PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

UNIVERSITY OF KASDI MERBAH OUARGLA

Faculty of Mathematics and Sciences of matter

DEPARTMENT OF MATHEMATICS

END OF STUDY MASTER THESIS

Speciality : Mathematics

Option : Modeling and Numerical Analysis

Prepared by : Imane Bendaoud

Theme

Approximation Theory via Deep Neural Networks and some applications

Publicly supported on: 27/06/2023

Before the jury:

Mr.Chacha A.Djamal	Prof. University of Kasdi Merbah-Ouargla	Chairman
Mr.Merabet Ismail	Prof University of Kasdi Merbah-Ouargla	Examiner
Mr.Bensayah Abdallah	M.C.A University of Kasdi Merbah-Ouargla	Supervisor

Promotion : 2022/2023

Dedication

In the name of Allah, the most gracious, most merciful, all the praise is due to him alone, the sustainer of the entire world.

This work is sincerely dedicated

to my parents,

to the soul of my dear uncle and grandfather

to all my family

Acknowledgments

My deepest and most sincere thanks go to the Almighty Allah for giving me the opportunity to carry out my studies, the strength and capacity to complete this work..

I would like to address my most sincere thanks to my supervisor, Mr.Bensayah Abdallah, this work would not have been accomplished without his vivacious guidance, constant support, and invaluable advices.

I would like to thank my bestfriend Sabrine Khelifa for her help, her encouragement when i was very disappointed and stressed.

All thanks and appreciation go to the members of the jury who devoted their precious time to read and evaluate this work.

Notations

I_n	the n-dimensional unit cube $[0,1]^n$
$C(I_n)$	The space of continuous functions on I_n
Γ	Gamma function
σ	activation function
θ	The ANN parameter: wieghts and biases
$\mathcal{L}_\theta$	Loss function
$\mathcal{T}$	Training points set
$\mathbb{T}^d$	d-dimensional torus

Abbreviations

ANN	Artificial Neural Network
FNN	Feed Forward Neural Network
MLP	Multi-Layer Perceptron
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Units
PINN	Pysics Informed Neural Network
L-BFGS	The Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm
Adam	Adaptive Moment Estimation
MSE	Mean ‘Square Error
lr	learning rate

Contents

Dedication 1

Acknowledgments 2

Symbols 3

Abbreviations 4

Introduction 6

1 Preliminairies 8

1.1 Fractional calculus 8

1.1.1 Some spacial functions 8

1.1.2 Fractional Integrals 10

1.1.3 Fractional Derivatives 10

1.1.4 Fractional integrals and fractional derivatives of a function with respect to another function 11

1.2 Fractal-Fractional Calculus 12

1.2.1 Fractal-Fractional Derivative 12

1.2.2 Fractal-Fractional Integral 13

1.3 Some useful theorems 13

2 Introduction to Artificial Neural Network 15

2.1 Artificial Neural Network 15

2.1.1 The Perceptron 16

2.1.2 Feed-Forward Neural Network (FNNs) 16

2.1.3 Activation Function 17

2.1.4 Backprobagation (Training phase) 19

2.2 Universal Approximation Property of ANN 20

2.2.1 Some relevant density theorems 22

2.3 Error estimation results 24

2.4 PINNs and deepxde library 27

2.4.1 Physics Informed Neural Network 27

2.4.2 Trainig the PINN 28

2.4.3 Application Library: DeepXde 31

3 Applications of Artificial Neural Network in Approximation 33

3.1 Heat Equation 33

3.2 Second Order ODE 34

3.3 Volterra Integration Differential Equation 35

3.4 Diffusion equation with hard initial and boundary conditions 36

3.5 Fractional Boundary Value Problem of nonlinear functional equation . . . 37

3.5.1 Approximation of the solution via artificial neural networks 38

3.6 Fractional integro-differential value problem 40

3.6.1 Approximation of solution via ANN 41

3.7 Fractal-fractional order mathematical model on the spread of Coronavirus (COVID-19) 42

3.7.1 Approximation of the solution via artificial neural networks 43

Conclusion 45

Introduction

The field of Approximation Theory has long been a central focus in mathematics, aiming to find effective methods for representing complex functions with simpler, more manageable models. This theory has numerous applications in science, engineering and other fields.

In recent years, the emergence of Artificial Neural Networks (ANNs) has revolutionized the landscape of Approximation Theory, offering a powerful and versatile tool for accurately approximating functions with unprecedented precision due to their ability to learn from data.

This thesis explores the utilization of ANNs within the context of approximation theory, delving into their capabilities, methodologies, and applications.

The motivation behind this research lies in the potential of ANNs to overcome the limitations of traditional approximation techniques when dealing with highly nonlinear and intricate functions. The key behind ANNs is their ability to approximate any continuos functions that were previously deemed challenging or computationally demanding. This property has been extensively studied in the literature, and several seminal works have contributed to our understanding of the approximation capabilities of neural networks.

In 1989, Cybenko demonstrated in his paper ”Approximation by superpositions of a sigmoidal function [?] that a single hidden layer feedforward neural network with sigmoid activation functions can approximate any continuous function to arbitrary precision. This result is known as the Universal Approximation Theorem and has been a cornerstone of neural network research ever since. Later, Hornik extended Cybenkos result by showing in his 1991 paper Approximation capabilities of multilayer feedforward networks [?] that a neural network with multiple hidden layers can also approximate any continuous function to arbitrary accuracy given a sufficient number of neurons. This result was further strengthened by Leshno et al. in their 1993 paper Multilayer feedforward networks with a nonpolynomial activation function can approximate any function [?], where they demonstrated that a neural network with a nonpolynomial activation function can also approximate any function to arbitrary precision.

In the decades that followed, researchers have continued to investigate the approximation capabilities of neural networks, exploring the role of network architecture, activation functions, and regularization techniques in determining their expressive power. More recent works like Kratsioss 2020 paper The Universal Approximation Property provides a comprehensive review of the approximation capabilities of various neural network architectures, while Lu et al.s 2017 paper The expressive power of neural networks: A view from the width [?] investigates the relationship between network width and approximation power. Hanin and Sellkes 2019 paper Approximating continuous functions by ReLU nets of minimal width [?] and Kidger and Lyonss 2020 paper Universal Approximation with Deep Narrow Networks [?] both focus on the approximation capabilities of deep narrow neural networks with ReLU activation functions.

This thesis is split into three main chapters as follows: The first chapter is devoted to recalling some notions and basic concepts in fractional calculus, and citing theorems that are used in the following chapters.

The second chapter is intended to explore the underlying principles of Approximation Theory via Artificial Neural Networks, discuss the different types of ANNs that exist and their respective architectures, the next section covers the main result of density in ANN, known as the universal approximation property, discuss its implications for approximation theory, review some error estimation results and finishing with exploring the application library DeepXde.

And in the third chapter, we provide numerical simulations using PINNs and DeepXde on ODE, IDE, PDE equations, some frational value problems and fractal-fractional equations system.

Chapter 1

Preliminairies

1.1 Fractional calculus

In this section, we present some definition and some primitive notions of fractional calculus.

1.1.1 Some spacial functions

Special function play an important role in the theory of differentiation of arbitrary order and in the theory of fractional differential equations.

Gamma Function

Gamma function is a commonly used extension of the factorial function to complex numbers. It is defined for all complex numbers except non-positive integers.
For complex numbers with a positive real part, the gamma function is defined via a convergent improper integral.

Definition 1.1 [?] The Gamma function $\Gamma(z)$ is defined by the integrals

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt, \qquad (Re(z) > 0) \tag{1.1}$$

where $t^{z-1} = e^{(z-1)log(t)}$

One of the basic properties of the gamma function is that it satisfies the following functiorial equation:

$$\Gamma(z+1) = z\Gamma(z) \tag{1.2}$$

The Gamma function is extended to the half-plane $Re(z) \leq 0$ by:

$$\Gamma(z) = \frac{\Gamma(z+n)}{(z)_n}, \quad (Re(z) > -n; \quad n \in \mathbb{N} \notin \mathbb{Z}_0^- = \{0, -1, -2, ...\}). \tag{1.3}$$

Here $(z)_n$ is the Pochhammer symbol, defined for complex $z \in \mathbb{C}$ and non-negative integer $n \in \mathbb{N}_0$ by

$$(z)_0 = 1 \quad \text{and} \quad (z)_n = z(z+1)...(z+n-1) \quad (n \in \mathbb{N}) \tag{1.4}$$

Equations 1.2 and 1.4 yield

$$\Gamma(n+1) = (1)_n = n! \quad (n \in \mathbb{N}) \tag{1.5}$$

Beta Funcion

Definition 1.2 [?] The Beta function is usually defined by

$$B(z,w) = \int_0^1 \tau^{z-1}(1-\tau)^{w-1}d\tau, \quad (Re(z) > 0, \quad Re(w) > 0) \tag{1.6}$$

The relationship between the Gamma function and the Beta function

The Beta function is connected with the Gamma functions by the relation

$$B(z,w) = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)} \quad (z, w \notin \mathbb{Z}_0^-) \tag{1.7}$$

1.1.2 Fractional Integrals

Riemann-Liouville fractional integrals

Definition 1.3 [?] The Riemann-Liouville fractional integrals $I^{\alpha}_{a+}f$ and $I^{\alpha}_{a-}f$ of order $\alpha \in \mathbb{C}(Re(\alpha) > 0)$ are defined by

$$(I^{\alpha}_{a-}f)(x) = \frac{1}{\Gamma(\alpha)} \int_a^x \frac{f(t)dt}{(x-t)^{1-\alpha}} \quad (x > \alpha; \quad Re(\alpha) > 0) \tag{1.8}$$

and

$$(I^{\alpha}_{a+}f)(x) = \frac{1}{\Gamma(\alpha)} \int_x^b \frac{f(t)dt}{(t-x)^{1-\alpha}} \quad (x < b; \quad Re(\alpha) > 0) \tag{1.9}$$

respectively. Here $\Gamma(\alpha)$ is the Gamma function 1.1.

These integrals are called the left-sided and the right-sided fractional integrals.

1.1.3 Fractional Derivatives

Riemann-Liouville fractional derivatives

Definition 1.4 [?] The Riemann-Liouville fractional derivatives $D^{\alpha}_{a+}y$ and $D^{\alpha}_{b-}y$ of order $\alpha \in \mathbb{C}(Re(\alpha) \geqq 0)$ are defined by

$$(D^{\alpha}_{a+}y)(x) = \frac{1}{\Gamma(n-\alpha)} (\frac{d}{dx})^n \int_a^x \frac{y(t)dt}{(x-t)^{\alpha-n+1}} \quad (n = [Re(\alpha)] + 1; \; x > a) \tag{1.10}$$

and

$$(D^{\alpha}_{b-}y)(x) = \frac{1}{\Gamma(n-\alpha)} (-\frac{d}{dx})^n \int_x^b \frac{y(t)dt}{(t-x)^{\alpha-n+1}} \quad (n = [Re(\alpha)] + 1; \; x < b) \tag{1.11}$$

respectively, where $[Re(\alpha)]$ means the integral part of $Re(\alpha)$).

In particular, when $\alpha = n \in \mathbb{N}_0$, then

$$(D^0_{a+}y)(x) = (D^0_{b-}y)(x) = y(x) \tag{1.12}$$

and

$$(D^n_{a+}y)(x) = y^{(n)}(x); \quad (D^n_{b-}y)(x) = (-1)^n y^{(n)}(x) \quad (n \in \mathbb{N}) \tag{1.13}$$

where $y^{(n)}(x)$ is the usual derivative of $y(x)$ of order n.

Caputo fractional derivative

Definition 1.5 [?] Let $t \in [a, b]$ be a finite interval of the real line $\mathbb{R}$, and let $(Re(\alpha) \geqq 0)$. If $y(x) \in AC^n[a, b]$, then the Caputo fractional derivative $({}^cD^\alpha_{a+}y)(x)$ and $({}^cD^\alpha_{b-}y)(x)$ exist almost everywhere on $[a, b]$.
If $\alpha \notin \mathbb{N}_0$, $({}^cD^\alpha_{a+}y)(x)$ and $({}^cD^\alpha_{b-}y)(x)$ are represented by

$$({}^cD^\alpha_{a+}y)(x) = \frac{1}{\Gamma(n-\alpha)} \int_a^x \frac{y^{(n)}(t)dt}{(x-t)^{\alpha-n+1}} =: (I^{n-\alpha}_{a+} D^n y)(x) \tag{1.14}$$

and

$$({}^cD^\alpha_{b-}y)(x) = \frac{(-1)^n}{\Gamma(n-\alpha)} \int_x^b \frac{y^{(n)}(t)dt}{(t-x)^{\alpha-n+1}} =: (-1)^n (I^{n-\alpha}_{b-} D^n y)(x) \tag{1.15}$$

respectively, where $D = d/x$ and $n = [Re(\alpha)] + 1$

1.1.4 Fractional integrals and fractional derivatives of a function with respect to another function

In this section we present the definitions and some properties of the fractional integrals and fractional derivatives of a function f with respect to another function g.

Definition 1.6 [?] Let $(a, b)(-\infty \leqq a < b \leqq \infty)$ be a finite interval of the real line $\mathbb{R}$ and $Re(\alpha) > 0$. Also, let $g(x)$ be an increasing and positive monotone function on $(a, b]$, having a continuous derivative $g'(x)$ on (a, b). The left- and right-sided fractional integrals of a function f with respect to another function g on $[a,b]$ are defined by

$$(I^{\alpha}_{a+;g}f)(x) := \frac{1}{\Gamma(\alpha)}\int_a^x \frac{g'(t)f(t)dt}{[g(x)-g(t)^{1-\alpha}]} \quad (x>0; Re(\alpha)>0) \tag{1.16}$$

and

$$(I^{\alpha}_{b-;g}f)(x) := \frac{1}{\Gamma(\alpha)}\int_x^b \frac{g'(t)f(t)dt}{[g(t)-g(x)^{1-\alpha}]} \quad (x<b; Re(\alpha)>0), \tag{1.17}$$

respectively.

Definition 1.7 [?] Let $g'(x) \neq 0(-\infty \leqq a < x < b \leqq \infty)$ and $Re(\alpha \neq 0)$. Also let $n = [Re(\alpha)+1]$ and $D = d/dx$. The Riemann-Liouvillle fractional derivatives of a function y with respect to g of order α $(Re(\alpha) \geqq 0; \alpha \neq 0)$, corresponding to the Riemann-Liouville integrals in 1.16 and 1.17 are defined by

$$(D^{\alpha}_{a+;g}y)(x) := \left(\frac{1}{g'(x)}\frac{d}{dt}\right)^n (I^{n-\alpha}_{a+;g}y)(x) = \frac{1}{\Gamma(n-\alpha)}\left(\frac{1}{g'(x)}\frac{d}{dt}\right)^n \int_a^x \frac{g'(t)y(t)dt}{[g(x)-g(t)]^{\alpha-n+1}} \quad (x>b) \tag{1.18}$$

and

$$(D^{\alpha}_{b-;g}y)(x) := \left(-\frac{1}{g'(x)}\frac{d}{dt}\right)^n (I^{n-\alpha}_{b-;g}y)(x) = \frac{1}{\Gamma(n-\alpha)}\left(-\frac{1}{g'(x)}\frac{d}{dt}\right)^n \int_x^b \frac{g'(t)y(t)dt}{[g(t)-g(x)]^{\alpha-n+1}} \quad (x<b) \tag{1.19}$$

1.2 Fractal-Fractional Calculus

1.2.1 Fractal-Fractional Derivative

Definition 1.8 [?] Suppose that y is continuous on an open interval (a,b) with order ν. Then the fractal-fractional derivative of the function y with order ϱ via the power law type kernel in the Riemann-Liouville sense is given by

$$^{\mathcal{FFP}}D^{\varrho,\nu}_{a,t}y(t) = \frac{1}{\Gamma(n-\varrho)}\frac{d}{dt^{\nu}}\int_a^t (t-x)^{n-\varrho-1}y(x)dx, \quad (n-1<\varrho,\nu \leq n \in \mathbb{N}) \tag{1.20}$$

where the fractal derivative is expressed as

$$\frac{dy(t)}{dt^{\nu}} = \lim_{t\to x} \frac{y(t)-y(x)}{x^{\nu}-t^{\varrho}}$$

1.2.2 Fractal-Fractional Integral

Definition 1.9 [?] Suppose that y is continuous function on the interval (a, b). Then the fractal-fractional integral of y with fractional order ϱ and fractal order ν via the power law type kernel is defined by

$$^{\mathcal{FFP}}I^{\varrho,\nu}_{a,t}y(t) = \frac{\nu}{\Gamma(\varrho)}\int_a^t x^{\nu-1}(t-x)^{\varrho-1}y(x)dx. \tag{1.21}$$

1.3 Some useful theorems

Definition 1.10 (discriminatory function)[?] We say that a function σ is discriminatory if given a measure $\mu \in M(I^n)$ such that

$$\int_{I^n} \sigma(w^T x + b)d\mu(x) = 0 \quad , \forall w \in \mathbb{R}^n, b \in \mathbb{R} \tag{1.22}$$

implies that $\mu = 0$

Definition 1.11 [?] We say that u is sigmoidal if

$$\sigma(t) \to \begin{cases} 1 & t \to +\infty \\ 0 & t \to -\infty \end{cases} \tag{1.23}$$

Lemma 1.1 [?] Any bounded, measurable sigmoidal function is discriminatory. In particular, any continuous sigmoidal function is discriminatory.

Theorem 1.1 (Hahn Banach Theorem) Let V be a normed vector space, $U \subset V$ a subspace of V. Let $L \in U^*$.Then there exists $L^{'} \in V^*$ that extends L to V and satisfies $||L^{'}||^*_V = ||L||^*_U$

Corollary 1.1 Let V be a normed vector space, $U \subset V$ a subspace of V. Let $x_0 \in V$ such that $d(x_0, U) = \gamma > 0$.

Then there exists $L \in V^{'}$ such that

- $||L||_V^{'} = 1$
- $L(x_0) = \gamma$
- $L(U) = 0$

Theorem 1.2 (Riesz-Representation Theorem) Let L be a bounded linear funtional on $C(I_n)$.Then there exists a unique $\mu \in M(I_n)$ such that

$$L(h) = \int_{I_n} h(x) d\mu(x) \forall h \in C(I_n)$$

Chapter 2

Introduction to Artificial Neural Network

2.1 Artificial Neural Network

Artificial Neural Networks (ANNs) are a type of machine learning model inspired by the structure and function of biological neurons found on the human brain. The basic idea is to create a network of simple processing elements, called artificial neurons or nodes, that can receive input data, process it, and produce output data. ANNs are a subset of deep learning models, which are neural networks with multiple layers. The history of ANNs dates back to the 1940s, when Warren McCulloch and Walter Pitts proposed a mathematical model of a biological neuron. Their model consisted of a simple binary threshold unit that received input signals and produced an output signal based on whether the sum of the inputs exceeded a certain threshold.

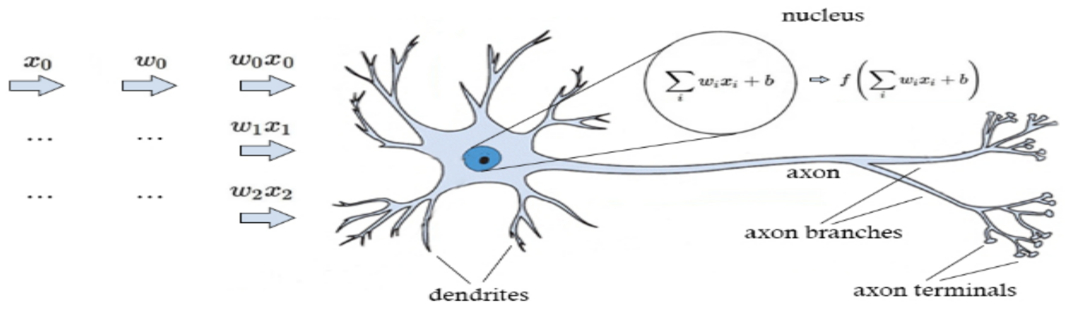


Figure 2.1: biological neuron

In the following decades, researchers developed various types of artificial neurons and architectures for neural networks,such as the perceptron, feedforward networks, and recurrent networks.

To understand the concept of a neuron working, we start with the simplest form of a neural network: Perceptron

2.1.1 The Perceptron

The perceptron is a type of artificial neural network and one of the earliest and simplest forms of neural network models. It was proposed by Frank Rosenblatt in the late 1950s as a binary classifier capable of learning from training data. The perceptron consists of a single layer of artificial neurons, also known as perceptron units or nodes.

The mathematical model of an artificial neuron is typically based on the concept of a weighted sum of inputs, followed by a nonlinear activation function.

Definition 2.1 [?] Let $d \in \mathbb{N}$. An artificial neuron $f : \mathbb{R}^d \to \mathbb{R}$ with weight $w \in \mathbb{R}^d$, bias $b \in \mathbb{R}$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is needed as the map

$$f(x) = \sigma(w.x + b) \qquad \text{for } x \in \mathbb{R}^d \tag{2.1}$$

Figure 2.2: Model of an artificial neuron taking an input feature of dimension $d = 3$.

2.1.2 Feed-Forward Neural Network (FNNs)

Also known as Multi-Layer Perceptron (MLPs), The neurons are typically organized into multiple layers, an input layer, one or more hidden layers and an output layer.

Definition 2.2 [?] Let L, d, $n_1, ..., n_L \in \mathbb{N}$ and $n_0 := d$. A L-layer feedforward neural network $\hat{u} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ with affine linear maps $A_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}, x \mapsto A_l(x) = W_l x + b_l$ with $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$, $b_l \in \mathbb{R}^{n_l}$ and activation function $\sigma_l : \mathbb{R} \to \mathbb{R}$, $l = 1, ..., L$ is defined as

Input layer: $\hat{u}^{(0)}(x) = x \in \mathbb{R}^d$

Hidden layer: $\hat{u}^{(l)}(x) = \sigma_l(W_l \hat{u}^{(l-1)}(x) + b_l) \in \mathbb{R}^{n_l} \qquad \text{for } 1 \leq l \leq L - 1$

Outout layer: $\hat{u}(x) = W_l \hat{u}^{(L-1)}(x) + b_L \in \mathbb{R}^{n_L}$

where the activation functions are used component-wise. Here, d is the dimension of the input layer, L denotes the number of layers also called depth of $\hat{u}$, $n_1, ..., n_{L-1}$ denote the number of neurons for each of the $L-1$ hidden layers, also called width of the respective layer. If $n_1 = ... = n_{L-1}$ then, n_i is called width of $\hat{u}$ for $i \in \{1, ..., L-1\}$. n_L is the dimension of the output layer. The matrices W_l contain the network's weights and the vector b_l is the biases.

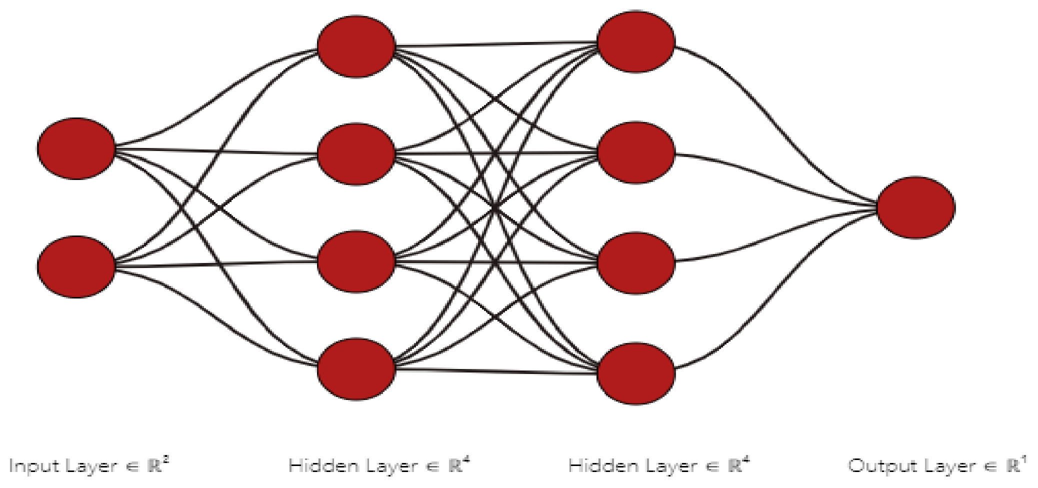


Figure 2.3: Multi-layer Neural Network

Figure 2.3 represent the general ANN architecture, which has 2 neurons in the input layer, 2 hidden layers each contains 4 neurons and an output layer.

2.1.3 Activation Function

Each neuron typically applies an activation function to the weighted sum of its inputs to produce an output.The activation function is usually a nonlinear function that introduces nonlinearity into the network, allowing it to model complex relationships between inputs and outputs and there are several activation functions of which we note the the common useful ones:

- The sigmoid function: (also known as Logistic function or Soft step). This is a very commonly used function that gives only values between 0 and 1, usually indicated by

$$\sigma(z) = f(z) = \frac{1}{1 + exp(-z)}$$

- The hyperbolic tangent function: Tanh has characteristics similar to sigmoid: it is nonlinear in nature, so we can stack layers. It is also bound to the range $[-1, 1]$.

$$f(z) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)} = \frac{exp(2x) - 1}{exp(2x) + 1}$$

- The rectified linear unit function: The rectifier is an activation function defined as the positive part of its argument:

$$f(z) = \max\{0, z\}$$

.

Commonly used activation functions are pictured in Figure(2.4)

Function.png Function.bb

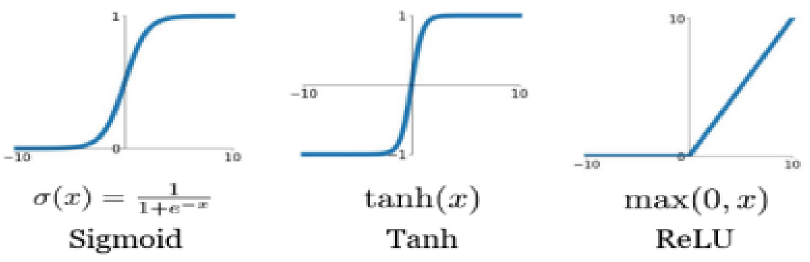


Figure 2.4: Common used Activation Function

All hidden layers usually use the same activation function. However, the output layer will typically use a different activation function from the hidden layers. The choice depends on the goal or type of prediction made by the model. In a neural network, the output of one layer of neurons is typically fed as input to the next layer of neurons, forming a hierarchical structure of processing.

2.1.4 Backprobagation (Training phase)

The weights and biases of the network are learned through a process called Back-probagation, which involves adjusting the parameters of the network to minimize a loss function that measures the difference between the network's predicted output and the true output. A loss function indicates how well an algorithm performs on a given data set, by minimizing this function, we can expect better predictions.

There are several loss functions that are used for machine learning applications according to which task they have to solve. Since we have a regression task the mean squared error (MSE) loss is a proper loss function.

Definition 2.3 [?](MSE Loss)

Let $\hat{u}_\theta : \mathbb{R}^d \to \mathbb{R}^{n_L}$ with $d, n_L \in \mathbb{N}$ be a FNN as defined in Definition (2.2). Further, let $\Omega \subset \mathbb{R}^d$ be a bounded domain and $u : \Omega \to \mathbb{R}^{n_L}$ be the desired truth function the FNN aims to approximate. Then, for a given set of training points $\mathcal{T} \subset \Omega$ the mean squared error loss is defined as:

$$\mathcal{L}_{\mathcal{T}}(\theta) = \frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} |\hat{u}_\theta(x) - u(x)|^2.$$

The training process of the network under the MSE loss can then be formulated as the optimization problem:

$$\min_\theta \mathcal{L}(\theta) \tag{2.2}$$

with loss function $\mathcal{L}$ as defined in definition (2.3) and θ representing the network's parameters.

To minimize the loss function, an optimization algorithm is needed in the hopes of improving the overall performance of the network on unseen data. The minimization is done using a gradient based optimisation algorithm, such as stochastic gradient descent (SGD), variants of SGD include Adaptive learning rate methods like Adam and RMSprop.

Starting from randomly initialized parameters $\theta(w, b)$,at each iteration i in general the parameters are updated as:

$$\theta_{i+1} = \theta_i - lr \times \nabla_{\theta_i}\mathcal{L} \tag{2.3}$$

where θ_i is the parameter value after the i-th training iteration, lr is an adaptive learning rate set automatically by the chosen optimizer,it controls the speed of the parameter update process and the learning speed of the model, thus the parameters are adjusted according to the results of each training iteration.

After the training phase is complete, the weights of the neural network are solidified and no longer change during the testing phase. The purpose of the testing phase is to evaluate how accurate the model is. This is accomplished by providing new data to the model, so the testing samples must be kept separate from the training data set. Again, the networks predictions will be compared to the expected output. But this is done only with the goal of evaluating the models accuracy, rather than as part of the process for improving the model itself.

In the next section, we will discuss the universal approximation property of ANNs, which is a fundamental theoretical result that underlies their success in many applications.

2.2 Universal Approximation Property of ANN

The Universal Approximation Property of artificial neural networks (ANNs) refers to their ability to approximate any continuous function to arbitrary accuracy, given enough neurons and appropriate activation functions. It is a key property that highlights the expressive power of ANNs.

Specifically, the Universal Approximation theorem states that a feedforward neural network with a single hidden layer can approximate any continuous function on a compact subset of Euclidean space.

It was first proved by George Cybenko in 1989 and later refined by several other researchers. It has since become a cornerstone result in the theory and practice of neural networks, contributing to their widespread adoption and exploration in various fields.

Theorem 2.1 [?] (Cybenko) Let σ be any continuous discriminatory function. Then finite sums of the form:

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j^T x + b_j) \tag{2.4}$$

where $w_j \in \mathbb{R}^n, \alpha_j, b_j \in \mathbb{R}$ are dense in $C(I^n)$. In other words, given any $\epsilon > 0$and $f \in C(I^n)$, there is a sum $G(x)$ of the above form such that

$$|G(x) - f(x)| < \epsilon, \quad \forall x \in I^n \tag{2.5}$$

Proof: Let $S \subset C(I_n)$ be the set of functions of the form (2.4), we want to prove that $R := \bar{S} = C(I_n)$.

Clearly S is a linear subspace of $C(I_n)$. Suppose that $R \subsetneq C(I_n)$, that is $\exists f \in C(I_n)$ such that $d(f, R) > 0$.

By the corollary of Hanh-Banach, $\exists L$ bounded linear functional on $C(I_n)$ such that $L \neq 0$, but $L(R) = L(S) = 0$. By the Riesz Representation theorem $\exists! \mu \in M(I_n)$ such that

$$L(h) = \int_{I_n} h(x) d\mu(x), \forall h \in C(I_n) \tag{2.6}$$

Since $L(R) = 0$ and since $\sigma(w^T x + b) \in R, \forall w, b$, then

$$0 = L(\sigma(w^T x + b)) = \int_{I_n} \sigma(w^T x + b) d\mu(x), \forall w, b \tag{2.7}$$

Since σ is discriminatory, (2.7) implies $\mu = 0$, which in turn implies $L = 0$, and this is contradiction. ■

Then, he specialized this result to sigmoidal function, using lemma (1.1). A straightforward combination of theorem (2.1) and lemma (1.1) shows that networks with one internal layer and an arbitrary continuous sigmoidal function can approximate continuous functions wtih arbitrary precision providing that no constraints are placed on the number of nodes or the size of the weights.

The neural network, used to demonstrate the Cybenkos theorem , has the following structure (n-dimensional inputs):

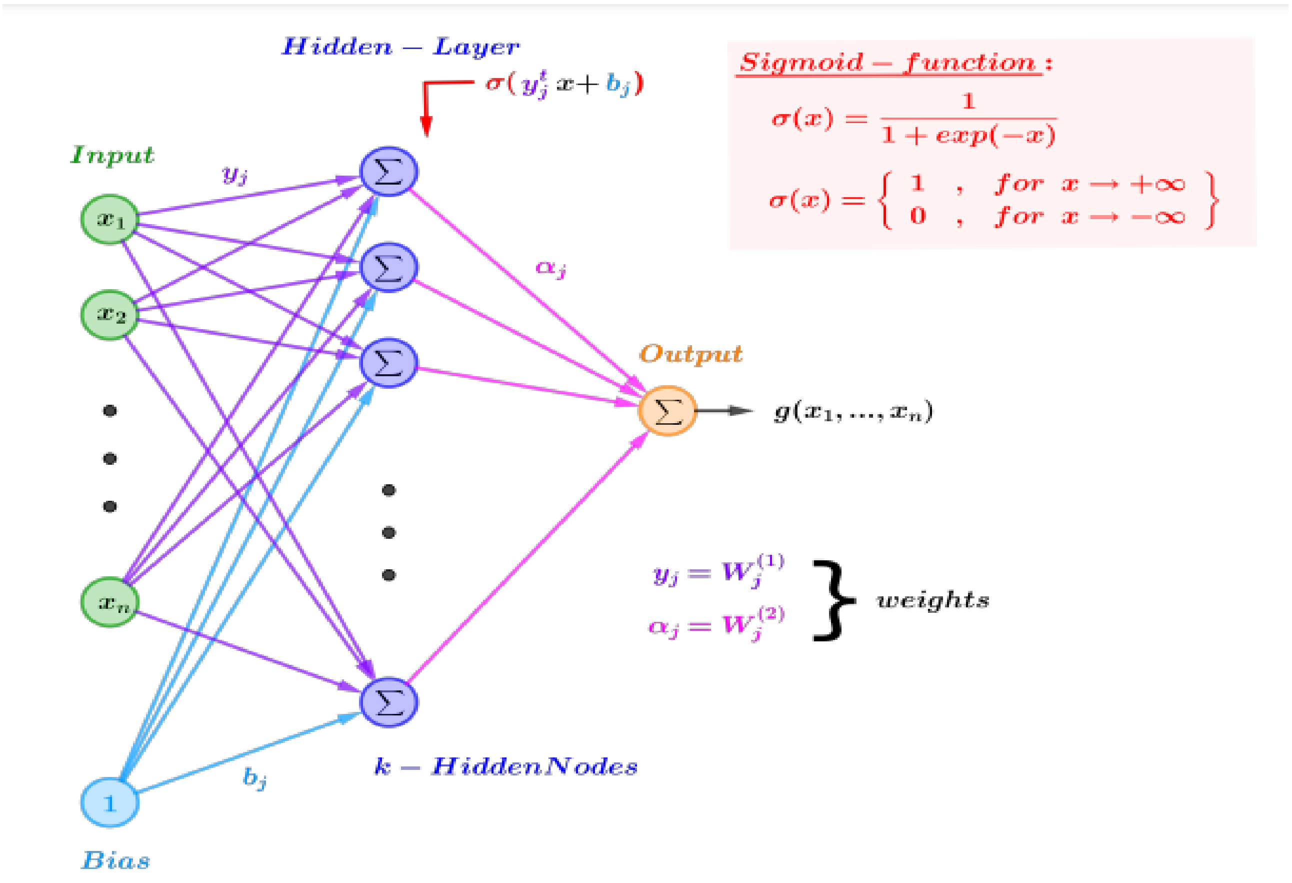


Figure 2.5: n-dimensional neural network

2.2.1 Some relevant density theorems

Hornik showed in his 1991 paper "Approximation capabilities of multilayer feedforward networks" [?], that it is not the specific choice of the activation function but rather the multilayer feed-forward architecture itself that gives neural networks the potential of being universal approximators. This result was further strengthened by Leshno et al. in their 1993 paper "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function" [?], where they demonstrated that a neural network with a non-polynomial activation function can also approximate any function to arbitrary precision.

In the decades that followed, researchers have continued to investigate the approximation capabilities of neural networks, exploring the role of network architecture, activation functions, and regularization techniques in determining their expressive power. More recent works have also investigated the role of network width and depth, showing that increasing the number of layers and/or neurons can improve the approximation capabilities of neural networks.

For example, Zhou Lu et al.s 2017 paper "The expressive power of neural networks: A view from the width" investigates the relationship between network width and approximation power, they showed that one can find a neural network that can approximate any Lebesgue-integrable function on n-dimensional input space with respect to L^1 distance if network depth is allowed to grow.

Theorem 2.2 [?] (Universal approximation theorem for width-bounded ReLU networks) For any Lebesgue-integrable function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, any $\epsilon > 0$, there exists a fully-connected ReLU network $\mathcal{A}$ with width $d_m \leq n+4$, such that the function $F_{\mathcal{A}}$ represented by this network satisfies:

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| dx < \epsilon$$

Hanin and Sellkes 2018 paper "Approximating continuous functions by ReLU nets of minimal width" [?] and Kidger and Lyonss 2020 paper "Universal Approximation with Deep Narrow Networks"[?] both focus on the approximation capabilities of deep narrow neural network, where the first one has shown that deep narrow networks with ReLU activation function are dense in $C(K;\mathbb{R}^m)$ for $K \subseteq \mathbb{R}^n$ compact, and require only width $n + m$. While the central result of the second one work yields the following universal approximation theorem for networks with bounded width.

Theorem 2.3 [?] Let $\rho : \mathbb{R} \longrightarrow \mathbb{R}$ be any nonaffine continuous function which is continuously differentiable at at least one point, with nonzero derivative at that point. Let $K \subseteq \mathbb{R}^n$be compact. Then $\mathcal{NN}^{\rho}_{n,m,n+m+2}$ is dense in $C(K, \mathbb{R}^m)$ with respect to the uniform norm.

This theorem result covers every activation function possible to use in practice, and also includes polynomial activation functions, which is unlike the classical version of the theorem.

The following refinement, specifies the optimal minimum width for which such an approximation is possible and is due to Sujin Park et al.

Theorem 2.4 [?] For any Bochner-Lebesgue p-integrable function $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ and any $\epsilon > 0$, there exists a fully-connected ReLU network F of width exactly $d_m = \max\{n + 1, m\}$, satisfying

$$\int_{\mathbb{R}^n} ||f(x) - F(x)||^p \mathrm{d}x < \epsilon$$

Moreover, there exists a function $f \in L^p(\mathbb{R}^n, \mathbb{R}^m)$ and some $\epsilon > 0$, for which there is no fully-connected ReLU network of width less than $d_m = \max\{n + 1, m\}$ satisfying the above approximation bound.

2.3 Error estimation results

Error estimation is a key measure in the process of model validation and verification for neural network. As Artificial neural networks are universal function approximators, it is also natural to use them as ansatz spaces for the solutions of partial differential equations (PDEs). In fact, the literature on the use of deep learning for numerical approximation of PDEs and error estimation has witnessed exponential growth in the last 2-3 years.

Recently,Tim.De Rick, A. D. Jagtap and S.Mishra in their 2022 article [?], have discussed the error esimation on the case of Navier-Stokes equation, they followed an approach where they proposed three theoritical questions, the first question about the smallness of the PDE residual in the class of neural networks, the second question is about

Here we have the Navier-Stokes equation :

$$\begin{cases} u_t + u.\nabla u + \nabla p = \nu \Delta u, & (x,t) \text{ in } D \times [0,T], \\ \mathrm{div}(u) = 0, & (x,t) \text{ in } D \times [0,T], \\ u(t=0) = u_0, & x \text{ in } D. \end{cases} \tag{2.8}$$

with the residuals:

$$\begin{aligned}
&\mathcal{R}_{PDE}[(u_\theta, q)](x,t) = (\partial_t u_\theta + u_\theta.\nabla u_\theta + \nabla q_\theta - \nu \Delta u_\theta)(x,t),\\
&\mathcal{R}_{div}[u_\theta](x,t) = \mathrm{div}(u_\theta)(x,t),\\
&\mathcal{R}_t[u_\theta](x) = u_\theta(x=0) - \varphi(x),
\end{aligned} \tag{2.9}$$

for $x \in D, t \in \mathcal{O} = [0,T]$, where $\varphi : D \longrightarrow \mathbb{R}^d$ is the initial condition.

Note that for the exact solution to the Navier-Stokes equations (2.8) it holds that $\mathcal{R}_{PDE}[(u,p)] = \mathcal{R}_{div}[u] = \mathcal{R}_t[u] = 0$. Hence, within the ANNs algorithm, one seeks to find a neural network (u_θ, p_θ),for which all residuals are simultaneously minimized, e.g. by minimizing the quantity,

$$\mathcal{E}_G^2(\theta) = \int_{D\times\mathcal{O}} |\mathcal{R}_{PDE}[(u_\theta,p_\theta)](x,t)|^2 dxdt + \int_{D\times\mathcal{O}} |\mathcal{R}_{div}[u_\theta](x,t)|^2 dxdt + \int_D |\mathcal{R}_t[u_\theta](x)|^2 dx. \tag{2.10}$$

This quantity refer to the genealisation error of the neural network. However, the quantity (2.10) involves integrals and can therefore not be directly minimized in practice. Instead, the integrals are approximated by numerical quadrature, the so-called training loss or training error $\theta \longmapsto \mathcal{E}_T(\theta, \mathcal{S})$, resulting in,

$$\begin{aligned}
\mathcal{E}_T(\theta,\mathcal{S})^2 &= \mathcal{E}_T^{PDE}(\theta,\mathcal{S}_{int})^2 + \mathcal{E}_T^{div}(\theta,\mathcal{S}_s)^2 + \mathcal{E}_T^{t}(\theta,\mathcal{S}_t)^2\\
&= \sum_{n=1}^{N_{int}} w_{int}^n |\mathcal{R}_{PDE}[(u_\theta,p_\theta)](t_{int}^n, x_{int}^n)|^2 + \sum_{n=1}^{N_{int}} w_{int}^n |\mathcal{R}_{div}[u_\theta](t_{int}^n, x_{int}^n)|^2\\
&+ \sum_{n=1}^{N_t} w_t^n |\mathcal{R}_t[u_\theta](x_t^n)|^2
\end{aligned} \tag{2.11}$$

where the training data set $\mathcal{S} = (\mathcal{S}_{int}, \mathcal{S}_s, \mathcal{S}_t)$ is chosen as quadrature points with respect to the relevant domain (resp. $D \times [0,T]$and D) and where the w_*^n are corresponding quadrature weights.

The authors have shown that there is for sufficiently smooth (Sobolev regular) initial data, there exists neural networks, with the tanh activation function and with two hidden layers, such that the resulting PDE residuals can be arbitrarily small

Theorem 2.5 [?] For every $N \in \mathbb{N}$, $d > 0$ and every $f \in W^{s,\infty}([0,1]^d)$, there exists a tanh neural network $\hat{f}$ with 2 hidden layers of width N^d such that for every $0 \leq k \leq s$ it holds that ,

$$||f - \hat{f}||_{W^{k,\infty}} \leq C(ln(cN))^k N^{-s+k}$$

where $c, C > 0$ are independent of N and explicitly known.

Proof: See [page 740, [?]] ■

As consequence, they obtained the following result:

Theorem 2.6 Let $d, r, k \in \mathbb{N}$, with $k \geq 3$, $u_0 \in H^r$ with $r > \frac{d}{r} + 2k$ and $\mathrm{div}(u_0) = 0$. For every $N \in \mathbb{N}$ there exists a tanh neural network (u_θ, p_θ) with 2 hidden layers of width N^{d+1} such that

$$||\mathcal{R}_{PDE}[(u_\theta, p_\theta)]||_{L^2} + ||\mathcal{R}_{div}[u_\theta]||_{L^2} + ||\mathcal{R}_t[u_\theta]||_{L^2} \leq C(ln(cN))^2 N^{-k+2}$$

Proof: See [page 9, [?]] ■

Then, they had shown that neural networks for which the residuals are small, will provide a good L^2 -approximation of the true solution of the Navier-Stokes equation.

Theorem 2.7 Let $d \in \mathbb{N}, D = \mathbb{T}^d$ and u in $C^1(D \times [0,T])$ be the classical solution of the Navier-Stokes equation 2.8. Let u_θ be a NN with parameters θ, then the resulting L^2-error is bounded as follows

$$||u - u_\theta||^2_{L^2} \leq C\Big(||\mathcal{R}_{div}||_{L^2} + ||\mathcal{R}_{PDE}||^2_{L^2} + ||\mathcal{R}_s||_{L^2}\Big)$$

Proof: See [page 12, [?]] ■

Combined by Theorem(2.7), they bound the total error in terms of the training error and size of the trainig set $\mathcal{S}$, obtained using accuracy of quadrature rules.

Theorem 2.8 Let $T > 0, d \in \mathbb{N}$, let $(u, p) \in C^4(\mathbb{T}^d \times [0, T])$ be the classical solution of the Navier-Stokes equation 2.8 and let (u_θ, p_θ) be a NN with parameters θ. Then the following error bound holds,

$$||u - u_\theta||_{L^2}^2 \leq C\Big(\mathcal{E}_T(\mathcal{S}) + M_t^{-\frac{2}{d}} + M_{int}^{-\frac{1}{d+1}} + M_s^{-\frac{1}{d}}\Big)$$

where M is the number of quadrature points.

Proof: See [page 15, [?]] ■

2.4 PINNs and deepxde library

2.4.1 Physics Informed Neural Network

Physics-Informed Neural Networks (PINNs), were first proposed in the 1990s as a machine learning framework for approximating solutions of differential equations. However, they were resurrected recently as a practical and computationally efficient paradigm for solving both forward and inverse problems for PDEs. Since then, there has been an explosive growth in designing and applying PINNs for a variety of applications involving PDEs.They were introduced in a 2018 paper by Raissi et al.

The idea behind PINNs is to incorporate physical laws and boundary conditions into the network architecture. PINNs have been applied to a wide range of problems, including science and engineering, including fluid dynamics, materials science, and biomedical engineering.

2.4.2 Trainig the PINN

Consider a physical system defined over a domain $\Omega \subset \mathbb{R}^d$, and governed by a PDE of the form

$$\begin{cases} \mathcal{F}[u(x,t)] = 0, \quad x \in \partial\Omega, \quad t \in [0,T] \\ \mathcal{B}[u(x,t)] = 0, \quad x \in \partial\Omega, \quad t \in [0,T] \\ \mathcal{I}[u(x,0)] = 0, \quad x \in \Omega \end{cases} \tag{2.12}$$

$\mathcal{F}$ is the PDE residual which contains several differential operators, $\mathcal{B}$ are the boundary conditions, and $\mathcal{I}$ is the initial condition.

In the framework of physics-informed neural networks, the solution of this forward problem is represented by a surrogate model $u_\theta(x,t)$ in the form of a fully-connected neural network that takes (x,t) as input and returns an approximation for u at this (x,t) as output. The parameter θ denotes the set of trainable parameters of the network (w,b) which propagates the input data through its l layers according to the sequence of operations

$$\begin{aligned} z^0 &= (x,t), \\ z^k &= \sigma(w^k z^{k-1} + b^k), \quad 1 \leq k \leq l-1, \\ z^l &= w^l z^{l-1} + b^l \end{aligned} \tag{2.13}$$

Each layer outputs a vector $z^k \in \mathbb{R}^{q_k}$, where q_k is the number of neurons, and is defined by a weigt matrix $w^k \in \mathbb{R}^{q_k \times q_{k-1}}$, a bias vector $b^k \in \mathbb{R}^{q_k}$, and a nonlinear activation function $\sigma(.)$. Finally, the output of the last layer is used to represent the predicted solution $z^l = u_\theta(x,t)$.

The parameters of the neural network $\theta = \left\{w^k, b\right\}_{k=1}^{l}$ are randomly initialized and iteratively updated by minimizing the loss function that enforces the PDE. The most common loss function is the mean squared error (MSE). The PINNs loss function consists of three error components, for the prediction of the neural network taking the form

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(\theta) + w_b \mathcal{L}_b(\theta) + w_i \mathcal{L}_i(\theta) \tag{2.14}$$

where

$$\mathcal{L}_f(\theta) = \frac{1}{N_f} \sum_{i=1}^{N_f} |\mathcal{F}[\hat{u}_\theta(x, y)]|_2 \tag{2.15}$$

$$\mathcal{L}_b(\theta) = \frac{1}{N_b} \sum_{i=1}^{N_b} |\mathcal{B}[\hat{u}_\theta(x, y)]|_2 \tag{2.16}$$

$$\mathcal{L}_i(\theta) = \frac{1}{N_i} \sum_{i=1}^{N_i} |\mathcal{I}[\hat{u}_\theta(x, y)]|_2 \tag{2.17}$$

Here $\mathcal{L}_f(\theta)$, $\mathcal{L}_b(\theta)$ and $\mathcal{L}_i(\theta)$ penalize the residuals of governing equations, the boundary conditions and the initial conditions, respectively with weights w_f, w_b, w_i and N_f, N_f and N_f are the numbers of data points for different terms.

In fact, when looking at the PINN loss, the term $w_f \mathcal{L}_f(\theta)$ acts as regularization with regularization parameter w_f in order to penalize those parameters θ which would lead to solutions not satisfying the PDE.

One advantage of PINNs by choosing neural networks as the surrogate of u is that we can take the derivatives of $\hat{u}$ with respect to its input x by applying the chain rule for differentiating compositions of functions using the automatic differentiation (AD), which is conveniently integrated in machine learning packages, such as TensorFlow and PyTorch

A critical underpinning of PINNs is the use of automatic differentiation (AD) to compute the loss (2.14). By using the chain rule to compose the derivatives of successive algebric operations, AD calculates the exact derivatives of the network output $u_\theta(x, t)$ with respect to its inputs x and t. Thus, the various loss components in (2.14) can be computed exactly without inheriting the truncation error incurred by standard numerical discretization schemes. Another advantage of computing derivatives with AD is that the residual points $\{x_i, t_i\}_{i=1}^{N_r}$ can be chosen arbitrarily, conferring PINNs their convenient mesh-free nature.

Now, we can find an optimum set of values θ^* that minimizes (2.14), by using gradient-based optimization, and the gradients of the loss function with respect to the weights are computed using backpropagation. The weights are then updated using the computed gradients (2.3) and a learning rate parameter that determines the step size of the update.

Tracking the behavior of the loss function $\mathcal{L}(\theta)$ allows the NN to adjust its internal parameters towards the values that lead to a smaller loss and thus to the best approximation of the PDE output by the NN.

(5).png (5).bb

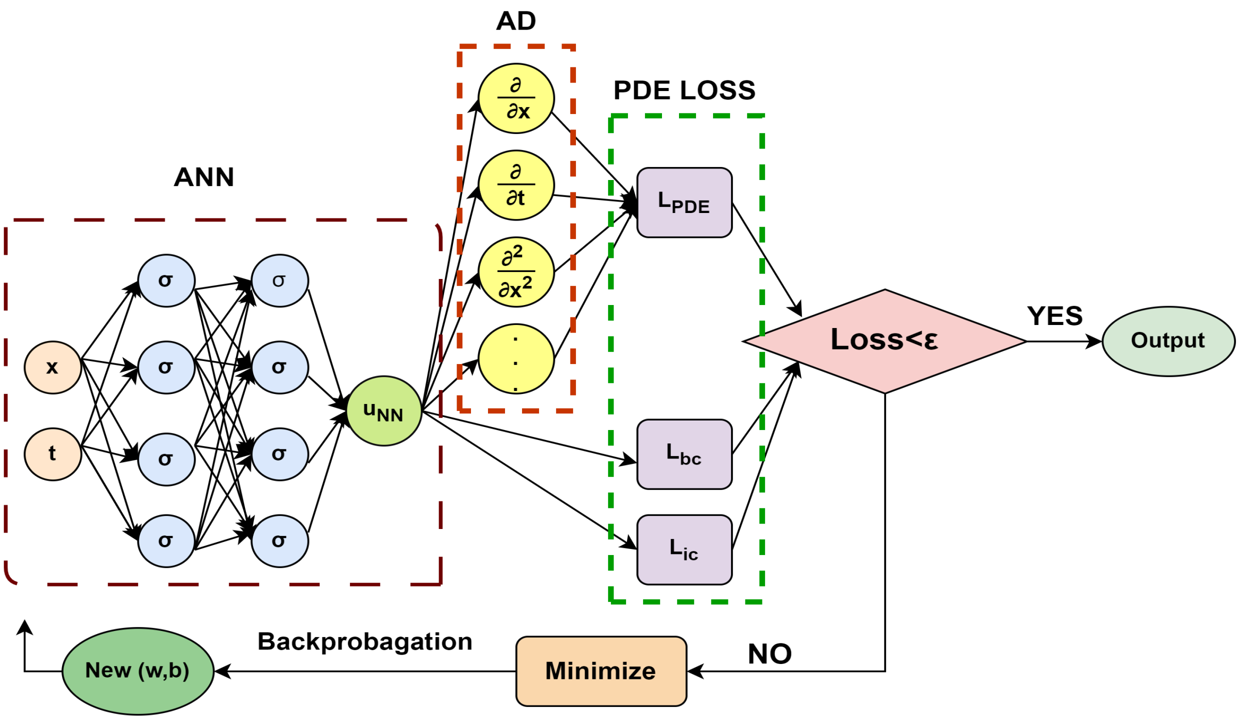


Figure 2.6: Neural network architecture diagrams of PINNs for investigating PDE solution

Here, we have an outline of an algorithm for training an ANN for PDE approximation using physics-informed neural networks (PINNs):

Taining an NN for PDE approxipmation Algorithm

1. Choose a suitable ANN architecture, including the number of layers, number of nodes per layer, and activation functions.

2. Formulate the PDE and its boundary conditions as a loss function, which measures

the difference between the predicted solution of the PDE and the true solution at the given input points.

3. Choose a suitable optimization algorithm, such as Adam or SGD.
4. Divide the available data into a training set and a testing set.
5. Iterate the following steps until convergence or a stopping criterion is met:
   (a) Sample input points from the domain of the PDE and its boundary conditions, as well as from the training data (if available).
   (b) Feed the input points into the network.
   (c) Compute the gradients of the loss function with respect to the weights using automatic differentiation.
   (d) Update the weights using the computed gradients and the chosen optimization algorithm.
6. Evaluate the performance of the trained network using a testing set or by solving the PDE at new input points.

2.4.3 Application Library: DeepXde

DeepXDE (Deep eXtreme Diffirential Equation) is a deep learning library specifically designed for solving partial differential equations (PDEs) using neural networks. It is an open-source Python library that provides a high-level interface for training neural networks to approximate solutions to PDEs.

DeepXDE makes the code stay compact and nice, resembling closely the mathematical formulation. Solving differential equations in DeepXDE is no more than specifying the problem using the build-in modules, including computational domain (geometry and time), PDE equations, boundary/initial conditions, constraints, training data, neural network architecture, and training hyperparameters. The workflow is shown in the following Procedure [?]:

1. Specify the computational domain using the geometry module.
2. Specify the PDE using the grammar of TensorFlow.
3. Specify the boundary and initial conditions.
4. Combine geometry, PDE and BCs/ICs together into data.PDE or data.TimePDE for time-independent or time-dependent problems, respectively. Specify the training data by either set the specific point locations or only set the number of points.
5. Construct a NN using the maps module.
6. Define a Model by combining the PDE problem in 4 and the NN in 5.
7. Call Model.compile to set the optimization hyperparameters such as optimizer and learning rate. The weights wf , wb can be set here by loss-weights.
8. Call Model.train to train the network from random initialization or a pretrained model using the argument model-restore-path. It is extremely flexible to monitor and modify the training behavior using callbacks.
9. Call model.predict to predict the PDE solution at diferent locations.

Chapter 3

Applications of Artificial Neural Network in Approximation

In this chapter we're going to apply the artificial neural network on some equations, problems and systems to see the performance of the method on solution approximation.

To illustrate the use of PINNs and deepxde, we provide some examples of numerical simulations of PDEs.

3.1 Heat Equation

Consider the one-dimensional heat equation:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha \dfrac{\partial^2 u}{\partial x^2} \qquad x \in [-1,1], \quad t \in [0,1] \\ u(0,t) = u(1,t) = 0 \\ u(x,0) = \sin(\pi x) \end{cases} \tag{3.1}$$

where $u(x,t)$ is the temperature at position x and time t, α is the thermal diffusivity, where $\alpha = 0.4$

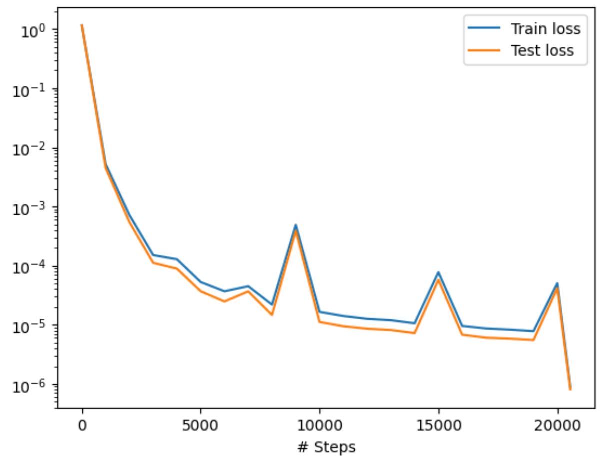
The exact solution is: $e^{-\pi^2 0.4t}\sin(\pi x)$

To solve this equation with PINNs and deepxde, we first define a neural network that takes as input (x, t) and outputs the temperature $u(x, t)$. We then train the neural network to minimize the loss function:

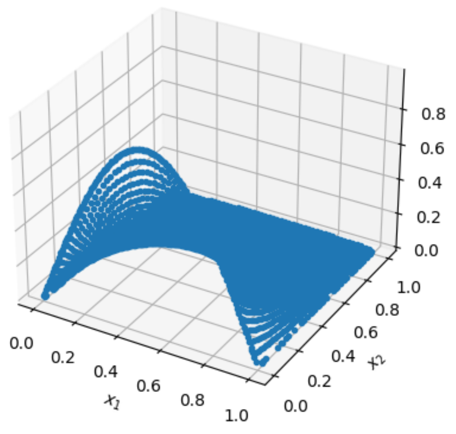
$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}|u(x_i, t_i) - u_i|^2 + \frac{1}{M}\sum_{j=1}^{M}|g_i(x_j, t_j)|^2,$$

where N is the number of data points, M is the number of boundary conditions, u_i is the temperature at (xi, ti) from the training data, $g_j(xj, tj)$ are the boundary conditions. The network is constructed of 2 inputs, 3 hidden layers, each one contains 20 neurons and an output u. The optimizer used in this structure is Adam with learning rate $(lr = 1e^{-3})$ After we continue to train the network using L-BFGS optimizer to achieve a smaller loss.

Here are the results:



(a) The loss history



(b) The predicted solution

3.2 Second Order ODE

We have the following ODE system:

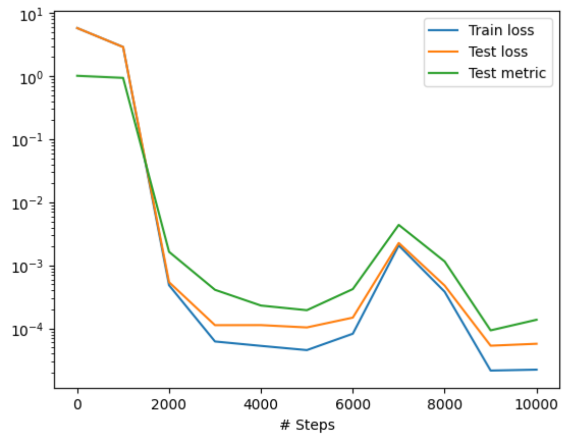
$$\begin{cases} y''(t) - 10y'(t) + 9y(t) = 5t, \quad t \in [0, 0.25] \\ \\ y(0) = -1, \; y'(t) = 2 \end{cases}$$

with the exact solution: $y(t) = \dfrac{50}{81} + 59t + \dfrac{31}{81}e^{9t}2e^{t}$.

We define a fully connected feedforward neural network with 3 hidden layers of 50 neurons each, using the hyperbolic tangent activation function and Glorot uniform weight initialization. We compile the model using the Adam optimizer with a learning rate of 0.001, we also compute the L^2 relative error as a metric during training.

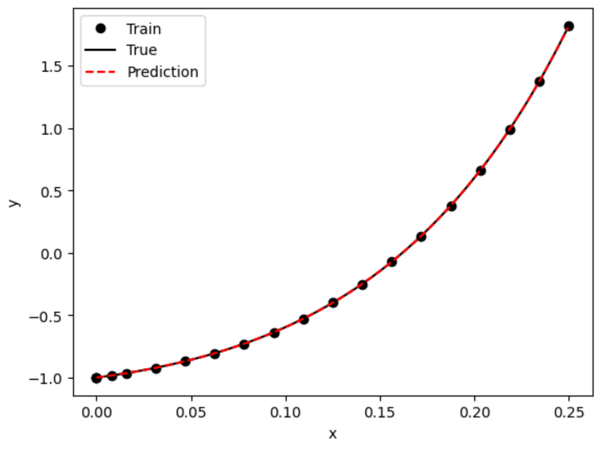
Here are the results:

loss.png loss.bb



(a) The loss history

sol.png sol.bb



(b) The predicted solution

3.3 Volterra Integration Differential Equation

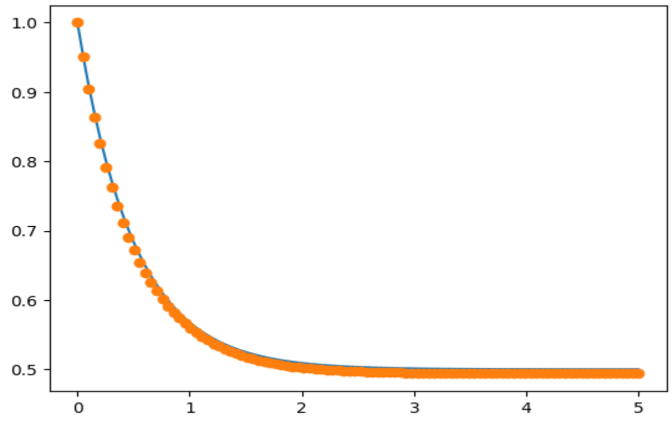
Cosnider the following Volterra IDE:

$$\frac{dy}{dx} + y(x) = \int_0^x e^{t-x} y(t)dt \tag{3.2}$$

with the initial condition $y(0) = 1$

Our neural network is constructed of 3 hidden layers with 20 neurons each with tangent hyperbolic activation function and Glorot uniform weight initialization. We compile the model using Adam optimizer.

Here are the result of the model:



10000.png 10000.bb

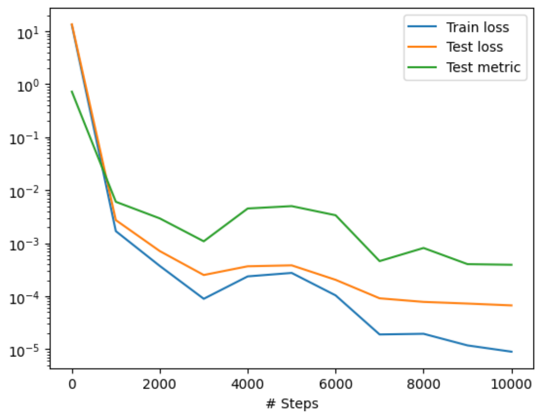
Figure 3.3: The predicted solution and The exact solution

3.4 Diffusion equation with hard initial and boundary conditions

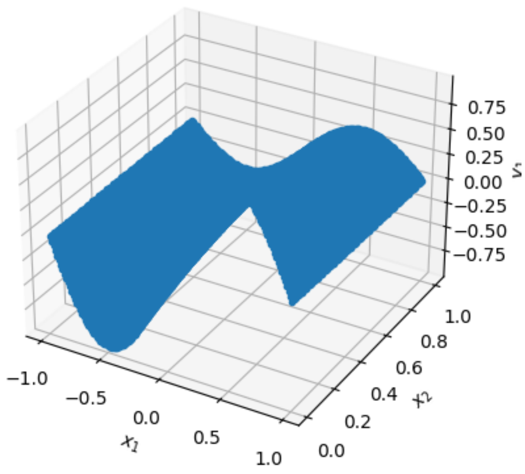
$$\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2} - e^{-t}(sin(\pi x) - \pi 2 sin(\pi x)), \quad x \in [-1, 1], \quad t \in [0, 1] \tag{3.3}$$

and the Dirichlet boundary condition $y(1, t) = y(1, t) = 0$

Here, we use a fully connected tangent hyperbolic neural network of depth 4 (i.e., 3 hidden layers) and width 32. We compile the model using Adam optimizer with learning rate of 0.001. We then train the model for 10000 iterations. Here are the predicted solution of (3.3)compared to the exact solution.



(a) The loss history



(b) The predicted solution

3.5 Fractional Boundary Value Problem of nonlinear functional equation

Consider the following problem

$$\begin{cases} \mathcal{D}_{0+}^{\sigma_1,\psi}(\mathcal{D}_{0+}^{\sigma_2,\psi}u)(t) = h(t,u(t)), & t \in [0,1], \\ u(0) = 0, \quad u(1) = \lambda \mathcal{I}_{0+}^{\sigma_3,\psi} g(\xi, u(\xi)), \end{cases} \tag{3.4}$$

where $0 < \sigma_1, \sigma_2, \xi < 1, \sigma_1 + \sigma_2 > 1$ and $h, g : [0,1] \times \mathbb{R} \to \mathbb{R}$, are two continuous functions, $\mathcal{D}_{0+}^{\sigma,\psi}$ is the ψ-Riemann-Liouville fractional derivative of order $\sigma \in \{\sigma_1, \sigma_2\}$ which is defined on chapter 1, which depends on an increasing function $\psi, \mathcal{I}_{0+}^{\sigma_3,\psi}$ is the ψ-Riemann-Liouville integral of order $\sigma_3 > 0$ and $\lambda > 0$.

Proposition 3.1 [?] Let $0 > \sigma_1, \sigma_2, \xi < 1$, $\sigma_1 + \sigma_2$, $\lambda, \sigma_3 > 0$, and $h, g : [0,1] \times \mathbb{R} \to \mathbb{R}$ are two continuous functions. Then the fractional boundary valus problem (3.4) is equivalent to the following integral equation

$$\begin{aligned} u(t) =& \frac{1}{\Gamma(\sigma_1+\sigma_2)} \int_0^t \psi'(s)(\psi(t)-\psi(s))^{\sigma_1+\sigma_2-1} h(s,u(s))ds \\ &- [\eta(t)]^{\sigma_1+\sigma_2-1} \Big[\frac{1}{\Gamma(\sigma_1+\sigma_2)} \int_0^1 \psi'(s)(\psi(1)-\psi(s))^{\sigma_1+\sigma_2-1} g(s,u(s))ds \\ &- \frac{\lambda}{\Gamma(\sigma_3)} \int_0^\xi \psi'(s)(\psi(\xi)-\psi(s))^{\sigma_3-1} g(s,u(s))ds \Big] \end{aligned} \tag{3.5}$$

where $\eta(t) = \Big(\frac{\psi(t)-\psi(0)}{\psi(1)-\psi(0)}\Big)^{\sigma_1+\sigma_2-1}$.

Proof: See [page 4, [?]] ■

Existence and uniquness result

Theorem 3.1 [?] Assume that the following assertions holds

$(\mathcal{AS}_1)$ There exists a real constant $\Lambda > 0$ such that

$$|h(t,u) - h(t,v)| \leq \Lambda |u-v|, \quad t \in [0,1], \quad u, v \in \mathbb{R}.$$

$(\mathcal{AS}_2)$ There existes an $\psi - RL$ integrable function $\Theta : [0,1] \to \mathbb{R}^+$ such that

$$|g(t,u) - g(t,v)| \leq \Theta(t)|u-v|, \quad t \in [0,1], \quad u, v \in \mathbb{R}.$$

$(\mathcal{AS}_3)$ The real constant k satisfies

$$0 < k = \frac{2\Lambda(\psi(1) - \psi(0))^{\sigma_1+\sigma_2}}{\Gamma(\sigma_1+\sigma_2+1)} + \lambda \mathcal{I}_{0+}^{\sigma_3,\psi}\Theta(\xi) < 1$$

Then, the FBVP (3.4) admits a unique solution.

Proof: See[page 6, [?] ■

3.5.1 Approximation of the solution via artificial neural networks

For our numerical application, we have the following FDP:

$$\begin{cases} \mathcal{D}_{0+}^{\frac{1}{2},t^2}(\mathcal{D}_{0+}^{\frac{3}{4},t^2}u)(t) = \frac{1}{4}u(t) + \frac{\Gamma(2.5)}{\Gamma(1.25)}t^{\frac{1}{2}} - \frac{1}{4}t^3, & t \in [0,1], \\ u(0) = 0, \quad u(1) = \frac{256}{3\pi+256}\int_0^{\frac{1}{2}} s(\frac{1}{4} - s^2)^{\frac{-1}{2}}(2+u(s))ds, \end{cases} \tag{3.6}$$

In this structure we have:

$\psi(t) = t^2, \quad = \frac{1}{2}, \quad \sigma_1 = \frac{1}{2}, \quad \sigma_2 = \frac{3}{4}, \quad \sigma_3 = \frac{1}{2}, \quad \lambda = \frac{128\sqrt{\pi}}{3\pi+256},$

$h(t,u(t)) = \frac{1}{4}u(t) + \frac{\Gamma(2.5)}{\Gamma(1.25)}t^{\frac{1}{2}} - \frac{1}{4}t^3$ and

$g(t,u(t)) = 2 + u(t)$

Applying the density result of Theorem (2.3) to the solution u of the integral equation (3.5) and its compact fixed point formulation $\mathcal{F}u(t) = u(t)$ where $\mathcal{F} : \mathbb{Y} \to \mathbb{Y}$ with $\mathbb{Y} = \mathcal{C}([0,1],\mathbb{R})$ equipped with the norm $||u|| = \max_{t\in[0,1]}|u(t)|$, is defined by (3.5), states that for all $\epsilon > 0$, there exists a function $u_h \in \mathcal{NN}_{1,1,4}^{\rho}$ such that $||\mathcal{F}u_h - u_h||_\infty < \epsilon$ which h depends on weights and biases.

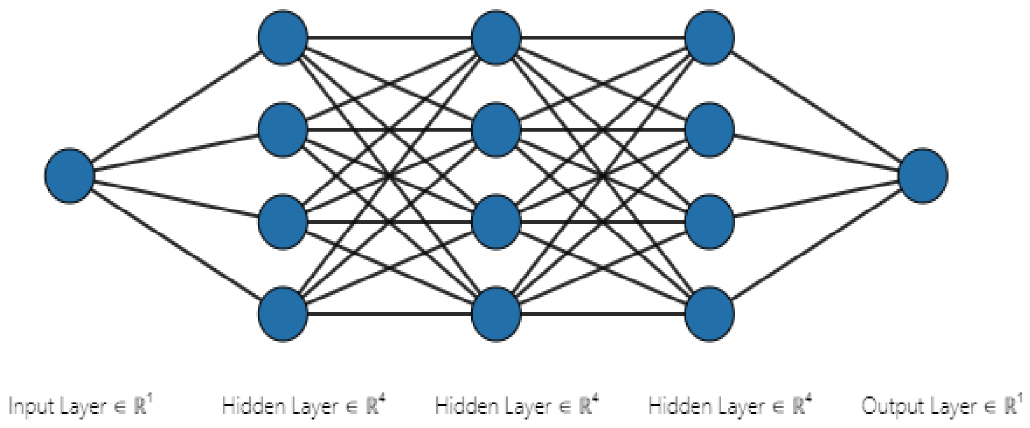


Figure 3.5: The correponding architecture of an artifcial neural networks of functions of the set $\mathcal{NN}^{\rho}_{1,1,4}$ with three hidden layers

.

Here are the yielded results:

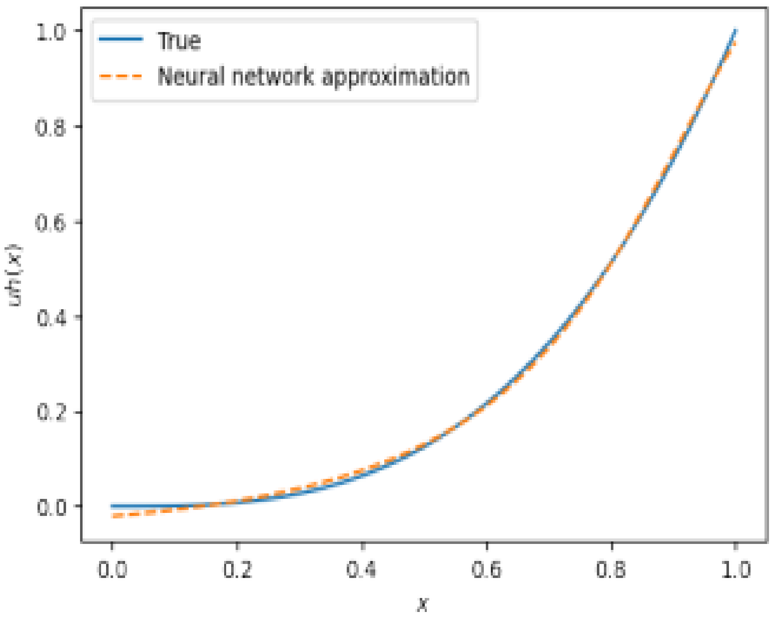


Figure 3.6: The exact soution compared to its approximate solution. Loss function= MSE, optimiser = Adam, learning rate = 0.01, number of epochs = 1000

3.6 Fractional integro-differential value problem

Consider the following IVP:

$$\begin{cases} {}^c\mathcal{D}^{\alpha}_{0+}\phi(x) + \sum_{i=0}^{m} \lambda_i \phi^{(i)}(x) + \mu h(\phi(x), \phi'(x), ..., \phi^{m}(x)) \\ + \int_a^b \mathcal{K}(x,t)\phi(t)dt = \xi(x), \qquad x \in [0,1], \phi(0) = d_0, \\ \phi(0) = d_0, \phi'(0) = d_1, ..., \phi^{(m)}(0) = d_m, \end{cases} \tag{3.7}$$

where $m \leq 1$, $m < \alpha < m+1$, $a, b, \mu, \lambda_i, d_i (i = 0, 1, ..., m)$ are known constant real numbers, $h : \mathbb{R}^{n+1} \longrightarrow \mathbb{R}$ is a continuous function and ${}^c\mathcal{D}^{l}_{0+}$ stands the Caputo fractional derivative of order l and $\mathcal{K}(x,t)$ is a separable kernel which can be expressed as follows:

$$\mathcal{K}(x,t) = \mathcal{V}(x)\phi(t)$$

We consider the Banach space $\mathcal{B} = C^2([0,1], \mathbb{R})$ (the space of all functions of class C^2 in $[0,1]$ with values in $\mathbb{R}$ endowed with the norm $|\phi| = \max_{x \in [0,1]} |\phi(x)|$, the operator $\mathcal{G} : \mathcal{B} \to \mathcal{B}$ by:

$$\begin{aligned} \mathcal{G}\phi(x) = & \sum_{k=0}^{2} \frac{d_k}{k!} x^k + I^{\alpha}_{0+}\xi(x) - \frac{\lambda_0}{\Gamma(\alpha)} \int_0^x (x-s)^{\alpha-1}\phi(s)ds - \sum_{k=0}^{2} \frac{\lambda_k}{\Gamma(\alpha-k)} \int_0^x (x-s)^{\alpha-k-1}\Big(\phi(s) \\ & - \sum_{0}^{1} \frac{d_j}{j!} s^j\Big) ds - \frac{\mu}{\Gamma(\alpha)} \int_0^x (x-s)^{\alpha-1} h(\phi(s), \phi'(s), \phi''(s)) ds \\ & - \frac{1}{\Gamma(\alpha)} \int_0^x (x-s)^{\alpha-1} v(s) ds \int_a^b \phi(t)\phi(t) dt. \end{aligned} \tag{3.8}$$

is equivalent to (3.7).

3.6.1 Approximation of solution via ANN

Here, we are interested in the following IVP:

$$\begin{cases} {}^c\mathcal{C}^{\frac{5}{2}}_{0+}\phi(x) + \phi''(x) + \sqrt{|\phi'(x)|} + 6^3\sqrt{1+\phi(x)} + sin\left(\dfrac{\phi''(x)}{6}\right) + e^x \int_0^1 t\phi(t)dt = \xi(x), \\ \phi(0) = -1, \quad \phi'(0) = 0, \quad \phi''(0) = 0. \end{cases} \tag{3.9}$$

where $\xi(x) = \dfrac{6}{\Gamma(1.5)}\sqrt{x} + (12+\sqrt{3})x + sinx - \dfrac{3}{10}e^x$

Using the density property in the space $C^2([0,1],\mathbb{R})$ of the set of mappings constructed by (ANN) and the continuity of the operator $\mathcal{G}$ and its fixed point equation $\mathcal{G}\phi(x) = \phi(x)$ where $\mathcal{G}$ is defined by (3.8), one affirms that for any $\epsilon > 0$, there exists a function $\phi_h \in \mathcal{NN}_{1,1,4}$ satisfies

$$||\mathcal{G}\phi_h - \phi_h||_2 < \epsilon$$

which h depends on weights and biases.

Here are the yeilded result:

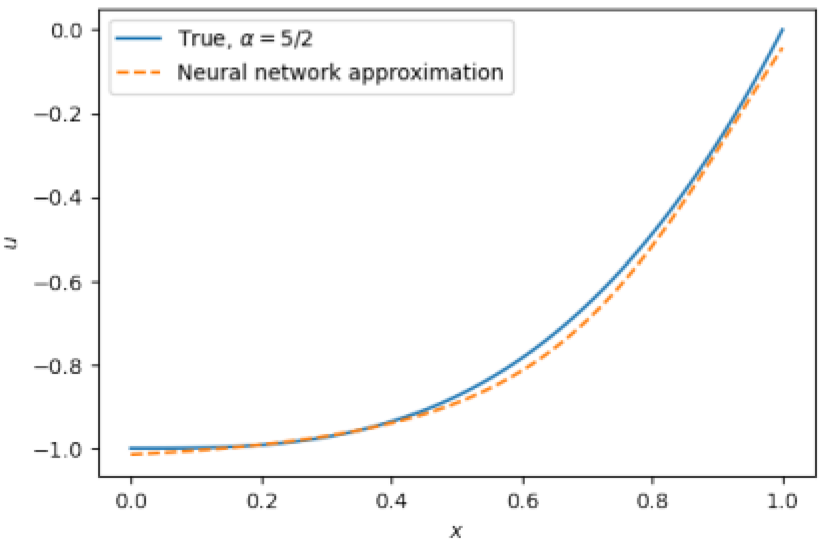


Figure 3.7: The exact solution compared to its (ANN) approximate solution

3.7 Fractal-fractional order mathematical model on the spread of Coronavirus (COVID-19)

Consider the following fractal-fractional order mathematical model on the spread of the Coronavirus (COVID-19):

$$\begin{cases} {}^{\mathcal{FPP}}\mathcal{D}_{0,t}^{\varrho,\nu} S(t) = \mu N(t) - \dfrac{\beta S(t) I(t)}{1+\alpha I(t)} - \mu S(t), \\ {}^{\mathcal{FPP}}\mathcal{D}_{0,t}^{\varrho,\nu} I(t) = \dfrac{\beta S(t) I(t)}{1+\alpha I(t)} - (\gamma+\mu) I(t), \\ {}^{\mathcal{FPP}}\mathcal{D}_{0,t}^{\varrho,\nu} R(t) = \gamma I(t) - \mu R(t), \end{cases} \tag{3.10}$$

endowded with the initial conditions

$$S(0) = S_0, I(0) = I_0, R(0) = R_0$$

where ${}^{\mathcal{FPP}}\mathcal{D}_{0,t}^{\varrho,\nu}$ is the fractal-fractional derivative with the fractional order $\varrho \in (0,1]$ and the fractal order $\nu \in (0,1]$ via the power law type kernel. Note that all parameters used in the model 3.10 are non-negative and the models state functions are defined by

$$N(t) = S(t) + I(t) + R(t),$$

where $N(t)$ is the total population at the time $t \in \mathbb{O} := [0,T], (T > 0)$, βI and $\frac{1}{1+\alpha I}$ represent respectively the infection force of the disease and the crowding effect. Moreover, the parameters μ, γ and β denote respectively, the death rate, the recovery rate and the transmission coefficient.

The fractal-fractional equations system (3.10) is equivalent to the following integral equations system:

$$\begin{cases} S(t) = S_0 + \dfrac{\nu}{\Gamma(\varrho)} \int_0^t v^{\nu-1}(t-v)^{\varrho-1}\mathcal{Z}_1(v, S(v), I(v), R(v))dv, \\ I(t) = I_0 + \dfrac{\nu}{\Gamma(\varrho)} \int_0^t v^{\nu-1}(t-v)^{\varrho-1}\mathcal{Z}_2(v, S(v), I(v), R(v))dv, \\ R(t) = R_0 + \dfrac{\nu}{\Gamma(\varrho)} \int_0^t v^{\nu-1}(t-v)^{\varrho-1}\mathcal{Z}_3(v, S(v), I(v), R(v))dv, \end{cases} \tag{3.11}$$

where $\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3$ are the second members of the system (3.10).

Existence Result

Theorem 3.2 Let $\mathcal{Z} \in C(\mathbb{O} \times X, X)$. Suppose that

$(\mathcal{HY}_1)$ There exist $\varphi \in L^1(\mathbb{O}, [0, +\infty))$ and an increasing function $\mathbb{B} \in C([0, +\infty), (0, +\infty))$ such that

$$\forall t \in \mathbb{O}, \text{and} \quad k \in X, \quad |\mathbb{B}(t, k(t)| \leq \varphi(t)\mathbb{B}(|k(t)|;$$

$(\mathcal{HY}_2)$ There exists $\alpha > 0$ so that

$$\alpha > k_0 + \frac{\varrho T^{v+\nu-1}\Gamma(\varrho}{\Gamma(\varrho+\nu)}\varphi_0^*\mathbb{B}(\alpha),$$

Then the given fractal fractional model of (COVID-19) (3.10) admits a solution on $\mathbb{O}$.

Uniquness Result

Theorem 3.3 Let the functions $S, I, R \in C(\mathbb{O}, \mathbb{R})$ and assume that $||S|| \leq \lambda_1, ||I|| \leq \lambda_2, ||R|| \leq \lambda_3$, for some constants $\lambda_1, \lambda_2, \lambda_3 > 0$. Then the given fractal-fractional model of (COVID-19) (3.10) has a solution if

$$\frac{\varrho T^{\varrho+\nu-1}\Gamma(\varrho)}{\Gamma(\varrho+\nu)} l_i < 1, \quad i \in \{1, 2, 3\}$$

3.7.1 Approximation of the solution via artificial neural networks

Using the numerical solutions obtained by Adams-Bashforth, we simulate and discuss the behavior of our mathematical model based on different values of the parameters $\mu = 0.15$, $\beta = 0.55$, $\alpha = 0.45$, $\gamma = 0.25$ and $T = 50$, with the following initial condition values for state functions $S(0) = 80$, $I(0) = 40$, $R(0) = 20$.

Applying the density result of Theorem (2.3) to the solution k of the system of integral equations (3.11) and its compact fixed point formulation $Gk(t) = k(t)$ where $G : \mathcal{X} \to \mathcal{X}$, $\mathcal{X} = \mathbb{K} \times \mathbb{K} \times \mathbb{K}$, with $\mathbb{K} = C(\mathbb{O}, \mathbb{R})$ is defined by

$$Gk(t) = k(0) + \frac{\nu}{\Gamma(\varrho)} \int_0^t v^{\nu-1}(t-v)^{\varrho-1} \mathcal{Z}(v, k(v))dv \tag{3.12}$$

where $k(0) = k_0 = (S(0), I(0), R(0))$ and $k(t) = (S(t), I(t), R(t))$

$$||k||_{\mathcal{X}} = ||(S, I, R)||_{\mathcal{X}} = \max\{|S(t)| + |I(t)| + |R(t)| : \quad t \in \mathbb{O}\}$$

and $\mathcal{Z} = (\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3)$, it states that for all $\epsilon > 0$, there exists a function $k_h \in \mathcal{NN}^{\rho}_{1,3,6}$ such that $||Gk_h - k_h||_2 < \epsilon$ where h depends on weights and biases.

Here are the yielded result:

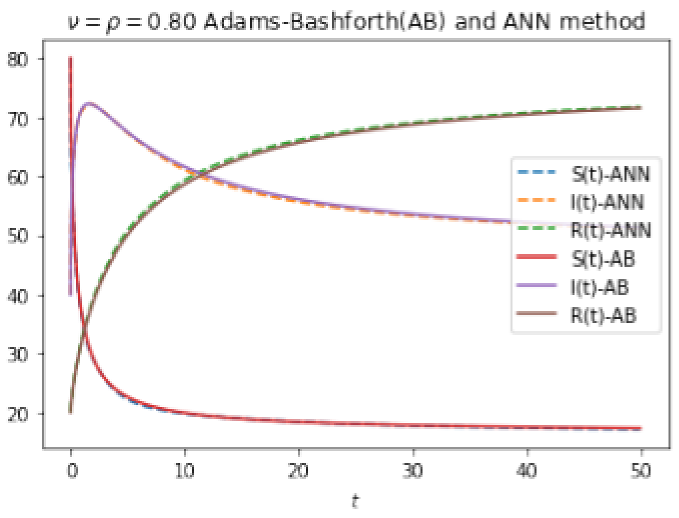


Figure 3.8: Numerical simulation for fractal-fractional dimension-order $\varrho = \nu = 0.80$ via two approximation methods. For ANN method, we used: learning rate=0.001, Optimizer = Adams, Number of epochs = 30000

Conclusion and percpective

In this master thesis, we have introduced the concept of Approximation Theory with Artificial Neural Networks (ANNs), including the universal approximation property of ANNs, review some error estimation results and providing numerical simulations with Physics-Informed Neural Networks (PINNs) using the deepxde library on several applications. Our results show that ANNs are a powerful tool for approximation theory, so it can be used to approximate functions and solutions of partial differential equations with high accuracy and efficiency. The use of PINNs and deepxde allows for the incorporation of physical constraints into the training process, which leads to more robust and accurate solutions. The study could be extended to use the ANN methods to solve more complex fractional boundary problems and variational inequalities problems.