Mohamat Aidil bin Mohamat Johari, PhD
Senior Lecturer
Faculty of Science
Universiti Putra Malaysia
(Member)

Ali Ahmadian, PhD
Senior Lecturer
College of Engineering and Aviation
Central Queensland University
Australia
(Member)

ZALILAH MOHD SHARIFF, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 18 April 2024

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF ABBREVIATIONS

AD	Automatic Differentiation
Adam	Adaptive Moment Estimation Method
ADM	Adomian Decomposition Method
AGM	Akbari-Ganji's Method
AI	Artificial Intelligence
ANN	Artificial Neural Network
ANN-SQP	Artificial Neural Network with Sequential Quadratic Programming
BeNN	Bernstein Neural Network
BFGS	Broyden–Fletcher–Goldfarb–Shanno
ChNN	Chebyshev Neural Network
CF	Caputo-Fabrizio
DBF	Deep Belief Network
DNN	Deep Neural Network
ELM	Extreme Learning Machine Algorithm
ESHLFNN	Extended Single Hidden Layer of Feedforward Neural Network
FC	Fractional Calculus
FDM	Finite Difference Method
FEM	Finite Element Method
FFD	Fractal-Fractional derivative

FDE/FDEs	Fractional Differential Equation/Fractional Differential Equations
FFDE/FFDEs	Fractal-Fractional Differential Equation/Fractal-Fractional Differential Equations
FDEC/FDEsC	Fractional Differential Equation in Caputo sense/Fractional Differential Equations in Caputo sense
FDECF/FDEsCF	Fractional Differential Equation in Caputo-Fabrizio sense/Fractional Differential Equations in Caputo-Fabrizio sense
FFDECP/FFDEsCP	Fractal-Fractional Differential Equation in Caputo sense with power law kernel/ Fractal-Fractional Differential Equations in Caputo sense with power law kernel
FPDEs	Fractional Partial Differential Equations
FPDEsC	Fractional Partial Differential Equations in Caputo sense
FNN	Feedforward Neural Network
FNNVA	Feedforward Neural Network with Vectorized Algorithm
FNN2HLVA	Feedforward Neural Network in Two Hidden Layers with Vectorized Algorithm
GA	Genetic Algorithm
GA-PS	Genetic Algorithm Hybrid with Pattern Search Technique
IVP	Initial-Value Problem
LeNN	Legendre Neural Network
LM	Levenberg-Marquardt Algorithm
ML	Machine Learning
MM	Momentum Method
MEMS	Microelectromechanical System
ODE/ODEs	Ordinary Differential Equation/Ordinary Differential Equations
PS	Pattern Search Technique
PSO-SA	Particle Swarm Optimization Algorithm with Simmulated Annealing

R-L	Riemann-Liouville
RNN	Recurrent Neural Network
RMSProp	Root Mean Squared Propagation
GD	Simple Gradient Descent
SHLFNNCP	Single Hidden Layer of Feedforward Neural Network Based on Chelyshkov Polynomial
VIM	Variational Iteration Method

CHAPTER 1

INTRODUCTION

1.1 Fractional Calculus

Fractional calculus (FC) is one of the branches of mathematics that deals with the theory and application of derivatives and integrals of arbitrary order (real or complex numbers) (Miller and Ross, 1993). It can be considered a modern version of mathematical knowledge as it overcomes the limitations of traditional or classical calculus, which is limited to dealing with integer-order derivatives and integrals. Although the extension of order may seem straightforward, the theory in FC is fundamentally different from traditional calculus, making it exclusive in applications.

Differing from traditional calculus, FC consists of an abundance of definitions. Here, the definitions represent the mathematical formulation of fractional operator that define as fractional differentiation or fractional integration. One of the attractive features in FC lies behind the fractional operator itself, which able to capture memory or hereditary effect when the transition of non-integer order takes place (Ford and Simpson, 2001). This property often related to the behaviour of process in a system in which the output let say $Y(t)$, at the current time t, is depend on the process occured in $\{\tau, Y(\tau)\}$ for entire time history, $\tau \in [t_0, t]$ (Tarasov, 2018). This concept is mathematically termed as nonlocal property, differing from traditional calculus, which is local and independent on the behaviour of a system in the history.

The foundation of knowledge in FC crucially depends on several basic functions commonly encountered in the definition of derivatives and integrals of arbitrary order. Here, the definition of the Gamma function is presented.

Definition 1.1 (Gamma Function) *(Milici et al., 2018).*

Gamma function or second Euler integral play the most important role in the theory of differentiation and integrals in FC. It has the following definition

$$\Gamma(q) = \int_0^\infty e^{-t} t^{q-1} dt. \tag{1.1}$$

By presenting the Gamma function, it is now possible to highlight some definitions related to fractional integrals and derivatives used in this research. To begin, let's introduce the classical definition of fractional integral known as Riemann-Liouville (R-L) fractional integral, presented as follows:

Definition 1.2 (Riemann-Liouville Fractional Integral) *(Podlubny, 1998).*

Let (a,b) is a finite interval in the real axis $\mathbb{R}$. Then, the Riemann-Liouville fractional integral with order $\alpha > 0$ is defined as

$${}_aJ_x^\alpha g(x) = \frac{1}{\Gamma(\alpha)} \int_a^x (x-t)^{\alpha-1} g(t) dt, \tag{1.2}$$

where $\Gamma(\cdot)$ denotes the Gamma function.

This definition serves as the cornerstone for most of the fractional derivatives that exist in FC, such as the Riemann-Liouville (R-L) fractional derivative and the Caputo fractional derivative.

Definition 1.3 (Riemann-Liouville Fractional Derivative) *(Podlubny, 1998).*

Let (a,b) is a finite interval in the real axis $\mathbb{R}$. Then, the Riemann-Liouville fractional derivative with order $\alpha > 0$ is defined as

$$\begin{aligned}{}_aD_x^{\alpha} g(x) &= \frac{d^m}{dt^m}\left[{}_aJ_x^{m-\alpha} g(x)\right] \\ &= \frac{1}{\Gamma(m-\alpha)}\frac{d^m}{dt^m}\int_a^x (x-t)^{m-\alpha-1} g(t)dt, \end{aligned} \tag{1.3}$$

where $m-1 < \alpha \leq m$, $m \in \mathbb{N}$.

Definition 1.4 (Caputo Fractional Derivative) *(Li and Zeng, 2015).*

Let (a,b) is a finite interval in the real axis $\mathbb{R}$. Then, the Caputo fractional derivative with order $\alpha > 0$ is defined as

$$\begin{aligned}{}_a^CD_x^{\alpha} g(x) &= {}_aJ_x^{(m-\alpha)}\left[g^{(m)}(x)\right] \\ &= \frac{1}{\Gamma(m-\alpha)}\int_a^x (x-t)^{m-\alpha-1} g^m(t)dt, \end{aligned} \tag{1.4}$$

where $m-1 < \alpha \leq m$, $m \in \mathbb{N}$.

Caputo and Fabrizio (2015) develop a new fractional derivative that have non-singular exponentional decaying kernel known as Caputo-Fabrizio (CF) fractional derivative. The main purpose of this new derivative is to get rid of the singularity of Caputo fractional derivative that often become challenges when designing numerical approximation at the endpoint of the singularity.

Let $H^1(a,b) = \{g | g \in L^2(a,b) \text{ and } g' \in L^2(a,b)\}$ where $L^2(a,b)$ is the space of square integrable functions on interval (a,b). Then, CF fractional derivative defined as follows:

Definition 1.5 (Caputo-Fabrizio Fractional Derivative) *(Caputo and Fabrizio, 2015).*

Let $g(t) \in H^1(a,b)$ and $\alpha \in (0,1)$. Then, the Caputo-Fabrizio fractional derivative with order α is defined as

$$ {}^{CF}_{a}D^{\alpha}_{x} g(x) = \frac{M(\alpha)}{1-\alpha} \int_a^x g'(t) e^{\left(-\alpha \frac{x-t}{1-\alpha}\right)} dt, \tag{1.5}$$

where $M(\alpha)$ is normalization function such that $M(0) = M(1) = 1$.

1.2 Fractal-Fractional Calculus

Despite the existence of the ground-breaking theory related on fractional derivative, there is another idea known as the fractal derivative or Hausdorff derivative (Chen, 2006). Fractal derivative or Hausdorff derivative can be defined by transforming the classical space-time derivative that scaled with integer dimension (g,t) to a fractal time (g,t^{β}) where β denotes fractal dimension in time (Allwright and Atangana, 2018; Chen, 2006). Mathematically, this can be defined as:

$$\frac{dg}{dt^{\beta}} = \lim_{t \to s} \frac{g(t) - g(s)}{t^{\beta} - s^{\beta}}, \quad \beta > 0. \tag{1.6}$$

The fractal derivative can also can be defined into fractal space-time (g^{ν}, t^{β}) which can be defined as

$$\frac{dg^{\nu}}{dt^{\beta}} = \lim_{t \to s} \frac{g^{\nu}(t) - g^{\nu}(s)}{t^{\beta} - s^{\beta}}, \quad \nu > 0, \quad \beta > 0. \tag{1.7}$$

where ν denotes fractal dimension in space. From the definition above, the fractal derivative differs from the traditional integer-order derivative in that the former represents the ratio of change of two quantities in fractal space, whereas the latter represents the change of a function (dependent variable) with the change of another quantity (independent variable) in ordinary space. In comparison with fractional derivatives, fractal derivatives are local operators, while fractional derivatives are global, as there is no convolution integral in (1.6) and (1.7).

Atangana (2017) combine the concept of fractal derivative and fractional derivative into single operator known as fractal-fractional derivative (FFD). One of the fractal-fractional derivative used in this study is defined in the following definition:

Definition 1.6 (Fractal-Fractional Derivative in Caputo sense with power law kernel) *(Atangana, 2017).*

Let $g(x)$ is a differentiable function in interval (a,b). If $g(x)$ is fractal differentiable with order β on interval (a,b), then, fractal-fractional derivative of order α in Caputo sense with power law kernel defined as

$$ {}_{a}^{FFDCP}D_{x}^{\alpha,\beta} g(x) = \frac{1}{\Gamma(m-\alpha)} \int_{a}^{x} (x-t)^{m-\alpha-1} \frac{dg}{dt^{\beta}} dt, \tag{1.8}$$

where $\dfrac{dg}{dt^{\beta}} = \lim\limits_{x\to t} \dfrac{g(x)-g(t)}{x^{\beta}-t^{\beta}}$, $m-1 < \alpha \leq m$, and $0 < m-1 < \beta \leq m$, $m \in \mathbb{N}$.

Since g is differentiable over (a,b), then

$$\begin{aligned} \frac{dg}{dt^{\beta}} &= \lim_{x\to t} \frac{g(x)-g(t)}{x^{\beta}-t^{\beta}}, \\ &= \frac{g'(t)}{\beta t^{\beta-1}}, \\ &= g'(t)\frac{t^{1-\beta}}{\beta}. \end{aligned} \tag{1.9}$$

1.3 Artificial Neural Network

ANN is an abstract computational model in machine learning that imitates the learning process in the organisational structure of the human brain (Guresen and Kayakutlu, 2011). It was created in 1943 by neuroscientist Warren S. McCulloch and logician Walter Pitts, who described the concept of ANN as a network of neuron cells in the brain that receive inputs, process the inputs, and produce outputs (McCulloch and Pitts, 1943). The basic components of ANN is called as artificial neuron or node. It consists of input, summing junction, activation function, bias and output as shown in Figure 1.1.

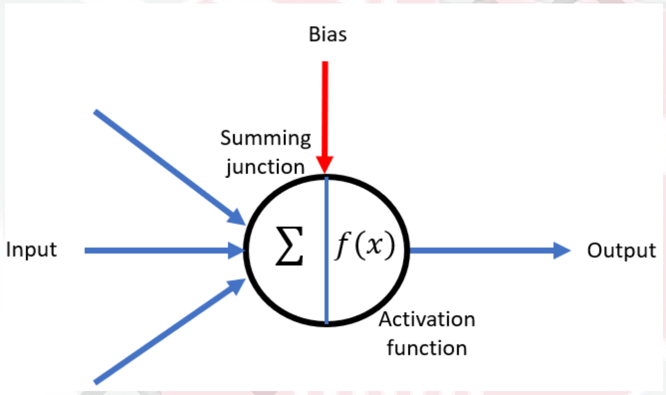


Figure 1.1: Artificial neuron/node.

1.3.1 Basic Concepts

Figure 1.2 shows a mathematical model of ANN. The circle and arrow denote the node and input flow respectively. Let x_1, x_2 and x_3 be the input of the ANN. While w_1, w_2 and w_3 are the connection weights. There also a bias denoted by b. The node for the input is labelled as a_1, a_2 and a_3 with just acceptance of the input from the outside.

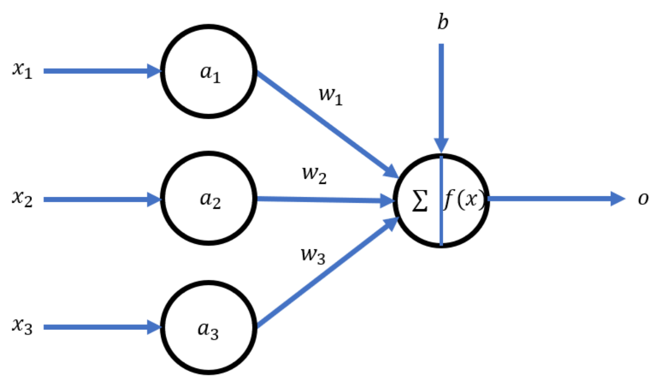


Figure 1.2: Mathematical model of ANN.

The following process, known as forward propagation is take place inside ANN:

i. The connection weight is multiplied to the input before reached to the node.

$$a_1 = w_1x_1, \tag{1.10}$$

$$a_2 = w_2x_2, \tag{1.11}$$

$$a_3 = w_3x_3, \tag{1.12}$$

ii. The collection of this weighted input are added to become weighted sum plus bias, which computed as follows:

$$\begin{aligned} v &= a_1 + a_2 + a_3 + b, \\ &= w_1x_1 + w_2x_2 + w_3x_3 + b. \end{aligned} \tag{1.13}$$

This equation can be summarized in matrix form:

$$v = wx + b. \tag{1.14}$$

where

$$w = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1.15}$$

iii. Applies the activation function to the weighted sum:

$$o = \phi(v). \tag{1.16}$$

where $\phi(\cdot)$ is the activation function.

A single node is insufficient for the practical problems, and networks with a large number of nodes are frequently used. The way in which nodes are connected determines how computations proceed and constitutes an important early design decision by neural network developer. This designation of ANN called as architecture of neural network. One of the example of the architecture of neural network can be shown in Figure 1.3.

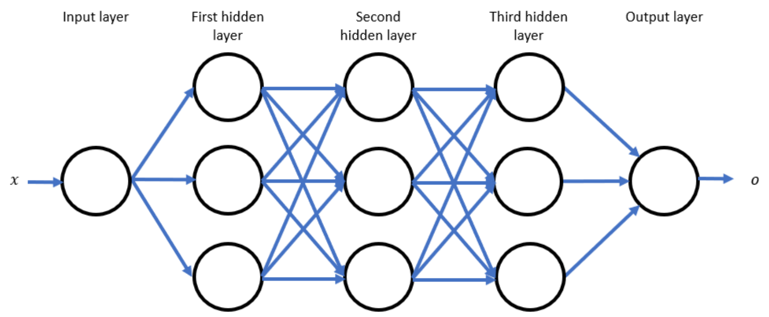


Figure 1.3: Feedforward neural network.

The group of the leftmost node is called the input layer. The input layer's nodes just act as a route for input signals to be sent to subsequent nodes. Here, the weighted sum and activation function are not calculated. The output layer, in contrast, refers to the collection of nodes at the rightmost position. The final output of the neural network is produced by these nodes. While hidden layers are those layers that exist between input and output layers which cannot be observed outside of the ANN.

In the early development of ANN, they had a very simple architecture with only an input layer and an output layer, which are called single layer ANN. The input layer does not count as a layer since it does not involve any mathematical computation in the node, rather than just receiving values from the outside. When hidden layers are added to this network, it produces a multilayer ANN. This network consists of an input layer, hidden layer(s), and an output layer. An ANN that has a single hidden layer is called a shallow neural network. A multilayer ANN that contains two or more hidden layers is called a deep neural network (DNN). The summary of the classification of ANNs based on layers can be seen in Figure 1.4.

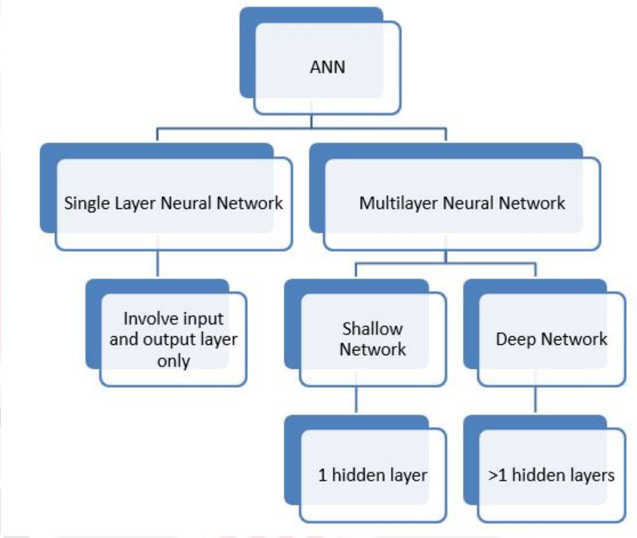

ANN
Single Layer Neural Network
Multilayer Neural Network
Involve input and output layer only
Shallow Network
Deep Network
1 hidden layer
>1 hidden layers

Figure 1.4: Types of ANN based on layer.

Additionally, ANN can be categorized based on how connections travel within linked nodes. One example is the feedforward neural network (FNN), as shown in Figure 1.3. During the data's journey, nodes in a layer receive input data from the previous layer and feed their output to the next layer. In FNN, there cannot be any route for data to travel to nodes in the same layer or the previous layer. Another type of ANN based on interconnection is known as a recurrent neural network (RNN). In this type of ANN, the output of nodes in a layer is allowed to flow through any layer. In other words, the data can travel freely either along the forward path or the backward path in any linked layer. RNN can be powerful, but they can also be extremely complicated. In fact, there are many types of ANNs, such as Hopfield networks, cellular neural networks, finite element neural networks, and so on (Yadav et al., 2015).

1.3.2 Learning Process

Learning process or training process in ANN involves several key steps:

i. *Preparation of input data*: The first step is to prepare the input data by collecting or generating a dataset that consists of sample data and corresponding target outputs. If the target output is provided, then the learning process is said to be supervised learning while if the target output is not provided the learning is said to be unsupervised learning (Kim, 2017).

ii. *Network initialization*: The next step is to define the architecture of the neural network, involving the number of layers, the number of neurons in each layer, and the types of connections between them. The network's parameters, such as weights and biases, are typically initialized with random numbers or using specific initialization techniques.

iii. *Forward propagation*: During this stage, the input data passes across the network, layer by layer, to produce an output. The activation function of each node in the network is applied to the weighted sum of its inputs. Forward propagation has been explained in Figure 1.2.

iv. *Error computation*: After forward propagation has been completed, the network's output is compared with the desired outputs from the training data. An error function is then used to measure the difference between the predicted and true outputs. Various types of error functions are available, such as the squared error function, mean squared error function, and others (Calin, 2020; Zurada, 1992).

v. *Backpropagation*: Backpropagation is technique to compute the gradients of the error function with respect to network parameters. The gradients are calculated by propagating the error backward through the network, using the basic chain rule. It can be implemented through automatic differentiation that will be discussed later.

vi. *Parameter adjustment*: The goal of the learning process in ANN is to minimize the error function by adjusting the network parameters. This situation is referred to as an optimization process in which minimization takes place. The network parameters that yield the minimum error function are called the argmin, as they are the arguments to the error function that give the minimum. To achieve this, the obtained gradient information is used to update the network's parameters. There are many types of optimization algorithms such as first-order optimization techniques, second-order optimization and heuristic optimization (Kochenderfer and Wheeler, 2019).

vii. *Epoch or iteration*: The iteration is repeated from steps iii to steps vi to allow network refine the parameters.

viii. *Hyperparameter tuning*: A hyperparameter is a parameter explicitly defined by the user to control the learning process. It is typically set manually, either by rule of thumb or through trial and error, to achieve the best optimal results in the training process. An example of a hyperparameter is the learning rate in first-order optimization techniques.

These steps are an overview of the common basic learning process that takes place in ANN. The algorithm can differ in key steps v, vii, and viii depending on the type of optimization used. The general basics of the learning process in ANN can be shown in Figure 1.5.

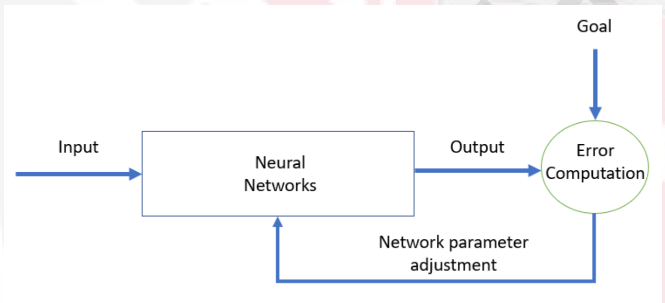


Figure 1.5: Learning process in ANN.

1.3.3 Activation Function

An activation function, also known as a transfer function, is a function that applied to the weighted sum of the input of a node to yield an output (Chakraverty and Mall, 2017). It incorporates non-linearity into the network, allowing it to learn intricate patterns and make accurate predictions. The activation function is typically employed after the linear transformation of the input data by the node's weights and biases to determine the node's output. There are a lot of different types of activation functions in the literature, but the most common ones are listed as follows:

i. Identity function:

$$\phi(x) = x. \tag{1.17}$$

ii. Sigmoid function:

$$\phi(x) = \frac{1}{1+e^{-x}}. \tag{1.18}$$

iii. Hyperbolic tangent function:

$$\phi(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}. \tag{1.19}$$

iv. Orthogonal polynomials:

- Legendre polynomial defined on the interval $[-1,1]$ can be determined with the aid of the following recurrence formula:

$$L_{i+1}(z) = \frac{2i+1}{i+1} z L_i(z) - \frac{i}{i+1} L_{i-1}(z), \quad i = 1,2,..., \tag{1.20}$$

  where $L_0(z) = 1$ and $L_1(z) = z$.

- Chebyshev polynomial defined on the interval $[-1,1]$ can be determined with the aid of the following recurrence formula:

$$T_{i+1}(z) = 2zT_i(z) - T_{i-1}(z), \quad i = 1,2,..., \tag{1.21}$$

  where $T_0(z) = 1$ and $T_1(z) = z$.

In actual practice, selecting an activation function depends on the nature of the problem. Each of them have their advantages and disadvantages, and it is usual practice to experiment with a variety of activation functions in order to identify the one that is most suitable for a certain task. For example, sigmoid function is continously differentiable, which gradient-based optimization procedure are applicable to implement it to ANN during training process.

1.3.4 Automatic Differentiation

Basically, derivatives can be computed in three ways which are manual differentiation, numerical differentiation and symbolic differentiation. Manual differentiation involves finding the derivative of a function by explicitly applying differentiation rules and algebraic manipulations to the function. It is usually done manually by hand. While numerical differentiation approximate the derivative of a function by using numerical techniques, such as finite differences. It involves discrete data points or values of a function to estimate the derivative. Symbolic differentiation involves finding the derivative of a function by manipulating its symbolic representation with the help of computer algebra systems.

Most of the learning process in ANN required the evaluation of gradient of an error function. Manual differentiation is obviously not practical and time consuming especially when deals with complicated functions. In comparison to the other options, numerical differentiation is the simplest way to implement but can be highly incorrect due to round-off and truncation errors. Besides it is not efficient and not appropriate for evaluating gradients in training of ANN that deals with many parameters (Jerrell, 1997). Symbolic differentiation aims to cover the weaknesses of the previous two methods, but often having an "expression swelling", where it refers to the phenomenon where the size or complexity of an expression grows significantly when differentiate (Corliss, 1988).

Automatic differentiation (AD) is another alternative of technique used to compute the derivatives of functions. It play an important role in many optimization algorithms and machine learning frameworks since it provides an efficient and accurate computation of gradients. AD can be defined as compositions of a finite set of elementary operations for which derivatives are known and combining the derivatives of the constituent operations through the chain rule gives the derivative of the overall composition (Baydin et al., 2018; Griewank and Walther, 2008). It is simple since it just involved the application of chain rule of basic elementary operation, i.e. addition, subtraction, division and multiplication and also elementary function, i.e. exponential, sin, cos and etc. It consists of two mode, the first one is forward mode AD and the other one is reverse-mode AD. In fact, backpropagation is the specialized version of reverse-mode AD (Baydin et al., 2018).

Consider example $f(x,y,z) = (x+y)z$. The general procedure on performing reverse-mode AD as follows:

Step 1: Identify the function that need to be differentiate and variables in the main function. Here the function have three independent variables x, y and z.

Step 2: Identify the elementary operations and functions involve inside the main function. Inside function f, there is two elementary operations which are addition between variable x and y and multiplication between $x+y$ and z.

Step 3: Choose one elementary operations involving variables in the function and then introduce new intermediate variables this operation. We let $q = x+y$ where q is a new variable.

Step 4: This intermediate variable take part in another elementary operation with other variables in the function. Then, assign the result to yet another new intermediate variable. Here, let $f = qz$ where f is another intermediate variable.

Step 5: Perform Step 4 until the last variable in the function operate with the last intermediate variable. Here, f is our last new intermediate variable.

Step 6: Sketch computational graph that has two elements: input variables and interior nodes representing operations as shown in Figure 1.6.

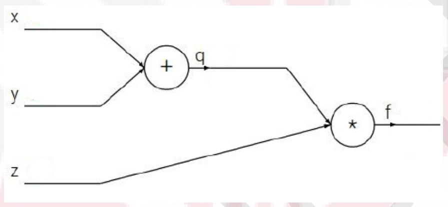


Figure 1.6: Computational graph.

Step 7: Let $x = -2$, $y = 5$ and $z = -4$. Perform forward propagation as shown in Figure 1.7.

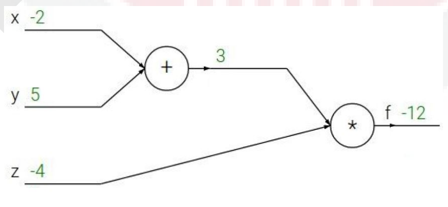


Figure 1.7: Computational graph with value at each node.

Step 8: Take the derivative of the last intermediate variable with respect to itself which the value is 1 as shown in Figure 1.8.

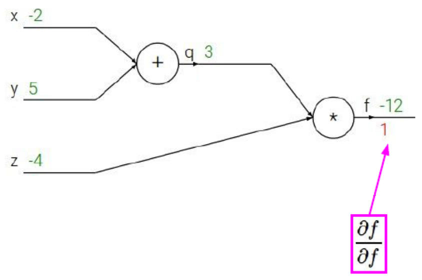


Figure 1.8: Computational graph.

Step 9: Finally, the derivative of f with respect to x, y and z can be performed by using chain rule.

The proposed steps provide a general overview of computing derivatives using reverse-mode AD. During the training process in ANN, the derivative of the error function with respect to the network parameters can be found through this procedure. Analogous to ANNs, the function f in the previous example represents the error function, where x, y, and z are the network parameters. In real situations, the form of the error function becomes very complex as the number of hidden layers increases. Symbolic differentiation is not practical for this situation as it takes much longer and faces the issue of "expression swelling". Breaking down the complexity of the error function into simple functions that involve basic operations may help expedite the computation.

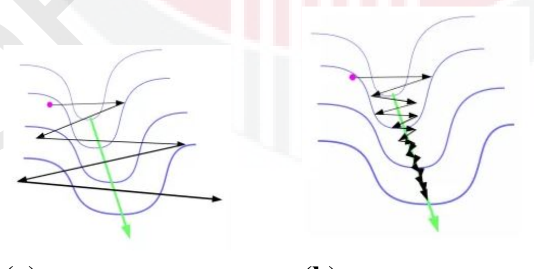
1.3.5 First-Order Optimization Methods

First-order optimization methods are one of the most widely used algorithms to optimize ANN. These methods update the network parameters in ANN iteratively based on the gradient of error function with respect to network parameters . The learning rate is the most important parameter in this method that determines the size of the update taken. Denoting n as a step taken, λ is the learning rate, E is the error function and w is the network weights, simple gradient descent (GD) can be formulated as follows

$$w^{n+1} = w^{n} - \lambda \frac{\partial E}{\partial w^{n}}, \tag{1.22}$$

where $\lambda > 0$ is learning rate.

If the learning rate is too small, then simple GD will have to go through many iterations to converge, which will take a long time. In the other way around, it makes simple GD diverge with larger values thus fail to reach the minimum. Both situation can be seen in Figure 1.9.



(a) **(b)**

Figure 1.9: Simple GD with (a) large learning rate and (b) small learning rate.

Simple GD with momentum or momentum method (MM) is a method that helps accelerate simple GD in the relevant direction and dampens oscillations as can be seen in Figure 1.10. By adding a fraction of the update vector of the past time step to the current update vector, the momentum formula update formula is

$$v^{n+1} = \gamma v^n - \lambda \frac{\partial E}{\partial w^n}, \tag{1.23}$$

$$w^{n+1} = w^n + v^{n+1}. \tag{1.24}$$

The momentum term that denoted by γ is usually set to 0.9 (Ruder, 2016). When this value become zero, it will return back to simple GD.

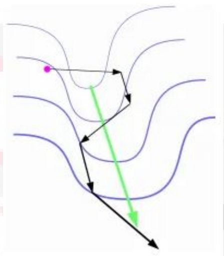


Figure 1.10: Momentum dampens the oscillation.

Simple GD and MM update the w with the constant or same learning rate. The adaptive subgradient method, or Adagrad, use different learning rate for network parameters based on the historical gradients. The Adagrad update formula is

$$v^n = v^{n-1} + \left(\frac{\partial E}{\partial w^n}\right)^2, \tag{1.25}$$

$$w^{n+1} = w^n - \frac{\lambda}{\sqrt{v^n + \varepsilon}} \frac{\partial E}{\partial w^n}, \tag{1.26}$$

where ε is small number usually chosen as 1×10^{-8} to avoid division by zero (Kochenderfer and Wheeler, 2019). From this formula, the learning rate has been change in such a way that it will decrease due to the summation of the previous square gradient for every iteration. However, the value of the square root at the denominator will be accumulated which then become large causes the learning rate become infinitesimally small. Thus the algorithm not able to make adjustment on the network parameters.

Root mean squared propagation or RMSProp extends Adagrad to avoid the effect of a monotonically decreasing learning rate. RMSProp maintains a decaying average of squared gradients, thus the formula can be updated as follows:

$$v^n = \sigma_1 v^{n-1} + (1-\sigma_1)\left(\frac{\partial E}{\partial w^n}\right)^2, \tag{1.27}$$

$$w^{n+1} = w^n - \frac{\lambda}{\sqrt{v^n + \varepsilon}}\frac{\partial E}{\partial w^n}, \tag{1.28}$$

where $\sigma_1 \in (0,1)$ is the forgetting factor which control the exponential decay rate, v is second moment of the gradient that initialized to be zero and ε is small number to avoid division by zero.

The adaptive moment estimation method, or Adam also adapts learning rates that stores both an exponentially decaying squared gradient like RMSProp, but also an exponentially decaying gradient like momentum. Initializing the gradient and squared gradient to zero introduces a bias. A bias correction step helps alleviate the issue. The Adam formula are:

$$\text{Biased decaying momentum}: v^n = \sigma_1 v^{n-1} + (1-\sigma_1)\frac{\partial E}{\partial w^n}, \tag{1.29}$$

$$\text{Biased decaying squared gradient}: u^n = \sigma_2 u^{n-1} + (1-\sigma_2)\left(\frac{\partial E}{\partial w^n}\right)^2, \tag{1.30}$$

$$\text{corrected decaying momentum}: \hat{v}^n = \frac{v_n}{1-\sigma_1}, \tag{1.31}$$

$$\text{corrected decaying squared gradient}: \hat{u}^n = \frac{u_n}{1-\sigma_2}, \tag{1.32}$$

$$\text{new update}: w^{n+1} = w^n - \frac{\lambda}{\sqrt{\hat{u}^n + \varepsilon}}\hat{v}^n, \tag{1.33}$$

where $\sigma_1, \sigma_2 \in (0,1)$ is the decay rates for the moment estimates and v and u biased estimate for the first moment and second moment of the gradient which both initialized to be zero.

1.3.6 Broyden-Fletcher-Goldfarb-Shanno Optimization Method

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method is one of the quasi-Newton algorithm categorized under second-order optimization methods. Compared from first-order optimization methods discussed in the previous section, second-order optimization methods provide additional second-order information for better training trajectory across the local curvature of error function in order to make the right step size to reach a local minimum. The fundamental of quasi-Newton methods is based on Newton method where the computation of true Hessian matrix or square matrix of second-order partial derivatives of a scalar-valued function is replaced by an approximation of Hessian to make it more practical in terms of simplicity and computational time.

Consider a vector function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with $m \geq n$. The goal is to minimized the vector function to find the optimal parameter, $\mathbf{w}^*$ from weights, $\mathbf{w} = [w_1, w_2, ...w_n]$ such that

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\{F(\mathbf{w})\}, \tag{1.34}$$

where

$$F(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\mathbf{w}))^2 = \frac{1}{2}||\mathbf{f}(\mathbf{w})||^2 = \frac{1}{2}\mathbf{f}^{\mathbf{T}}(\mathbf{w})\mathbf{f}(\mathbf{w}). \tag{1.35}$$

Provided that $\mathbf{f}$ has continous partial derivatives, the Taylor series for f can be written as

$$\mathbf{f}(\mathbf{w}+\mathbf{h}) = \mathbf{f}(\mathbf{w}) + \mathbf{J}(\mathbf{w})\mathbf{h} + \text{ higher order terms,} \tag{1.36}$$

where $\mathbf{J}(\mathbf{w}) \in \mathbb{R}^{m\times n}$ is the Jacobian. This matrix contain the first partial derivatives of the function components,

$$\mathbf{J}(\mathbf{w})_{ij} = \frac{\partial f_i}{\partial w_j}(\mathbf{w}). \tag{1.37}$$

As can be seen in (1.35), $F : \mathbb{R}^n \to \mathbb{R}$. Then, the first derivative or gradient of F is given by

$$\frac{\partial F}{\partial w_j}(\mathbf{w}) = \sum_{i=1}^{m} f_i(\mathbf{w}) \frac{\partial f_i}{\partial w_j}(\mathbf{w}), \tag{1.38}$$

that can be rewrite as

$$\frac{\partial \mathbf{F}}{\partial \mathbf{w}}(\mathbf{w}) = \mathbf{J}(\mathbf{w})^T \mathbf{f}(\mathbf{w}), \tag{1.39}$$

also from (1.38), the second derivative of F is given by

$$\frac{\partial^2 F}{\partial w_j \partial w_k}(\mathbf{w}) = \sum_{i=1}^{m} \left(\frac{\partial f_i}{\partial w_j}(\mathbf{w}) \frac{\partial f_i}{\partial w_k}(\mathbf{w}) + f_i(x) \frac{\partial^2 f_i}{\partial w_j \partial w_k}(\mathbf{w}) \right), \tag{1.40}$$

that can be rewrite as Hessian matrix, **H**

$$\mathbf{H} = \mathbf{J}(\mathbf{w})^T \mathbf{J}(\mathbf{w}) + \sum_{i=1}^{m} f_i(w) \mathbf{f}_i''(\mathbf{w}). \tag{1.41}$$

The weight updating formula for the Newton method is given as

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \lambda \mathbf{H}(\mathbf{w}_k)^{-1} \frac{\partial \mathbf{F}}{\partial \mathbf{w}_k}(\mathbf{w}_k), \tag{1.42}$$

where λ is the learning rate is set by default as 1. However, computing the inverse

Hessian is very expensive.

The variants under second-order optimization lies on updating this approximation of Hessian matrix, in replace with the exact Hessian matrix. Besides, the matrix must be symmetric and positive definite so that it is nonsingular (Rafati and Marica, 2020; Tan and Lim, 2019). Since $\mathbf{H}(\mathbf{w}_k) = \mathbf{H}_k$, it can be computed at each iteration through the following approximation

$$\mathbf{H}_k = \mathbf{H}_{k-1} + \frac{\phi\phi^T}{\phi^T\delta} - \frac{\mathbf{H}_{k-1}\delta(\mathbf{H}_{k-1}\delta)^T}{\delta^T\mathbf{H}_{k-1}\delta} \tag{1.43}$$

where $\delta = \mathbf{w}_k - \mathbf{w}_{k-1} =$ and $\phi = \frac{\partial \mathbf{F}}{\partial \mathbf{w}_k}(\mathbf{w}_k) - \frac{\partial \mathbf{F}}{\partial \mathbf{w}_{k-1}}(\mathbf{w}_{k-1})$. By finding this inverse, the following update of $\mathbf{w}_{k+1}$ in (1.42) can be obtained.

Alternatively, the inverse of Hessian matrix also can be approximate directly through BFGS. Let $\mathbf{H}(\mathbf{w}_k)^{-1} = \mathbf{Q}_k$. The inverse Hessian matrix can computed at each iteration through

$$\mathbf{Q}_k = \left(I - \frac{\delta\phi^T}{\phi^T\delta}\right)\mathbf{Q}_{k-1}\left(I - \frac{\phi\delta^T}{\phi^T\delta}\right) + \frac{\delta\phi^T}{\phi^T\delta}, \tag{1.44}$$

Substituting into the update equation in (1.42) yields

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \lambda\mathbf{Q}_k\frac{\partial \mathbf{F}}{\partial \mathbf{w}_k}(\mathbf{w}_k). \tag{1.45}$$

1.3.7 Extreme Learning Machine Algorithm

Extreme Learning Machine algorithm (ELM) is a ML algorithm that introduced by Huang et al. (2006), provide an en extremely fast and efficient scheme as it categorize under optimization free approach. The objective of ELM is to find the weights and biases from the hidden layer to the output layer, while the other network parameters are randomly initialized and remain fixed. ELM is non-iterative process that involve the implementation of Moore-Penrose generalized inverse during finding the unknown network parameters.

Let input $\mathbf{x} = (x_1, ..., x_i)^T$, where x_i denotes the input value, o_j denotes the jth output of the FNN, κ_i is the ith weight between the output and the hidden layer, a_i is the weight between input layer and hidden layer and b_i is the bias in ith hidden layer, then FNN can be expressed as:

$$\sum_{i=1}^{N} \kappa_i g(w_i x_j + b_i) = o_j, \quad j = 1, ..., d. \tag{1.46}$$

If target $\mathbf{t} = (t_1, ..., t_d)^T$, where t denotes the target with N hidden nodes and d is the number of training data, the aim of ELM is to minimize error between target and output of the FNN by minimizing the following objective function:

$$E = \sum_{j=1}^{d} (o_j - t_j)^2. \tag{1.47}$$

According to Huang et al. (2006), FNN with one hidden layer able to approximate all training data with zero error,

$$\sum_{j=1}^{d} ||o_j - t_j|| = 0, \tag{1.48}$$

so there exist a set of w_i, b_i and κ_i that satisfy

$$\sum_{i=1}^{N} \kappa_i g\left(w_i x_j + b_i\right) = t_j, \quad j = 1, ..., d. \tag{1.49}$$

The above formula can be reformulated as linear system

$$\mathbf{H}\kappa = \mathbf{T}, \tag{1.50}$$

where

$$\mathbf{H}(w_1,...,w_N,b_1,...,b_N,x_1,...,x_d) = \begin{bmatrix} g(w_1x_1+b_1) & \cdots & g(w_Nx_1+b_N) \\ \vdots & \cdots & \vdots \\ g(w_1x_d+b_1) & \cdots & g(w_Nx_d+b_N) \end{bmatrix}, \tag{1.51}$$

$$\kappa = \begin{bmatrix} \kappa_1 \\ \vdots \\ \kappa_N \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}. \tag{1.52}$$

It cannot guarantee that the matrix **H** are non-degenerate or square matrix, thus it is impossible to find its inverse. Hence by using minimal norm least square solution of the system. Then, the following solution of κ can be obtained

$$\kappa = \mathbf{H}^{\dagger}\mathbf{T}. \tag{1.53}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse matrix.

1.4 Statement of the Problem

Selecting appropriate configurations in architecture of FNN, such as the number of hidden layers, when solving differential equations may enhance solution accuracy. However, it is noteworthy that these investigations have mainly concentrated on ODEs. The question of whether including more than one hidden layer in FNN is necessary for solving FDEs remains unanswered.

Besides, a critical concern when working with FNN with more than one hidden layer is computational time, due to the iterative looping statements used in computer programs. The increased number of hidden layers results in a significant increase in execution time, often lasting for hours.

In aspect of optimization, there is less concern on realizing the capability of first-order optimization method on solving FDEs. Since there are many types of them, it is necessary to select only several its variant and compare their performances. This style of study has not yet been performed in any previous of studies, not only in ODEs but also in FDEs.

Next, a specific variant of second-order optimization methods, namely the BFGS method, has proven effective in solving FDEs in the Caputo sense when employed with FNNs in a single hidden layer. However, the effectiveness of this optimization method in solving FDEs remains a subject of inquiry when dealing with FNNs equipped with two hidden layers.

So far, the suggested optimizations are iterative. An alternative is the ELM, a non-iterative approach demonstrating significant results in terms of accuracy and computational efficiency in solving ODEs. However, the potential of ELM in solving FDEs using FNN remains unexplored. Additionally, the combined impact of implementing orthogonal polynomials as a hidden layer along with ELM for solving FDEs is unknown and requires further exploration.

1.5 Objective of the Study

In this study, the aim of the research is to propose several schemes based on one of the variants in ANN known as FNN to solve FDEs and FFDEs. Several objectives of this research are highlighted as follows:

i. To develop FNN with vectorized algorithm (FNNVA) for solving FDEs in Caputo sense (FDEsC) using first-order optimization techniques.

ii. To construct a single hidden layer of FNN based on Chelyshkov Polynomial (SHLFNNCP) for solving FDEs in Caputo sense (FDEsC) using ELM.

iii. To design extended single hidden layer of FNN (ESHLFNN) for solving FDEs in Caputo-Fabrizio sense (FDEsCF) using BFGS method.

iv. To extend FNN in two hidden layers with vectorized algorithm (FNN2HLVA) for solving FFDEs in Caputo sense with power law kernel (FFDEsCP) using adaptive moment estimation method (Adam).

1.6 Scope of the Study

This research is focused to solve initial value problem (IVP) for both FDEs and FFDEs. In the aspect of ANN architecture, this research consider fully connected feedforward neural network, in which the information or value from the input layer is in forward pass. There is no path of value going to the node on the same layer or previous layer. Besides, unsupervised learning is considered as there is no targeted solution considered.

1.7 Significance of the Study

The proposed schemes prioritize speed and efficiency in addressing FDEsC and FFDEsCP by emphasizing a vectorized algorithm. This algorithm employs vector and matrix computations to reduce computational load, particularly in FNN parts like forward propagation, avoiding conventional looping statements. Additionally, the incorporation of Hessian information via the BFGS method, as the third objective, aims to speed up the training process by reducing the required iterations. Notably, the scheme associated with the second objective eliminates the need for an iterative optimization procedure in adjusting network parameters, leading to a clear reduction in computational time. In summary, these schemes offer time savings and user-friendly solutions for solving FDEs and FFDEs.

Furthermore, traditional numerical methods typically provide a discrete solution, requiring repetitive algorithm runs for different step sizes, which obviously time inefficient process. The proposed schemes in all objectives, however, naturally generate continuous solutions. These solutions extend beyond discretized points, encompassing values within reasonable absolute error from the exact solution. Consequently, researchers avoid the time-consuming repetition of procedures.

Last but not least, the implementation of proposed scheme for all the objectives are not restricted for only specific type of problems but it also can be extended to various type of them such as multi-order FDEs or system of FFDEs. Consequently, this versatility allows researchers to save a considerable amount of time and effort, as they need only understand and implement one method to address various types of problems, as opposed to grappling with different numerical techniques for each distinct problem they encounter.

1.8 Organization of the Thesis

The thesis is organized as follows:

In Chapter 2, a historical reviews of fractional calculus is presented. Then, the emergence of fractal-fractional calculus is described. Next, a review regarding on ANN on solving differential equation is briefly discussed. The topic covers thereotical studies involving ANN as a universal approximator, ANN on solving ODEs, ANN for solving FDEs and ANN for solving FFDEs. The theoretical difficulties regarding on the convergences and stability of ANN is also highlighted. Finally, research gap is discussed.

Chapter 3 describes the new numerical scheme based on FNN to solve FDEsC using selected first-order optimization techniques which are simple GD, MM and Adam. At the first stage, architecture of FNN with many hidden layers is exposed. Then, the methodology of the scheme is briefly discussed through two stages: method formulation and vectorized algorithm. The method formulation involved the form of FDEC that will be solved, the approximation of Caputo fractional derivatives and learning algorithm. The vectorized algorithm is then designed to make the training process work efficiently. Four problems involving FDEsC then are listed. Finally, the designed scheme is implement on these problems to show the effectiveness of the designed scheme through investigation on different number of nodes, hidden layer, learning rate and first-order optimization techniques that has been extensively discussed based on the presented numerical results.

Chapter 4 discusses the newly single hidden layer of FNN based on Chelyshov polynomial for solving FDEsC using ELM. A basic preliminaries involve in this chapter is listed at the beginning of this chapter including the definition of Chelyshkov polynomials and some properties of the Caputo derivative. The architecture of this type of FNN that make use Chelyshkov polynomials as activation function is then presented. The methodology for solving specific type of FDEC using the proposed scheme is then briefly explained. To investigate the applicability and performance of the proposed method, five real world application problems are solved. The discussion has been discussed at the end of the chapter based on the numerical results obtained.

Chapter 5 introduces a new scheme based on extended single layer of FNN for solving FDEsCF using BFGS method. At first, an architecture of extended single layer of FNN with appropriate selection of activation function is presented. Then, the proposed scheme is ddiscussed through three stages. At first stage, the method formulation involving type of FDECF, construction of approximation solution and approximation of Caputo-Fabrizio derivative is derived. Then, the vectorization is derived for forward propagation, approximate solution and error function. In the last stage, the learning solver that use BFGS method is described. Finally, five problems are presented and tested with the scheme alongside with the discussion.

Chapter 6 present new numerical scheme based on FNN in two hidden layer using Adam to solve FFDEsCP. At initial stage, the architecture of FNN in two hidden layer is presented. This include the forward propagation and activation function used in the architecture. Then, the methodology involving method framework and vectorized algorithm is briefly explained. Three problems involving FFDEsCP are presented to investigate the applicability of the proposed scheme. Finally, the numerical results is provided and discussed.

Finally, the overall discussion and finding in the thesis is concluded in the Chapter 7. At first, several conclusions can be drawn from this work is highlighted. Then, some potential future works that could be expanded further are suggested to interested researcher in the recommendation section.

REFERENCES

Akgül, A. (2021). Analysis and New Applications of Fractal Fractional Differential Equations With Power Law Kernel. *Discrete & Continuous Dynamical Systems-S*, 14(10):3401.

Akgül, A., Inc, M., Karatas, E., and Baleanu, D. (2015). Numerical Solutions of Fractional Differential Equations of Lane-Emden Type by an Accurate Technique. *Advances in Difference Equations*, 2015(1):1–12.

Al-Sharif, M., Ahmed, A., and Salim, M. (2020). An Integral Operational Matrix of Fractional-Order Chelyshkov Functions and Its Applications. *Symmetry*, 12(11):1755.

Allwright, A. and Atangana, A. (2018). Fractal Advection-Dispersion Equation for Groundwater Transport in Fractured Aquifers With Self-Similarities. *The European Physical Journal Plus*, 133:1–20.

Arshad, S., Saleem, I., Akgül, A., Huang, J., Tang, Y., and Eldin, S. M. (2023). A Novel Numerical Method for Solving the Caputo-Fabrizio Fractional Differential Equation. *AIMS Math*, 8:9535–9556.

Aruldoss, R. and Balaji, K. (2022). Numerical Inversion of Laplace Transform via Wavelet Operational Matrix and Its Applications to Fractional Differential Equations. *International Journal of Applied and Computational Mathematics*, 8(1):1–17.

Atangana, A. (2017). Fractal-Fractional Differentiation and Integration: Connecting Fractal Calculus and Fractional Calculus to Predict Complex System. *Chaos, Solitons & Fractals*, 102:396–406.

Atangana, A. and Baleanu, D. (2016). New Fractional Derivatives with Nonlocal and Non-Singular Kernel: Theory and Application to Heat Transfer Model. *arXiv preprint arXiv:1602.03408*.

Atangana, A. and Qureshi, S. (2019). Modeling Attractors of Chaotic Dynamical Systems With Fractal–Fractional Operators. *Chaos, Solitons & Fractals*, 123:320–337.

Bagley, R. L. and Torvik, P. J. (1983). Fractional Calculus-A Different Approach to the Analysis of Viscoelastically Damped Structures. *American Institute of Aeronautics and Astronautics*, 21(5):741–748.

Barron, A. R. (1993). Universal Approximation Bounds for Superpositions of a Sigmoidal Function. *IEEE Transactions on Information Theory*, 39(3):930–945.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: A Survey. *Journal of Marchine Learning Research*, 18:1–43.

Bengio, Y. (2012). *Practical Recommendations for Gradient-Based Training of Deep Architectures*. Springer, Berlin, Heidelberg.

Berg, J. and Nyström, K. (2018). A Unified Deep Artificial Neural Network Approach to Partial Differential Equations in Complex Geometries. *Neurocomputing*, 317:28–41.

Brouers, F. et al. (2014). The Fractal (BSf) Kinetics Equation and Its Approximations. *Journal of Modern Physics*, 5(16):1594.

Brouers, F. and Sotolongo-Costa, O. (2006). Generalized Fractal Kinetics in Complex Systems (Application to Biophysics and Biotechnology). *Physica A: Statistical Mechanics and its Applications*, 368(1):165–175.

Burden, R. L., Faires, J. D., and Burden, A. M. (2015). *Numerical Analysis*. Cengage learning.

Burnett, D. (1987). *Finite Element Analysis: From Concepts to Applications Solutions Manual*. Addison-Wesley Longman.

Cai, W., Chen, W., and Xu, W. (2018). The Fractal Derivative Wave Equation: Application to Clinical Amplitude/velocity Reconstruction Imaging. *The Journal of the Acoustical Society of America*, 143(3):1559–1566.

Calin, O. (2020). *Deep Learning Architectures*. Springer.

Cao, J., Wang, Z., and Xu, C. (2020). A High-Order Scheme for Fractional Ordinary Differential Equations With the Caputo–Fabrizio Derivative. *Communications on Applied Mathematics and Computation*, 2(2):179–199.

Caputo, M. and Fabrizio, M. (2015). A New Definition of Fractional Derivative Without Singular Kernel. *Progress in Fractional Differentiation & Applications*, 1(2):73–85.

Chakraverty, S. and Mall, S. (2017). *Artificial Neural Networks for Engineers and Scientists: Solving Ordinary Differential Equations*. CRC Press.

Chelyshkov, V. S. (2006). Alternative Orthogonal Polynomials and Quadratures. *Electron. Trans. Numer. Anal*, 25(7):17–26.

Chen, W. (2006). Time–Space Fabric Underlying Anomalous Diffusion. *Chaos, Solitons & Fractals*, 28(4):923–929.

Chen, W., Sun, H., Zhang, X., and Korošak, D. (2010a). Anomalous Diffusion Modeling by Fractal and Fractional Derivatives. *Computers & Mathematics with Applications*, 59(5):1754–1758.

Chen, W., Zhang, X.-D., and Korošak, D. (2010b). Investigation on Fractional and Fractal Derivative Relaxation-Oscillation Models. *International Journal of Nonlinear Sciences and Numerical Simulation*, 11(1):3–10.

Corliss, G. F. (1988). *Application of Differentiation Arithmetic*. Academic Press, Boston.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Datta, L. (2020). A Survey on Activation Functions and Their Relation With Xavier and He Normal Initialization. *arXiv preprint arXiv:2004.06632*.

Debnath, L. (2004). A Brief Historical Introduction to Fractional Calculus. *International Journal of Mathematical Education in Science and Technology*, 35(4):487–501.

Deshi, A. and Gudodagi, G. (2021). Numerical Solution of Bagley–Torvik, Nonlinear and Higher Order Fractional Differential Equations using Haar Wavelet. *SeMA Journal*, 79(4):663–675.

Diethelm, K. (2010). *The Analysis of Fractional Differential Equations: An Application-Oriented Exposition Using Differential Operators of Caputo Type*. Lecture Notes in Mathematics. Springer Berlin Heidelberg.

Diethelm, K., Ford, N. J., and Freed, A. D. (2002). A Predictor-Corrector Approach for the Numerical Solution of Fractional Differential Equations. *Nonlinear Dynamics*, 29(1):3–22.

Domingues, J. C. (2005). *Landmark Writings in Western Mathematics 1640-1940*. Elsevier Science.

Dormand, J. R. (1996). *Numerical Methods for Differential Equations: A Computational Approach*, volume 3. CRC press.

Dufera, T. T. (2021). Deep Neural Network for System of Ordinary Differential Equations: Vectorized Algorithm and Simulation. *Machine Learning with Applications*, 5:100058.

Emden, R. (1907). *Gaskugeln: Anwendungen der mechanischen Wärmetheorie auf kosmologische und meteorologische Probleme*. BG Teubner.

Euler, L. (1738). De progressionibus transcendentibus seu quarum termini generales algebraice dari nequeunt. *Commentarii Academiae Scientiarum Petropolitanae*, 5:36–57.

Fitt, A., Goodwin, A., Ronaldson, K., and Wakeham, W. (2009). A Fractional Differential Equation for a MEMS Viscometer Used in the Oil Industry. *Journal of Computational and Applied Mathematics*, 229(2):373–381.

Ford, N. J. and Simpson, A. C. (2001). The Numerical Solution of Fractional Differential Equations: Speed Versus Accuracy. *Numerical Algorithms*, 26(4):333–346.

Fouladi, S., Ebadi, M., Safaei, A. A., Bajuri, M. Y., and Ahmadian, A. (2021). Efficient Deep Neural Networks for Classification of COVID-19 Based on CT Images: Virtualization via Software Defined Radio. *Computer Communications*, 176:234–248.

Fourier, J. B. J., Darboux, G., et al. (1822). *Théorie analytique de la chaleur*, volume 504. Didot Paris.

Glorot, X. and Bengio, Y. (2010). Understanding The Difficulty of Training Deep Feedforward Neural Networks. *Journal of Machine Learning Research - Proceedings Track*, 9:249–256.

Gómez-Aguilar, J. and Atangana, A. (2021). New Chaotic Attractors: Application of Fractal-Fractional Differentiation and Integration. *Mathematical Methods in the Applied Sciences*, 44(4):3036–3065.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT press.

Griewank, A. and Walther, A. (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM.

Guirao, J. L., Sabir, Z., Raja, M. A. Z., and Baleanu, D. (2022). Design of Neuro-Swarming Computational Solver for the Fractional Bagley–Torvik Mathematical Model. *The European Physical Journal Plus*, 137(2):245.

Guresen, E. and Kayakutlu, G. (2011). Definition of Artificial Neural Networks With Comparison to Other Networks. *Procedia Computer Science*, 3:426–433.

Haber, E. and Ruthotto, L. (2017). Stable Architectures for Deep Neural Networks. *Inverse Problems*, 34(1):014004.

Hamid, M., Usman, M., Haq, R., and Wang, W. (2020). A Chelyshkov Polynomial Based Algorithm to Analyze the Transport Dynamics and Anomalous Diffusion in Fractional Model. *Physica A: Statistical Mechanics and Its Applications*, 551:124227.

Hayou, S., Doucet, A., and Rousseau, J. (2019). On the Impact of the Activation Function on Deep Neural Networks Training. *Proceedings of Machine Learning Research*, 97:2672–2680.

He, J.-H. (2018). Fractal Calculus and Its Geometrical Explanation. *Results in Physics*, 10:272–276.

Heydari, M. H., Atangana, A., and Avazzadeh, Z. (2021). Chebyshev Polynomials for the Numerical Solution of Fractal–Fractional Model of Nonlinear Ginzburg–Landau Equation. *Engineering with Computers*, 37(2):1377–1388.

Hoffer, E., Banner, R., Golan, I., and Soudry, D. (2018). Norm Matters: Efficient and Accurate Normalization Schemes in Deep Networks. *Advances in Neural Information Processing Systems*, 31.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks*, 2(5):359–366.

Hornik, K., Stinchcombe, M., and White, H. (1990). Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks. *Neural Networks*, 3(5):551–560.

Hu, Z. and Tu, X. (2015). A New Discrete Economic Model Involving Generalized Fractal Derivative. *Advances in Difference Equations*, 2015:1–11.

Huang, G., Huang, G.-B., Song, S., and You, K. (2015). Trends in Extreme Learning Machines: A Review. *Neural Networks*, 61:32–48.

Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2006). Extreme Learning Machine: Theory and Applications. *Neurocomputing*, 70(1-3):489–501.

Izadi, M. (2020). Fractional Polynomial Approximations to the Solution of Fractional Riccati Equation. *Punjab University Journal of Mathematics*, 51(11).

Izadi, M. (2021). Comparison of Various Fractional Basis Functions for Solving Fractional-Order Logistic Population Model. *Facta Universitatis, Series: Mathematics and Informatics*, 35(4):1181–1198.

Jafarian, A., Mokhtarpour, M., and Baleanu, D. (2017). Artificial Neural Network Approach for a Class of Fractional Ordinary Differential Equation. *Neural Computing and Applications*, 28(4):765–773.

Jafarian, A., Nia, S. M., Golmankhaneh, A. K., and Baleanu, D. (2018). On Artificial Neural Networks Approach With New Cost Functions. *Applied Mathematics and Computation*, 339:546–555.

Jerrell, M. E. (1997). Automatic Differentiation and Interval Arithmetic for Estimation of Disequilibrium Models. *Computational Economics*, 10:295–316.

Karlsson, L. and Bonde, O. (2020). *A Comparison of Selected Optimization Methods for Neural Networks*. Degree Project in Technology.

Kazemi, B. F. and Jafari, H. (2017). Error Estimate of the MQ-RBF Collocation Method for Fractional Differential Equations With Caputo–Fabrizio Derivative. *Mathematical Sciences*, 11(4):297–305.

Kilbas, A. A., Srivastava, H. M., and Trujillo, J. J. (2006). *Theory and Applications of Fractional Differential Equations*. Elsevier Science.

Kim, P. (2017). *Matlab Deep Learning*. Apress Berkeley, CA.

Kochenderfer, M. J. and Wheeler, T. A. (2019). *Algorithms for Optimization*. Mit Press.

Kosmidis, K. and Macheras, P. (2018). On the Dilemma of Fractal or Fractional Kinetics in Drug Release Studies: A Comparison Between Weibull and Mittag-Leffler Functions. *International Journal of Pharmaceutics*, 543(1-2):269–273.

Kumar, S., Kumar, A., and Odibat, Z. M. (2017). A Nonlinear Fractional Model to Describe the Population Dynamics of Two Interacting Species. *Mathematical Methods in the Applied Sciences*, 40(11):4134–4148.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000.

Lane, H. J. (1870). On the Theoretical Temperature of the Sun, Under the Hypothesis of a Gaseous Mass Maintaining Its Volume by Its Internal Heat, and Depending on the Laws of Gases as Known to Terrestrial Experiment. *American Journal of Science*, 2(148):57–74.

Laplace, P. S. (2012). *A Philosophical Essay on Probabilities*. Courier Corporation.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. R. (2002). *Efficient Backprop*. Springer, Berlin, Heidelberg.

Lee, H. and Kang, I. S. (1990). Neural Algorithm for Solving Differential Equations. *Journal of Computational Physics*, 91(1):110–131.

Li, C. and Zeng, F. (2015). *Numerical Methods for Fractional Calculus*, volume 24. CRC Press.

Liang, Y., Allen, Q. Y., Chen, W., Gatto, R. G., Colon-Perez, L., Mareci, T. H., and Magin, R. L. (2016). A Fractal Derivative Model for the Characterization of Anomalous Diffusion in Magnetic Resonance Imaging. *Communications in Nonlinear Science and Numerical Simulation*, 39:529–537.

Malek, A. and Beidokhti, R. S. (2006). Numerical Solution for High Order Differential Equations Using a Hybrid Neural Network—Optimization Method. *Applied Mathematics and Computation*, 183(1):260–271.

Mall, S. and Chakraverty, S. (2013). Comparison of Artificial Neural Network Architecture in Solving Ordinary Differential Equations. *Advances in Artificial Neural Systems*, 2013.

Mall, S. and Chakraverty, S. (2014). Chebyshev Neural Network Based Model for Solving Lane–Emden Type Equations. *Applied Mathematics and Computation*, 247:100–114.

Mall, S. and Chakraverty, S. (2016). Application of Legendre Neural Network for Solving Ordinary Differential Equations. *Applied Soft Computing*, 43:347–356.

Mall, S. and Chakraverty, S. (2017). Single Layer Chebyshev Neural Network Model for Solving Elliptic Partial Differential Equations. *Neural Processing Letters*, 45(3):825–840.

Mall, S. and Chakraverty, S. (2018). Artificial Neural Network Approach for Solving Fractional Order Initial Value Problems. *arXiv preprint arXiv:1810.04992*.

McCulloch, W. S. and Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Meade Jr, A. J. and Fernandez, A. A. (1994). The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks. *Mathematical and Computer Modelling*, 19(12):1–25.

Mechee, M. S. and Senu, N. (2012). Numerical Study of Fractional Differential Equations of Lane-Emden Type by Method of Collocation. *Applied Mathematics*, 3(8):851–856.

Mekkaoui, T., Atangana, A., and Araz, S. İ. (2021). Predictor–Corrector for Non-Linear Differential and Integral Equation With Fractal–Fractional Operators. *Engineering with Computers*, 37(3):2359–2368.

Michoski, C., Milosavljević, M., Oliver, T., and Hatch, D. R. (2020). Solving Differential Equations Using Deep Neural Networks. *Neurocomputing*, 399:193–212.

Milici, C., Drăgănescu, G., and Machado, J. T. (2018). *Introduction to Fractional Differential Equations*. Springer Cham.

Miller, K. S. and Ross, B. (1993). *An Introduction to The Fractional Calculus and Fractional Differential Equations*. Wiley.

Minai, A. A. and Williams, R. D. (1993). On the Derivatives of the Sigmoid. *Neural Networks*, 6(6):845–853.

Nottale, L. (1988). On Time in Microphysics. *Academie des Sciences Paris Comptes Rendus Serie Sciences Mathematiques*, 306(5):341–346.

Odibat, Z. and Momani, S. (2008). Modified Homotopy Perturbation Method: Application to Quadratic Riccati Differential Equation of Fractional Order. *Chaos, Solitons & Fractals*, 36(1):167–174.

Pakdaman, M., Ahmadian, A., Effati, S., Salahshour, S., and Baleanu, D. (2017). Solving Differential Equations of Fractional Order Using an Optimization Technique Based on Training Artificial Neural Network. *Applied Mathematics and Computation*, 293:81–95.

Panghal, S. and Kumar, M. (2021a). Optimization Free Neural Network Approach for Solving Ordinary and Partial Differential Equations. *Engineering with Computers*, 37(4):2989–3002.

Panghal, S. and Kumar, M. (2021b). Neural network method: delay and system of delay differential equations. *Engineering with Computers*, 38(Suppl 3):2423–2432.

Parisi, D. R., Mariani, M. C., and Laborde, M. A. (2003). Solving Differential Equations With Unsupervised Neural Networks. *Chemical Engineering and Processing: Process Intensification*, 42(8-9):715–721.

Pinkus, A. (1999). Approximation Theory of the MLP Model in Neural Networks. *Acta Numerica*, 8:143–195.

Podlubny, I. (1998). *Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications*. Elsevier.

Podlubny, I., Magin, R. L., and Trymorush, I. (2017). Niels Henrik Abel and the birth of fractional calculus. *Fractional Calculus and Applied Analysis*, 20(5):1068–1075.

Qu, H. and Liu, X. (2015). A Numerical Method for Solving Fractional Differential Equations by Using Neural Network. *Advances in Mathematical Physics*, 2015:1–12.

Rafati, J. and Marica, R. F. (2020). Quasi-Newton Optimization Methods for Deep Learning Applications. *Deep Learning Applications*, 1098:9–38.

Raissi, M., Perdikaris, P., and Karniadakis, G. (2019). A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations. *J. Comput. Phys*, 378:686–707.

Raja, M. A. Z., Khan, J. A., and Qureshi, I. M. (2010a). A New Stochastic Approach for Solution of Riccati Differential Equation of Fractional Order. *Annals of Mathematics and Artificial Intelligence*, 60(3):229–250.

Raja, M. A. Z., Khan, J. A., and Qureshi, I. M. (2010b). Heuristic Computational Approach Using Swarm Intelligence in Solving Fractional Differential Equations. In *Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 2023–2026.

Raja, M. A. Z., Khan, J. A., and Qureshi, I. M. (2011). Solution of Fractional Order System of Bagley-Torvik Equation Using Evolutionary Computational Intelligence. *Mathematical Problems in Engineering*, 2011.

Raja, M. A. Z., Manzar, M. A., and Samar, R. (2015). An Efficient Computational Intelligence Approach for Solving Fractional Order Riccati Equations Using ANN and SQP. *Applied Mathematical Modelling*, 39(10-11):3075–3093.

Raja, M. A. Z., Samar, R., Manzar, M. A., and Shah, S. M. (2017). Design of Unsupervised Fractional Neural Network Model Optimized With Interior Point Algorithm for Solving Bagley–Torvik Equation. *Mathematics and Computers in Simulation*, 132:139–158.

Rostami, F. and Jafarian, A. (2018). A New Artificial Neural Network Structure for Solving High-Order Linear Fractional Differential Equations. *International Journal of Computer Mathematics*, 95(3):528–539.

Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms. *arXiv preprint arXiv:1609.04747*.

Samko, S. G., Kilbas, A. A., Marichev, O. I., et al. (1993). *Fractional Integrals and Derivatives*, volume 1. Gordon and Breach Science Publishers, Yverdon Yverdon-les-Bains, Switzerland.

Shah, F. A., Abass, R., and Debnath, L. (2017). Numerical Solution of Fractional Differential Equations Using Haar Wavelet Operational Matrix Method. *International Journal of Applied and Computational Mathematics*, 3(3):2423–2445.

Shloof, A., Senu, N., Ahmadian, A., Pakdaman, M., and Salahshour, S. (2022). A New Iterative Technique for Solving Fractal-Fractional Differential Equations Based on Artificial Neural Network in the New Generalized Caputo Sense. *Engineering with Computers*, 39(1):505–515.

Sirignano, J. and DGM, K. S. (2017). A Deep Learning Algorithm for Solving Partial Differential Equations. *ArXiv e-prints*.

Sun, H., Meerschaert, M. M., Zhang, Y., Zhu, J., and Chen, W. (2013). A Fractal Richards' Equation to Capture the Non-Boltzmann Scaling of Water Transport in Unsaturated Media. *Advances in Water Resources*, 52:292–295.

Talaei, Y. and Asgari, M. (2018). An Operational Matrix Based on Chelyshkov Polynomials for Solving Multi-Order Fractional Differential Equations. *Neural Computing and Applications*, 30(5):1369–1376.

Tan, H. H. and Lim, K. H. (2019). Review of Second-Order Optimization Techniques in Artificial Neural Networks Backpropagation. In *IOP Conference Series: Materials Science and Engineering*, page 012003. IOP Publishing.

Tarasov, V. E. (2018). No Nonlocality. No Fractional Derivative. *Communications in Nonlinear Science and Numerical Simulation*, 62:157–163.

Toh, Y. T., Phang, C., and Loh, J. R. (2019). New Predictor-Corrector Scheme for Solving Nonlinear Differential Equations With Caputo-Fabrizio Operator. *Mathematical Methods in the Applied Sciences*, 42(1):175–185.

Torvik, P. J. and Bagley, R. L. (1984). On the Appearance of the Fractional Derivative in the Behavior of Real Materials. *Journal of Applied Mechanics*, 51(2):294–298.

Tsoulos, I. G., Gavrilis, D., and Glavas, E. (2009). Solving Differential Equations With Constructed Neural Networks. *Neurocomputing*, 72(10-12):2385–2391.

Tulbure, A.-A., Tulbure, A.-A., and Dulf, E.-H. (2022). A Review on Modern Defect Detection Models Using DCNNs–Deep Convolutional Neural Networks. *Journal of Advanced Research*, 35:33–48.

Verma, A. and Kumar, M. (2021). Numerical Solution of Bagley–Torvik Equations Using Legendre Artificial Neural Network Method. *Evolutionary Intelligence*, 14:2027–2037.

Vidal, R., Bruna, J., Giryes, R., and Soatto, S. (2017). Mathematics of Deep Learning. *arXiv preprint arXiv:1712.04741*.

Wang, L., Zeng, X., Yang, H., Lv, X., Guo, F., Shi, Y., and Hanif, A. (2021). Investigation and Application of Fractal Theory in Cement-Based Materials: A Review. *Fractal and Fractional*, 5(4):247.

Yadav, N., Yadav, A., and Kumar, M. (2015). *An Introduction to Neural Network Methods for Differential Equations*. Springer Dordrecht.

Yang, Y., Hou, M., and Luo, J. (2018). A Novel Improved Extreme Learning Machine Algorithm in Solving Ordinary Differential Equations by Legendre Neural Network Methods. *Advances in Difference Equations*, 2018(1):1–24.

Yazdi, H. S., Pakdaman, M., and Modaghegh, H. (2011). Unsupervised Kernel Least Mean Square Algorithm for Solving Ordinary Differential Equations. *Neurocomputing*, 74(12-13):2062–2071.

Yun, C., Sra, S., and Jadbabaie, A. (2018). Small Nonlinearities in Activation Functions Create Bad Local Minima in Neural Networks. *arXiv preprint arXiv:1802.03487*.

Zahra, W. K., Elkholy, S. M., et al. (2012). The Use of Cubic Splines in the Numerical Solution of Fractional Differential Equations. *International Journal of Mathematics and Mathematical Sciences*, 2012:1–16.

Zúñiga-Aguilar, C., Coronel-Escamilla, A., Gómez-Aguilar, J., Alvarado-Martínez, V., and Romero-Ugalde, H. (2018). New Numerical Approximation for Solving Fractional Delay Differential Equations of Variable Order Using Artificial Neural Networks. *The European Physical Journal Plus*, 133(2):1–16.

Zúñiga-Aguilar, C., Gómez-Aguilar, J., Romero-Ugalde, H., Escobar-Jiménez, R., Fernández-Anaya, G., and Alsaadi, F. E. (2021). Numerical Solution of Fractal-Fractional Mittag–Leffler Differential Equations With Variable-Order Using Artificial Neural Networks. *Engineering with Computers*, 38:2669–2682.

Zúñiga-Aguilar, C., Romero-Ugalde, H., Gómez-Aguilar, J., Escobar-Jiménez, R., and Valtierra-Rodríguez, M. (2017). Solving Fractional Differential Equations of Variable-Order Involving Operators With Mittag-Leffler Kernel Using Artificial Neural Networks. *Chaos, Solitons & Fractals*, 103:382–403.

Zurada, J. (1992). *Introduction to Artificial Neural Systems*. West Group.