DATA-DRIVEN DEEP LEARNING ALGORITHMS FOR DYNAMICAL SYSTEMS

By

Ebenezer Otoge Oluwasakin

Dissertation Defense

Submitted in Partial Fulfillment

of the Requirement for the Degree of

Doctor of Computational and Data Science

Middle Tennessee State University

March, 2024

Dissertation Committee:

Prof. Abdul Khaliq (Mathematical Sciences), Chair.

Prof. Wandi Ding (Mathematical Sciences).

Prof. Jing Kong (Chemistry)

ABSTRACT

Artificial neural networks have revolutionized scientific problem-solving, offering reliable techniques for modeling and understanding complex processes. This trend is most apparent in dynamical system simulation, where experts continuously seek methods to increase accuracy and efficiency. Learning the time-varying parameters of a dynamical system is essential, especially when trying to understand how a system reacts to different situations over time. We present three deep learning algorithm approaches to address the learning of time-varying parameters from a dynamical system. The first algorithm, a logistic-informed neural network, is motivated by using physics-informed neural networks on logistic differential equations to predict the number of individuals infected by the COVID-19 Omicron variant. This algorithm learns the time-varying parameters of four mathematical models to predict individuals infected with the COVID-19 Omicron in a country with strict and partial mitigation measures. The second algorithm, optimized physics-informed neural networks, allows us to understand nonlinear dynamics in various fields, such as biochemistry, ecology, and epidemiology. By optimizing the model, a constant parameter of a system of ordinary differential equations can be learned as a time-varying parameter, revealing hidden patterns in complex systems. Finally, we tackle the difficulties of learning the time-varying parameter and solving stiff dynamical systems by introducing a novel approach called physics-informed transfer learning neural network. This model is developed by transferring prior knowledge of optimized neural network parameters and pre-training using physics-informed neural networks with a simple adaptive scheme to learn the time-varying parameters of stiff dynamical systems. These algorithms improve neural network's capacities for analyzing dynamic systems, learning time-varying parameters, and predicting system behavior. We are confident that our research significantly contributes to the journey toward more complex and accurate modeling of dynamic processes.

This work is dedicated to:

my wife, Janet Oluwasakin, and my dad's Pastor. Oyebola Tolorunlogo and Mr. Julius Otegbade

my mom, Deaconess Grace Tolorunlogo, and my sisters, Mercy, Goodness, and Susan Tolorunlogo

my brothers, Dunsin and Darasimi Tolorunlogo.

my beloved pastor, Pastor (Mrs) Temitayo Falade, and Mr. Adeniyi Falade.

ACKNOWLEDGMENTS

I am grateful to God for His wisdom and guidance throughout my Ph.D. journey. Numerous individuals have supported my path to whom I owe a great deal of gratitude. Firstly, I extend my deepest appreciation to my advisor, Professor Abdul Khaliq, for his unwavering care, guidance, and mentorship, which have been instrumental in my progress. I am also thankful for the support and encouragement of Professor John Wallin, the program director. My gratitude extends to Professor Wandi Ding and Professor Jing Kong for their willingness to serve on my committee.

I am privileged to have been a part of the CRUNCH group at Brown University, under the direction of Professor George Em Karniadakis. This groups weekly seminars on cutting-edge computational scientific research and their pioneering work on physics-informed neural networks have not only inspired me but also significantly influenced my own research. I am grateful to the faculty and my colleagues in the University Studies and Computational and Data Science Program, including Dr. Rebecca Calahan, Dr. Jeremy Strayer, Dr. Olumoyin Kayode, and Dr. Thomas Torku, for their continual encouragement. I also acknowledge Dr. Khalid Furati of KFUPM for his valuable insights that have significantly shaped my research direction.

I am also pleased to acknowledge the support of my beloved wife, Janet Oluwasakin, and the unwavering backing of my parents, Pastor and Deaconess Tolorunlogo. My siblings, in-laws, and friends have also supported me throughout this journey. Additionally, I am thankful for the spiritual nourishment provided by the pastor and congregation of the Redeemed Christian Church of God (Agape) in Clarksville, Tennessee. Lastly, my most profound gratitude goes to God Almighty, from whom all wisdom flows.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In many scientific and engineering fields, understanding and predicting a wide range of complex problems is all about understanding and predicting dynamic systems. These systems often employ ordinary differential equations (ODEs) to describe and model the intricate relationships inherent in various natural and artificial processes. One such common usage is seen in compartmental models, extensively used across fields such as ecology [54], epidemiology [62], chemical engineering [58], mathematical biology [69], and economics. These models require careful estimation of numerous unknown parameters for their validity, often derived from data collected from experiments or real-world observations. This process, termed calibration of the dynamical system, constitutes a challenging optimization problem due to its iterative nature and potential convergence issues. Keeping the gap between observed data and model forecasts as small as possible is also important.

This makes the accurate identification of parameters a key step in developing models. Moreover, process optimization has greatly increased the demand for dynamical system models in chemical process engineering that accurately identify the parameters [49]. However, models with non-linear parameters, like ordinary differential equations (ODEs) or differential-algebraic equations (DAEs), make parameter estimation even more challenging. There are a lot of different system identification techniques out there, such as the Gauss-Newton method [39], multiple shooting, recursive estimation [26], maximum likelihood estimation, Markov chain Monte Carlo-based Bayesian inference [20, 29], finite element methods [57], and collocation methods [38]. However, each has some problems, like needing a lot of computing power and relying on initial conditions. Despite the challenges, the role of parameter identification in model accuracy and predictability remains paramount, calling for continued innovation and research in this fundamental mathematical field.

Artificial intelligence (AI) has modified many fields by allowing scientists to model complex problems. They are also becoming more and more useful for solving difficult scientific problems. Still, humans continue searching for quicker, more accurate approaches to simulate dynamic systems. Artificial intelligence (AI), specifically implementing machine learning and deep learning, has greatly increased applications for interpreting complex structures [56, 11]. This is particularly genuine for dynamic systems, which present complex interactions requiring analytic tools. Although these AI models have demonstrated remarkable efficacy in data fitting and short-term prediction tasks, their implementation may need to be improved due to certain obstacles. One major barrier is that it takes work for AI models to understand the underlying structures of a dynamic system's activities [70, 71, 47]. The resulting AI model predictions and real system behaviors may be at odds with one another. Additionally, the performance of these models relies on the quantity and quality of input data, potentially resulting in inadequate insights when the data fails to portray the system's dynamics.

Chapter 1 provides a comprehensive literature review that lays the foundational base for this study. Section 1.1 introduces the concept of deep learning, providing a detailed overview of its key principles and its relevance to the wider scope of our research. Section 1.2 presents an in-depth analysis of various neural network architectures. This section critically examines the perceptron, single-layer perceptron, and multi-layer perceptron and discusses activation functions and backpropagation. In Section 1.3, the focus shifts to deep neural networks. Here, we explore their role in accurately learning the parameters and dynamics present in systems of differential equations. Finally, Section 1.4 highlights the vital role of data-driven simulations. It demonstrates how these simulations are integral to the efficacy and accuracy of our computational models. This chapter aims to provide a robust theoretical framework that supports the subsequent chapters of this study.

1.1 Deep Learning

Deep learning involves training multi-layer perceptron models to understand and represent data effectively. It's becoming increasingly popular for its ability to manage complex and high-dimensional problems. This growing field extends across various domains, including science and engineering, geography, chemistry, physics, finance, and business. One of the key reasons for its widespread use is its ability to perform tasks in scientific applications, such as driving autonomous vehicles, detecting objects, and recognizing images.

Its versatility in handling a wide array of tasks, from simple classification to complex predictive modeling, makes it a valuable tool in technology and research. Its applications in healthcare, such as disease detection and drug discovery, have shown promising results. Moreover, in the field of natural language processing, deep learning has revolutionized how machines understand and generate human language, enabling more intuitive and efficient human-computer interactions. Additionally, deep neural networks are proficient in solving both partial and ordinary differential equations, expanding their utility in mathematical and scientific problem-solving. The primary models in deep learning, renowned for their effectiveness and versatility, are known as neural networks [70]. These models have garnered significant attention and widespread use in various fields due to their robustness in handling complex tasks.

1.2 Neural Network Architectures

Many problems in real life need solutions. Some problems are easy to solve because they are linear. The linear regression model can solve this type of linear problem; however, some complex problems in real life are not easy to solve because they are non-linear. The neural network (NN) is good at solving both linear and non-linear problems. Neural networks form the base of deep learning, a subset of machine learning where the algorithm mimics the structure or reflects the human brain's behaviors [70]. In other words, a neural network works like the human brain, which takes in a piece of information (data) called the input with a target output, processes the information through different thinking (logic) called the hidden layer, and gives out a result called the actual

output. It is a mathematical procedure optimized or taught to produce the desired output. This can be used for anything that needs to make some prediction based on evidence that consistently takes the same form of representation.

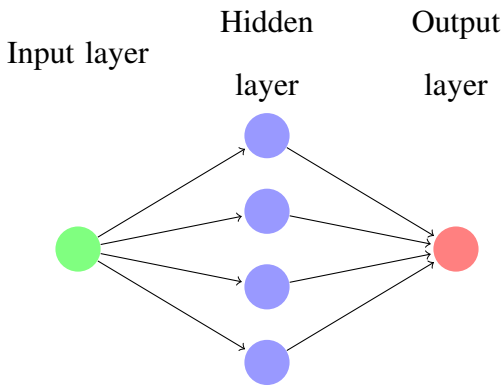


Figure 1.1: A Simple Neural Network

Figure 1.1 above shows a simple neural network consisting of the input, hidden, and output layers. The network always has one input layer and one or many hidden layers with one output layer. The input layer takes in the data or information provided to the neurons, which pass through the hidden layers. We have the weight, bias, and activation functions inside the hidden layers. These three work together to produce the output. This process is called the perceptron.

1.2.1 The Perceptron

The perceptron is a neural network (NN) that takes in dataset or information as inputs, performs some tasks, or does certain computations or algorithms on the input data to produce actual output. This computation or algorithm empowers neurons to learn and process components in the training set one at a time. Perceptron is used when a dataset is linearly separable. It has two parameters which are the weights and the learning rate. The weight is the number of dimensions, also known as the coefficient. The learning rate helps train the neural network and shows how fast the perceptron converges. The higher the learning rate, the faster it will take steps to the optimal line, but it may make it overshoot, which can result in mistakes. The lower the learning rate, the slower it will take to the optimal line, but it helps make fewer or reduce mistakes. Perceptron was developed and

introduced by Frank Rosenblatt in 1958 [15]. Although before Rosenblatt introduced perceptron, McCulloch-Pitts(MCP) introduced a neuron in 1943 [67] that works with binary inputs $I_n \in \{0,1\}$ where one is true, and 0 is false. The neuron's function becomes

$$y = \begin{cases} 1 & \sum_{n=1}^{K} I_n w_n > \beta \\ 0 & otherwise \end{cases}$$

Where β is the threshold, the MCP neuron shows that the weight and the threshold may be the same value, while the MCP neuron extension done by Rosenblatt states that the weight and thresholds may have a different value. There are two types of perceptron, single-layer and multi-layer perceptron.

1.2.2 Single-layer Perceptron

A single-layer perceptron can learn only linearly separable patterns.

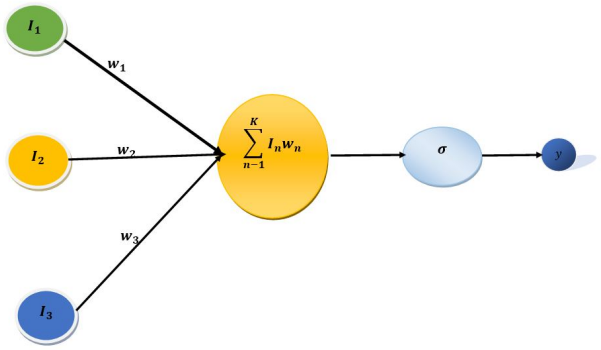


Figure 1.2: Single-layer Perceptron

Let

$$m = \sum_{n=1}^{K} I_n w_n$$

The single-layer perceptron in Figure 1.2 produces the following output:

$$y = \sigma(m)$$

Where I is the input, w is the weight, and σ is the activation function.

1.2.3 Multi-layer Perceptron

A more complex problem that is not linearly separable can be solved by linking multiple perceptrons, called multi-layer perceptrons. A multi-layer perceptron is a neural network with two or more layers with greater processing power to solve complex problems.

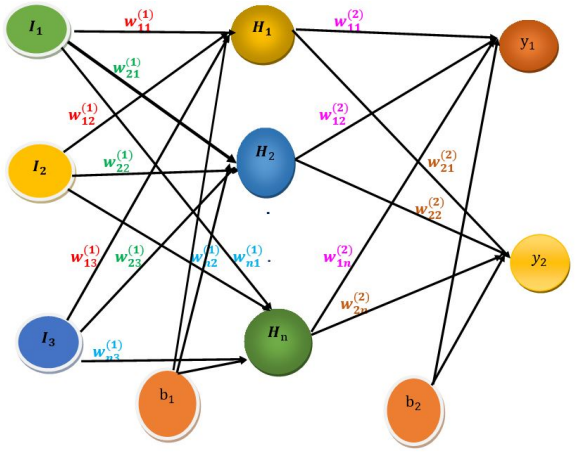


Figure 1.3: Multi-layer Perceptron

The multi-layer perceptron in Figure 1.3 produces the following output:

$$H_1 = \sigma(I_1 w_{11}^{(1)} + I_2 w_{12}^{(1)} + I_3 w_{13}^{(1)} + I_4 w_{14}^{(1)} + \ldots + I_n w_{1n}^{(1)} + b_1)$$

$$H_2 = \sigma(I_1 w_{21}^{(1)} + I_2 w_{22}^{(1)} + I_3 w_{23}^{(1)} + I_4 w_{24}^{(1)} + \ldots + I_n w_{2n}^{(1)} + b_1)$$

$$y_1 = \sigma(H_1 * w_{11}^{(2)} + H_2 * w_{12}^{(2)} + b_2)$$

On the other hand, the matrix form of multi-layer perceptron is given by

$$I = \begin{pmatrix} I_1 \\ I_2 \\ I_3 \\ . \\ . \\ , \\ I_n \end{pmatrix}, \quad W^{(k)} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & ... & w_{1n} \\ w_{21} & w_{22} & w_{23} & ... & w_{2n} \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ w_{n1} & w_{n2} & w_{n3} & ... & w_{nn} \end{pmatrix}, \quad H = \begin{pmatrix} H_1 \\ H_2 \\ H_3 \\ . \\ . \\ , \\ H_n \end{pmatrix}$$

where $k = 1, 2, 3, ..., n$

$$B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ . \\ . \\ , \\ b_n \end{pmatrix}, \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ , \\ y_n \end{pmatrix}$$

Finally the output of multi-layer perceptron is given by

$$Y = \sigma\left(\sum_{n=1}^{K} I_n W_{n,n}^{(K)} + B_n \right)$$

where Y is called the actual output or the predicted function.

- I is the input dataset or information received.

- $W_{n,n}^{(k)}$ is the weights in NN. Weight is a parameter in neural networks, and they are learnable. They are also applied within the hidden layer to transform the input data. Before learning begins in NN, the weights are randomized. The weight is then adjusted during learning to get the correct output. The impact of weight value on the input will always affect

the output. A large weight value will have more impact on the output, while a low weight value will have no impact on the input, resulting in higher errors.

- B is the bias in NN. It is also a parameter that can be learned. The bias is added to the input data product and weight to get the hidden layers. Bias is also adjusted during the NN train to get the desired output. It makes up the difference between the actual and target output and represents how far predictions are from their values.

- H is the hidden layer. The hidden layers can be called a linear equation because they compare the weight action on the input data with the addition of the bias. The computation of the NN is done in the hidden layers, and from the hidden layer, outputs are generated. The outputs are always functions. Therefore, before the outputs are generated, an activation function acts on the linear equation at the hidden layer that transforms the linear to the non-linear.

- σ is the activation function.

1.2.4 Activation Function

The activation function of a perceptron defines the output of the neurons given a set of inputs. It is helpful in the output layer since the output of a problem is always a function. The activation function is also needed in the hidden layers. If the activation function is removed from the hidden layers, the output becomes a simple linear equation; for this reason, the hidden layer is not needed. This shows that there cannot be hidden layers without an activation function. In a word, a NN is just a linear regression model without the activation function. The main role of the activation function is to introduce non-linearity in the NN and make it perform and learn complex tasks. Non-linearity is introduced because, in real life, we know that complex problems cannot be solved by simple linear equations or regression. There are many activation functions. We have three major activation functions in NN. They are

- **Sigmoid or Logistic function**: A sigmoid function is an activation function in which the domain is from $-\infty$ to ∞ and ranges is from 0 to 1. The output value of a NN using a

sigmoid function as an activation function is between 0 and 1. It is used in the hidden layer and output layer for classification.

$$\sigma(H) = \frac{1}{1+e^{-H}}$$

$$\sigma'(H) = \frac{e^{-H}}{(1+e^{-H})^2} = \sigma(H)(1-\sigma(H))$$

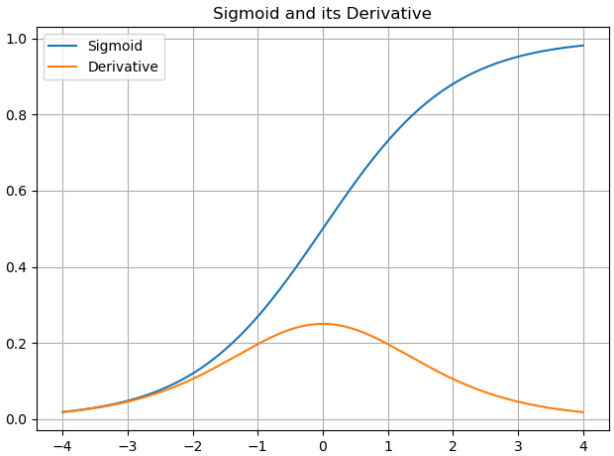


Figure 1.4: The Sigmoid and it's Derivative Function Graph

- **Hyperbolic Tangent Function**($tanh(H)$): A hyperbolic tangent function is an activation function in which the domain is from $-\infty$ to ∞ and ranges is from -1 to 1. The output value of a NN using a hyperbolic tangent function as an activation function is between -1 and 1. It is used in the hidden layer and output layer for classification.

$$\sigma(H) = \frac{e^{H}-e^{-H}}{e^{H}+e^{-H}} = tanh(H)$$

$$\sigma'(H) = 1-\sigma(H)^2$$

- **The Rectified Linear Unit(ReLU)**: The rectified linear unit function is an activation function in which the domain is from $-\infty$ to ∞ and ranges is $[0,\infty)$. The output value of a NN

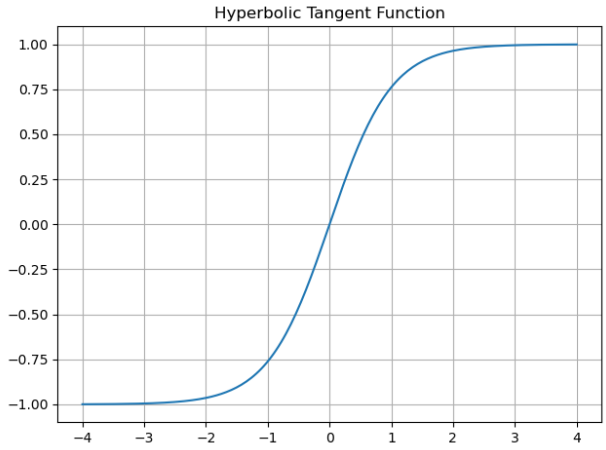


Figure 1.5: The Hyperbolic Tangent Function Graph

using a rectified linear unit function as an activation function is the $max\{0,H\}$. It is used in the hidden and output layers for regression (only positive output).

$$ReLU(H) = max\{0,H\}$$

$$ReLU'(H) = \begin{cases} 1 & if \quad H \geq 0 \\ 0 & if \quad H < 0 \end{cases}$$

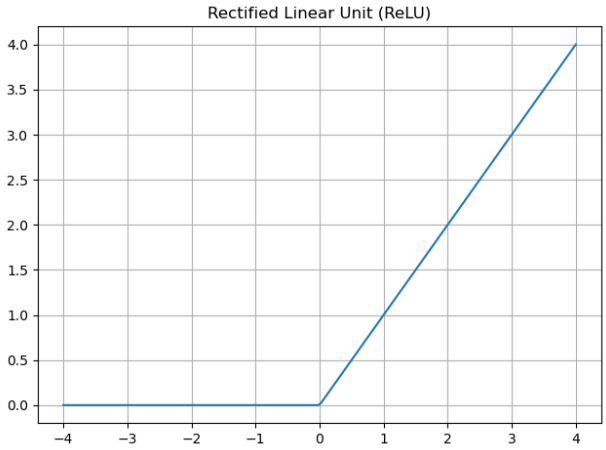


Figure 1.6: The Rectified Linear Unit Graph

- **The Gaussian Error Linear Unit (GELU)**: The Gaussian Error Linear Unit function is an activation function in which the domain is from $-\infty$ to ∞ and ranges are $(-\infty, \infty)$. It's a smooth, non-linear activation function that weights inputs by their magnitude rather than gating them in the binary fashion of ReLU.

$$GELU(H) = H\Phi(H)$$

where $\Phi(H)$ is the cumulative distribution function of the standard Gaussian.

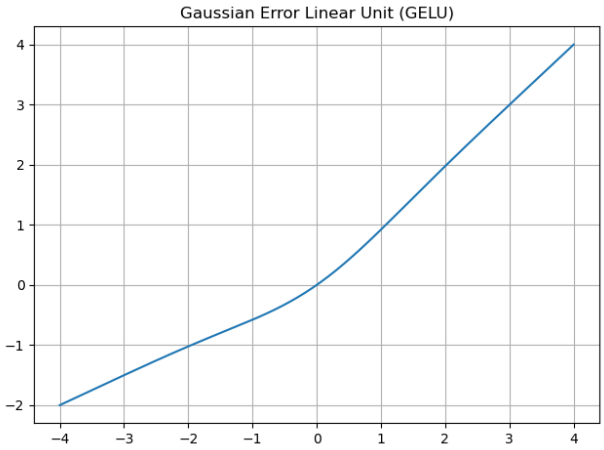


Figure 1.7: The Gaussian Error Linear Unit Graph

1.3 Artificial Neural Networks

Artificial neural networks, commonly referred to as neural networks (NNs), are computational models inspired by the biological neural networks present within the human brain [67]. These systems can learn to carry out tasks by processing a range of examples, typically without explicit, task-specific programming. Artificial neural networks (ANNs) have gained popularity because of their potential to overcome common problems in the field, such as the necessity of massive training datasets and high processing costs. It has been proposed that ANNs can be used to pioneering effect in estimating parameters in systems of differential equations. The advent of ANNs traces back to the 1940s, primarily attributed to the research article by McCulloch and Pitts [67]. However, it

is only in recent years that ANNs have seen widespread acceptance and use, owing to significant advancements in computational capabilities and data storage.

ANNs essentially with multiple hidden layers, have seen a surge in application across different sectors, including computer vision, image processing [40], pattern recognition [46], and cybersecurity [44]. The architecture of ANNs allows them to capture more variance, contributing to their success. The effectiveness of compartmental models, which are often described by systems of ordinary differential equations, and the effectiveness of the models in understanding and predicting the behaviors of dynamical systems have been demonstrated. However, they are not without their weaknesses. Researchers are increasingly gravitating toward combining compartmental and models to improve their abilities to analyze and predict the behavior of dynamical systems. While there are numerous types of artificial neural networks, this discussion primarily focuses on feedforward neural networks (FNN), Backpropagation, deep neural networks (DNNs), and physics-informed neural networks (PINNs).

1.3.1 Feedforward neural networks

The feedforward neural network (FNN) is an artificial neural network with a simplistic architecture and uncomplicated node connections that are not cyclic. FNN is used to estimate the target function by recursively applying a number of activation functions to the input and then producing a value or vector that strongly matches the target values. Input, hidden, and output layers make up the network. The input layer receives data, applies some neurons, and then propagates the results to the hidden layers. Input neurons make up the input layer, introducing training data into the network for later processing by hidden layers. Between the input layer and output layer are hidden layers where some linear or non-linear activation functions are applied to transform the data.

The primary objective of Rosenblatt's perceptron model, as outlined in [15], is to refine the weights of the perceptron in such a way that the output it generates closely approximates the desired target output. The difference between the actual output the perceptron produces and the intended target output can be significant at first, especially before the learning process. To illustrate

this, consider a scenario where we iterate through M learning examples, each of which is the jth instance in a set of input data $(x_1, x_2, ..., x_n)^j$. If the perceptron has not been learned, the error would be that actual output y_j differs from target output d_j

$$e_j = d_j - y_j$$

The essence of the neural network perceptron lies in its ability to reduce this error, thereby aligning the actual output more closely with the target output. Ideally, this error diminishes to a value approaching zero. Achieving this necessitates the adjustment or updating of the perceptron's weights. The e_j is the error, and it is used to update the value of the weight. The procedure for updating or improving the weight values is as follows: Given that

$$\frac{\partial y}{\partial x} = \frac{f(x+h) - f(x)}{h} \tag{1.1}$$

Simplifying (1.1):

$$f(x+h) = f(x) + h\frac{\partial y}{\partial x} \tag{1.2}$$

From (1.2), let $x = w_n(weight)$, $f(x) = w_{n,j}$, $h = \beta$, $y = E$ *and* $f(x+h) = w_{n+1}$

Now,

$$w_{n,j+1} = w_{n,j} + \beta\frac{\partial E_j}{\partial w_{n,j}} \tag{1.3}$$

$w_{n,j+1}$ is called updated weight; $w_{n,j}$ is called old weight; β is called the learning rate and $\beta > 0$; E_j is the total loss function or mean squared error and $\frac{\partial E_j}{\partial w_{n,j}}$ is called the gradient descent or the stochastic gradient descent.

- **learning rate(β)**: The learning rate helps in training the neural network and shows how fast

the perceptron converges. It is the step taken to reach the global minima.

- **Total loss function or cost function**: The total loss function (mean square error) is the sum of the distance between the dependent data, called the target output, and the predicted function, called the actual output. The total loss function determines how well the initial line fits the data.

$$E_j = \frac{1}{p} \sum_{n=1}^{j} (\Delta_n)^2$$

$$E_j = \frac{1}{p} \sum_{n=1}^{j} (d_n - Y)^2$$

$$Y = \sigma\left(\sum_{n=1}^{j} I_{n,n} W_{n,n}^{(j)} + B_n \right)$$

  Where d_n is the dependent data called the target output, Y is the predicted function called the actual output, and p is the number of input data points or length of the data point. Although there are different cost functions, the most popular is the mean squared error.

- **Gradient Descent(GD)**: This is the process of finding the best-fit line for a given training data set. Suppose an M dataset is provided, and consider all the data points in the dataset at once. The $\frac{\partial E_j}{\partial w_{n,j}}$ will be seen as gradient descent.

$$GD = \sum_{n=1}^{j} (d_n - y_n)^2$$

- **Stochastic Gradient Descent (SDE)**: Stochastic gradient descent is an optimization procedure for minimizing the loss or cost function of a predictive model with regard to a training dataset. Suppose an M dataset is provided, and consider only one data point at a time in the dataset. The $\frac{\partial E_j}{\partial w_{n,j}}$ will be seen as stochastic gradient descent

$$SGD = (d_n - y_n)^2$$

The graph in Figure 1.8 below shows the cost function on the y-axis and weight on the x-axis, and inside the graph, we have a curve that shows the initial weight and the global minima. The primary purpose of updating the weight is to make a function to take the minimum value at a point called the global minima. However, there is a local minimum where the function may have a minimum value at different points, which may appear minima but may not be the minimum value. Global minima and local minima are the differences between gradient descent and stochastic gradient descent

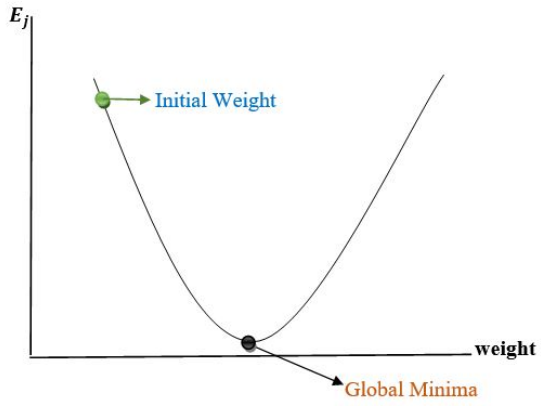


Figure 1.8: Graph showing Initial weight and Global minima

During the model's training, the gradient descent can find local minima most of the time because the gradient considers all the data points at once and does not point in the steepest descent direction. At the same time, the stochastic gradient descent can find the global minima because the gradient considers only one data point at a time and also optimizes the procedure that will minimize the total loss function. To consider one point at a time in a dataset set will require a high iteration count called an epoch. The larger the number of epochs, the better the optimization procedure for minimizing the total loss function, and the closer the function will be to the minimum value point.

From equation (1.3) $w_{n,j+1} = w_{n,j} + \beta \frac{\partial E_j}{\partial w_{n,j}}$ is called the updated weight equation. The question is how do we get $w_{n,j+1}$, which is the updated weight? To get the updated weight, we need to

get $\frac{\partial E_j}{\partial w_{n,j}}$, which is the GD or SGD. The cost function

$$E_j = \frac{1}{p}\sum_{n=1}^{j}(d_n - Y)^2$$

$$E_j = \frac{1}{p}\sum_{n=1}^{j}\left(d_n - \sigma\left(\sum_{n=1}^{j} I_n w_{n,n}^{(j)} + B_n\right)\right)^2$$

$$\frac{\partial E_j}{\partial w_{n,j}} = \frac{2}{p}\sum_{n=1}^{j} -I_{n,n}\left(d_n - \sigma\left(\sum_{n=1}^{j} I_n w_{n,n}^{(j)} + B_n\right)\right)$$

$$\frac{\partial E_j}{\partial B_n} = \frac{2}{p}\sum_{n=1}^{j} -\left(d_n - \sigma\left(\sum_{n=1}^{j} I_n w_{n,n}^{(j)} + B_n\right)\right)$$

Lets consider kink over M learning example with input data $(I_1, I_2)^j$ in which $I_{j,3} = 1$, where bias is zero and has a target output d_j. The cost function will be

$$E_j = \frac{1}{p}\sum_{n=1}^{j}\left(d_n - \sigma\left(\sum_{n=1}^{3} I_{n,j} w_{n,n}^{(j)}\right)\right)^2$$

Taking Sigmoid function as the activation function

$$\sigma(H) = \frac{1}{1+e^{-H}}$$

Where the domain of the sigmoid function is $(-\infty, \infty)$, and range is $(0,1)$.

$$\sigma'(H) = \frac{e^{-H}}{(1+e^{-H})^2} = (1 - \sigma(H))\sigma(H)$$

To update the value of weight in this example, we need to find GD = $\frac{dE_j}{dw_{n,n}}$

Let

$$T_j = d_j - y_j$$

Then,

$$E_j = \frac{1}{2} {T_j}^2$$

$$\frac{\partial E_j}{\partial T_j} = T_j$$

$$y_j = \sigma \left(\sum_{n=1}^{3} w_{n,n} I_{n,j} \right)$$

Therefore,

$$T_j = d_j - \sigma \left(\sum_{n=1}^{3} w_{n,n} I_{n,j} \right)$$

$$\frac{\partial T_j}{\partial w_{(n,n)}} = \sigma' \left(\sum_{n=1}^{3} w_{n,n} I_{n,j} \right) . \ -I_{j,n}$$

$$\frac{\partial E_j}{\partial w_{n,n}} = T_j . \ \sigma' \left(\sum_{n=1}^{3} w_{n,n} I_{n,j} \right) . \ -I_{j,n}$$

$$\frac{\partial E_j}{\partial w_{n,n}} = \left(d_j - y_j\right) . \sigma' \left(\sum_{n=1}^{3} w_{n,n} I_{j,n} \right) . \ -I_{n,j}$$

$$\frac{\partial E_j}{\partial w_{n,n}} = \ -I_{n,j}\ e_j \sigma' \left(\sum_{n=1}^{3} w_{n,n} I_{n,j} \right)$$

Let

$$\delta_j = e_j \sigma' \left(\sum_{n=1}^{3} w_{n,n} I_{n,j} \right)$$

$$\frac{\partial E_j}{\partial w_{n,n}} = \ -I_{j,n} \delta_j$$

Equ (1.3) becomes

$$w_{n,j+1} = w_{n,j} - \beta I_{n,j} \delta_j \tag{1.4}$$

Equation (1.4) is called the updated weight equation.

The primary objective in this context is to navigate towards the global minimum starting from the initial weight values. This journey involves taking incremental steps. If these steps are of a fixed size, as illustrated in Figure 1.8, there is a risk of bypassing the global minimum. Such a scenario could lead to the failure of the gradient descent algorithm to converge to the optimal solution. As Figure 1.9 demonstrates, a step that overshoots or misses the global minimum can result in a continual upward trajectory. This means that the algorithm keeps moving away from the global minimum, potentially leading to an inconclusive search for the optimal solution. Thus, careful consideration of step size and approach is crucial to ensure that the gradient descent method

effectively identifies the global minimum.

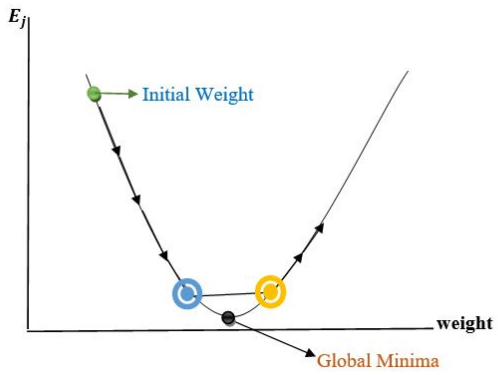


Figure 1.9: Graph showing the Global minima missed

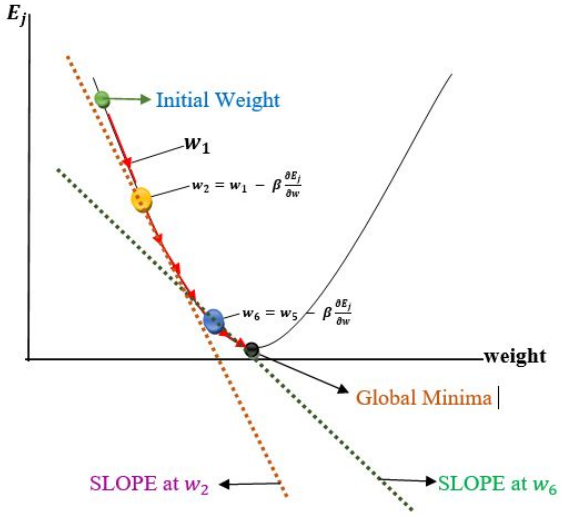


Figure 1.10: Graph showing the global minima reached

To address this issue, it is necessary to decrease the step size. This reduction can be achieved by computing the gradient or slope at each point, as depicted in Figure 1.10. The learning rate, a key parameter in this process, works in conjunction with the calculated slope to determine the size of each step taken during the optimization process. The goal is to reach the global minimum, a point where the solution is optimized. Attaining the global minimum allows us to accurately determine the precise weight values needed to solve the problem effectively. This approach ensures that the algorithm converges on the most efficient and accurate solution. The weighted sum of the input

data through the activation function is generated in the first hidden layer and propagated through the other hidden layers. Each output from each hidden layer is given a bias term. In order to obtain the desired result, the output layer processes the net output from the final hidden layer. If the task is a classification task, the output layer will provide discrete outcomes; however, if the goal is a regression task, the output layer will produce a continuous-valued outcome to produce the desired output [5].

Hyperparameter tuning, which includes figuring out how many layers, neurons per layer, learning rate, activation function, and a loss function optimizer to use, is essential in training a neural network. Approximate solutions to differential equations have been learned using FNN. By restricting the residual, FNN was utilized in [47] to build differential equation solvers and parameter estimation. The mathematical formula that changes the data from one layer to the next is defined as [13]

$$y_n^{k+1} = \sum_{m}^{z_k} W_{m,n}^k \, \sigma_{k-1}(y_m^k) + b^k$$

Where y_m^k, $m = 1,2,,z_k$ denotes the output of the m^{th} node in $(k-1)^{th}$ layer, σ represents the activation function, z_k represents the number of neurons in the kth layers, b^k are the bias in the k^{th} layers and $W_{m,n}^k$ are the weights between the nodes m^{th} and n^{th}. In this study, the hyperbolic tangent function was used as the activation function.

$$\sigma(H) = \frac{e^H - e^{-H}}{e^H + e^{-H}}$$

The output layer's output values are used to generate a loss function, which is used to estimate the model's error. Optimizers like Adam or gradient descent [5] are used to minimize the loss function so that the result is near what we observed. Optimizer and loss function choices vary depending on the problem at hand.

1.3.2 Backpropagation.

It is important to minimize the total loss function with respect to the parameter bias and weight to get accurate results. In the process of minimizing the loss function, gradient descent was used to train a neural network to find all the weights and biases of the network. However, one of the key problems with gradient descent is that for each iteration, one needs the derivative of the loss function with respect to each one of the network parameters, which is very expensive. One of the major innovations in neural networks was the creation of an algorithm called backpropagation. Backpropagation dramatically reduces gradient descent's complexity and makes it more efficient and tractable. The analysis below will show how backpropagation works.

Let's consider two inputs, I_1 and I_2, from Figure 1.2 and hidden layers H_1 and H_2 and y_1 as the actual output of the network. To compute the backpropagation, we need to compute the gradient descent of $w^2_{1,1}$

Using Chain rule:

$$\frac{\partial E_j}{\partial w^{(2)}_{1,1}} = \frac{\partial e_1}{\partial y_1}\frac{\partial y_1}{\partial y^*_1}\frac{\partial y^*_1}{\partial w^{(2)}_{1,1}}$$

Where

$$y^*_1 = \sigma(H_1)w^{(2)}_{1,1} + \sigma(H_2)w^{(2)}_{1,2} + b_2$$

$$y_1 = \sigma(y^*_1)$$

Then

$$e_1 = \| d_1 - y_1 \|^2$$

$$\frac{\partial E_j}{\partial w^{(2)}_{1,1}} = -2(d_1 - y_1)\sigma(y^*_1)(1 - \sigma(y^*_1))\sigma(H_1)$$

Thus, for any weight or bias in the network,

$$\frac{\partial E_j}{\partial w^{(l)}_{i,j}} = \delta^{(l)}_{j,i} T^{(l-1)}_j$$

$$\frac{\partial E_j}{\partial b_i^{(l)}} = \delta_i^{(l)}$$

$w_{i,j}^{(l)}$ is the weight from jth neuron in layer, l 1 to ith neuron in layer l, $\delta_i^{(l)}$ is the δ of ith in layer l, $b_i^{(l)}$ is the bias of ith neuron in layer l and $T_j^{(l-1)}$ is the activation of jth neuron in layer $l-1$.

The key question we face is determining how to calculate the error gradient, denoted as δ, for the preceding layers in a neural network. To ascertain the δ value for a previous layer, we utilize the already computed δ from the current layer. This calculation involves the activation function output from the previous layer, the known weights of the current layer, and the application of the chain rule in calculus. In essence, the δ for the previous layer is derived by integrating these elements. This process ensures that the error signal is accurately back-propagated through the network, allowing for effective adjustment of weights in each layer to minimize the overall error. This methodology is fundamental in the training of neural networks, ensuring that each layer's parameters are optimally tuned in relation to the observed error.

$$\delta_i^{(l)} = \sum_j \delta_j^{(l+1)} \, w_{i,j}^{l+1} T_i^{(l)} (1 - T_i^l)$$

$\delta_j^{(l+1)}$ is the δ of jth in layer $l+1$. Finally, backpropagation works this way:

- Initialize neural networks with random weights and bias.
- Compute the gradient with respect to each bias and weight and find the average of the gradient over the training set.

$$\frac{\partial E_j}{\partial w_{i,j}^{(l)}} = \frac{1}{n} \sum \frac{\partial E_j}{\partial w_{i,j}^{(l)}}$$

$$\frac{\partial E_j}{\partial b_i^{(l)}} = \frac{1}{n} \sum \frac{\partial E_j}{\partial b_i^{(l)}}$$

- Compute δ for all neurons in the current and previous layers

- Update the weight and bias using gradient descent.

$$w_{i,j+1}^{(l)} = w_{i,j}^{(l)} - \beta \frac{\partial E_j}{\partial w_{i,j}^{(l)}}$$

$$b_{i+1}^{(l)} = b_i^{(l)} - \beta \frac{\partial E_j}{\partial b_i^{(l)}}$$

1.3.3 Deep neural networks

Deep neural networks (DNN) are a type of artificial neural network with multiple layers between the input and output layers. These layers, known as hidden layers, allow DNNs to learn high-level features from the input data. DNNs are a cornerstone of deep learning and are widely used for tasks such as image recognition, speech recognition, natural language processing, and many other applications. A deep neural network is a framework that mimics how the brain works. Suppose you want to use a deep neural network to solve any problem. In that case, the first step is to decide which of the many different neural network architectures will perform best for your particular problem domain. Since deep learning and deep neural networks [71] are known to be universal approximators of continuous functions, these techniques have found applications in function approximation tasks. Consider a deep neural network with similar architecture in [56].

$$\theta_n^l = \sigma^l \left(\sum_{k=1}^{l} a_k^l W_k^l + b_k^l \right)$$

where θ_n^l denotes the predicted solution, a_k^l are the input, σ^l is the activation function, W_k^l and b_k^l are the weight, and the bias, respectively. Using the DNN approach to solve and identify the parameters of a dynamical system, we find the best network parameters ω representing the biases and weights of the network that minimize the loss function. In a DNN, the loss function is typically a measure of the discrepancy between the predictions of the network and the true output values from the training data.

$$L(\omega;\mu) = \frac{1}{n}\left(\sum_{i=1}^{\mu} ||\theta(t_n;\omega) - \theta(t_n)||^2\right)$$

and

$$\theta(t_n;\omega) = \frac{d\theta(t_n;\omega)}{dt} - f\left(t_n, \theta(t_n;\omega), \delta(t_n;\omega)\right)$$

where $\theta(t_n)$ are the output values, $\theta(t_n;\omega)$ are the predicted output values, f represents some form of model dynamics or external influences on the system described by the differential equation and n is the number of instances.

1.3.4 Physics-Informed Neural Network

Physics-informed neural networks represent an advanced deep-learning technique that integrates knowledge from physics principles to provide a robust approximation of function and parameter identification. PINNs have emerged as a powerful tool among data-driven deep neural networks due to their versatility in solving inverse and forward problems. Physics-informed neural networks are universal function approximators that can incorporate any physical principles that govern a given dataset into the learning process and can be characterized by partial differential equations. The Physics-Informed Neural Network (PINN), which was first proposed [47], is one of the most effective data-driven deep neural networks in recent years. It was developed to learn the differential system's parameters from data (inverse problem) or use it as a differential system solver (forward problem). The Physics-Informed Neural Network (PINN) is a data-driven algorithm for approximating differential equation solutions and identifying parameters. It could use any neural network architecture, such as the FNN, as the main framework. The activation functions and optimization methods used in PINN are the same as those used in conventional deep learning techniques. However, the loss function, composed of initial values, boundary conditions, and physical constraints, is the most intriguing aspect of this algorithm. The neural network's outputs are constrained to satisfy the system of differential equations by penalizing differential equation residuals in the loss function. Let us consider a nonlinear differential equation of the general form

$$\frac{dv(t)}{dt} + \Phi(v(t,\eta)) = 0, \quad t \in [0,T] \tag{1.5}$$

where $v(t)$ denote the solution, $\Phi[.;\eta]$ is a non-linear differential operator and ϕ is a subset on the domain of $\mathbb{R}$ and the model parameters η is fixed. Therefore, the loss is minimized to obtain weights and biases for the neural network. This is called forward PINN. The loss function of PINN is

$$Loss = \lambda_{data} + \lambda_{res}, \tag{1.6}$$

λ_{data} represents the contribution loss value from the predicted and actual data. λ_{res} gives the loss from the underlying physics of the model.

1.3.5 Transfer learning

Transfer learning (TL) is a machine learning technique where a model developed for a particular task is reused as the starting point for a model on a second task. It's a popular approach in deep learning because it can train deep neural networks with comparatively little data. Transfer learning techniques aim to efficiently achieve training convergence by transferring information between the appropriate base and target tasks. This is useful since gathering large datasets is only sometimes feasible [76, 55, 36]. It allows for less data, reduces training time, can leverage state-of-the-art pre-trained models, and is effective in domains where data is scarce or expensive to collect. However, it requires a relevant pre-trained model, the new task must be somewhat similar to the original task, and there is a risk of negative transfer if the tasks are too different.

Basic Concept

- **Pre-training:** In transfer learning, you start with a pre-trained model. This model has been trained on a large, generic dataset and has learned a wide range of features that can be useful for various tasks. Common examples are models trained on ImageNet for image tasks or

BERT for natural language processing tasks.

- **Fine-tuning:** The second step in transfer learning is fine-tuning the pre-trained model for a specific task. This involves training the model on a dataset from the new task. This fine-tuning can be a short process depending on the size and similarity of the new dataset to the original dataset.

- **Representation Learning:** In deep learning, transfer learning leverages learned features (representations) from an earlier model. Mathematically, this can be seen as learning a function $f(x)$ from a pre-trained model, where x is the input and $f(x)$ is the feature representation.

- **Objective Function Adaptation:** In fine-tuning, we typically adapt the last few layers of the model to suit the new task better. This involves changing the objective function. If the original model was trained to minimize a function $\lambda(y, h(f(x)))$, where y is the true label and $h(f(x))$ is the model output, the new model might be trained to minimize $\lambda_{new}(y_{new}, h_{new}(f(x))$, where y_{new} and h_{new} are the labels and model output for the new task.

- **Optimization:** During fine-tuning, the optimization algorithm (e.g., SGD, Adam) will update the weights of the model, typically with a lower learning rate, to prevent overwriting the learned features too quickly.

1.4 Data-Driven Simulations

Data-driven simulations represent a modern approach in computational science, where the focus is on using extensive datasets to inform and drive the simulation process. Traditional simulations frequently rely on predefined models of physical processes or systems, and these models are typically subject to differential equations. The solutions to these equations are approximated using various numerical methods, providing insights into the behavior of the system under study. However, data-driven simulations shift the paradigm by prioritizing data as the primary source of information. In this approach, the discovery of model parameters and the model's resolution are entirely dependent

on the data collected. This method leverages large volumes of data, often gathered from real-world observations or experiments, to directly influence and shape the simulation.

The advantage of data-driven simulations lies in their ability to adapt and evolve based on incoming data, making them particularly suited for complex systems where traditional modeling may be limited or infeasible. These simulations can uncover hidden patterns and relationships within the data, which can lead to more accurate and reliable models. Furthermore, data-driven simulations are integral in fields like machine learning [10] and artificial intelligence, where they contribute to the development of predictive models and decision-making algorithms. In areas such as climate modeling, biomedical engineering, and financial forecasting, data-driven simulations offer a robust tool for analyzing vast datasets, enabling researchers to simulate scenarios and predict outcomes with higher precision. Oluwasakin et al. used data-driven simulations to model the parameters and dynamics of the COVID-19 model, completely relying on available data [13, 25].

1.4.1 Gathering Data

This phase involves extensive data collection about the system or process of interest. Data sources are diverse and can include controlled experiments, real-world observations, or previously existing databases. This stage may also involve preprocessing steps to clean and organize the data, ensuring its quality and relevance for the analysis. Raw data often contains noise and may be incomplete, necessitating a process known as data preprocessing to transform it into a more useful format for analysis and inference [39]. This step is crucial for enhancing the results of data-driven simulations. Data preprocessing is applicable to both structured and unstructured data types. Techniques used in data preprocessing include scaling, standardization, and normalization, each playing a vital role in refining the data for more effective analysis. In addition to cleaning the data, proper preprocessing makes sure it is in a format that simulation algorithms can process and analyze quickly.

1.4.2 Learning from Data and Making Predictions

Specialized algorithms, often from the fields of machine learning and deep learning, are employed to analyze the data. These algorithms are adaptive; they don't rely on already established models

but aim to uncover patterns or relationships directly from the dataset. This stage is crucial for understanding complex systems where predefined models might be inadequate or nonexistent. After the learning phase, these algorithms are capable of making predictions or simulating scenarios. For instance, in meteorology, after analyzing past weather data, the simulation might predict future weather patterns. This step is pivotal for forecasting and decision-making in various fields, from climate science to financial markets. Advanced algorithms, like Regularized Physics-Informed Neural Networks (RPINN), are used in engineering and healthcare to create models that find the best answers to difficult problems [47]. RPINN, in particular, is effective in discovering compliance rates and identifying critical dynamics in systems of parameters. Additionally, variations of recurrent neural networks like long-short-term memory (LSTM) and attention-based LSTM (A-LSTM) are employed to predict future dynamics in behavior and epidemiological parameters [34, 2, 5], as well as in modeling the dynamics of epidemic models. These advanced tools play a significant role in enhancing the precision and effectiveness of simulations in these fields.

1.4.3 Adapting and Improving

As new data is acquired, continuously updating and refining models in data-driven simulations is crucial for maintaining relevance and accuracy. This iterative process allows simulations to adapt to new information and improve over time, which is particularly vital in dynamic fields where conditions constantly change. This adaptability enables simulations to respond to the latest trends and anomalies in the data, ensuring that their insights and predictions are up-to-date. Additionally, it allows for the refinement of models to account for new variables or changing relationships within the data, enhancing the precision and reliability of the simulations.

1.4.4 Error Metrics

This subsection discusses the error metrics used for data-driven simulations, examining when and why specific metrics should be employed. These error metrics provide valuable insights into the accuracy and performance of data-driven models. By evaluating the models based on these metrics, researchers can determine the quality of their predictions and compare the effectiveness of

different approaches. In the context of learning the time-varying parameters for dynamical systems, it is important to consider the specific characteristics of the problem and the goals of the analysis when selecting the appropriate error metrics. For instance, if θ symbolizes the actual data and $\hat{\theta}$ represents the data predicted by the model, these error metrics are applied in our data-driven simulations:

- **Explained Variance (EV):** The explained variance is equivalent to the coefficient of determination R^2 used for linear regression. Therefore, the algorithm correctly predicted the targets when the EV came close to 1. The optimum metric for nonlinear regression is the explained variance, which is the variation in the predicted θ that the neural network algorithm has been able to explain. Therefore, for nonlinear regression, the EV is ideal. The formula for explained variance is given by:

$$EV = 1 - \frac{Var(\theta - \hat{\theta})}{Var(\theta)}$$

- **Mean Absolute Percentage Error (MAPE):** The key benefit of utilizing MAPE is that it is useful for quantile regression and is the relative error in the mean absolute error. It is a weighted mean absolute error. The formula for the mean absolute percentage error is given by:

$$MAPE = \frac{100}{N} \sum_{j=1}^{N} \left| \frac{\theta_j - \hat{\theta}_j}{\theta} \right| \%$$

- **Mean squared error (MSE):** mean squared error (MSE) measures the amount of error in the models. It assesses the average squared difference between the observed and predicted values.

$$MSE = \frac{1}{N} \sum_{j=1}^{N} (\theta_j - \hat{\theta})^2$$

- **Root Mean Squared Error (RMSE):** When testing model accuracy, the RMSE, a scale-dependent quantity, is used to compare prediction errors from various models or model con-

figurations for a specific variable. That is, the RMSE is in the same units as the predicted variable, providing a straightforward interpretation of the model's prediction error magnitude. The root mean squared error (RMSE) is obtained by taking the square root of the mean squared error (MSE). The formula for root mean squared error is given by:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(\theta_j - \hat{\theta})^2}$$

- **Mean absolute error (MAE):** A significant advantage of using the mean absolute error (MAE) is its relevance to quantile regression. MAE represents an average of absolute differences between predicted and actual values. The formula for mean absolute error is given by:

$$MAE = \frac{1}{N}\sum_{j=1}^{N}\left|\theta_j - \hat{\theta}_j\right|$$

- **Mean Absolute Percentage Error (MAPE)**: This is especially useful when comparing performance across different data scales since this metric evaluates prediction accuracy as a percent of the true value itself, i.e.

$$MAPE = \frac{100}{N}\sum_{j=1}^{N}\left|\frac{\theta_j - \hat{\theta}_j}{\theta}\right|\%$$

- **Coefficient of determination (R^2):** In trend analysis, researchers rely significantly on the coefficient of determination, or R-squared, which measures the linear relationship between two variables. the algorithm correctly predicted the targets when the R^2 comes close to 1. The formula for the coefficient of determination is given by:

$$R^2 = 1 - \frac{RSS}{TSS}$$

where RSS represents the residual sum of squares and TSS represents the total sum of squares.

CHAPTER 2

Learning the Time-varying Parameters of Transmission Rate

2.1 Introduction

The Chinese city of Wuhan was the site of the first incidence of COVID-19 [65]. However, on a global scale, the disease broke out and spread fast, creating one of human history's most deadly pandemics [53]. COVID-19 is always a threat to human health because it spreads quickly, has terrible effects on health, and changes its genetic makeup. In addition, human-to-human transmission through the air may be facilitated by the virus [72]. As a result, on the 31st of January 2020, the WHO proclaimed a global health emergency. Despite careful supervision, the disease has spread to over a hundred nations worldwide since its discovery and has become a global pandemic [31]. As of 2021, numerous dominant mutant variants of COVID-19 have emerged. These numerous cases of infection with mutating variants have been recorded in most countries. Many of these cases originated in the United Kingdom and South Africa [12]. In addition, a new COVID-19 variant called Omicron surfaced towards the end of 2021, and it was first detected in South Africa in November 2021 and later spread to Europe in the same month. Numerous countries worldwide, including China, Brazil, and India, have experienced outbreaks of the epidemic caused by the Omicron variant. According to reports, the Omicron variant infection can be transferred by anyone who has come into contact with it, regardless of whether or not they have been vaccinated [7]. In addition, the Omicron variant spreads more quickly than other COVID-19 variants. The success of the measures put in place can only be ascertained by looking into COVID-19's dynamic behavior.

A prediction model is required to prevent the disease's spread. Unfortunately, due to limited public information on emerging epidemics and diseases, creating realistic models during public health emergencies will be difficult [28]. Some epidemics can be predicted, while others cannot but are still likely. Predicting turning points and epidemic waves helps create and evaluate re-

sponse plans [21]. Many researchers from the most severe countries and around the world have created COVID-19 trend-predicting methodologies. In order to better understand and control this pandemic, an infectious disease model, estimation methods, and forecasting tools have been created. Mathematical models, machine learning, and deep learning have been used to predict how an epidemic will change over time [17, 9]. Kermack and McKendrick introduced a mathematical model called the Susceptible-Infectious-Recovery (SIR) model [62]. This variational model calculates the estimated number of people infected and later recovered from the disease. Under specific circumstances, the logistic model can be derived from the SIR model. The logistic model is frequently used in fitting regression models to time series data because its underlying theory is straightforward and efficient. Fitting and analyzing epidemic prediction has been done using logistic, Bertalanffy, and Gompertz models [48]. The logistic model outperformed the other two models in terms of prediction accuracy. However, the key drawbacks of these three models are that they are only applicable during certain stages of an outbreak and only if enough data is available. The logistic growth model, the generalized Richards model, and the generalized logistic growth model were all computed to report the number of infected cases in the COVID-19 epidemics [68]. The logistic model provides upper and lower bounds for predictions for the number of infected cases among the three models.

In this chapter, we develop algorithms and a model that will capture various time-dependent factors to enhance the accuracy of predictions regarding individuals infected with the COVID-19 Omicron variant. We present a logistic-informed neural network (LINN), a deep-learning neural network algorithm that is based on using a physics-informed neutral network (PINN) on logistic differential equations [13]. The LINN algorithms are a viable choice to learn the time-varying transmission rate and identify the governments effects on mitigation measures in the data. Some well-known logistical data regarding infectious diseases are added to the LINN loss function. The LINN algorithm was employed to learn the parameters of the four mathematical models to check if these models can improve in predicting the daily and cumulative number of individuals infected with the COVID-19 Omicron variant in a country with partial mitigation measures. We employ

cubic-spline interpolation to create enough training data to detect hidden characteristics in the training data. Given the limitations of the rational and birational models in accurately predicting the number of COVID-19 infections in countries with partial mitigation measures, we introduce the time-series model. This model leverages neural network techniques to capture the dynamic, time-dependent patterns present in the data. This model aims to enhance predictions' accuracy for partial and strict mitigation measures, specifically for the COVID-19 Omicron variant infections. The learned time-varying parameters and the analytical solutions of the mathematical models are used to predict the daily and time that a plateau will be reached, as well as the cumulative number of people who have been reported to be infected with the COVID-19 Omicron variant in a given country.

2.2 Materials and Methods

2.2.1 Mathematical models

In the discipline of epidemiology, modeling practice may be traced back to the early part of the twentieth century. The theoretical foundations of contemporary epidemiology are based on modeling disease propagation and demonstrating that the disease will become extinct if particular circumstances are met. The infectious diseases caused by the COVID-19 Omicron variant, where the disease is transmitted from one host to another, are examined in this section. Kermack and McKendrick are credited with developing some of the first known mathematical models [62]. They evaluated several models that are called SIR models without vital dynamics. These models are based on healthy, infected, and immune individuals living in a stable population (no births or deaths).

Let us assume that we have a constant number of people, N, and that they are separated into the three states: susceptible (S), infected (I), and recovered (R). This simple compartmentalization captures individuals' fundamental stages during an outbreak. The transition from susceptible to infected and then from infected to recovered is based on real-world observations. When a susceptible person comes into contact with an infected person, they may become infected. After some time,

an infected person will either recover or, in more detailed models, may pass away. The SIR model uses ordinary differential equations, which can be solved both analytically (in simpler cases) and numerically (for more detailed or extended models). This allows for both general insight into the behavior of outbreaks and detailed predictions for specific cases. The model takes into account the basic dynamics of disease transmission. The rate at which susceptibles become infected depends on the contact rate with infected individuals, which is captured by the αSI term. Infected individuals don't stay infected indefinitely. They either recover or pass away after some time. The SIR model accounts for this with the recovery rate, Θ. Therefore, the SIR model has the following system of ordinary differential equations.

$$\begin{aligned} \frac{dS(t)}{dt} &= -\frac{\alpha(t)S(t)I(t)}{N} - \Gamma(t)R(t) \\ \frac{dI(t)}{dt} &= \frac{\alpha(t)S(t)I(t)}{N} - \Theta(t)I(t) \\ \frac{dR(t)}{dt} &= \Theta(t)I(t) + \Gamma(t)R(t) \end{aligned} \tag{2.1}$$

$\alpha(t)$ represents the transmission rate at time t, $\Gamma(t)$ represents the rate of loss of immunity at time t, and $\Theta(t)$ represents the recovery rate at time t. Let's consider the simplest of these models, in which an infected individual remains infectious, which captures the transmission rate between susceptible and infected individuals. The model has the following system of ordinary differential equations.

$$\begin{aligned} \frac{dS(t)}{dt} &= -\frac{\alpha(t)S(t)I(t)}{N} \\ \frac{dI(t)}{dt} &= \frac{\alpha(t)S(t)I(t)}{N} \end{aligned} \tag{2.2}$$

Since $N = S(t) + I(t)$, the above model is reduced to a simple logistic differential equation model.

$$\frac{dI(t)}{dt} = \alpha(t)I(t)\left(1 - \frac{I(t)}{N_p}\right) \tag{2.3}$$

Therefore, the ordinary differential equation (2.3) satisfies the daily or cumulative number of individuals reported to be infected by Omicron at time t. Where $I(t)$ represents the daily and the

cumulative number of individuals reported to be infected by Omicron at time t, N_p is the constant parameter, and $\alpha(t)$ is the time-dependent function. The constant and the function $\alpha(t)$ depend on the fundamental properties of the particular virus and the daily effect of the various steps taken by the country to limit the spread of the virus [17].

2.2.2 Learning Time-Varying Transmission Rate

The fact that $\alpha(t)$ is time-dependent indicates various time-dependent elements, including integrating the effect of public health efforts and the public's response to the actions [73] and the effect of various government policies to prevent the virus from spreading further also depends on t [17]. Early in COVID-19 transmission, the key public health activity was lockdown, followed by early detection of infectives, social distancing, contact tracing, masking, etc. Suppose $\alpha(t)$ is replaced by a constant k, then the analytical solution to (2.3) becomes

$$I(t) = \frac{N_p}{1 + \gamma e^{-kt}} \tag{2.4}$$

I represents the daily and the cumulative number of Omicron infected cases at time t, k indicates the growth rate of Omicron cases, and N_p represents the total number of Omicron infected cases in the final phase of the epidemic.

Let's differentiate equation (2.3) with respect to t where $\alpha(t)$ equals k. We get,

$$\frac{d^2I(t)}{dt^2} = k\frac{dI(t)}{dt}\left(1 - \frac{2I(t)}{N_p}\right) \tag{2.5}$$

Therefore, when $\frac{d^2I(t)}{dt^2} = 0$, then $\frac{dI(t)}{dt} = 0$ or $I(t) = \frac{N_p}{2}$. Where $I(t) = \frac{N_p}{2}$ is called the infection point, meaning that the point where the growth curve changes its concavity.

- When $I < \frac{N_p}{2}$ it concave upward which mean that $\frac{dI}{dt}$ increases with time.
- When $I > \frac{N_p}{2}$ it concave downward, meaning that $\frac{dI}{dt}$ decreases with time.

Now, Putting $I(t) = \frac{N_p}{2}$ into equation (2.3) where $\alpha(t)$ is k, then we have

$$\frac{dI(t)}{dt} = \frac{kN_p}{4} \tag{2.6}$$

Where $\frac{dI(t)}{dt} = \frac{kN_p}{4}$ is the growth rate at its maximum point.

Now putting, $I = \frac{N_p}{2}$ into $I = \frac{N_p}{1+\gamma e^{-\int_0^T k\,dt}}$, we have

$$1 + \gamma e^{-\int_0^T k\,dt} = 2$$

then

$$T = \frac{ln(\gamma)}{k}$$

where $T = \frac{ln(\gamma)}{k}$ is the estimation of the time at which the epidemic reaches its maximum growth rate. Therefore, the slope of equation (2.6) is symmetrical at the inflection point (peak time) for the value of $I = \frac{N_p}{2}$ defined as $T = \frac{ln(\gamma)}{k}$ where the curve assumes its sigmoid-shaped. When the values of $I(t)$ are less than the N_p, then

$$\frac{dI(t)}{dt} = kI(t)$$

meaning that the daily and the cumulative number of individuals reported to be infected by a viral epidemic will grow exponentially. The epidemic's peak marks a turning point regarding the total number of infected cases. After this point, a basic exponential growth curve can no longer represent the epidemic's time evolution since the number of infected people is not growing at this point. Instead, during this phase, the number of daily cases of infection will decrease. The constant formula is enough to describe how $I(t)$ viral epidemics change over time. For example, [3] the constant model provides a good fit for COVID-19 pandemic data and accurately predicts. The parameters k, γ and N_p generally remain unchanged if we use short series of existing data of the Omicron, that is using the subset of the existing data, but when considering a long series of exist-

ing data where the remaining available data is added the constant model does not make accurate predicting in which the constant model underestimates $I(t)$ [17].

These results in a question that was raised [17] since the constant model gives a lower bound of $I(t)$, can there be a mathematical model that can give an upper bound of $I(t)$ and capture the remaining data that will provide a good fit and more accurate prediction of a long series of existing data. After testing more than 50 distinct forms of $\alpha(t)$, Fokas et al [17] found two innovative formulae called rational and birational that address the question. The analytical solution for equation (2.3) gives

$$I(t) = \frac{N_p}{1+\gamma e^{-\eta}} \tag{2.7}$$

where

$$\eta = \int \alpha(t)dt \tag{2.8}$$

Given that $\alpha(t) = \frac{ab}{1+bt}$ which is an algebraic function, equation (2.8) becomes

$$\eta = \int \frac{ab}{1+bt}dt = ln|1+bt|^a \tag{2.9}$$

Therefore,

$$I(t) = \frac{N_p}{1+\gamma(1+bt)^{-a}}. \tag{2.10}$$

Where (2.10) is called the rational model with N_p, γ, b and a as the parameter.

Given that

$$\alpha(t) = \begin{cases} \frac{ab}{1+bt}, & t \leq M \\ \frac{a_1 b_1}{1+b_1 t}\frac{1}{1+(1-(d_1/N_p))(1+b_1 t)^{-a_1}}, & t > M \end{cases} \tag{2.11}$$

which is a piece-wise function, then equation (2.8) becomes

$$I(t) = \begin{cases} \frac{d}{1+\gamma(1+bt)^{-a}}, & t \leq M \\ \frac{d}{1+\gamma(1+bM)^{-a}} - \frac{d_1}{1+\gamma_1(1+b_1M)^{-a_1}} + \frac{d_1}{1+\gamma_1(1+b_1t)^{-a_1}}, & t > M \end{cases} \tag{2.12}$$

Where (2.12) is called the birational model and a, a_1, b, b_1, d, d_1, γ, and γ_1 are the parameters. M is a constant parameter in the vicinity of T.

2.2.3 Learning Time-Series Transmission Rate

Equation (2.3) can also be referred to as a Riccati equation in [17] since it is defined by the time-dependent function $\alpha(t)$ and the constant parameter N_p. Riccati equation is an equation that has a single parameter. When it comes to simulating infectious diseases mathematically, the Riccati equation with constant coefficients appears in numerous places. For example, the mathematical model's Riccati equation with constant coefficients describes the dynamic relationship between parasites and their hosts [52]. Taking $\alpha(t)$ as a subject of the formula in equation (2.3), we have

$$\alpha(t) = \frac{\frac{dI(t)}{dt}}{I(t)\left(1 - \frac{I(t)}{N_p}\right)} \tag{2.13}$$

The fact that $\alpha(t)$ depends on time reflects several time-dependent factors, such as the effects of the government's different actions depending on t, and also includes the effects of public health actions and how people reacted to them. For example, lockdown was the first public health precaution used early in the COVID-19 outbreak. Other actions like social isolation, contact tracing, masking, and early infection identification. These show that there may be multiple forms of $\alpha(t)$. Therefore, $\alpha(t)$ was learned as a time series; This was done by guiding the deep learning neural network toward learning the form of alpha from the data. As a result, the time series model is a neural network based where the neural network learns every vital information of $\alpha(t)$ going on in the data.

2.3 Logistic Informed Neural Network (LINN) for Time-varying Transmission Rate.

The Physics-Informed Neural Network (PINN), which has a Feedforward Neural Network architecture, is the best way to learn the parameters of infectious disease models. Given the limitations of existing models, the Logistic Informed Neural Network (LINN) is introduced. This model is inspired by applying PINN to logistic models, intending to surmount the constraints inherent to conventional statistical techniques. The LINN algorithm is specifically designed to learn the time-varying transmission rate and identify from data the effects of the wide range of different measures that the given country has developed to prevent the viral infection of Omicron from spreading. We delve into the intricate technical details of LINNs implementation, elaborating on its unique computational processes and algorithms and illustrating how it stands out in learning the time-dependent function depending on the essential characteristics of the particular virus and the daily and cumulative effect of the variety of different mitigation measures taken by the given country for the prevention of the spread of the viral infection from the available data. The comparative analysis elucidates LINNs superiority in handling specific data types and scenarios, underscoring its practical applicability and impact in real-world situations. However, despite its advancements, LINN may have limitations, including potential challenges with data dependency, interpretability, and adaptability to ever-evolving viral strains and diverse preventive measures. It might also face difficulties generalizing findings across varied populations and healthcare systems and may require continuous refinement and validation to maintain its reliability and accuracy.

We employed four hidden layers with 64 neurons each in all of the simulations discussed in this study, and the training loss was minimized in 50,000 iterations. The hyperbolic tangent function is the activation function for the hidden and output layers. In our study, the loss function of LINN and the fundamental framework, which is FNN, are shown in Figure 2.1. This figure divides the LINN's loss function into two components called RE_{loss} and ME_{loss}. Subtracting the right side from the left side of the logistic model equation (2.4) and the analytical solution produces the residuals represented by the RE_{loss}. The mean squared error between the outputs of the neural network and the data is denoted as ME_{loss}.

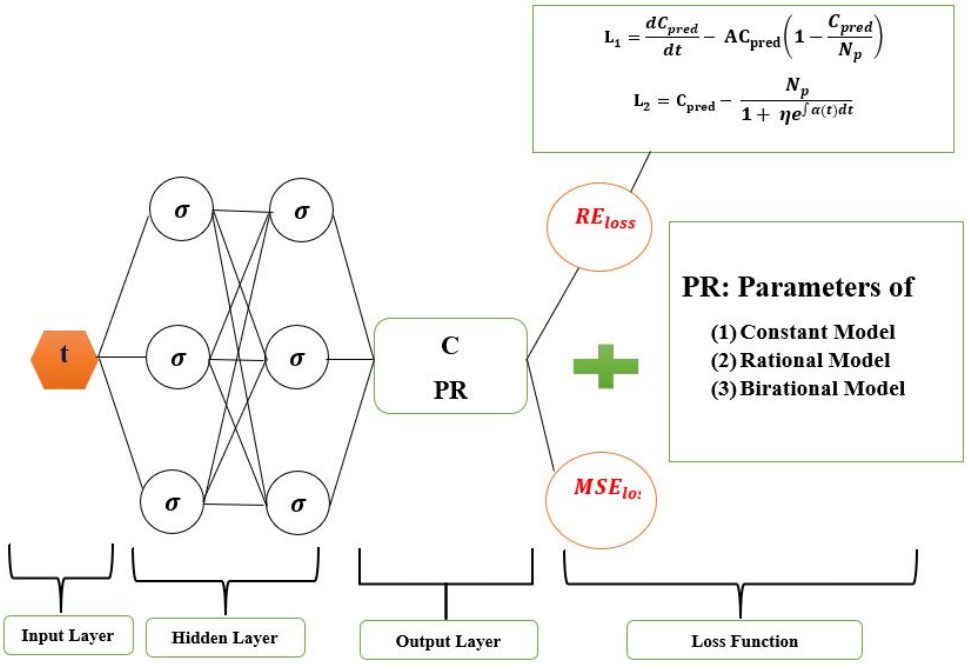


Figure 2.1: Logistic Informed Neural Network schematic diagram with non-linear time-varying transmission rate.

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n) - I_{pred}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{2}\sum_{n=1}^{X} ||L_i(t_n)||_2^2$$

where the residual L_i, $i = 1,2$ is as follows:

$$L_1 = \frac{dI(t)}{dt} - AI(t)\left(1 - \frac{I(t)}{N_p}\right)$$

$$L_2 = I(t) - \frac{N_p}{1 + \gamma e^{-\int A dt}}$$

Where A represents the time-varying transmission rate.

2.3.1 Parameters Identification Algorithms.

This section will discuss the logistic informed neural network (LINN) algorithms for learning the parameters of constant, rational, and birational models and the time-varying transmission rate.

2.3.2 Constant Model.

We provide LINN algorithm 1 to learn the parameters of the constant model. We set $\alpha(t) = k$ in (2.8), which results in a constant model. The analytical solution, which contains three parameters, is learned and obtained. The learned daily and cumulative infection solution is matched against the daily and cumulative infection data. We implement algorithm 1 using publicly available Omicron data. The output of LINN is the learned solution of the constant model denoted by $I(t_n;\delta;\theta), n = 1,...,X$. Where δ represents the neural network weights and biases and θ represents the constant model solution parameters. X is the number of the training set. A cubic spline is used to create the training data, represented by $\hat{I}(t_n), n = 1,...X$.

We notice that training data for all compartments in the constant model is unavailable; however, the constant model residual is included in the MSE loss function, which allows LINN to capture the interactions between the compartments. The LINN algorithm 1 for learning the optimal parameters of the constant model is shown below. The MSE loss function for LINN with a time-varying transmission rate is:

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n;\delta;\theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{2}\sum_{n=1}^{X} ||L_i(t_n;\delta;\theta)||_2^2$$

where the residual L_i, $i = 1,2$, is as follows:

$$L_1 = \frac{dI(t_n;\delta;\theta)}{dt} - kI(t_n;\delta;\theta)\left(1 - \frac{I(t_n;\delta;\theta)}{N_p}\right)$$

$$L_2 = I(t_n;\delta;\theta) - \frac{N_p}{1+\gamma e^{-kt_n}}$$

Algorithm 1 LINN algorithm for Constant Model

1: Construct LINN for Constant Model
2: Specify the input: $t_n, n = 1, ..., X$
Initialize LINN parameter: δ
Initialize Constant model parameter: $\theta = (k, \eta, N_p)$
Output layer: $I(t_n; \delta; \theta), n = 1, ..., X$
3: Specify the training set
Training data: using cubic spline to generate $\hat{I}(t_n), n = 1, ..., X$ given from the dataset.
Split data into $x\%$ training data and $(100 - x)\%$ testing data
l = length(data)
Train = $data[0 : l * x\%]$; Test = $data[l * x\% : l]$
4: Train the neural network
Specify an MSE loss function

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n; \delta; \theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{2}\sum_{n=1}^{X} ||L_i(t_n; \delta; \theta)||_2^2$$

Minimize the MSE loss function: compute $argmin_{\{\delta;\theta\}}$(MSE) using Adam Optimizer.
5: Return LINN solution
$I(t_n; \delta; \theta), n = 1, ..., X$
Parameters: N_p, k, η

2.3.3 Rational Model.

We implement the LINN algorithm 2 to learn the parameters of the rational model. We set $\alpha(t) = \frac{ab}{1+bt}$ in (2.8), which we call the Rational model. The analytical solution containing four parameters is learned and obtained. The learned daily and cumulative infective solutions are matched against the daily and cumulative infective data. The output of LINN is the learned solution of the Rational model denoted by $I(t_n; \delta; \theta), n = 1, ..., X$, where δ represents the neural network weights and biases and θ represents the Rational model solution parameters. X is the number of the training set. The same network also produces the time-varying transmission rate denoted by $\alpha(t_n; \delta; \theta) = \frac{ab}{1+bt_n}, n = 1, ..., X$.

Algorithm 2 LINN algorithm for Rational Model with time-varying transmission rate

1: Construct LINN for Rational Model
   Specify the input: $t_n, n = 1, ..., X$
   Initialize LINN parameter: δ
   Initialize Rational model parameter: $\theta = (a, b, \eta, N_p)$
   Output layer: $I(t_n; \delta; \theta)$ and $\alpha(t_n; \delta; \theta) = \frac{ab}{1+bt_n}, n = 1, ..., X$
2: Specify the training set
   Training data: using cubic spline to generate $\hat{I}(t_n), n = 1, ..., X$ given from the dataset.
   Split data into $x\%$ training data and $(100 - x)\%$ testing data
   l = length(data)
   Train = $data[0 : l * x\%]$; Test = $data[l * x\% : l]$
3: Train the neural network
   Specify an MSE loss function

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n; \delta; \theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{2}\sum_{n=1}^{X} ||L_i(t_n; \delta; \theta)||_2^2$$

   Minimize the MSE loss function: compute $argmin_{\{\delta;\theta\}}$(MSE) using Adam Optimizer.
4: Return LINN solution
   $I(t_n; \delta; \theta), n = 1, ..., X$
   $\alpha(t_n; \delta; \theta), n = 1, ..., X$
   Parameters: a, b, N_p, η

2.3.4 Birational Model.

We introduced the LINN algorithm 3 to learn the parameters of the Birational model. We set

$$\alpha(t) = \begin{cases} \frac{ab}{1+bt}, & t \leq M \\ \frac{a_1 b_1}{1+b_1 t} \frac{1}{1+(1-(d_1/N_p))(1+b_1 t)^{-a_1}}, & t > M \end{cases}$$

in (2.8), which we call the Birational model. The analytical solution containing eight parameters is learned and obtained.

The learned daily and cumulative infective solutions are matched against the daily and cumulative infective data.

The output of LINN is the learned solutions of the Birational model denoted by $I(t_n; \delta; \theta), n = 1, ..., X$. The same network also produces the time-varying transmission rate denoted by

Algorithm 3 LINN algorithm for Birational Model with time-varying transmission rate

1: Construct LINN for Birational Model
Specify the input: $t_n, n = 1, ..., X$
Initialize LINN parameter: δ
Initialize Birational model parameter: $\theta = (a, b, \eta, d, a_1, b_1, \eta_1, d_1)$
Output layer: $I(t_n; \delta; \theta)$ and

$$\alpha(t_n;\delta;\theta) = \begin{cases} \frac{ab}{1+bt}, & t \le X \\ \frac{a_1 b_1}{1+b_1 t_n} \frac{1}{1+(1-(d_1/N_p))(1+b_1 t_n)^{-a_1}}, & t > X \end{cases}$$

2: Specify the training set
Training data: using cubic spline to generate $\hat{I}(t_n), n = 1, ..., X$ given from the dataset.
Split data into $x\%$ training data and $(100-x)\%$ testing data
l = length(data)
Train = $data[0 : l * x\%]$; Test = $data[l * x\% : l]$

3: Train the neural network
Specify an MSE loss function

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n;\delta;\theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{3}\sum_{n=1}^{X} ||L_i(t_n;\delta;\theta)||_2^2$$

Minimize the MSE loss function: compute $argmin_{\{\delta;\theta\}}$(MSE) using Adam Optimizer.

4: Return LINN solution
$I(t_n; \delta; \theta), n = 1, ..., X$
$\alpha(t_n; \delta; \theta), n = 1, ..., X$
Parameters: $a, b, d, \eta, a_1, b_1, d_1, \eta_1$

$$\alpha(t_n;\delta;\theta) = \begin{cases} \frac{ab}{1+bt_n}, & t_n \le M \\ \frac{a_1 b_1}{1+b_1 t_n} \frac{1}{1+(1-(d_1/N_p))(1+b_1 t_n)^{-a_1}}, & t_n > M \end{cases}, n = 1, ..., X.$$

The cubic spline is used to create the training data, represented by $\hat{I}(t_n), n = 1, ...X$. The MSE loss function of LINN for the Birational Model with a time-varying transmission rate is:

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n;\delta;\theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{3}\sum_{n=1}^{X} ||L_i(t_n;\delta;\theta)||_2^2$$

where the residual L_i, $i = 1, 2, 3$ is as follows:

$$L_1 = \frac{dI(t_n;\delta;\theta)}{dt} - \alpha(t_n;\delta;\theta) I(t_n;\delta;\theta)\left(1 - \frac{I(t_n;\delta;\theta)}{N_p}\right)$$

$$L_2 = I(t_n;\delta;\theta) - \frac{d}{1+\gamma(1+bt_n)^{-a}} \quad t_n \leq X$$

$$L_3 = I(t_n;\delta;\theta) - \left(\frac{d}{1+\gamma(1+bX)^{-a}} - \frac{d_1}{1+\gamma_1(1+b_1X)^{-}a_1} + \frac{d_1}{1+\gamma_1(1+b_1t_n)^{-a_1}}\right) \quad t_n > X$$

2.4 Logistic Informed Neural Network for Time-dependent Transmission Rate.

We offer a Logistic Informed Neural Network (LINN) with two networks, shown in Figure 2.2. The first network learned the constant parameter for the Omicron infection's daily and cumulative initial value. The second network learned the form of the time-dependent function that helps capture the various time-dependent factors or elements, including the impact of public health actions and the public response to the actions in preventing the spread of Omicron. The LINN algorithm 4 is an excellent choice for learning the time-series transmission rate (the form of the time-dependent function in the transmission of Omicron) that is robust enough to adjust to whatever information is present in the data. We employed four hidden layers with 64 neurons each to achieve good accuracy in the neural network. As a result, the training loss was minimized using 50,000 epochs. The hyperbolic tangent function is the activation function for the hidden and output layers. The learned daily and cumulative infective solutions are matched against the daily and cumulative infective data.

The Logistic Informed Neural Network (LINN) is adapted for the logistic differential equation (2.3), where the mean square error (MSE) of this neural networks loss function includes the known logistic dynamics. At the same time, the time-series transmission rate detects various time-dependent factors in the Omicron-infected data. LINN's output is the model's learned solution, denoted by $I(t_n;\delta;\theta), n = 1,...,X$. Where δ represents the neural network weights and biases and θ represents the time-series model solution parameter. X is the number of the training set. The second network representing the time-series transmission rate is denoted by $\alpha(t_n;\Phi), n = 1,...,X$, The parameter Φ represents the weights and biases of the network. The cubic spline is used to create the training data, which are represented by $\hat{I}(t_n), n = 1,...X$. We notice that training data for all compartments in the constant model are unavailable; however, the logistic model residual is included in the MSE loss function, which allows LINN to capture the interactions between the

Algorithm 4 LINN algorithm for Time-series Model with time-series transmission rate

1: Construct LINN
   Specify the input: $t_n, n = 1, ..., X$
   Initialize LINN parameter: δ
   Initialize Time-series model parameter: $\theta = N_p$
   Output layer: $I(t_n; \delta; \theta), n = 1, ..., X$
2: Construct neural network: α
   Specify the input: $t_n, n = 1, ..., X$
   Initialize LINN parameter: Φ
   Output layer: $\alpha(t_n; \Phi), n = 1, ..., X$
3: Specify the training set
   Training data: using cubic spline to generate $\hat{I}(t_n), n = 1, ..., X$ given from the dataset.
   Split data into $x\%$ training data and $(100 - x)\%$ testing data
   l = length(data)
   Train = $data[0 : l * x\%]$; Test = $data[l * x\% : l]$
4: Train the neural network
   Specify an MSE loss function

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n; \delta; \theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{1}\sum_{n=1}^{X} ||L_i(t_n; \delta; \theta, \Phi)||_2^2$$

   Minimize the MSE loss function: compute $argmin_{\{\delta;\theta\}}$(MSE) using Adam Optimizer.
5: Return LINN solution
   $I(t_n; \delta; \theta), n = 1, ..., X$
   Parameters: N_p
6: Return Time-series alpha:
   $\alpha(t_n; \Phi), n = 1, ..., X$

compartments.

The MSE loss function for LINN with the time-series transmission rate is:

$$MSE = \frac{1}{X}\sum_{n=1}^{X} ||I(t_n; \delta; \theta) - \hat{I}(t_n)||_2^2 + \frac{1}{X}\sum_{i=1}^{1}\sum_{n=1}^{X} ||L_i(t_n; \delta; \theta)||_2^2$$

where the residual L_i, $i = 1$ is as follows:

$$L_1 = \frac{dI(t_n; \delta; \theta)}{dt} - \alpha(t_n; \Phi) I(t_n; \delta; \theta)\left(1 - \frac{I(t_n; \delta; \theta)}{N_p}\right)$$

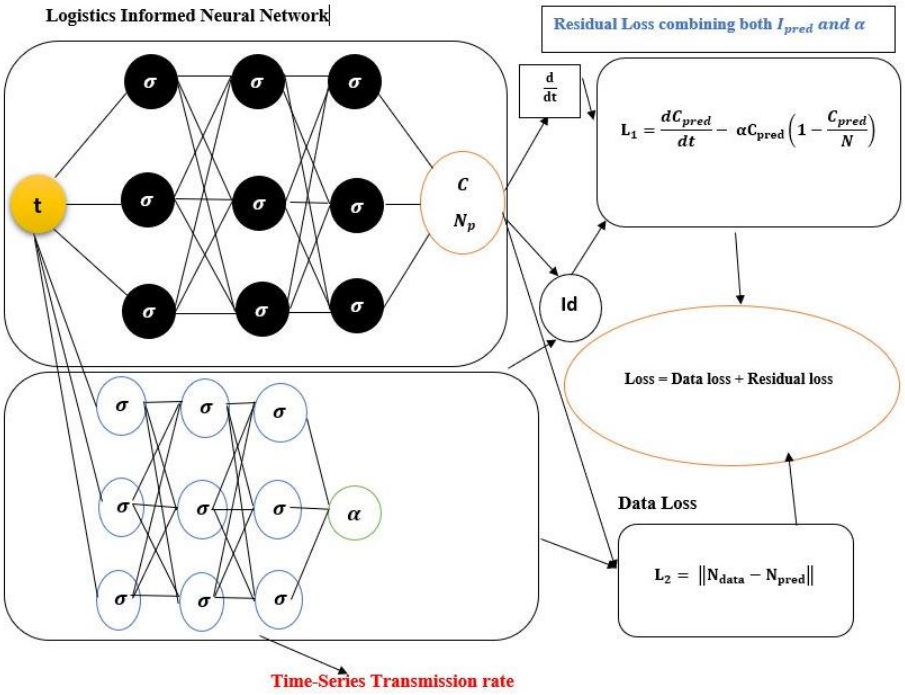


Figure 2.2: Schematic diagram of the Logistic Informed Neural Network with non-linear time-series transmission rate.

2.5 Data and Data Preprocessing

The data for the Omicron variant for China, Italy, and Portugal were obtained from the World Health Organization (WHO) [63]. The data include daily new infected cases, and we consider 160 days from March 25 to August 31, 2022. Furthermore, the cumulative Omicron data was also considered and obtained via calculation. According to WHO [64], the Omicron(B.1.1.529) variant includes BA.1, BA.2, BA.3, BA.4, and BA.5. As a result, it is observed that the daily new infected cases and cumulative number of individuals infected with COVID-19 in these countries are different. In March 2022, the Omicron variant BA.4 and BA.5 were detected in Europe, where the proportion increased rapidly. According to [42], the SARS-CoV-2 Omicron variant BA.4 and BA.5 dominate and increase in Portugal with a proportion of around 87%, followed by a high proportion in Italy, Germany, and some other countries in Europe [14]. The Omicron variants were discovered in China in December 2021 and later dominated and spread rapidly since February 2022, according to the National Health Commission of the Peoples Republic of China [41].

The daily Omicron data is preprocessed by implementing several data transformation and normalization steps to ensure its suitability for modeling. After figuring out the rolling mean to smooth out short-term changes, the data is reshaped into a two-dimensional array, leaving out the first six elements so it can be used in the next stages of processing. The first six elements of the original data are also reshaped similarly. Subsequently, these reshaped segments are vertically stacked to form a modified data array, ensuring that the initial segments of the original data are included. This modified data structure is advantageous as it maintains the integrity of the original data while integrating the smoothed elements, providing a balanced and coherent dataset for analysis. In the final preprocessing step, the data is normalized to a scale between 0 and 1, using a min-max normalization technique. Normalizing the data is crucial as it brings all the variables to a comparable scale, preventing any one variable from disproportionately influencing the model due to its scale. This transformation maintains the relative dispersion of the data points, ensuring more stable and meaningful analysis results.

The resulting new data array is now well-suited for model development, offering a refined and well-scaled representation of the initial Omicron data. The cumulative Omicron data is preprocessed by implementing several procedures to refine the data and prepare it for subsequent modeling tasks. Initially, the data for time (t) and total cases (I) are reshaped into one-dimensional arrays, which is instrumental in streamlining the data structure and facilitating easier manipulation and analysis in the subsequent stages. Post-reshaping, the data undergoes normalization, a pivotal step in scaling the data within a specific range between 0 and 1. This normalization is executed through a min-max scaling technique, where each data point is transformed based on the minimum and maximum values within the dataset. This process is crucial as it mitigates the risk of any variable disproportionately influencing the model due to its original scale, ensuring more reliable and accurate model outcomes.

After normalization, cubic spline interpolation is leveraged to enrich the data points available for training. This method is renowned for generating additional data points within the datasets range, enhancing the models access to information during the training phase. In this specific case,

it creates a smoother curve by fitting a cubic polynomial between each pair of data points in the new data array and subsequently generates 1000 new data points for 0 to 160. This interpolation makes the dataset fuller and smoother. This lets the model pick up more subtle and nuanced patterns in the daily and cumulative Omicron data, which improves the accuracy and reliability of the analysis and modeling tasks that come next. The processed cumulative infected data was split into 70% and 30% training and validation using random splitting and passed into LINN to learn the parameters of the mathematical models.

2.6 Results and Discussion

The data simulation results of the mathematical models, the prediction of the time that a plateau will be reached, and the cumulative number of individuals reported to be infected by the Omicron variant for China, Italy, and Portugal are presented in this section.

2.6.1 Parameter Identification and Data-Driven Simulations

The parameters of the mathematical models are learned using the validation data. For reliability and accuracy of the learned parameters, the LINN is run five independent times to quantify the uncertainty in the learned parameters. The situation with epidemics in different countries is affected by many external factors, such as the number of reports, the measures taken to deal with the epidemic, population movement measures, etc. Therefore, our model considers these factors, making our models suitable for prediction. The learned parameters from the testing data and the analytical solution to the mathematical models are used to predict the time that a plateau will be reached and the cumulative number of individuals reported to be infected by the Omicron variant in a given country. The time-series model was used to make a short prediction of the daily and the time that a plateau will be reached, as well as the cumulative number of individuals reported to be infected by the Omicron variant in a given country. In addition, the relative error in the numerical solution is used to verify the accuracy of the LINN-learned parameters with respect to the analytical solutions of each model. Therefore, the relative error metric showing the fitting accuracy of each mathematical model using the LINN algorithm was provided.

Table 2.1: Comparative analysis of model parameters and error metrics of daily Omicron infections in China.

Params	**Constant**	**Params**	**Rational**	**Params**	**Birational**	**Params**	**Time-series**
N	44743	N	44444	d	364597	N	82837
k	0.1783	a	26.9413	a	6.3973	RMSE	386
γ	1020.4299	γ	7521.3825	γ	2006.2519	MAPE	0.0202
RMSE	9349	b	0.0101	b	0.02868	EV	0.9997
MAPE	0.2517	RMSE	9142	d_1	85368		
EV	0.8558	MAPE	0.2357	a_1	6.9310		
		EV	0.8621	γ_1	1501.4508		
				b_1	0.02449		
				RMSE	2864		
				MAPE	0.1667		
				EV	0.9869		

Tables 2.1, 2.2, and 2.3 present the parameters and error metrics showing the four different mathematical models applied to the fitting accuracy of the daily Omicron infection data in China, Italy, and Portugal. These tables show how the four distinct mathematical models are evaluated to determine the fitting accuracy of the observed data. It also provides insights into the effectiveness of different models in capturing the infection dynamics. The table illustrates the superiority of the time-series model in terms of reduced error metrics and higher variance explained, demonstrating its optimal fit and reliability in modeling the epidemics daily data in China, Italy, and Portugal.

The tables show that the birational model has eight parameters, the rational model has four parameters, the constant model has three, and the time-series model has one parameter. The error metrics demonstrate that the time-series model is the best because of its smallest root mean squared error and mean absolute percentage error values. Furthermore, the time series model has the greatest explained variance, which shows that the time-series model fits the data better than the rest of the models, making the model preferable to the other three mathematical models when considering the daily data of an epidemic.

Figures 2.3a-d, 2.4a-d, and 2.5a-d also show the data fitting using the four mathematical models with the original daily Omicron infective data of China, Italy, and Portugal. These figures show

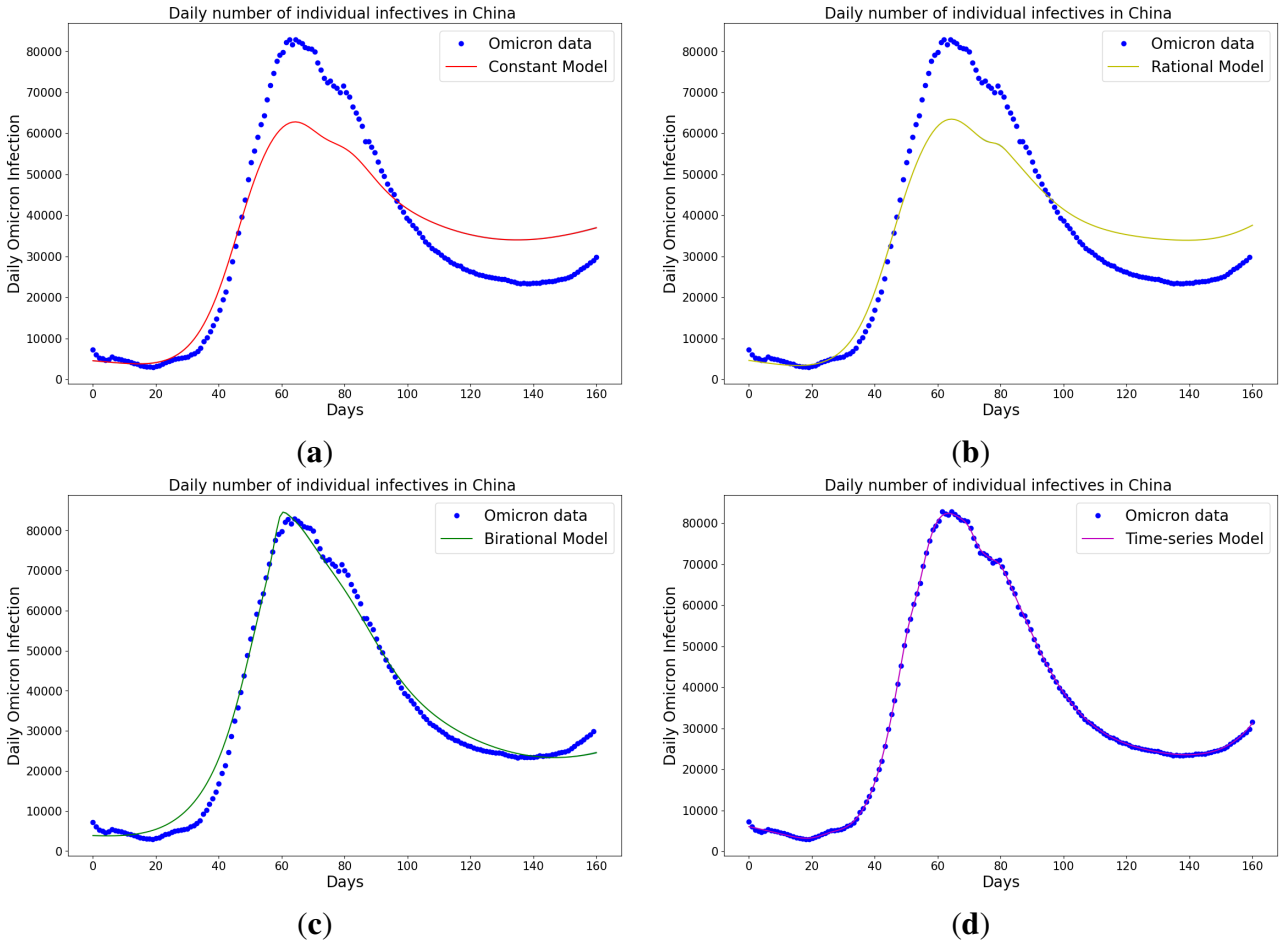


Figure 2.3: Simulation results of China daily Omicron data from 25th of March to 31 August 2022. The graph of daily Omicron infectives data and the learned infectives of **(a)** Constant model using LINN algorithm 1; **(b)** Rational model using LINN algorithm 2; **(c)** Birational model using LINN algorithm 3; **(d)** Time-series model using LINN algorithm 4.

how closely each model accurately fits the data. Notably, the time-series model fits the observed data better than the other models, which shows how well it can learn and describe how the Omicron variant infection spreads in China, Italy, and Portugal. This excellent fit shows that the model captures the complex patterns and trends of daily Omicron infections in China, Italy, and Portugal. The time-series model better fits the daily Omicron infective data than the other three mathematical models. The parameter $\alpha(t)$ that indicates the transmission rate is critical because it determines the epidemic's trend. The learned transmission rate of rational and birational in Figures 2.6a-b, 2.7a-b, and 2.8a-b shows an exponential decay. In addition, it was observed from Figures 2.5(c), 2.6(c), and 2.7(c) that the learned time-series model in this paper was able to capture the form of

Table 2.2: Comparative analysis of model parameters and error metrics of daily Omicron infections in Italy.

Params	**Constant**	**Params**	**Rational**	**Params**	**Birational**	**Params**	**Time-series**
N	86344	N	84524	d	175403	N	175649
k	0.2601	a	24.1175	a	13.5611	RMSE	1035
γ	980.6510	γ	7519.4065	γ	5227.3434	MAPE	0.01607
RMSE	21897	b	0.0177	b	0.02696	EV	0.9995
MAPE	0.2451	RMSE	20385	d_1	1859879		
EV	0.7866	MAPE	0.2302	a_1	7.1762		
		EV	0.8151	γ_1	9519.0627		
				b_1	0.07002		
				RMSE	7833		
				MAPE	0.1377		
				EV	0.9728		

$\alpha(t)$ and the information going on in daily Omicron infection. This means that the time-series model will be able to capture any sudden change in the trend of newly infected daily cases. The figures illustrate the learning and representation of the daily Omicron variant's transmission rate from observed data. It is evident that the rational and birational models exhibit a trend in $\alpha(t)$ that suggests exponential decay, whereas the time-series model reveals a different, more varied trend in $\alpha(t)$.

The parameters, plateau days, plateau cases, and error metrics for the four models used to fit China's, Italy's, and Portugal's cumulative Omicron infection data can be seen in Tables 2.4, 2.5, and 2.6. These tables show how the four distinct mathematical models are evaluated to determine the fitting accuracy of the observed data and predict the plateau days and cases. It also provides insights into the effectiveness of different models in capturing the infection dynamics. The table illustrates the superiority of the time-series model in terms of reduced error metrics and higher variance, demonstrating its optimal fit and reliability in modeling the epidemics cumulative data in China, Italy, and Portugal.

In addition, the inflection point T on the tables was calculated by fitting all the cumulative infection data, namely data up to August 31, 2022. The table shows that the birational model has

Table 2.3: Comparative analysis of model parameters and error metrics of daily Omicron infections in Portugal.

Params	**Constant**	**Params**	**Rational**	**Params**	**Birational**	**Params**	**Time-series**
N	11393	N	1165436	d	67997	N	27541
k	0.2795	a	0.4740	a	1.3654	RMSE	377
γ	278.8262	γ	8871.1400	γ	5497.3608	MAPE	0.02786
RMSE	4350	b	45.5159	b	6.5310	EV	0.9973
MAPE	0.6676	RMSE	4633	d_1	169104		
EV	0.6889	MAPE	0.6111	a_1	3.5296		
		EV	0.6712	γ_1	2570.4033		
				b_1	0.2312		
				RMSE	1633		
				MAPE	0.1067		
				EV	0.9575		

eight parameters, the rational model has four parameters, the constant model has three parameters, and the time-series model has one constant parameter. The error metrics demonstrate that the time-series model is the best because of its smallest root mean squared error and mean absolute percentage error values. Furthermore, the time series model has the greatest explained variance and the coefficient of determination values, which show that the time-series model fits the data better than the rest of the models, making the model preferable to the other three mathematical models when considering the cumulative data of an epidemic.

The four mathematical models were used to fit the original cumulative Omicron infective data from China, Italy, and Portugal. These models are shown in Figures 2.9ad, 2.10ad, and 2.11ad. Furthermore, the time-series model better fits the cumulative Omicron infective data than the other three mathematical models. This figure illustrates the learning and representation of the cumulative Omicron variant's transmission rate from observed data. It is evident that the rational and birational models exhibit a trend in $\alpha(t)$ that suggests exponential decay, whereas the time-series model reveals a different, more varied trend in $\alpha(t)$. It shows how closely each model accurately fits the data. Notably, the time-series model fits the observed data better than the other models, which shows how well it can learn and describe how the Omicron variant infection spreads in China, Italy,

Table 2.4: Comparative analysis of model parameters, plateau days, plateau cases, and error metric for constant, rational, birational, and time-series models for the cumulative Omicron infections in China.

Params	**Constant**	**Params**	**Rational**	**Params**	**Birational**	**Params**	**Time-series**
N	5544704	N	6243040	d	4303425	N	7320999
k	0.09262	a	5.0366	a	2.00762	RMSE	1657179
γ	954.2391	γ	1505.8875	γ	2499.9048	MAPE	0.3406
RMSE	1737625	b	0.04187	b	0.3706	EV	0.7808
MAPE	0.3609	RMSE	1664443	d_1	6919512	plateau(days)	390
EV	0.7565	MAPE	0.3334	a_1	3.8630	cases	7320636
plateau(days)	122	EV	0.7728	γ_1	1202.6787		
cases	5488350	plateau(days)	282	b_1	0.06890		
T	74	cases	6221115	RMSE	1657179		
				MAPE	0.3403		
				EV	0.7792		
				plateau(days)	317		
				cases	6458279		

and Portugal. This excellent fit shows that the model captures the complex patterns and trends of cumulative data on Omicron infections in Portugal

The learned rational and birational transmission rate in Figures 2.12a-b, 2.13a-b, and 2.14a-b shows an exponential decay for the cumulative Omicron infection in China, Italy, and Portugal. In addition, it was observed from Figures 2.12(c), 2.13(c), and 2.14(c) that the learned time-series model in this paper was able to capture the form of $\alpha(t)$ and the information going on in the cumulative Omicron infection in China, Italy, and Portugal.

This means that the time-series model will be able to capture any sudden change in the trend of newly infected cumulative cases. This distinction implies that the time-series model has successfully captured extensive information on the ongoing mitigation measures evident in the data. Basically, this difference shows that the time-series model is better at capturing the details and nuances of the data, pointing to its enhanced reliability in representing the real-world dynamics of the Omicron variant's spread.

Table 2.5: Comparative analysis of model parameters, plateau days, plateau cases, and error metric for constant, rational, birational, and time-series models for the cumulative Omicron infections in Italy.

Params	**Constant**	**Params**	**Rational**	**Params**	**Birational**	**Params**	**Time-series**
N	9502524	N	13375081	d	73618	N	13338639
k	0.1182	a	2.2535	a	14.8388	RMSE	2604505
γ	953.5184	γ	7499.3847	γ	1998.6043	MAPE	0.3933
RMSE	2605441	b	0.6809	b	4.8570	EV	0.8957
MAPE	0.4474	RMSE	2652081	d_1	18112258	plateau(days)	381
EV	0.8953	MAPE	0.3515	a_1	1.5291	cases	13337516
plateau(days)	96	EV	0.8832	γ_1	996.7389		
cases	9396847	plateau(days)	442	b_1	1.1285		
T	58	cases	13121787	RMSE	2651400		
				MAPE	0.4851		
				EV	0.8940		
				plateau(days)	641		
				cases	15409748		

Table 2.6: Comparative analysis of model parameters, plateau days, plateau cases, and error metric for constant, rational, birational, and time-series models for the cumulative Omicron infections in Portugal.

Params	**Constant**	**Params**	**Rational**	**Params**	**Birational**	**Params**	**Time-series**
N	2556208	N	2901213	d	21614	N	3396848
k	0.1280	a	3.8416	a	14.5922	RMSE	720538
γ	1954.1612	γ	7499.4426	γ	2498.6008	MAPE	0.4133
RMSE	743396	b	0.1478	b	4.6981	EV	0.8821
MAPE	0.4732	RMSE	723210	d_1	18112258	plateau(days)	312
EV	0.8490	MAPE	0.4209	a_1	3.9029	cases	3395243
plateau(days)	94	EV	0.8787	γ_1	999.9216		
cases	2526824	plateau(days)	305	b_1	0.0797		
T	59	cases	2892395	RMSE	723022		
				MAPE	0.4972		
				EV	0.8747		
				plateau(days)	308		
				cases	2911888		

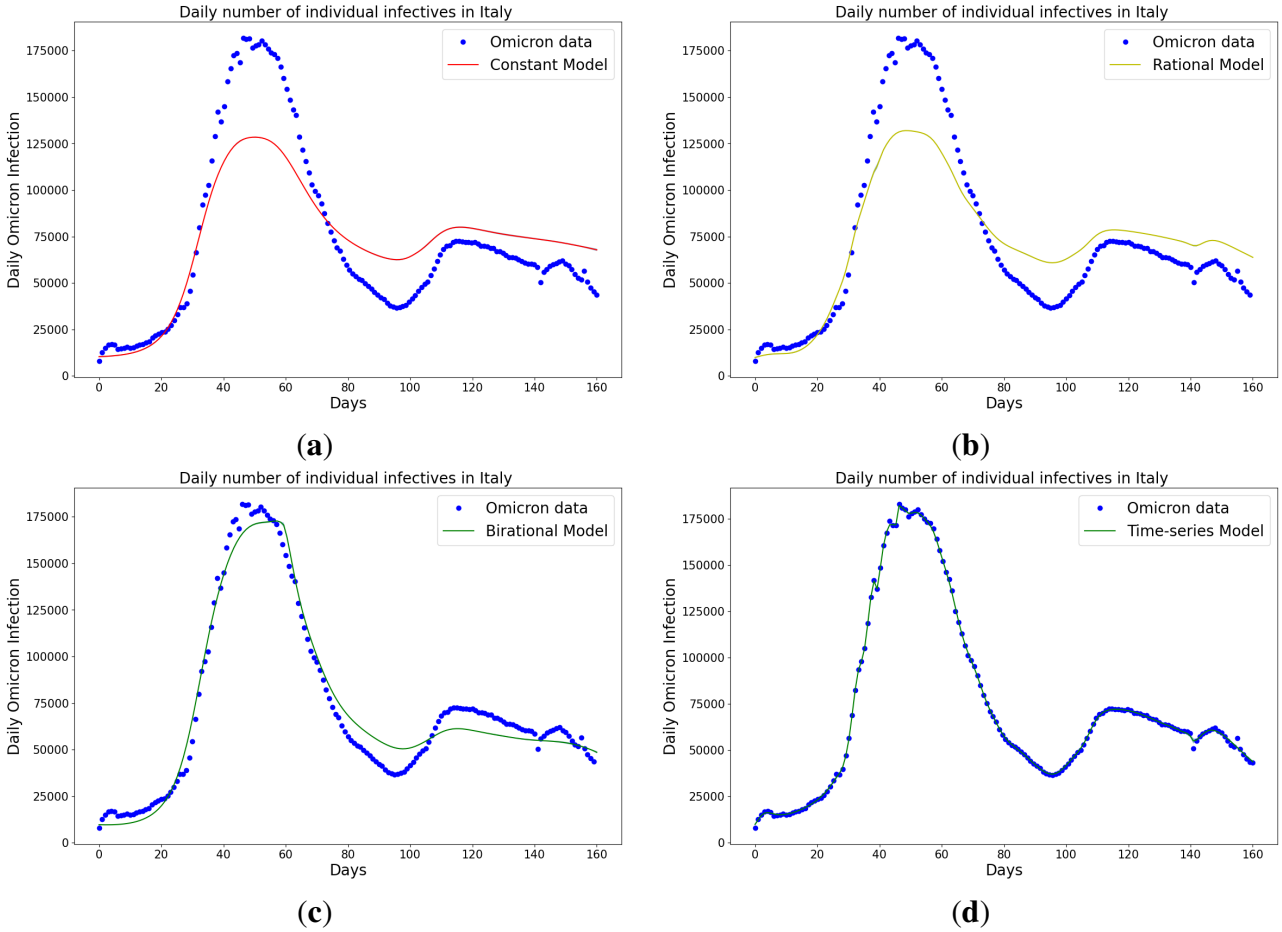


Figure 2.4: Simulation results of Italy daily Omicron data from 30th of November 2021 to 8th of May 2022: **(a)** The graph of daily Omicron infective data and the learned infectives using the constant model by the LINN algorithm 1; **(b)** The graph of daily Omicron infective data and the learned infectives using the rational model by the LINN algorithm 2; **(c)** The graph of daily Omicron infective data and the learned infectives using the birational model by the LINN algorithm 3; **(d)** The graph of daily Omicron infective data and the learned infectives using the time series model by the LINN algorithm 4.

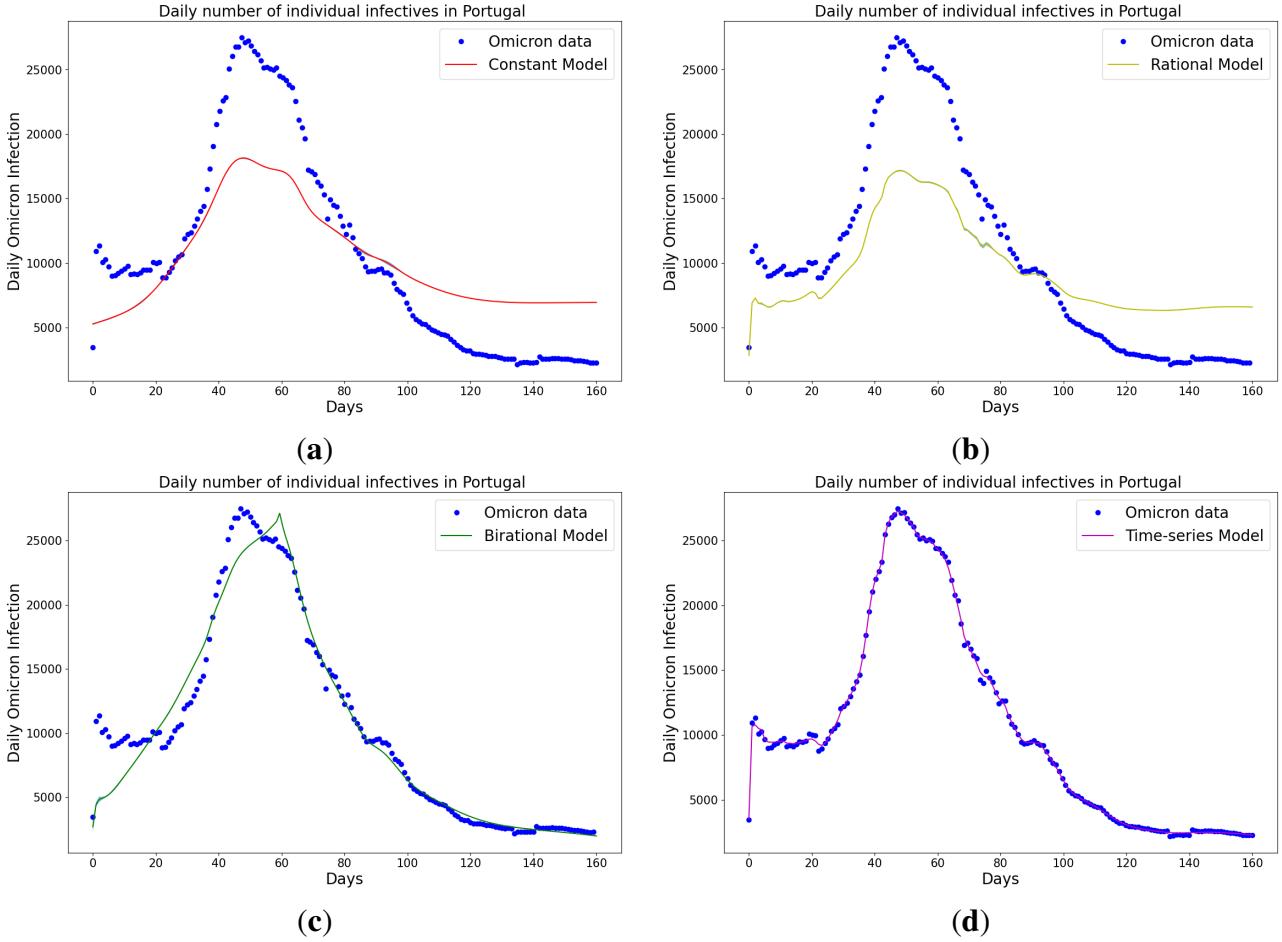


Figure 2.5: Simulation results of Portugal daily Omicron data from 5th of April to 11th of September 2022: **(a)** The graph of daily Omicron infectives data and the learned infectives using the constant model by the LINN algorithm 1; **(b)** The graph of daily Omicron infectives data and the learned infectives using the rational model by the LINN algorithm 2; **(c)** The graph of daily Omicron infectives data and the learned infectives using the birational model by the LINN algorithm 3; **(d)** The graph of daily Omicron infectives data and the learned infectives using the time series model by the LINN algorithm 4.

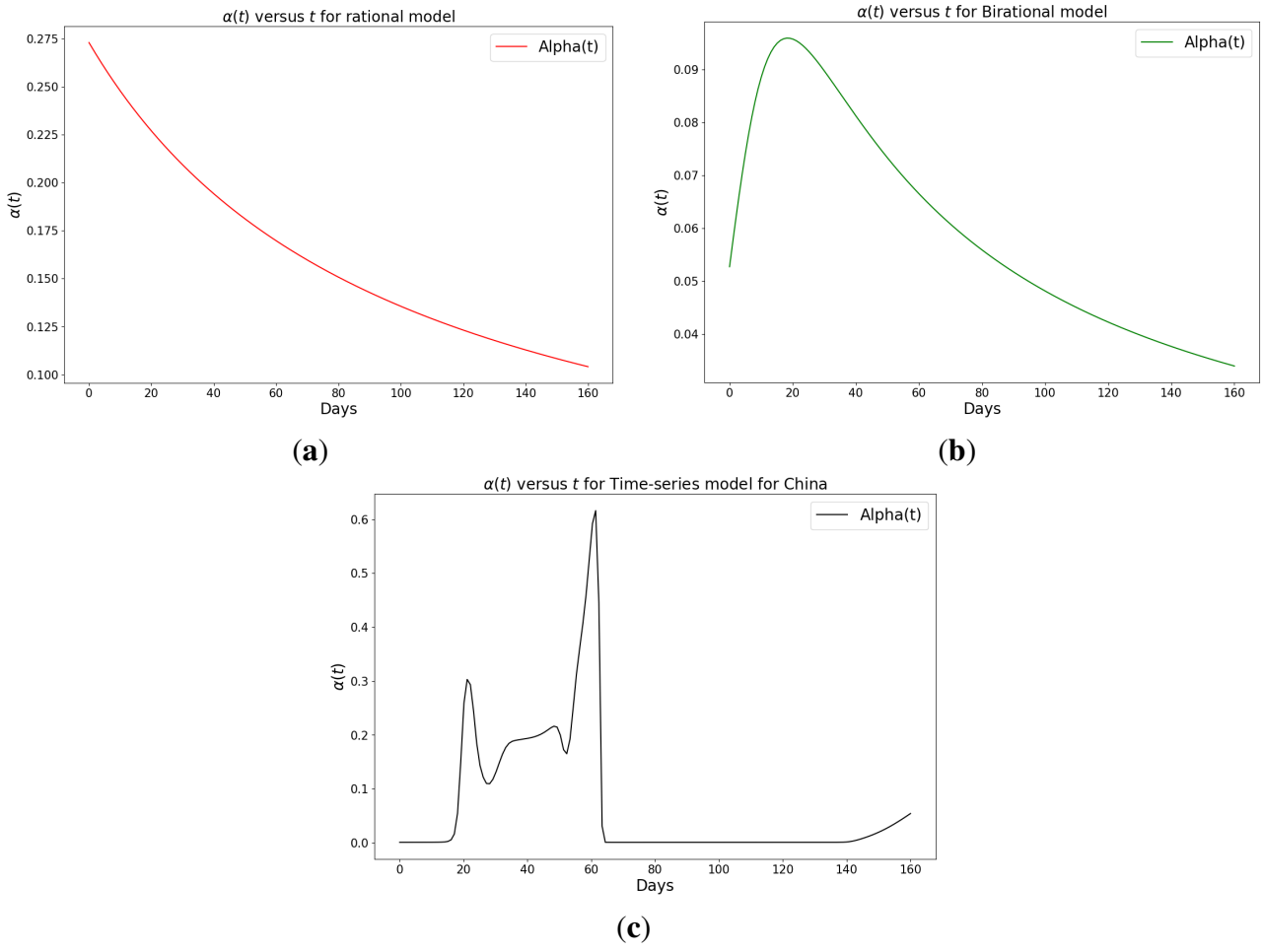


Figure 2.6: Simulation results of the rate of transmission ($\alpha(t)$) for the daily Omicron infection in China using: **(a)** Rational model; **(b)** Birational model; **(c)** Time-series model.

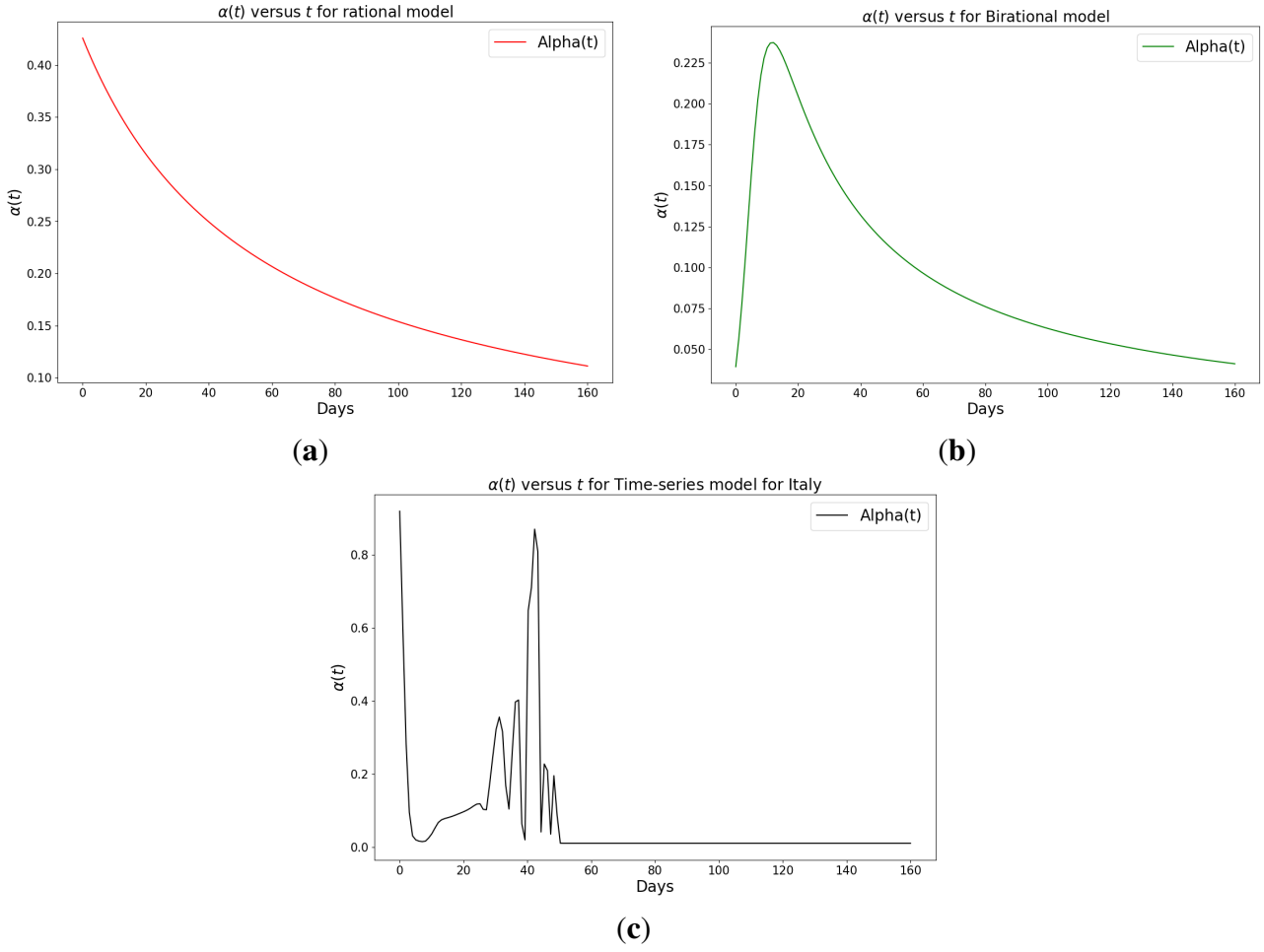


Figure 2.7: Simulation results of the rate of transmission ($\alpha(t)$) for the daily Omicron infection in Italy using: **(a)** Rational model; **(b)** Birational model; **(c)** Time-series model.

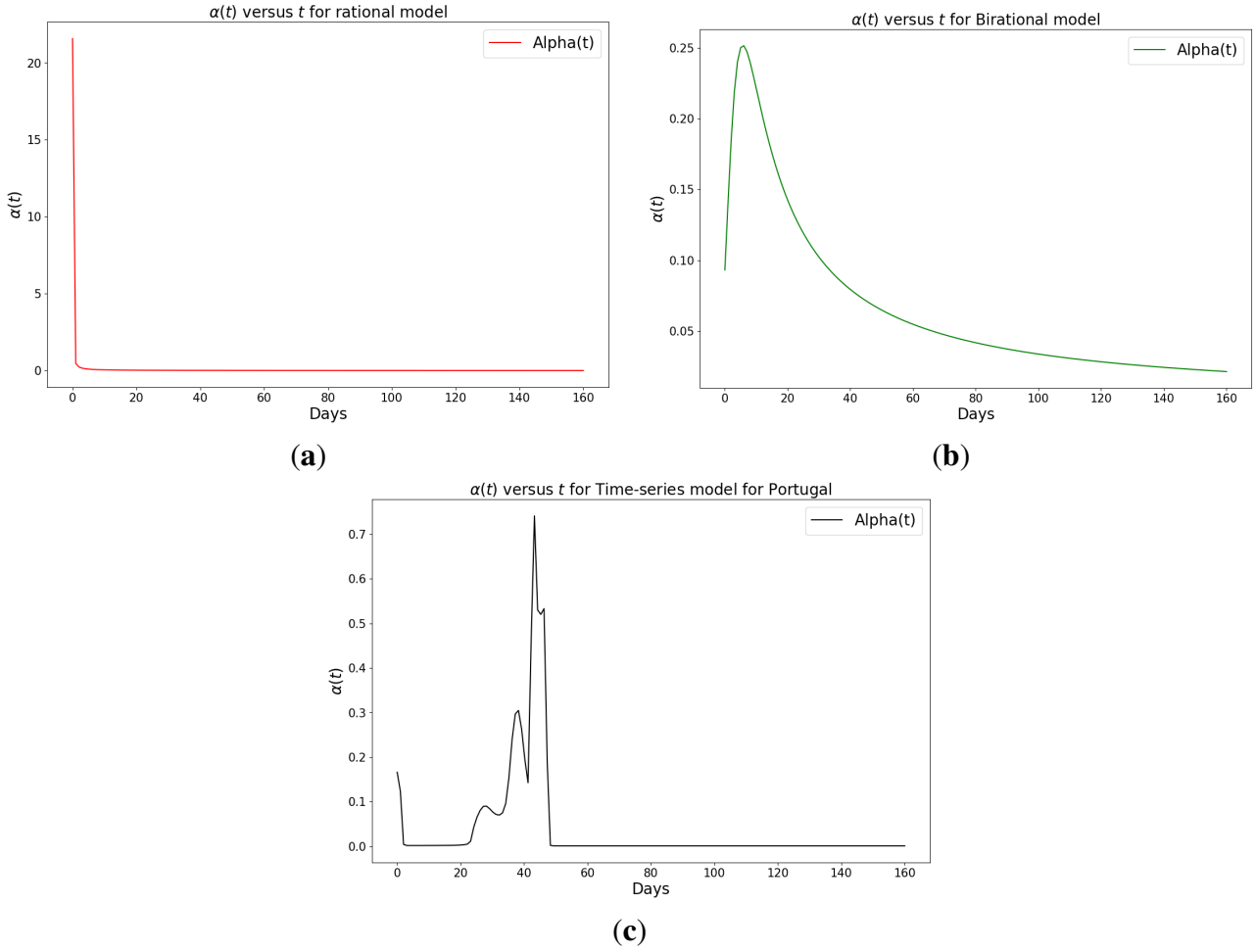


Figure 2.8: Simulation results of the rate of transmission ($\alpha(t)$) for the daily Omicron infection in Portugal using: **(a)** Rational model; **(b)** Birational model; **(c)** Time-series model.

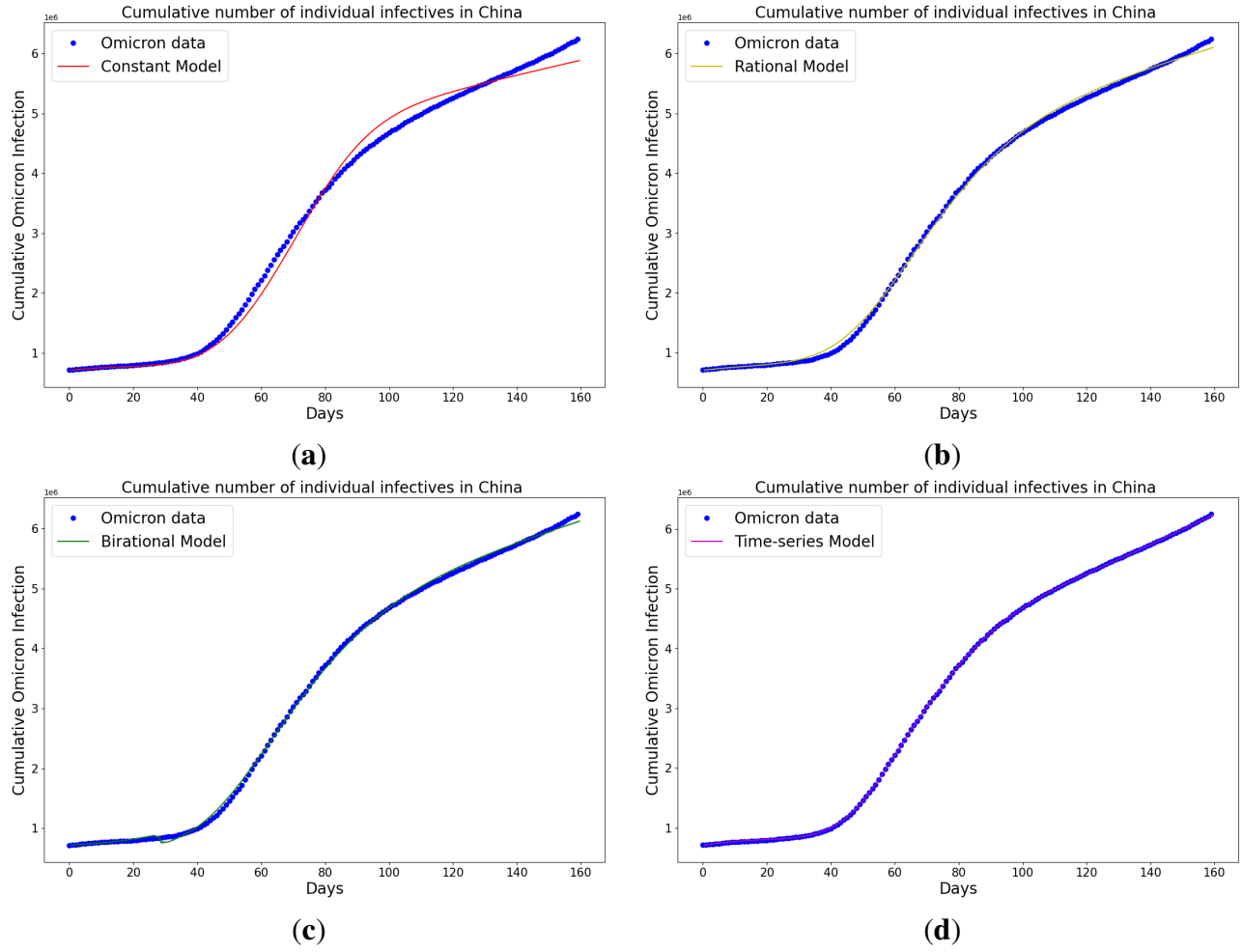


Figure 2.9: Simulation results of China cumulative Omicron data from 25th of March to 31 of August 2022. The graph of the cumulative Omicron infective data and the learned infectives using **(a)** the constant model by the LINN algorithm 1; **(b)** The rational model by the LINN algorithm 2; **(c)** Birational model by the LINN algorithm 3; **(d)** Time series model by the LINN algorithm 4.

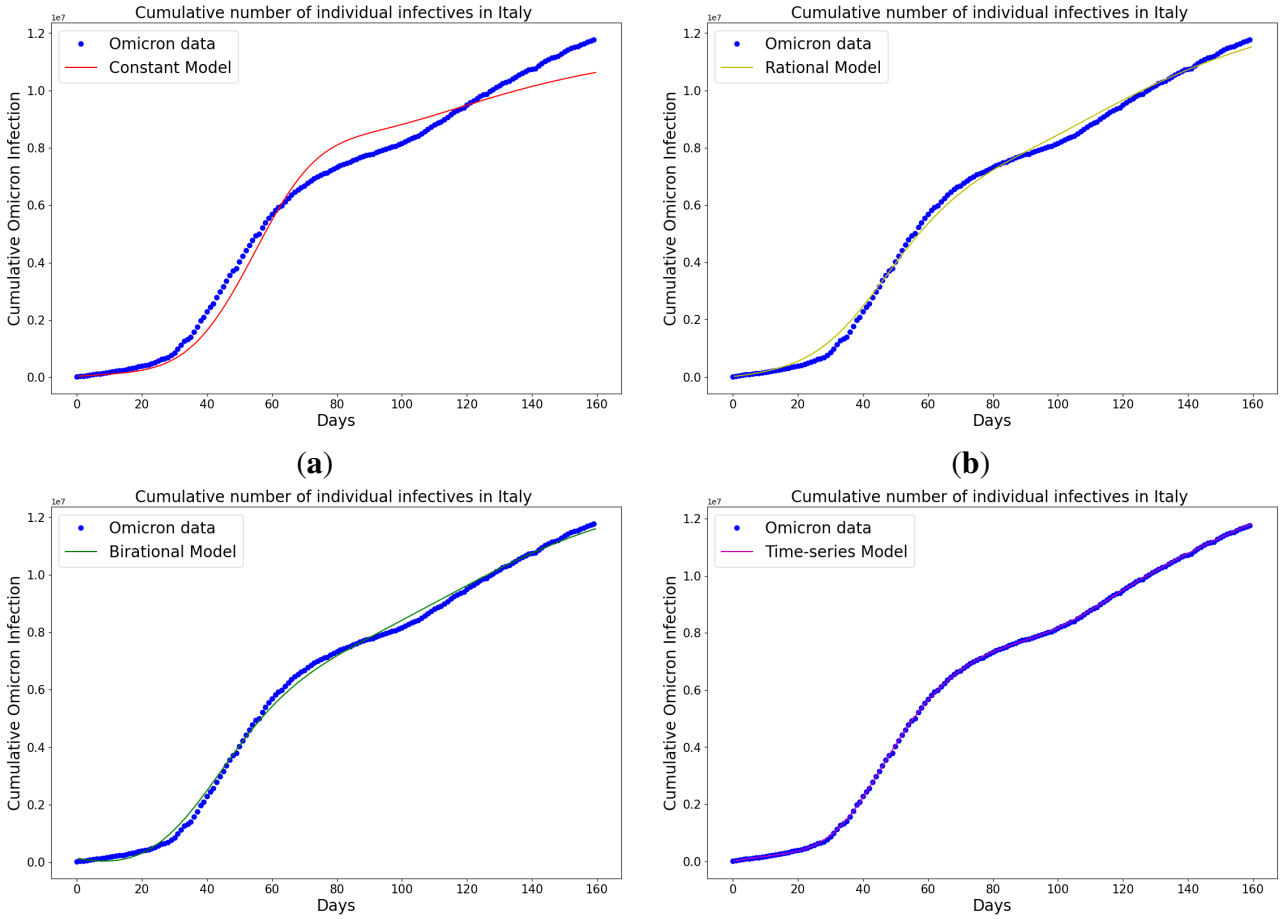


Figure 2.10: Simulation results of Italy cumulative Omicron data from 30th of November 2021 to 8th of May 2022. The graph of the cumulative Omicron infective data and the learned infectives using: **(a)** Constant model; **(b)** Rational model; **(c)** Birational model; **(d)** Time series model.

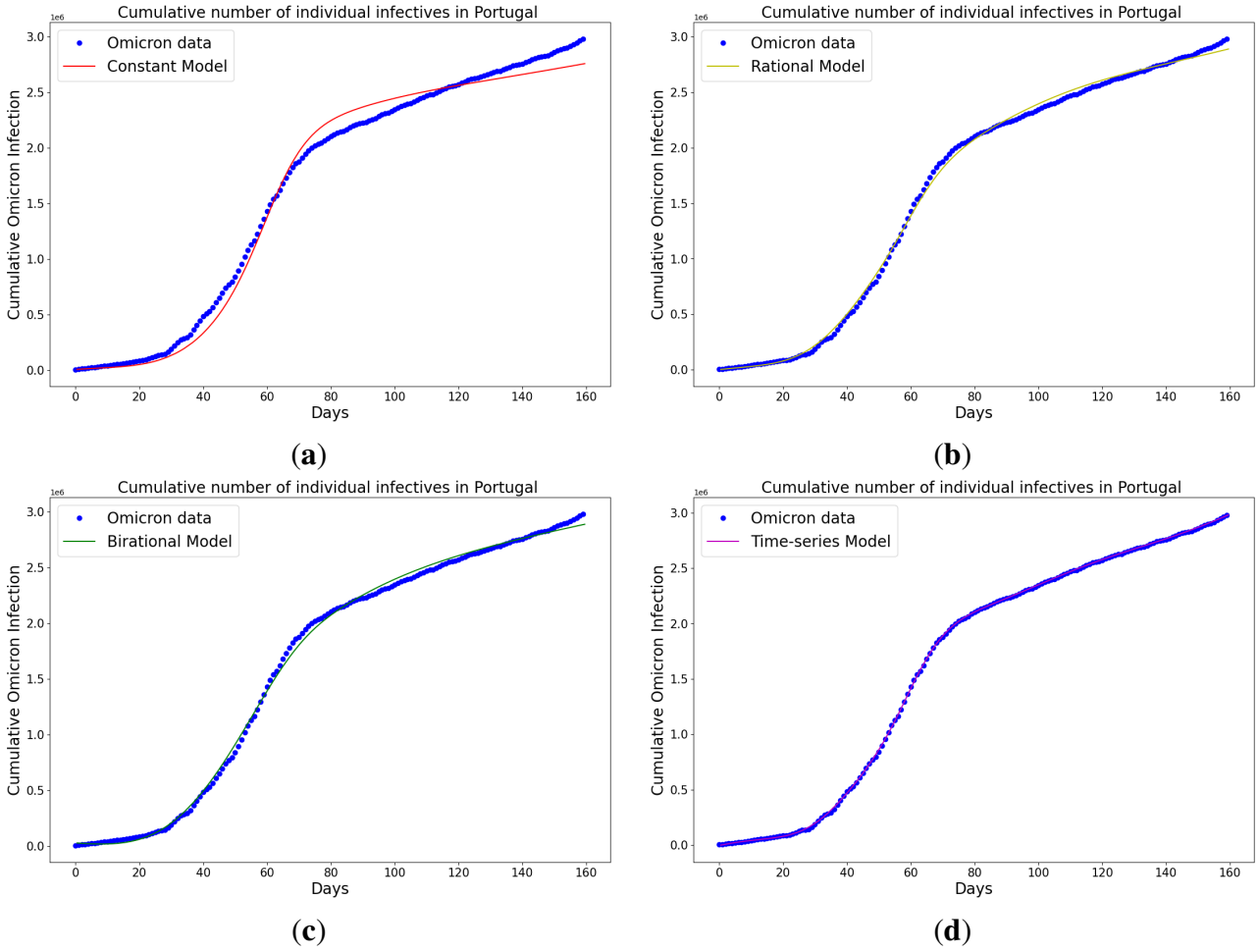


Figure 2.11: Simulation results of Portugal cumulative Omicron data from 5th of April to 11th of September 2022. The graph of the cumulative Omicron infectives data and the learned infectives using: **(a)** Constant model; **(b)** Rational model; **(c)** Birational model; **(d)** Time series model.

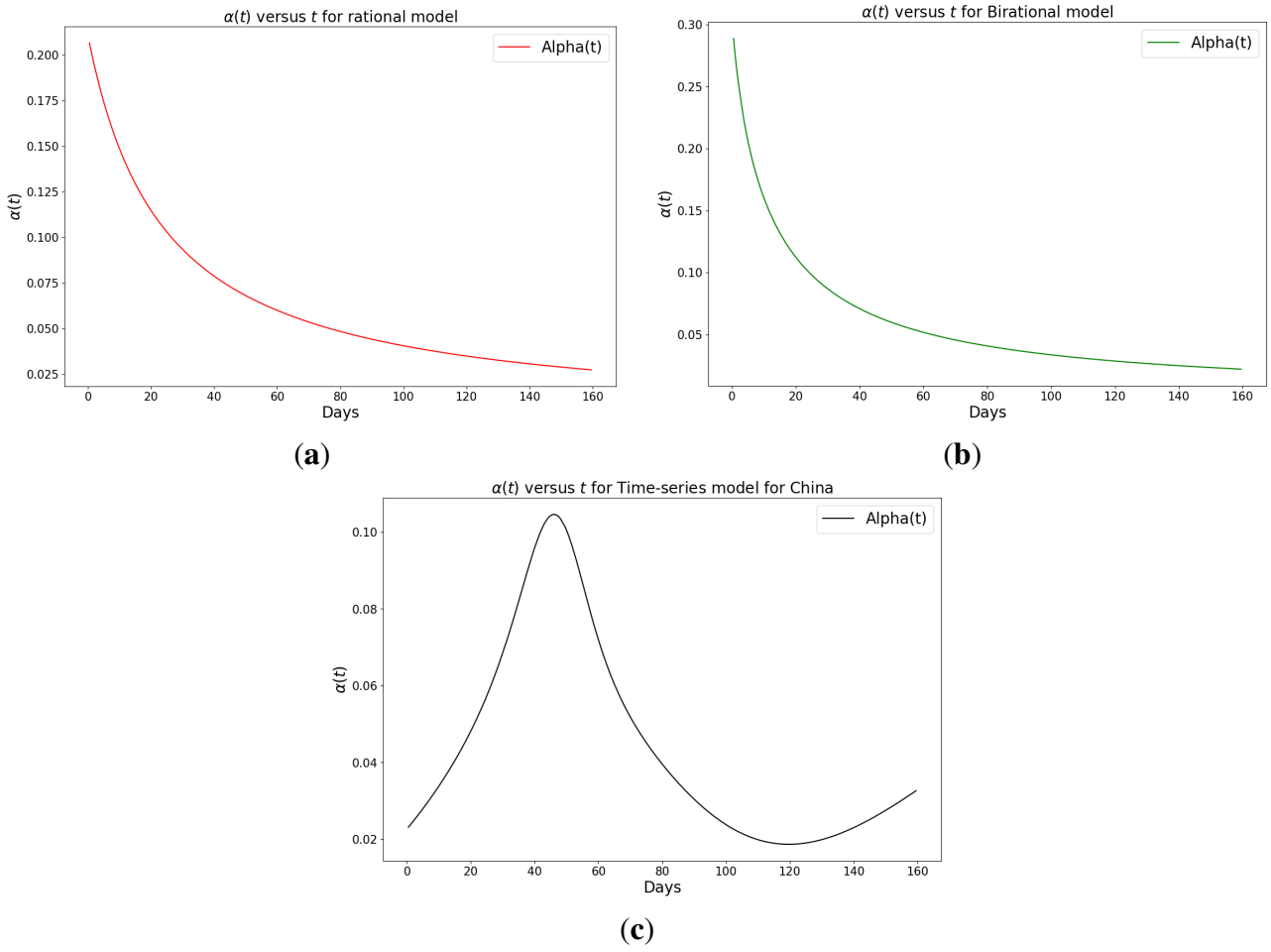


Figure 2.12: Simulation results of the rate of transmission ($\alpha(t)$) for the cumulative Omicron infection in China using: **(a)** Rational model; **(b)** Birational model; **(c)** Time-series model.

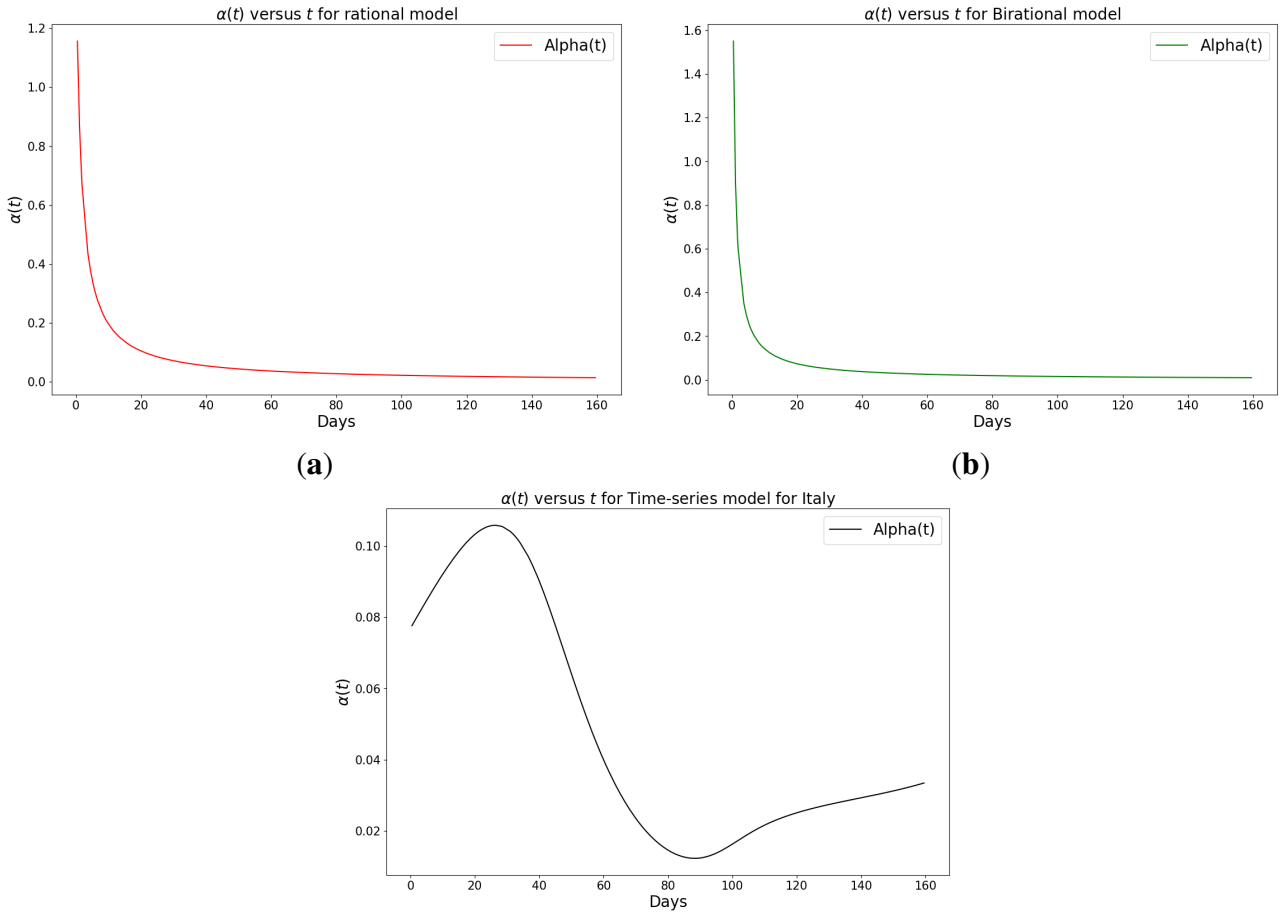


Figure 2.13: Simulation results of the rate of transmission ($\alpha(t)$) for the cumulative Omicron infection in Italy using: **(a)** Rational model; **(b)** Birational model; **(c)** Time-series model.

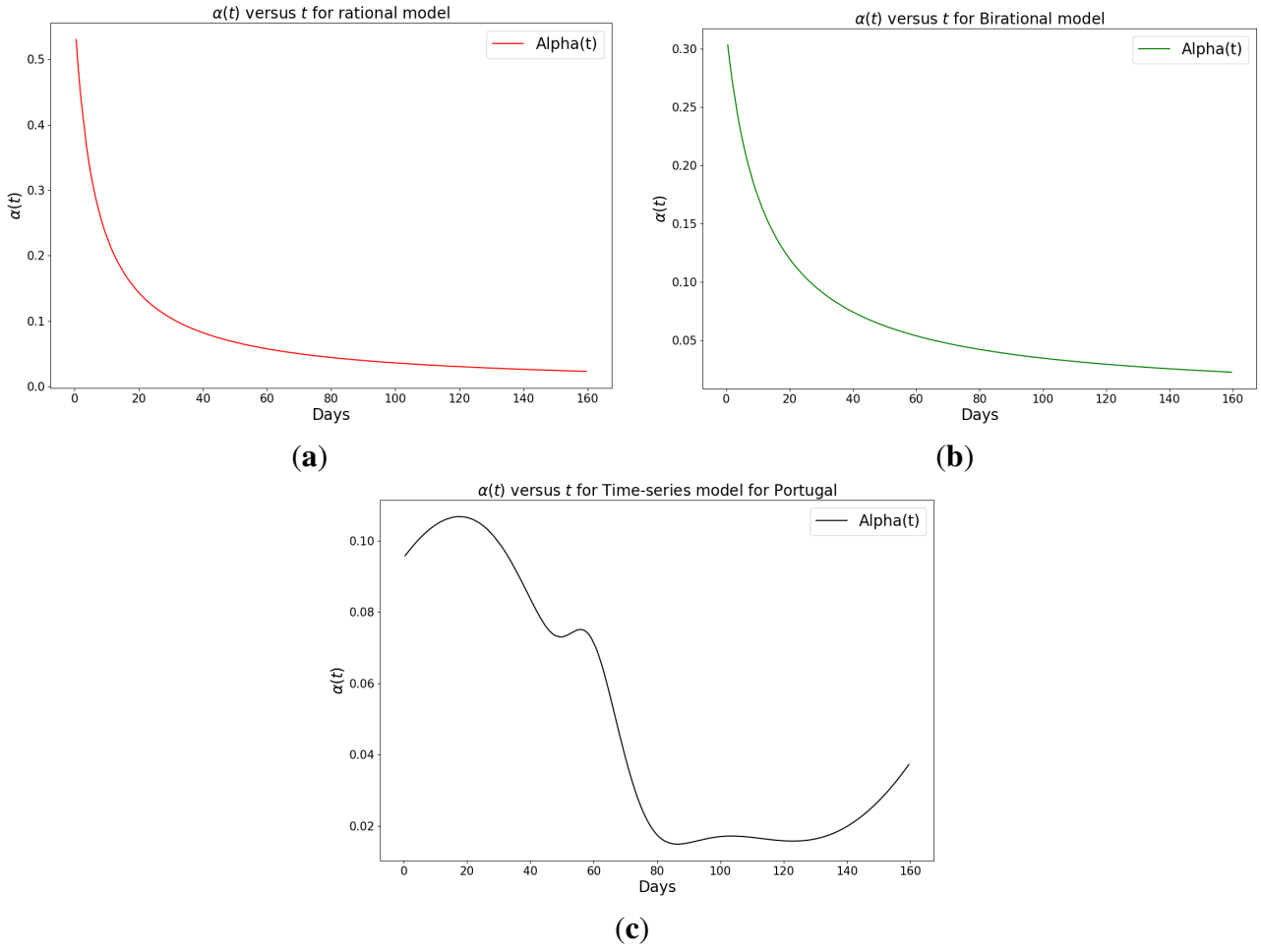


Figure 2.14: Simulation results of the rate of transmission ($\alpha(t)$) for the cumulative Omicron infection in Portugal using: **(a)** Rational model; **(b)** Birational model; **(c)** Time-series model.

2.6.2 Prediction of Daily and the Cumulative Number of Omicron Infections.

As the data fitting of LINN for the mathematical models and the parameters have been obtained, predictions for the daily, the time that a plateau will be reached, and the cumulative number of individuals reported to be infected by the Omicron variant in a given country can be made. The logistic difference equation is a nonlinear ODE with a constant parameter and a time-dependent function. The time-dependent functions result in the constant, rational, and birational formula in which the analytical solutions to the logistic differential equation called constant, rational, and birational models are obtained. The parameters obtained from these three mathematical models are added to the analytical solution to obtain the prediction results.

We could obtain the prediction for the time that a plateau will be reached and the cumulative number of individuals reported to be infected by the Omicron variant in a given country. Still, the daily prediction could not be obtained. However, to obtain both the daily and cumulative prediction results, the initial conditions for all of the compartments, as well as the model parameters, should be known. We obtained the initial values, the constant parameter, and the time-dependent function from the training data using LINN for the time-series model. The learned parameter, together with initial values and the time-dependent functions of the time-series model, was passed into a differential equation solver to predict the daily, the time that a plateau will be reached and the cumulative number of individuals reported to be infected by the Omicron variant in a given country.

Figure 2.15 shows the predicted daily number of individuals infected with the COVID-19 Omicron variant in China, Italy, and Portugal. The 14-day prediction was based on the learned time-series model. The model predicts that on September 14, 2022, the daily number of individuals infected with the COVID-19 Omicron variant in China will be 48053, in Italy will be 16,203, and in Portugal will be 2,301. On September 14, 2022, the record shows that 49311 individuals were infected with the Omicron variant in China, 12,081 in Italy, and 2671 in Portugal. The time-dependent function ($\alpha(t)$) and the constant parameter were both used to achieve this success. The success in fitting the daily data for the COVID-19 Omicron variant enabled this approach with the time-series model. Using the time-series model, the outcomes derived from the constant parame-

ter and the time-dependent function were then integrated into the logistic differential equation to formulate predictions for the subsequent 14 days. We noticed that the infection numbers in China continue to rise, contrasting with Italy, where the numbers are declining, and Portugal where the numbers seem stable. These trends accurately reflect the real-life situations of the COVID-19 Omicron variant infection in these countries

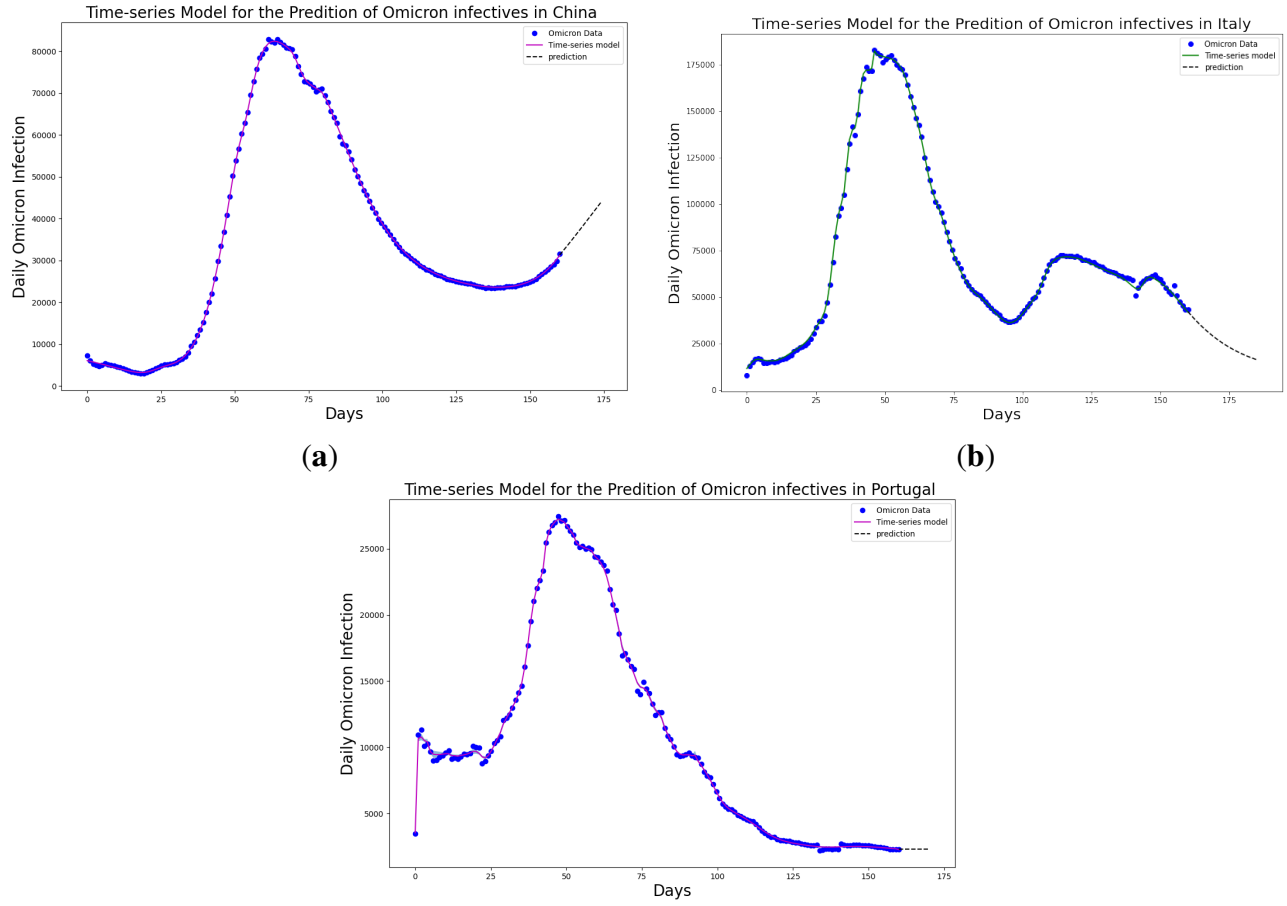


Figure 2.15: The prediction for the 14-day daily number of individuals reported to be infected by COVID-19 Omicron variant using time-series model in: **(a)** China; **(b)** Italy; **(c)** Portugal.

The predictions displayed in Figure 2.16 were made possible by employing the learned parameters of each model, combined with their analytical solutions. Its evident from the figure that the time-series model excels both in fitting the data and in predicting the time when a plateau will be reached and the cumulative number of individuals reported to be infected by the Omicron variant in the given country. The plateau is when the rate of change of the people reported to be infected

is 5% of the maximum infection rate.

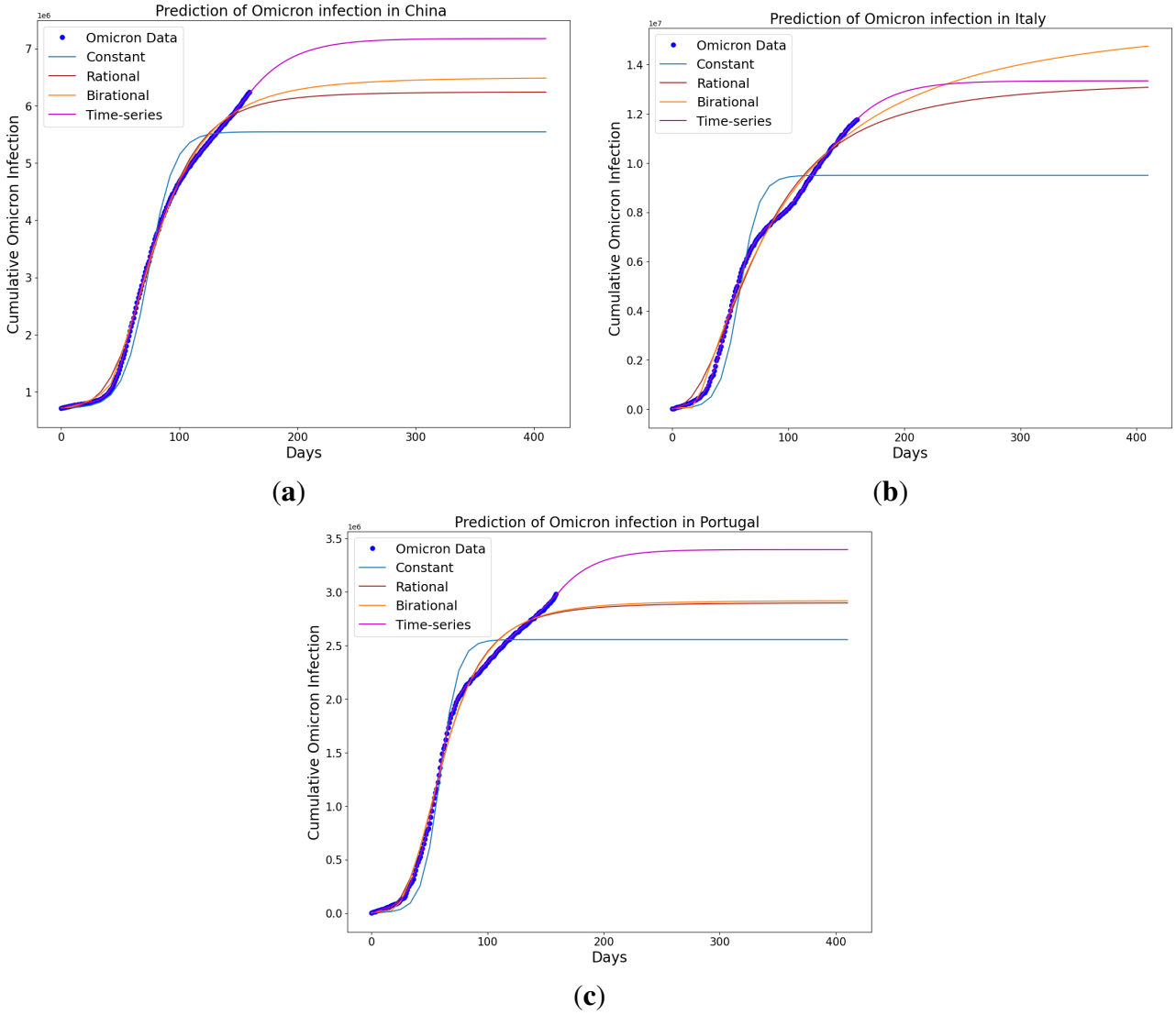


Figure 2.16: The mathematical model prediction for the time that a plateau will be reached as well as the cumulative number of individuals reported to be infected by the Omicron variant in: **(a)** China; **(b)** Italy; **(c)** Portugal.

Additionally, observations revealed a consistent increase in the predicted trends for Italy when using the rational and birational models. These models struggled to perform accurately due to the implementation of only partial mitigation measures in Italy at that time. However, these models demonstrated better accuracy in predicting the time when a plateau will be reached and the cumulative number of individuals reported to be infected by the Omicron variant in China and Portugal, which is attributed to the strict mitigation measures enforced in these countries. Lastly,

the constant model tended to underestimate the predictions, failing to account for the long series of existing data on the epidemics in the given country.

The constant model in Table 2.4 predicted that the Omicron outbreak in China would plateau on June 24, 2022 (122 days), with 5,488,350 cumulative cases of infected people. The rational model predicted that the Omicron outbreak in China would plateau on December 31, 2022 (282 days after March 24, 2022), with 6,2211,15 cumulative cases of people to be infected. The birational model predicted that the Omicron outbreak in China would plateau on March 4, 2022 (317 days after March 24, 2022), with 6,458,279 cumulative cases of people being infected. Finally, the time-series model predicted that the Omicron outbreak in China would plateau on April 18, 2022 (390 days after March 24, 2022), with 7,320,636 cumulative cases of people to be infected. Therefore, the above analysis shows that the Constant model underestimated the actual plateau days and the cumulative number of individuals reported being infected by the COVID-19 Omicron variant in China on August 31, 2022, the last day of acquired data for this study. However, the cumulative number of individuals reported to be infected by the Omicron epidemic in China on the last day of acquired data for this study was 6,243,423.

The constant model in Table 2.5 predicted that the Omicron outbreak in Italy would plateau on March 5, 2022 (96 days), with 9,396,847 cumulative cases of people being infected. The rational model predicted that the Omicron outbreak in Italy would plateau on February 15, 2023 (442 days after November 30, 2021), with 13,121,787 cumulative cases of people being infected. The birational model predicted that the Omicron outbreak in Italy would plateau on September 01, 2023 (641 days after November 30, 2021), with 15,409,748 cumulative cases of people being infected. Finally, the time-series model predicted that the Omicron outbreak in Italy would plateau on December 15, 2022 (381 days after March 24, 2022), with 13,337,516 cumulative cases of people to be infected. Therefore, the above analysis shows that the constant model underestimated the actual plateau days and the cumulative number of individuals reported being infected by the COVID-19 Omicron variant in Italy on August 31, 2022, the last day of acquired data for this study. However, the cumulative number of individuals reported to be infected by the Omicron epidemic

in Italy on the last day of acquired data for this study was 11,772,882.

The constant model in Table 2.6 predicted that the Omicron outbreak in Portugal would plateau on July 2, 2022 (94 days), with 2,526,824 cumulative cases of people being infected. The rational model predicted that the Omicron outbreak in Portugal would plateau on March 2, 2023 (305 days after May 1, 2022), with 2,892,395 cumulative cases of people being infected. The birational model predicted that the Omicron outbreak in Portugal would plateau on March 5, 2023 (308 days after May 1, 2022), with 2,911,888 cumulative cases of people being infected. Finally, the time-series model predicted that the Omicron outbreak in Portugal would plateau on March 8, 2023 (312 days after May 1, 2022), with 3,387,466 cumulative cases of people to be infected. Therefore, the above analysis shows that the Constant model underestimated the actual plateau days and the cumulative number of individuals reported being infected by the COVID-19 Omicron variant in Portugal on August 31, 2022, the last day of acquired data for this study. However, the cumulative number of individuals reported to be infected by the Omicron epidemic in Portugal on the last day of acquired data for this study was 2,980,125.

The Omicron variant's spread analysis reveals interesting patterns when observed through different predictive models. The constant model provides a lower bound, while the time-series model provides an upper bound for China, Italy, and Portugal when the prediction curves of these models are plotted against the actual data. China and Portugal exhibited predictive accuracy across the rational, birational, and time-series models. These models were adept at predicting the plateau, which is when the infection rate stabilizes, and the cumulative number of infected individuals no longer increases significantly. This showcases the robustness of these models in understanding and extrapolating the spread pattern, at least in the contexts of China and Portugal. Furthermore, due to the partial mitigation measures in Italy, we could not obtain an accurate curve for predicting cumulative Omicron infection using the constant, rational, and birational models. However, we obtained an accurate curve using the time-series model because the learned time-series model captured the form of the transmission rate and the information going on Italy's Omicron variant data.

To optimize modeling and predictive efforts related to the spread of infectious diseases like

the Omicron variant, practitioners should prioritize deploying a time-series model that is neural network-based. This model has shown that they are better at adapting to different situations and being accurate, especially when only some mitigation measures are in place. They have done this by getting the best results for different error metrics and fitting well with daily epidemic data. It is important to update and test these models against changing real-world data on a regular basis. This keeps the predictions accurate and up-to-date and considers that infectious diseases are always changing and that infection trends can change quickly. Additionally, a sophisticated understanding and comprehensive incorporation of external and country-specific factors, such as government measures, population movements, and reporting structures, is fundamental to enhancing the reliability and accuracy of the models.

Special attention must be paid to understanding and learning about important parameters like the transmission rate, which is a key factor in figuring out epidemic trends and helping to predict sudden changes in infection patterns accurately. A comprehensive approach involving the comparative use of multiple models provides nuanced insights and a more reliable understanding of infection spread patterns and potential outcomes. The constant model, in spite of its underestimating tendency, offers a valuable baseline for effective risk assessment and resource allocation. For models to be more reliable in different situations, they need to be carefully integrated with different types of mitigation measures and external factors, and they need to be constantly adapted to each region's specific situations and characteristics. The insights obtained from such refined models should subsequently guide proactive and effective intervention strategies and resource allocation, enabling a more informed and enlightened response to epidemics. In conclusion, a structured, iterative, and multifaceted approach, leveraging the strengths of time-series and constant models while incorporating critical parameters and contextual nuances, will significantly elevate the efficacy of predictive modeling in the spread of infectious diseases.

2.6.3 Error Metrics of the Neural Network Training

The neural network training and validation performance is demonstrated in Figures 2.17 and 2.18, where the random splits [8] have been used to generate training and validation for the cumulative infected Omicron dataset in China. Figures 2.17 and 2.18 present the training and testing of MSE and RMSE at different epochs.

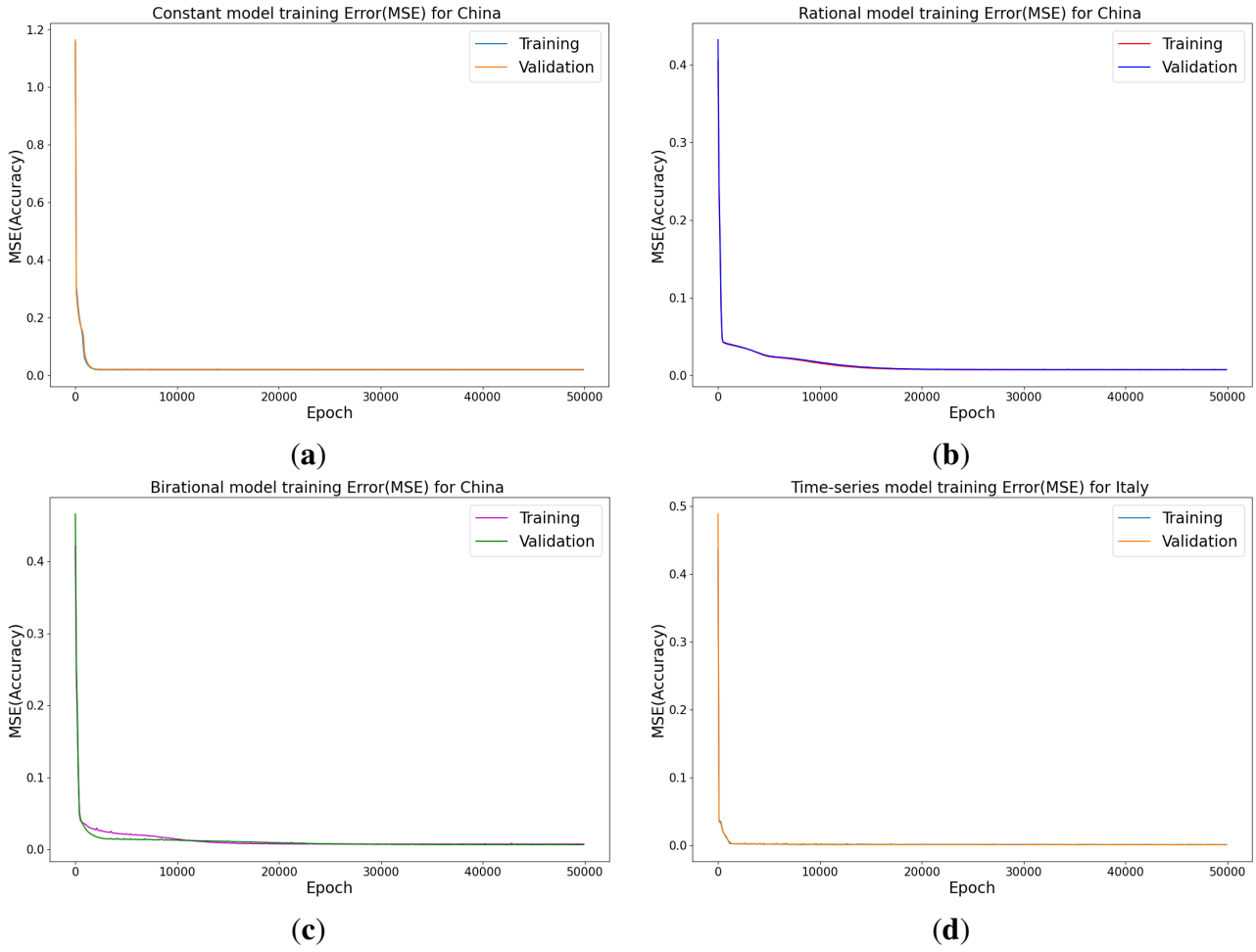


Figure 2.17: Error metrics for the infected cases using the random splits for China COVID-19 Omicron data, where we use 30% of the dataset for testing. Training and testing errors in LINN for nonlinear time-varying transmission rates. MSE at different epochs, using four hidden layers, learning rate 0.001 and 64 neurons per layer in (a) Constant model; (b) Rational model; (c) Birational model; (d) Time-series model.

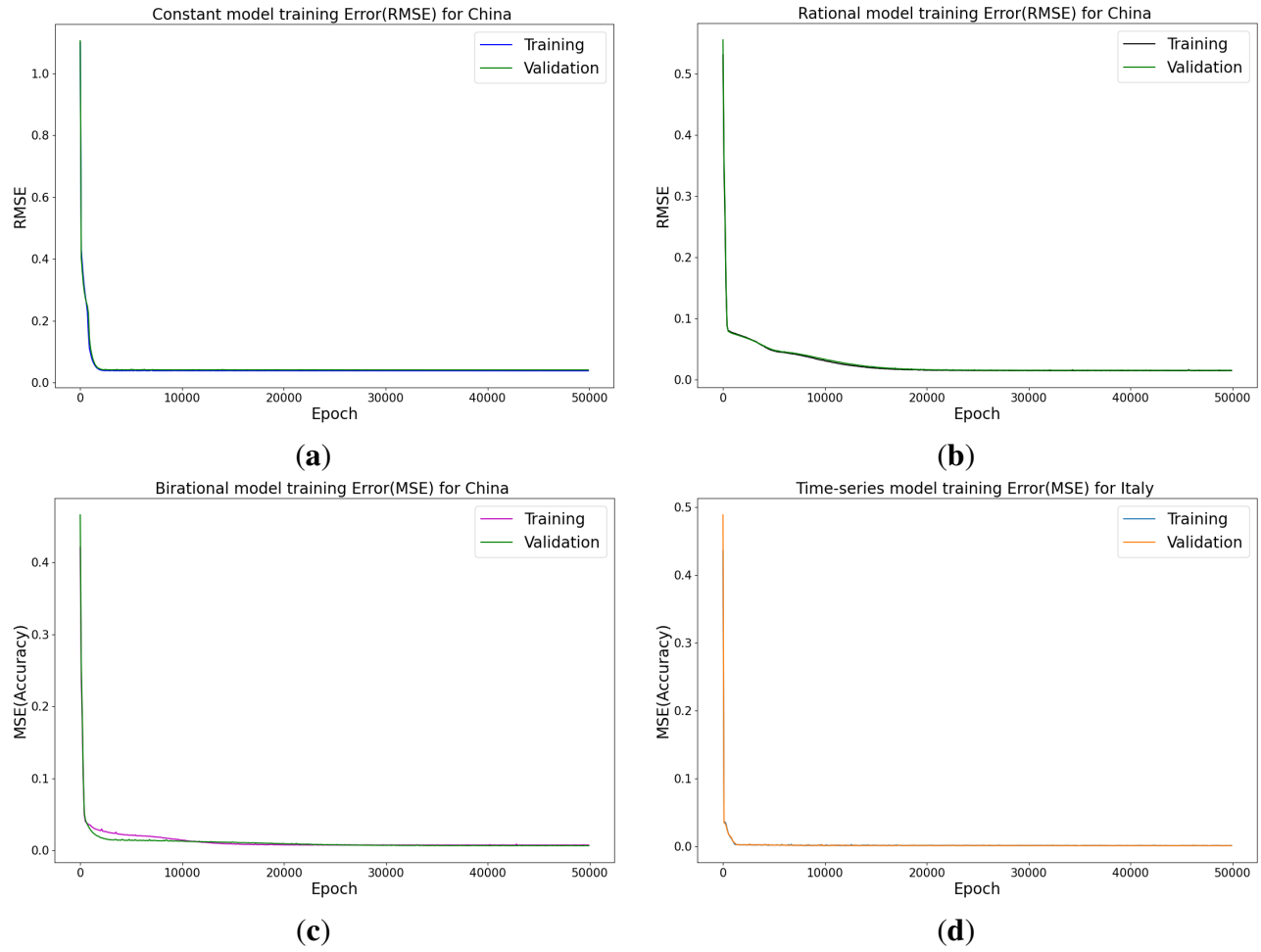


Figure 2.18: RMSE at different epochs, using three hidden layers, learning rate 0.001 and 64 neurons per layer in: (a) Constant model; (b) Rational model; (c) Birational model; (d) Time-series model.

2.7 Conclusions

We have shown a data-driven, deep-learning algorithm based on a logistically informed neural network that uses daily and cumulative infective data in a logistic model to find patterns in the transmission rate. This logistics-informed neural network was made to learn the time-varying transmission rate parameters of the constant, rational, and birational models and the time-series model. The algorithm can be adapted to most logistical models. Using the learned constant parameters of the mathematical models, the analytical solution was used to predict the daily and cumulative number of individuals reported to be infected by the COVID-19 Omicron variant and plateau characteristics. However, the rational and birational models couldn't accurately predict the

daily and cumulative number of individuals reported to be infected with the COVID-19 Omicron variant in a given country with partial mitigation measured. Instead, the time-series model introduced could predict the time that a plateau will be reached as well as the daily and cumulative number of individuals infected with the COVID-19 Omicron variant in the country, observing partial mitigation measures as shown in Figures 2.15 and 2.16. The error metrics in the simulations of each model were also computed and compared. Tables 2.1-2.3, Figures 2.3d, 2.4d, and 2.5d of the daily COVID-19 Omicron variant, and Tables 2.4-2.6 and Figures 2.9d, 2.10d, and 2.11d of the cumulative COVID-19 Omicron variant have shown that the time-series model performs better in accurately fitting the data. Furthermore, Figures 2.6d2.8d and 2.12d2.14d show how the time-series model and the logistic-informed neural network could learn the dynamics of time-dependent functions from the data. In addition, the models and the proposed model demonstrated how the dependence on time reflects various time-dependent elements, including the impact of public actions on the rate of transmission of the COVID-19 Omicron variant and various mitigation measures. Finally, the results and metrics error have shown that the time-series model performs better in fitting and prediction than other mathematical models.

The outstanding achievement in this research is that we introduced a time-series model that learns the form of the time-dependent function of the logistic differential equation from the COVID-19 Omicron variant infection data, which also provides an accurate prediction. Since the logistic-informed neural network algorithm and deep learning have been shown to be good at figuring out transmission rate patterns and predicting infection trends, future research can look into adding other analytical methods, machine learning, and statistical methods to improve the accuracy of predictions and the stability of the model. Employing ensemble learning techniques, where multiple models are integrated, could provide more reliable and diversified insights, compensating for individual model limitations. Bayesian methods and probabilistic models can be explored to incorporate the uncertainty inherently present in epidemiological data, allowing for more nuanced predictions. The methodology can be expanded and refined in several ways for future work. The logistic-informed neural network, along with the time-series model, could be applied to other infec-

tious diseases to understand their transmission dynamics and predict their spread. It would also be beneficial to incorporate more diverse datasets, including different demographic, socioeconomic, and geographic variables, to make the model more generalized and applicable across varied contexts. The model could be refined by integrating real-time data and continually updating the model parameters to account for the evolving nature of infectious diseases and their transmission patterns. Also, exploring the impact of different public health interventions and government policies on the transmission rate could provide valuable insights for designing effective strategies to control the spread of infectious diseases. Additionally, the developed models can be validated against more extensive and diverse datasets, and their performance can be compared with other state-of-the-art models. Furthermore, integrating multidisciplinary knowledge, from epidemiology to social science, can enrich the models contextual understanding and improve its predictive capabilities. Finally, the focus can also be on developing user-friendly tools and applications based on this methodology, which can aid policymakers, researchers, and healthcare professionals in making informed decisions and implementing effective interventions to control the spread of infectious diseases.

CHAPTER 3

Learning the Time-varying Parameters of Dynamical Systems

3.1 Introduction

Artificial neural networks have changed many fields by giving scientists a strong way to model complex phenomena. They are also gaining great utility in solving complex scientific problems. People are still trying to find faster and more accurate ways to simulate dynamic systems. This section investigates the adaptive potential of physics-informed neural networks, a specialized subset of artificial neural networks, in modeling complex dynamical systems with enhanced speed and accuracy. These networks incorporate known physics laws into the learning process, ensuring predictions remain consistent with basic principles essential in continuously handling scientific phenomena. This study focuses on optimizing the application of this specialized network for simultaneous system dynamics simulations and learning time-varying parameters, especially when the number of unknown elements in the system matches the number of undetermined parameters.

Furthermore, we investigate inequality conditions between parameters and equations, optimizing network architecture to enhance convergence speed, computational efficiency, and accuracy in learning the time-varying parameter. Our approach enhances the algorithm's efficiency and accuracy, ensuring optimal use of computational resources and yielding good results. Extensive experiments are conducted on four different dynamic systems: the first irreversible chain, the biomass transfer, the Brusselsator model, and the Lotka-Volterra model, using synthetically generated data to validate our approach using synthetically generated data to validate our approach. In addition, we apply our method to susceptibility models, using real-world COVID-19 data to understand the time-varying parameters of epidemic spread.

A detailed comparison between the performance of our method and fully connected deep neural networks is presented, testing the accuracy and computational efficiency in parameter identification and system dynamics capture. The results show that Optimized physics-informed neural networks

perform better than fully connected deep neural networks, especially as network depth increases, making it ideal for modeling complex real-time systems. This highlights the effectiveness of Optimized physics-informed neural networks in scientific modeling under conditions of equilibrium of unknown parameters. Furthermore, it provides a fast, accurate, and efficient alternative for analyzing dynamic systems.

3.2 Systems of Ordinary Differential Equation

Consider a dynamical system characterized by n ordinary differential equations encompassing m undetermined parameters (see also, [56]).

$$\frac{d\theta(t)}{dt} = \Theta(t, \theta(t), \delta(t)), \quad t \in (t_0, t_{end}), \tag{3.1}$$

$$\theta(t_0) = \theta_0$$

These equations make up the mathematical model that describes the system's behavior being studied, with m parameters that define the system's characteristics. In this context, $\theta(t) = [\theta_1(t), \theta_2(t), \theta_3(t), \ldots, \theta_n(t)]^T$ is a vector field composed of n components. Further, let $\delta(t) = [\delta_1(t), \delta_2(t), \ldots, \delta_m(t)] \in \mathbb{R}^m$ be the vector encompassing the unknown parameters, and $\theta_0 \in \mathbb{R}^n$ be the initial condition. On the right-hand side of (3.1), we have the vector-valued function

$$\Theta(t, \theta(t), \delta(t)) = \begin{bmatrix} \Theta_1(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t)) \\ \Theta_2(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t)) \\ \vdots \\ \Theta_n(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t)) \end{bmatrix}$$

not necessarily linear, consisting of n components designated as

$$\Theta_k(t, \theta_1(t), \theta_2(t), \ldots, \theta_n(t), \delta_1(t), \delta_2(t), \ldots, \delta_m(t))$$

with $k = 1, \ldots, n$. The data for the model equation will be represented by $\theta^*(t)$.

Using the data measured at P distinct time points for the given model, $(t_i, \theta^*(t_i))$ for $i = 1, 2, \ldots, P$, our goal is to develop Optimized Optimized physics-informed neural network (OPINN) algorithm that estimates both the unidentified parameters $\delta(t)$ and the solution $\theta(t)$ across all time points t. Understanding the value and change of these parameters over time can give useful information about the system's dynamic behavior, which can be used to improve predictive models and solutions.

3.2.1 Parameter Identification of Dynamical Systems Model Using OPINN

Let us consider a dynamical system (3.1). The goal is to use the OPINN approach to solve and identify the parameters of the dynamical system model. This can be conducted by finding the best network parameters ω, representing the biases and weights network that minimizes the loss function. We offer Optimized physics-informed neural network shown in Figure 3.1, with two networks to do this. The first network outputs are $\theta(t_n; \omega)$, which admits t as the input data. The second network learns the data's parameters $\delta(t_n; \omega)$. The following loss function was minimized.

$$\zeta(\omega; \kappa) = \zeta_t(\omega; \kappa_t) + \zeta_r(\omega; \kappa_r) \tag{3.2}$$

$\zeta_t(\omega; \kappa_t)$ is called the training loss and $\zeta_r(\omega; \kappa_r)$ is called the residual loss. $\kappa = \kappa_r \cup \kappa_t$ in the total training datasets. We define the following:

$$\zeta_r(\omega; \kappa_r) = \frac{1}{\kappa_r} \sum_{i=1}^{p} \sum_{n=1}^{\kappa_r} |r_i(t_n)|^2 \tag{3.3}$$

where

$$r_i(t_n) = \frac{d\theta(t_n;\omega)}{dt} - \Theta\Big(t_n, \theta(t_n;\omega), \delta(t_n;\omega)\Big) \tag{3.4}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t} ||\theta(t_n;\omega) - \theta(t_n)||^2\right). \tag{3.5}$$

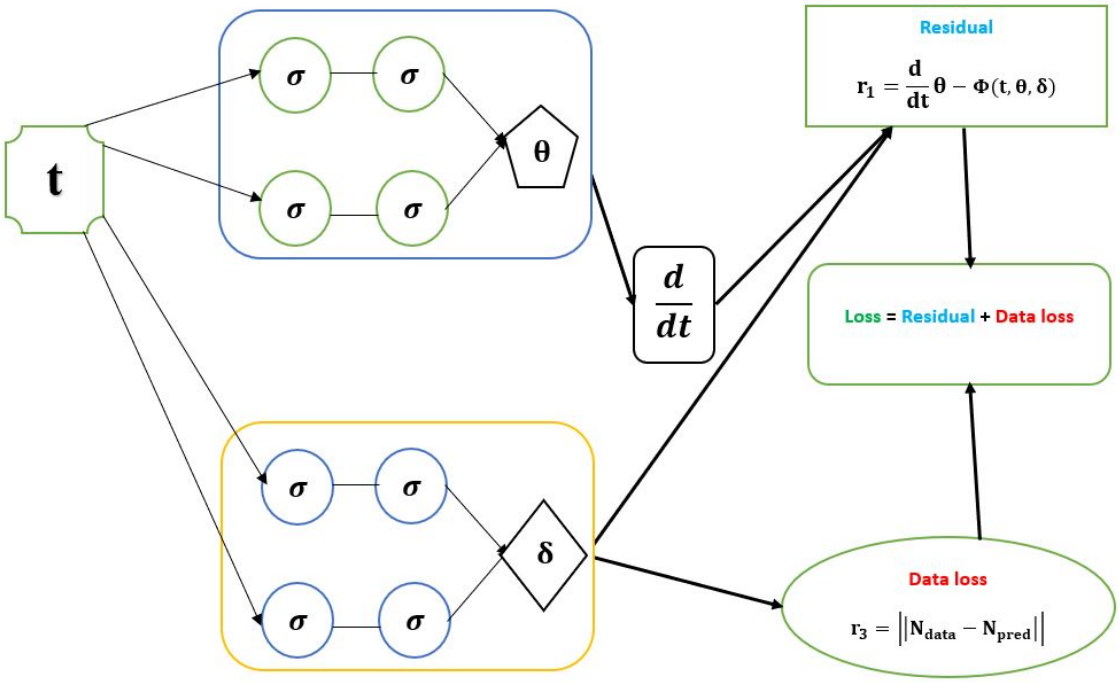


Figure 3.1: Schematic diagram of the OPINN with the parameters of a dynamical system of ODE model.

The network is trained to minimize (3.3); an optimizer such as Adams is the most optimal weight and bias to minimize (3.3). The neural network also obtains the optimal values for the model parameters. The $\theta(t_n)$ represents the loss function's training data. We identify the parameter δ by solving Equation (3.5). When trying to address the problem (3.5), we can categorize it into two possible scenarios:

Scenario 1: In this particular instance, we assume the right-hand side function of a dynamical

system (3.1) has an equal number of parameters and equations. This scenario shows that a solution exists. In other words, there is an exact solution [56].

Scenario 2: In this context, the number of parameters does not match the number of dynamical system equations [56].

```
import numpy as np
import pandas as pd
import timeit
import time
import matplotlib.pyplot as plt
import scipy.io
import tensorflow as tf


class PINN:
    def __init__(self, ...):  # Initialization with parameters and network setup
        # Network architecture setup
        self.weights, self.biases = self.initialize_NN(layers)
        ...
        self.sess = tf.Session(...)  # TensorFlow session

    def initialize_NN(self, layers):  # Initialize weights and biases
        ...

    def neural_net(self, ...): # Building the network for the input and the output solution
        num_layers = len(layers1)
        ...

    def neural_net1(self, ...): # Building the network for learning the time-varing parameter
        ...

    def loss(self, ...): # Loss function
        ...

    def train(self, nIter):  # Training process
        for it in range(nIter):
            ...

    def predict(self, t_star):  # Prediction method
        ...

# Usage
model = PINN(...)
model.train(nIter)  # Training the model
z1_pred, z2_pred, ... = model.predict(t_data)  # Making predictions
```

Figure 3.2: PINN source code.

The implementation of Optimized physics-informed neural networks (OPINNs) is carried out using Python. Its simplicity and extensive library ecosystem make it ideal for developing sophisticated models like OPINNs [43]. Key to our implementation is TensorFlow, which offers robust

tools for building and training neural network models. TensorFlow's compatibility with Python and its efficient handling of large-scale numerical computations enable the effective construction of OPINN architectures. Additionally, we employ NumPy, a Python library that supports large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these arrays. This is crucial for handling the numerical aspects inherent in OPINN development. We also utilize Pandas, a library for data manipulation and analysis. Pandas provide structures and functions ideal for modifying numerical tables and time series, making them useful for preprocessing and organizing the data that neural networks need to learn. The synergy of these libraries in Python creates a powerful toolkit for developing and implementing OPINNs.

The source code in Figure 3.2 provides a comprehensive example of Optimized physics-informed neural network implementation, illustrating the practical application of parameter identification in dynamical systems. This Python-based code features a custom OPINN class, encompassing the complete workflow of constructing, training, and deploying the neural network for predictive analysis. At the core of this class is the initialization method, which is responsible for configuring the network's layers, weights, and biases. This setup is facilitated by the initialize_NN function, leveraging TensorFlows robust computational power for efficient network architecture development. The functions neural_net and neural_net1 each articulate distinct network structures tailored for specific model segments. The train and loss functions represent the heart of the learning process, where TensorFlows optimization algorithms are utilized to refine the network by minimizing the loss function. This method effectively enables the network to learn the intricacies of the dynamical system under study. Finally, the predict function demonstrates the practical use of the trained network in generating predictions for new datasets. This implementation not only showcases the coding aspects of OPINNs but also exemplifies the fusion of theoretical principles with their practical application in advanced computational modeling.

3.3 Computational Simulations of Dynamical Systems

This section explores Optimized physics-informed neural networks (OPINNs) for parameter identification and solution in various dynamical systems. Parameter identification is crucial, especially when working with ordinary differential equations (ODEs), as each model is essentially unique. ODEs showcase their practicality when the parameters are correctly determined, emphasizing this methodology's significance. The identification of parameters becomes evident as we develop a model employing ordinary differential equations (ODEs) to elucidate the fundamental occurrences. These parameters include different characteristics of the modeled system, such as rate constants, initial conditions, and coefficients, defining the form, rate of change, and stability of the solution. The parameters that make up an ODE can also determine the computational techniques used to solve it and have a big impact on the quality of the solution. Consequently, parameter identification is crucial for making insightful predictions and adequately assessing the simulation of the modeled dynamical system. In our discussion, we consider five benchmark problems derived from the existing literature. For each example, we will specify the choice of neurons, the number of hidden layers, activation functions, and other network parameters. Two of these five problems have an exact analytical solution, which will be leveraged to validate the model's accuracy. In addition, the relative error metric showing the fitting accuracy of each dynamical system using the OPINN algorithm will be provided.

3.3.1 First-Order Irreversible Chain Reactions

First-order irreversible chain reactions are a type of chemical reaction where a reactant undergoes a series of transformations to form a product, with the process continuing in a chain-like manner. A common example of such a reaction is the decomposition of hydrogen peroxide (H_2O_2) in the presence of an iodide ion as a catalyst. One of the most fundamental studies on chain reactions was conducted by Polanyi and Wigner [19], who published a series of papers in the 1920s and 1930s on the theory of chain reactions. Consider the subsequent first-order irreversible chain reactions [66, 61].

$$\mathscr{A} \xrightarrow{k_1} \mathscr{B} \xrightarrow{k_2} \mathscr{C}$$

The first step is a first-order reaction where species $\mathscr{A}$ transforms into species $\mathscr{B}$ at a rate determined by the rate constant k_1. The second is another first-order reaction where species $\mathscr{B}$ transforms into species $\mathscr{C}$ at a rate determined by the rate constant k_2. In both steps, the reaction rate depends on the concentration of a single reactant (either A or B), which is characteristic of first-order reactions and characterized by the following model:

$$\begin{aligned} \frac{dz_1(t)}{dt} &= -k_1(t)z_1(t) \\ \frac{dz_2(t)}{dt} &= k_1(t)z_1(t) - k_2(t)z_2(t). \end{aligned} \tag{3.6}$$

where z_1 and z_2 symbolize the concentrations of chemical species $\mathscr{A}$ and $\mathscr{B}$, respectively and k_1, k_2 denote the rate constants which are the parameters required to be learned. Given that $k_1 = 5$, $k_2 = 1$ and $z(t = 0) = [1, 0]$, then the observed data z_1 and z_2 are given in Table 3.1. The data was taken from reference [56].

The analytical solution of (3.6) is given as follows:

Table 3.1: Observed values of z_1 and z_2 at different time points.

t	z_1	z_2
0.0	1.000000	0.000000
0.1	0.606531	0.372883
0.2	0.367879	0.563564
0.3	0.223130	0.647110
0.4	0.135335	0.668731
0.5	0.082085	0.655557
0.6	0.049787	0.623781
0.7	0.030197	0.582985
0.8	0.018316	0.538767
0.9	0.011109	0.494326
1.0	0.006738	0.451427

$$
\begin{aligned}
z_1(t) &= e^{-5t} \\
z_2(t) &= \frac{5}{4}e^{-5t}(-1+e^{4t}),
\end{aligned}
\tag{3.7}
$$

and the analytical derivatives of (3.7) are given by:

$$
\begin{aligned}
\frac{dz_1(t)}{dt} &= -5e^{-5t} \\
\frac{dz_2(t)}{dt} &= 5e^{-t} - \frac{25}{4}e^{-5t}(-1+e^{4t}).
\end{aligned}
\tag{3.8}
$$

The OPINNs approach to solve and identify the parameters of the first-order irreversible chain reactions model is performed by finding the best network parameters, ω, which represents the biases and weights network that minimizes the loss function. We offer a Optimized physics-informed neural network (OPINN) shown in Figure 3.3 with three networks to do this. The first network outputs are $z_{1NN}(t_n;\omega)$ and $z_{2NN}(t_n;\omega)$, which admits t as the input data. The second and third networks learn the parameters $k_1(t_n;\Phi)$ and $k_2(t_n;\Phi)$ from the data. The neural network also obtains the optimal values for the model parameters. The $z_1(t_n)$ and $z_2(t_n)$ represent the loss function's training data. Finally, the first-order irreversible chain reaction model parameters' mean value was obtained using OPINN with one layer of 40 neurons, 90,000 epochs, sigmoid activation function used, and 10^{-3} learning rate. The OPINN Algorithm 10 for learning the optimal parameters of the first-order irreversible chain reactions model (3.6) is shown below.

$$
\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{2}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2 \tag{3.9}
$$

where

$$
\begin{aligned}
r_1(t_n) &= \frac{dz_{1NN}(t_n;\omega)}{dt} + k_1(t_n;\Phi)z_{1NN}(t_n;\omega) \\
r_2(t_n) &= \frac{dz_{2NN}(t_n;\omega)}{dt} - \left(k_1(t_n;\Phi)z_{1NN}(t_n;\omega) - k_2(t_n;\Phi)z_{2NN}(t_n;\omega)\right)
\end{aligned}
\tag{3.10}
$$

and

$$
\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t}|z_{1NN}(t_n;\omega)-z_1(t_n)|^2 + \sum_{n=1}^{\kappa_t}|z_{2NN}(t_n;\omega)-z_2(t_n)|^2\right) \tag{3.11}
$$

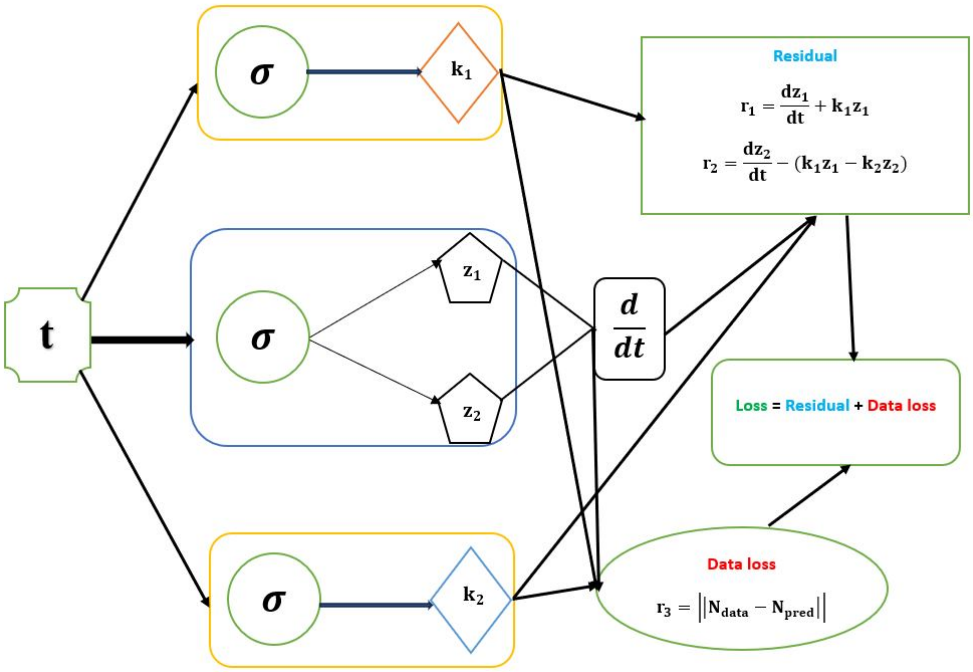


Figure 3.3: Schematic diagram of the OPINN with the parameters of first-order irreversible chain reactions model.

Two distinct scenarios were examined in the study of the first-order irreversible chain reactions model, with results detailed in Figures 3.4–3.7. For Scenario 1, the system was solved using Equation (3.10) at $t^* = 0.28$ and 0.19, while for Scenario 2, it was solved at $t^* = 0.22$ and 0.27. Figure 3.4 shows the obtained solution of the actual output and the predicted output of the first-order irreversible chain reactions model from the data. The phase space plot of the actual output of species $z_1(t)$ against species $z_2(t)$ and the predicted output of species $z_1(t)$ against species $z_2(t)$ is shown in Figure 3.5. This visualization provides insight into the relationships and behaviors of the species involved. Figure 3.6 shows the learned time-varying parameter k_1 and k_2 values of the first-order irreversible chain reaction models, shedding light on how these parameters change over time in the first-order irreversible chain reaction models. The $k_1(t)$ curve appears to have an oscillatory behavior, and the oscillation frequency seems consistent throughout the curve.

Algorithm 5 OPINN algorithm for learning the parameters of the first-order irreversible chain reactions model

1: Construct OPINN

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: ω

Output layer: $z_{1NN}(t_n;\omega)$ and $z_{2NN}(t_n;\omega), n = 1, \ldots, X$

2: Construct neural network: $k_1(t)$

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: Φ

Output layer: $k_1(t_n;\Phi), n = 1, \ldots, X$

3: Construct neural network: $k_2(t)$

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: Φ

Output layer: $k_2(t_n;\Phi), n = 1, \ldots, X$

4: Specify the training set

$\kappa = \kappa_r \cup \kappa_t$ of the NN

5: Train the neural network

Specify a loss function

$$\zeta(\omega;\kappa) = \zeta_t(\omega;\kappa_t) + \zeta_r(\beta;\alpha_r)$$
$$r_1(t_n) = \frac{dz_{1NN}(t_n;\omega)}{dt} + k_1(t_n;\Phi)z_{1NN}(t_n;\omega)$$
$$r_2(t_n) = \frac{dz_{2NN}(t_n;\omega)}{dt} - \left(k_1(t_n;\Phi)z_{1NN}(t_n;\omega) - k_2(t_n;\Phi)z_{2NN}(t_n;\omega)\right)$$

Minimize $min_\kappa \zeta(\omega;\kappa)$ using Adam Optimizer.

6: Return OPINN solution

$z_{1NN}(t_n;\omega)$ and $z_{2NN}(t_n;\omega), n = 1, \ldots, X$

7: Return Parameter k_1:

$k_1(t_n;\Phi), n = 1, \ldots, X$

8: Return Parameter k_2:

$k_2(t_n;\Phi), n = 1, \ldots, X$

Upon examining the amplitude of oscillations over time, there is a noticeable variation in amplitude. While there are some fluctuations, there appears to be an amplitude increase toward the end of the interval. Considering the observations, a potential functional form for the curve could be a simple sinusoidal oscillation with a constant offset. A common form for such a function is:

$$k_1(t) = Asin(\lambda t + \alpha) + C.$$

where A is the amplitude, λ determines the frequency, α is the phase shift, and C is the vertical shift (which should be close to 5, given the curve's behavior). The curve $k_2(t)$ appears to have an initial rise, followed by a decrease, then stabilizes after a certain time. It appears to resemble an exponential decay combined with an exponential rise. A potential functional form for such a curve could be:

$$k_2(t) = ae^{-bt} + ce^{dt} + e.$$

where a and c are the amplitudes of the respective exponential terms, b and d are the rate constants for the decaying and rising terms, and e is a constant offset.

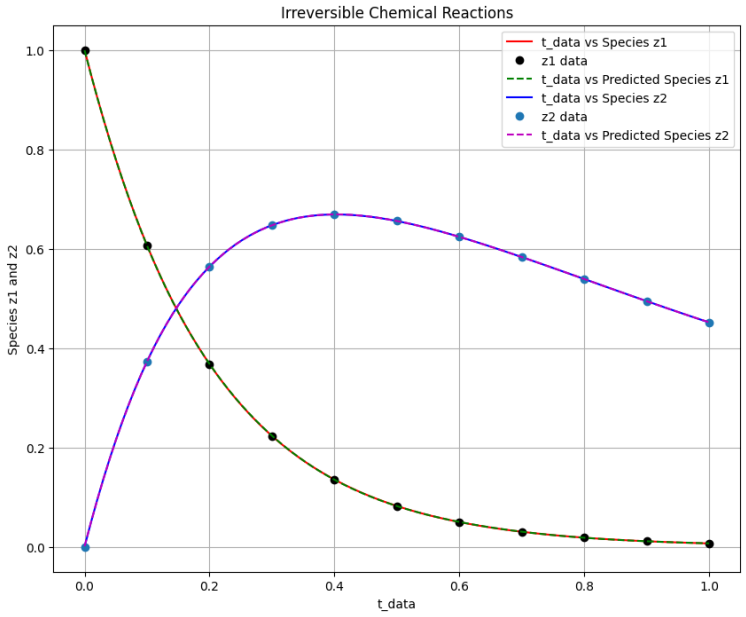


Figure 3.4: The first-order irreversible chain reactions solution of the actual output of species z_1, z_2 against t_{data} and the predicted output of species z_1, z_2 against t_{data}.

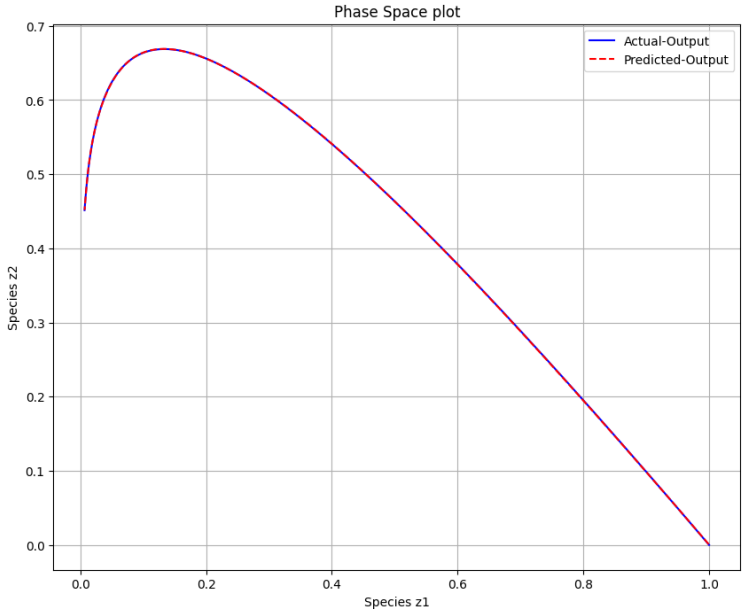
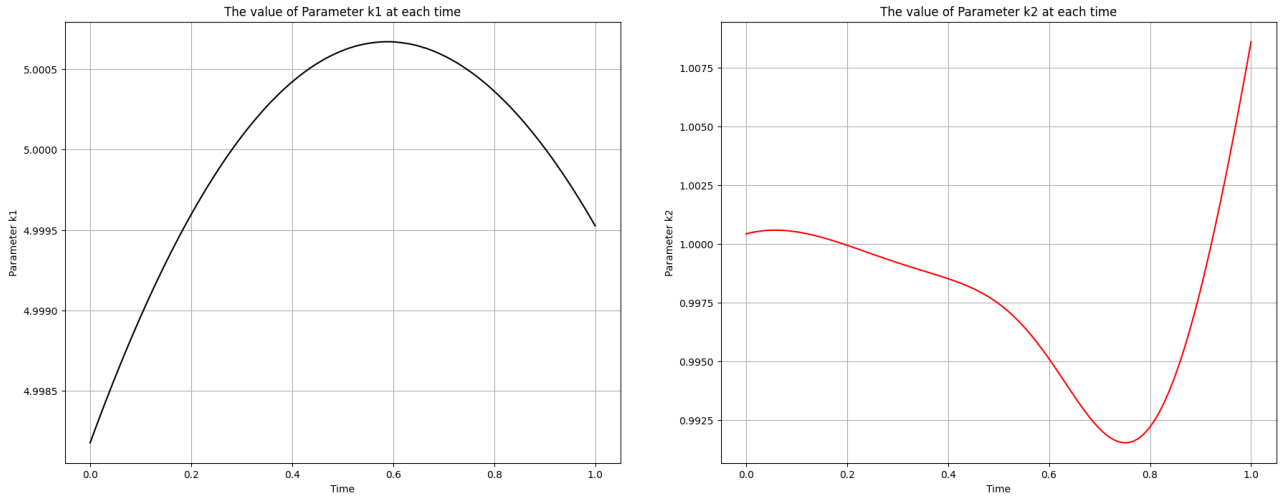
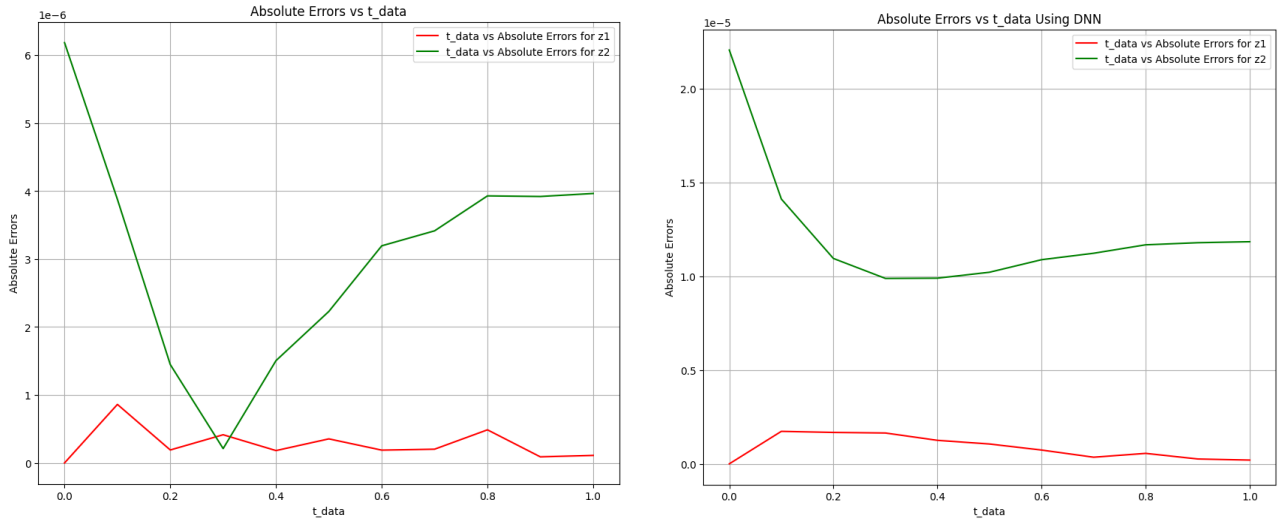


Figure 3.5: The phase space plot of the actual output of species z_1 against species z_2 and the predicted output of species z_1 against species z_2.



(**a**) (**b**)

Figure 3.6: The learned time-varying parameter values of the first-order irreversible chain reactions model. (**a**) Time-varying parameter k_1. (**b**) Time-varying parameter k_2.



(**a**) (**b**)

Figure 3.7: Absolute error plot between the data, OPINN solution, and DNN solution. (**a**) Absolute error plot using OPINN. (**b**) Absolute error plot using DNN.

Table 3.2: Comparison of the two approaches using OPINN.

	$k_1(t)$	$k_2(t)$
True Values	5.0	1.0
Approach 1	5.0000086	0.99999505
Error of Approach 1	0.0000086	0.00000495
Approach 2	5.0005193	1.0001917
Error of Approach 1	0.0005193	0.0001917
	$z_1(t)$	$z_2(t)$
MAE of Approach 1	2.7979×10^{-7}	3.0801×10^{-6}
MAE of Approach 2	1.7826×10^{-5}	3.7310×10^{-5}
MSE of Approach 1	1.3095×10^{-13}	1.1916×10^{-11}
MSE of Approach 2	4.8358×10^{-10}	2.0368×10^{-9}
RMSE of Approach 1	3.6187×10^{-7}	3.4519×10^{-6}
RMSE of Approach 2	2.1991×10^{-5}	4.5131×10^{-5}

Table 3.2 shows the obtained time-varying parameters and error metrics through the approaches described in Scenario 1 and Scenario 2. We observe that Scenario 1 proved more accurate than Scenario 2. The accuracy of the OPINN technique is evaluated by comparing the predicted concentrations of z_1 and z_2 with the actual values obtained from the data. These metrics provide insights

into the fitting accuracy of the OPINN technique. The results of the simulations show that the OPINN algorithm can accurately predict the concentrations and learn the time-varying parameters of the first-order irreversible chain reactions model. The obtained time-varying parameter values are close to the true values, and the error metrics indicate high accuracy and good fitting of the model to the data.

In addition, we compared the results we achieved using Optimized physics-informed neural networks to those obtained using deep neural networks, as shown in Tables 3.3 and 3.4. This comparison study used a setup of 90,000 epochs and one hidden layer containing 40 neurons, with processing time measured in seconds.

Figure 3.7 explores the model's accuracy by showing the absolute error between the target and the predictions. Specifically, Figure 3.7a indicates that the absolute error is bounded by 1 $\times 10^{-6}$ when using OPINN, while Figure 3.7b reveals that the error is bounded by 1 $\times 10^{-5}$ using DNN. This comparative analysis highlights the difference in accuracy between the two techniques, OPINN and DNN, used in the modeling process. Finally, the results show that the OPINN technique achieves comparable or better parameter estimation accuracy than the DNN technique. Additionally, the OPINN technique exhibits faster computation times, making it an efficient choice for solving and identifying parameters in dynamical systems.

Table 3.3: Comparison of obtained results using OPINN vs. DNN through the approaches described in Scenario 1. The table demonstrates a significant 67.7% improvement in computational efficiency when employing OPINNs compared to DNNs, highlighting the enhanced performance and time-saving capabilities of OPINNs in dynamic system modeling.

	True Values	**OPINN**	**DNN**	**Epochs**	**CPU Using OPINN**	**CPU Using DNN**
k_1	5.0	5.0000086	5.0000643			
				90,000	40s	124s
k_2	1.0	0.99999505	1.0000190			

Table 3.4: Comparison of obtained error results using OPINN vs. DNN. This table presents a detailed analysis of the error metrics for both OPINN and DNN models across two variables, z_1 and z_2. The results highlight the superior accuracy of OPINNs, as evidenced by significantly lower error values across all metrics compared to DNNs, underscoring the effectiveness of OPINNs in dynamic system modeling.

Error Metrics	$z_1(OPINN)$	$z_1(DNN)$	$z_2(OPINN)$	$z_2(DNN)$
MAE	2.7979×10^{-7}	2.1615×10^{-6}	3.0801×10^{-6}	2.3330×10^{-5}
MSE	1.3095×10^{-13}	7.0444×10^{-12}	1.1916×10^{-11}	5.6775×10^{-10}
RMSE	3.6187×10^{-7}	2.6541×10^{-6}	3.4519×10^{-6}	2.3827×10^{-5}

Furthermore, we also conducted a detailed investigation on the impact of augmenting the number of layers in the network called shallow vs. deep layers. Specifically, we focused on network architectures consisting of one, two, and three hidden layers, implemented using Optimized physics-informed neural networks. Subsequently, we compared the results obtained with deep neural networks. The respective outcomes of these analyses are succinctly presented in Table 3.5. The results indicate that the OPINN with one hidden layer achieves good accuracy, and increasing the number of layers does not significantly improve performance. Interestingly, the three hidden layer OPINN model showed a lower computation time (9 s) than the two layer model (13 s). There are various reasons for this unexpected outcome. Firstly, the increased depth (three layers) might enable more efficient learning and quicker convergence to a solution, thereby reducing the required epochs for training. Although the table shows more epochs for the two-layer configuration than the three-layer one, the actual computation time (CPU time) is less for the latter. This suggests that each epoch in the three-layer configuration is computationally less expensive.

Moreover, the three-layer network might better capture the complexities of the modeled dynamical system, leading to more efficient computation per epoch. This is indirectly supported by parameter estimation accuracy (k_1 and k_2), which are very close to their ideal values in the three-layer setup, indicating a high-quality learning process. Therefore, the reduction in computation time for the three-layer networks compared to the two-layer ones can be attributed to more effi-

Table 3.5: Comparison of OPINN vs. DNN based on shallow vs. deep layers through the approaches described in Scenario 1. This table details the accuracy of parameter estimation (k_1 and k_2) and computational performance across various network depths. Notably, it showcases the increasing computational efficiency of OPINNs over DNNs, with improvements of $66.67\%, 60.61\%$, and 62.5% for different layer configurations. These findings emphasize the enhanced efficiency and adaptability of OPINNs in handling complex dynamical systems, especially in deeper network architectures.

Layers	**Neurons**	k_1 **(OPINN)**	k_1 **(DNN)**	k_2 **(OPINN)**	k_2 **(DNN)**	**Epochs**	**CPU OPINN**	**CPU DNN**
1	40	4.99815	4.9925046	1.0002131	0.9969400	21378	10s	30s
2	40, 40	5.0000534	4.9999063	1.00021	1.0000036	24358	13s	33s
3	40, 40, 40	4.999991	5.0050346	0.9999847	1.0018185	16021	9s	24s

cient learning per epoch, potentially due to better representation and processing of the complex relationships in the data by the deeper network architecture.

3.3.2 Brusselator Model

The Brusselator model is a simple mathematical model for the behavior of chemical oscillations. Ilya Prigogine and Lefever first proposed it in 1968 [22]. The model comprises two chemical species, U and V, that interact through non-linear differential equations. The equations describe how the concentrations of the two species change over time, taking into account chemical reactions, diffusion, and other factors. The Brusselator is known for its ability to make complex, self-sustaining oscillations in the concentrations of the two chemical species. As a result, it has been used to study a wide range of phenomena in chemistry, biology, and physics, including patterns of gene expression and the behavior of simple electronic circuits [74, 45]. The Brusselator model is useful in various fields because it is a simple yet rich model that can exhibit a wide range of complex behaviors.

The Brusselator reaction is a simple model of autocatalytic chemical reactions, which is a chemical reaction catalyzed by one of the products of the reaction. The reaction has been used as a model to study the emergence of self-organizing behavior in chemical systems [51]. It has been applied to various fields, including chemistry, physics, biology, and engineering. The Brusselator reaction consists of the following:

$$A \longrightarrow U \rightarrow \frac{du}{dt} = k_a a$$

$$2U + V \longrightarrow 3U \rightarrow \left(\frac{du}{dt} = k_b u^2 v, \frac{dv}{dt} = -k_b v^2 y \right)$$

$$B + U \longrightarrow V + C \rightarrow \left(\frac{du}{dt} = -k_c bu, \frac{dv}{dt} = -k_c bu \right)$$

$$U \longrightarrow D \rightarrow \frac{du}{dt} = -k_a u.$$

The overall reaction is $A + B \longrightarrow C + D$ with intermediary species U and V. A and B are reactants while C and D are products. During the chemical reaction, suppose the concentration reactant A and B are kept constant; therefore, the ordinary differential equation version of the Brusselator is given below:

$$\begin{aligned} \frac{du(t)}{dt} &= a(t) + u(t)^2 v(t) - (b(t) + 1)u(t) \\ \frac{dv(t)}{dt} &= bu(t) - u(t)^2 v(t). \end{aligned} \tag{3.12}$$

where $u(t)$ and $v(t)$ are the concentrations of the two chemical species involved in the reaction, and $a(t)$ and $b(t)$ are the time-varying parameters that control the system's behavior. The Brusselator model exhibits various dynamic behaviors, depending on the values of the a and b parameters [22]. They demonstrated that the system could exhibit stable, steady states, oscillations, and chaotic behavior. Finally, the parameters of the Brusselator system are also important in understanding the behavior of real-world chemical systems, which can be modeled using similar equations. Understanding how the parameters of such a system affect its behavior can provide insights into the underlying chemical reactions and help predict and control the system's behavior.

Our primary aim is to learn the time-varying parameters of the Brusselator model. Here, we used synthetic data that was generated by using the numerical solution of the Brusselator model derived from an ODE-solver on the Brusselator model, where we assumed the initial conditions for the first species and the second species variables to be $u(0) = 1$ and $v(0) = 1$, respectively. We choose $a = 1.0$ and $b = 2.5$. This problem was solved on the time interval $[0.T]$, where T = 30. We

used a grid with $P_x = 1000$ points on the time interval, and every second-time step, the numerical solution vector is taken to obtain the observation or synthetic data. We use the OPINN architecture shown in Figure 3.3 with three networks to learn the time-varying parameters of the Brusselator model. The first network outputs are $u_{NN}(t_n;\omega)$ and $v_{NN}(t_n;\omega)$, which admits t as the input data. The second and third networks learn the parameters $a(t_n;\Phi)$ and $b(t_n;\Phi)$ from the data. We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ which was minimized as follows:

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{2}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2. \tag{3.13}$$

where

-

$$\begin{aligned} r_1(t) &= \frac{du_{NN}(t_n;\omega)}{dt} - \Big(a(t_n;\Phi) - (b(t_n;\Phi)+1)u_{NN}(t_n;\omega) + u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big) \\ r_2(t) &= \frac{dv_{NN}(t)}{dt} - \Big(b(t_n;\Phi)u_{NN}(t_n;\omega) - u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big) \end{aligned} \tag{3.14}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\Bigg(\sum_{n=1}^{\kappa_t}||u_{NN}(t_n;\omega) - u(t_n)||_2^2 + \sum_{n=1}^{k_t}||v_{NN}(t_n;\omega) - v(t_n)||_2^2\Bigg). \tag{3.15}$$

The time-varying parameters of the Brusselator model were obtained after using OPINN and the approach of Scenario 2 with three hidden layers, 64 neurons per layer, 50,000 epochs, the tanh activation function was used, and the learning rate was 10^{-3}. The OPINN Algorithm 6 is an excellent choice for learning the optimal parameters of the Brusselator model. The maximum parameter value a was solved at $t^* = 6.37$ and parameter b at $t^* = 13.76$. Figure 3.8 shows the obtained solution of the actual and predicted output of the Brusselator model after the parameters were learned with the same initial condition used to generate the data. The phase space plot of the actual output of species $u(t)$ against species $v(t)$ and the predicted output of species $u_{NN}(t)$ against species $v_{NN}(t)$ is shown in Figure 3.9.

Figure 3.10 shows the learned time-varying parameter values of $a(t)$ and $b(t)$ of the Brusselator

model. The $a(t)$ curve appears to oscillate with an overall decreasing trend. A potential functional form for this curve is:

$$a(t) = e^{-\alpha\beta(t)} \tag{3.16}$$

$$b(t) = e^{\alpha\beta(t)}. \tag{3.17}$$

where α is the constant coefficient, its value will determine the growth or decay rate. Since α is negative in (3.16), the function represents decay, and if positive, the function represents growth. The magnitude of α will determine how fast this growth or decay occurs. The $b(t)$ curve also appears to oscillate with an overall increasing trend. Since α is positive in (3.17), the function represents growth, and if negative, the function represents decay. $\beta(t)$ is a time-dependent function that appears as the exponent of the exponential term. It plays a crucial role in determining the behavior of the curve over time. Table 3.6 shows the parameters and error metrics obtained through the approaches described in Scenario 2. The results showed that the OPINN algorithm successfully learned the parameters of the Brusselator model. The obtained parameter values were close to the true values used to generate the synthetic data. The accuracy of the predictions was evaluated using various error metrics, such as mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). The error metrics indicated a high level of accuracy in the predictions, with very small errors.

Furthermore, we conducted a comparison between the results obtained using Optimized physics-informed neural networks (OPINN) and those obtained through deep neural networks (DNN), as detailed in Tables 3.7 and 3.8. This comparative analysis was carried out under the specific setup of 50,000 epochs, three hidden layers containing 64 neurons, and employing the tanh activation function. Processing time, measured in seconds, was also considered in this comparison.

Algorithm 6 OPINN algorithm for learning the parameters of the Brusselator model

1: Construct OPINN

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: ω

Output layer: $u_{NN}(t_n)$ and $v_{NN}(t_n), n = 1, \ldots, X$

2: Construct neural network: $a(t)$

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: Φ

Output layer: $a(t_n), n = 1, \ldots, X$

3: Construct neural network: $b(t)$

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: Φ

Output layer: $b(t_n), n = 1, \ldots, X$

4: Specify the training set

$k = k_r \cup k_t$ of the NN

5: Train the neural network

Specify a loss function

$$\zeta(\omega;\kappa) = \zeta_t(\omega;\kappa_t) + \zeta_r(\omega;\kappa_r)$$

$$r_1(t) = \frac{du_{NN}(t_n;\omega)}{dt} - \Big(a(t_n;\Phi) - (b(t_n;\Phi)+1)u_{NN}(t_n;\omega) + u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big)$$

$$r_2(t) = \frac{dv_{NN}(t_n;\omega)}{dt} - \Big(b(t_n;\Phi)u_{NN}(t_n;\omega) - u_{NN}(t_n;\omega)^2 v_{NN}(t_n;\omega)\Big)$$

Minimize $min_\kappa \zeta(\omega;\kappa)$ using Adam Optimizer.

6: Return OPINN solution

$u_{NN}(t_n;\omega)$ and $v_{NN}(t_n;\omega), n = 1, \ldots, X$

7: Return Parameter a:

$a(t_n;\Phi), n = 1, \ldots, X$

8: Return Parameter b:

$b(t_n;\Phi), n = 1, \ldots, X$

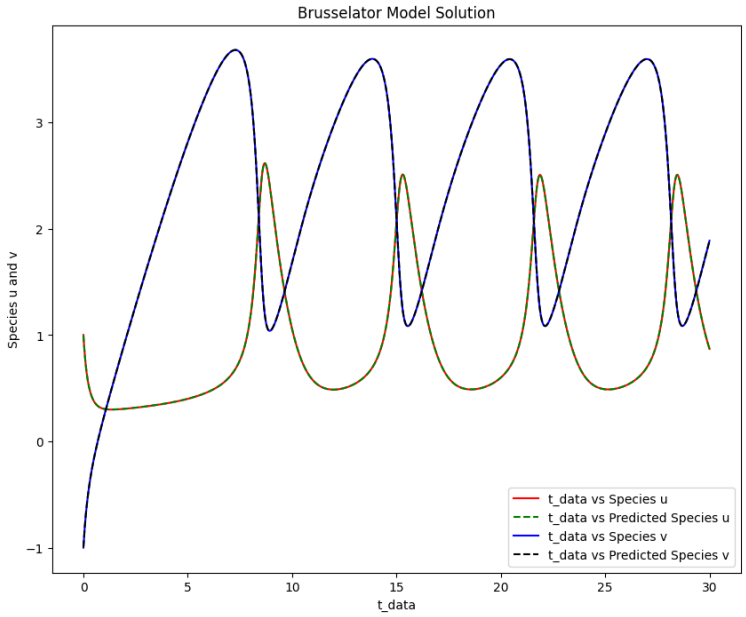


Figure 3.8: The Brusselator model solution of the actual output of species u,v against t_{data} and the predicted output of species u,v against t_{data}.

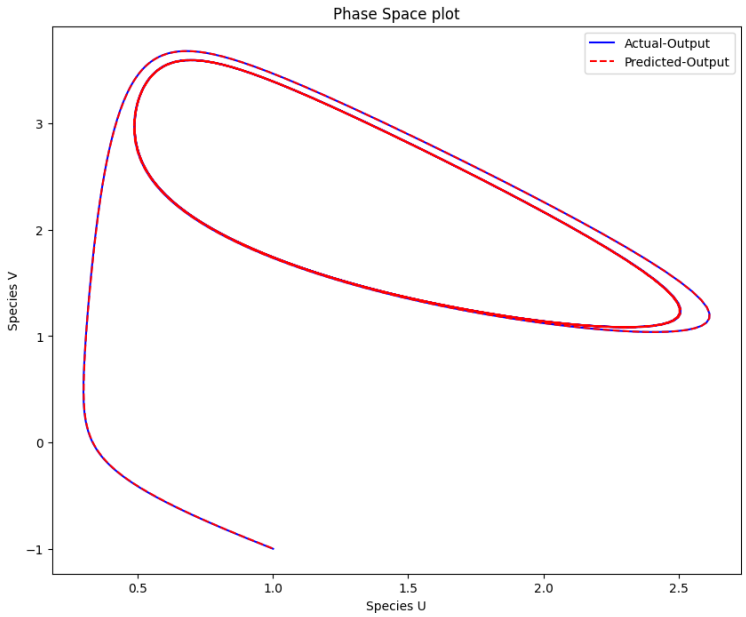
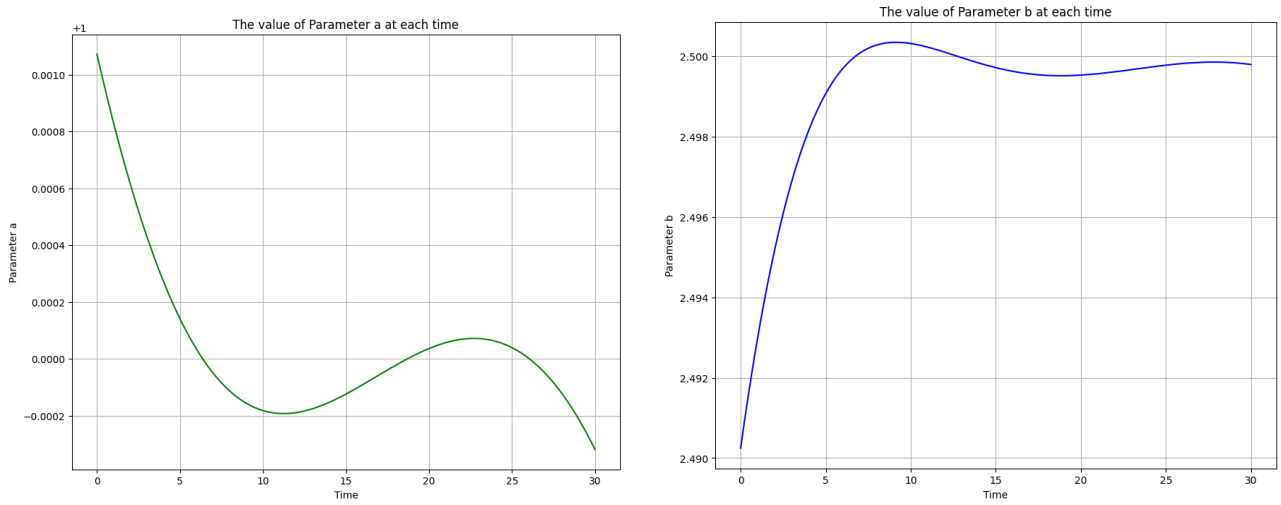


Figure 3.9: The phase space plot of the actual output of species u against species v and the predicted output of species u against species v.



(**a**) (**b**)

Figure 3.10: The learned time-varying parameter values of the Brusselator model. (**a**) Time-varying parameter a. (**b**) Time-varying parameter b.

Table 3.6: The optimal parameter estimation and the error metrics using OPINN.

	a	b
True Values	1.0	2.5
Approach 2	1.0000001	2.5
Error of Approach 2	0.0000001	0.0000
	u	v
MAE of Approach 2	1.5247×10^{-6}	1.7520×10^{-6}
MSE of Approach 2	6.5662×10^{-12}	9.5430×10^{-12}
RMSE of Approach 2	2.5625×10^{-6}	3.0892×10^{-6}

The study revealed that OPINN not only achieved faster processing times than DNN but also showed that the optimal parameters learned through OPINN were closer to the actual values than those determined by DNN. This emphasizes the efficiency and accuracy advantages of OPINN in this particular application.

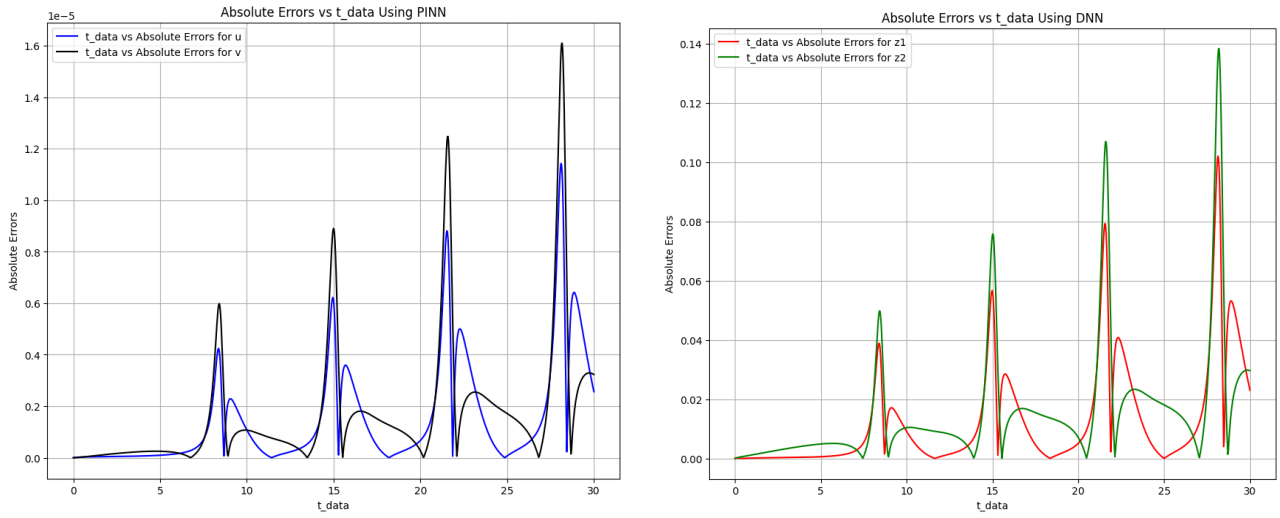
Table 3.7: Comparison of obtained results using OPINN vs. DNN through the approaches described in Scenario 1. This table illustrates the accuracy in parameter estimation and the significant computational efficiency improvement of 88.7% with OPINNs over DNNs, highlighting the robustness and speed of OPINNs in complex system modeling.

	True Values	**OPINN**	**DNN**	**Epochs**	**CPU Using OPINN**	**CPU Using DNN**
a	1.0	1.0000001	1.0038480			
				50,000	111s	982s
b	2.5	2.5	2.5088659			

Table 3.8: Comparison of obtained error results using OPINN vs. DNN.

Error Metrics	$u(OPINN)$	$u(DNN)$	$v(OPINN)$	$v(DNN)$
MAE	1.5247×10^{-6}	3.6030×10^{-2}	1.7520×10^{-6}	4.5508×10^{-2}
MSE	6.5662×10^{-12}	3.8405×10^{-3}	9.5430×10^{-12}	$5.4941x10^{-3}$
RMSE	2.5624×10^{-6}	6.1972×10^{-2}	3.0891×10^{-6}	7.4122×10^{-2}

In addition, the model's accuracy is further illustrated in Figure 3.11, where the absolute error between the target and predictions made by Optimized physics-informed neural networks (OPINN) and deep neural networks (DNN) is presented. The comparative analysis demonstrates that the error was significantly reduced when OPINN was utilized instead of DNN. This discrepancy highlights the difference in accuracy between the two techniques employed in the modeling process. Furthermore, the results demonstrate that OPINN matches but often exceeds DNN regarding parameter estimation accuracy. Finally, faster computation times make OPINN an especially efficient choice for solving and pinpointing parameters in dynamical systems.



(**a**) (**b**)

Figure 3.11: Absolute error plot between the data, OPINN solution, and DNN solution for the Brusselator model. (**a**) Absolute error plot using OPINN. (**b**) Absolute error plot using DNN.

Finally, an analysis of data-driven simulations for training with varying numbers of epochs, such as 30,000, 40,000, and 50,000, using three layers with 64 neurons on the Brusselator model to learn time-varying parameters is presented in Table 3.9. The overall loss decreases as the number of epochs increases from 30,000 to 50,000. This indicates that the model benefits from a longer training duration, refining its weights and biases more effectively to minimize the difference between its predictions and the actual data. Therefore, increasing the number of epochs generally leads to a decrease in loss and better performance in predicting the parameters of u and v.

Table 3.9: Analysis of the Brusselator model using different epochs.

Epochs	**Loss**	**Error u**	**Error v**
50,000	2.7514×10^{-2}	7.9001×10^{-4}	5.5722×10^{-4}
40,000	1.9048×10^{-1}	1.5938×10^{-3}	3.3815×10^{-3}
30,000	1.5272×10^{-1}	2.8571×10^{-3}	5.4818×10^{-3}

3.3.3 Biomass Transfer

Biomass transfer involves the movement of organic material from one organism to another within an ecosystem. This concept is key to understanding the flow of energy within an ecosystem, as energy is transferred from producers (like plants) to primary consumers (like herbivores) and then on to secondary and tertiary consumers (like carnivores and omnivores). Imagine a forest in Europe populated by one or two species of trees. We pick out some of the eldest among them, those on the brink of their life cycle and anticipated to wither in the coming years. We then observe their journey from living, thriving trees to becoming lifeless entities. Over time, these deceased trees decompose and topple due to natural seasonal changes and biological influences. Ultimately, these fallen trees transform into nutrient-rich humus, completing their life cycle [18]. Differential equations are often used to model biomass transfer as they can describe the rate of change of a quantity (in this case, biomass) over time.

Let us consider the dynamical system model of a biomass transfer

$$\begin{aligned} \frac{du_1(t)}{dt} &= -a(t)u_1(t) + b(t)u_2(t) \\ \frac{du_2(t)}{dt} &= -b(t)u_2(t) + c(t)u_3(t) \\ \frac{du_3(t)}{dt} &= -c(t)u_3(t). \end{aligned} \tag{3.18}$$

where variable $u_1(t)$, $u_2(t)$ and $u_3(t)$ is defined by

$u_1(t)$: biomass decayed into humus per time

$u_2(t)$: biomass of dead trees per time

$u_2(t)$: biomass of living trees

t : time in decades

and the variable a, b and c are the parameters required to be estimated. Given that $a = 1$, $b = 3$,

$c = 5$ and the initial conditions of $u(t = 0) = [0, 0, 1]$, then the observed data [56] are given in Table 3.10.

Table 3.10: Observed values of u_1, u_2 and u_3 at different time points.

t	$u1$	$u2$	$u3$
0.0	0.000000	0.000000	1.000000
0.1	0.055747	0.335719	0.606531
0.2	0.166850	0.452330	0.367879
0.3	0.282767	0.458599	0.223130
0.4	0.381125	0.414647	0.135335
0.5	0.454416	0.352613	0.082085
0.6	0.502502	0.288780	0.049787
0.7	0.528506	0.230648	0.030197
0.8	0.536641	0.181006	0.018316
0.9	0.531127	0.140241	0.011109
1.0	0.515706	0.107623	0.006738

The analytical solution of (18) is given as follows:

$$\begin{aligned} u_1(t) &= \frac{15}{8}\left(e^{-5t} - 2e^{-3t} + e^{-t}\right) \\ u_2(t) &= \frac{5}{2}\left(-e^{-5t} + e^{-3t}\right) \\ u_3(t) &= e^{-5t}. \end{aligned} \tag{3.19}$$

The OPINN approach to solve and identify the parameters of the biomass transfer model is made by finding the best network parameters, ω, representing the biases and weights network that minimizes the loss function. Following a similar procedure in the previous application, we offer OPINN with four networks to do this. The first network outputs are $u_{1NN}(t_n;\omega)$, $u_{2NN}(t_n;\omega)$ and $u_{3NN}(t_n;\omega)$, which admits t as the input data. The second, third and fourth networks learn the parameters $a(t_n;\Phi)$, $b(t_n;\Phi)$ and $c(t_n;\Phi)$ from the data.

We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows:

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{3}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2. \tag{3.20}$$

where

$$
\begin{aligned}
r_1(t_n) &= \frac{du_{1NN}(t_n;\omega)}{dt} - \Big(-a(t_n;\Phi)u_{1NN}(t_n;\omega) + b(t_n;\Phi)u_{2NN}(t_n;\omega) \Big) \\
r_2(t_n) &= \frac{du_{2NN}(t_n;\omega)}{dt} - \Big(-b(t_n;\Phi)u_{2NN}(t_n;\omega) + c(t_n;\Phi)u_{3NN}(t_n;\omega) \Big) \\
r_3(t_n) &= \frac{du_{3NN}(t_n;\omega)}{dt} - \Big(-c(t_n;\Phi)u_{3NN}(t_n;\omega) \Big)
\end{aligned}
\tag{3.21}
$$

and

$$
\begin{aligned}
\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t} \Bigg(& \sum_{n=1}^{\kappa_t} |u_{1NN}(t_n;\omega) - u_1(t_n)|^2 + \sum_{n=1}^{\kappa_t} |u_{2NN}(t_n;\omega) - u_2(t_n)|^2 \\
& + \sum_{n=1}^{\kappa_t} |u_{3NN}(t_n;\omega) - u_3(t_n)|^2 \Bigg).
\end{aligned}
\tag{3.22}
$$

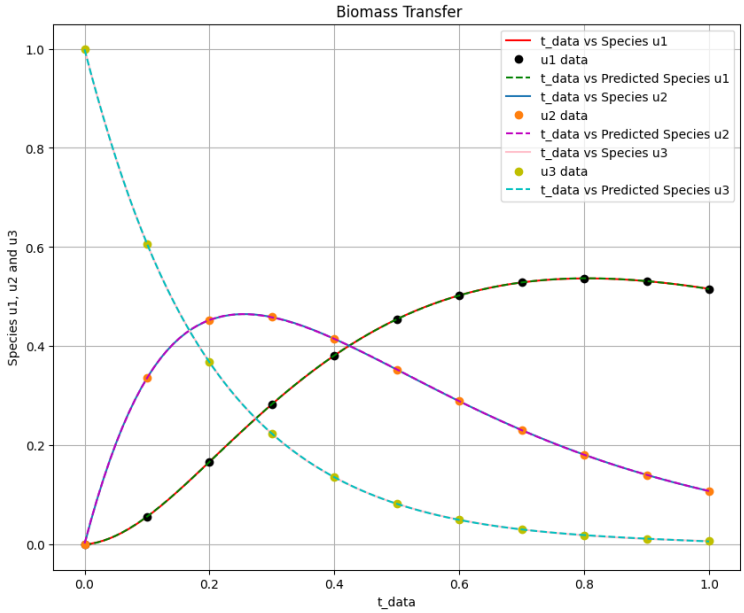


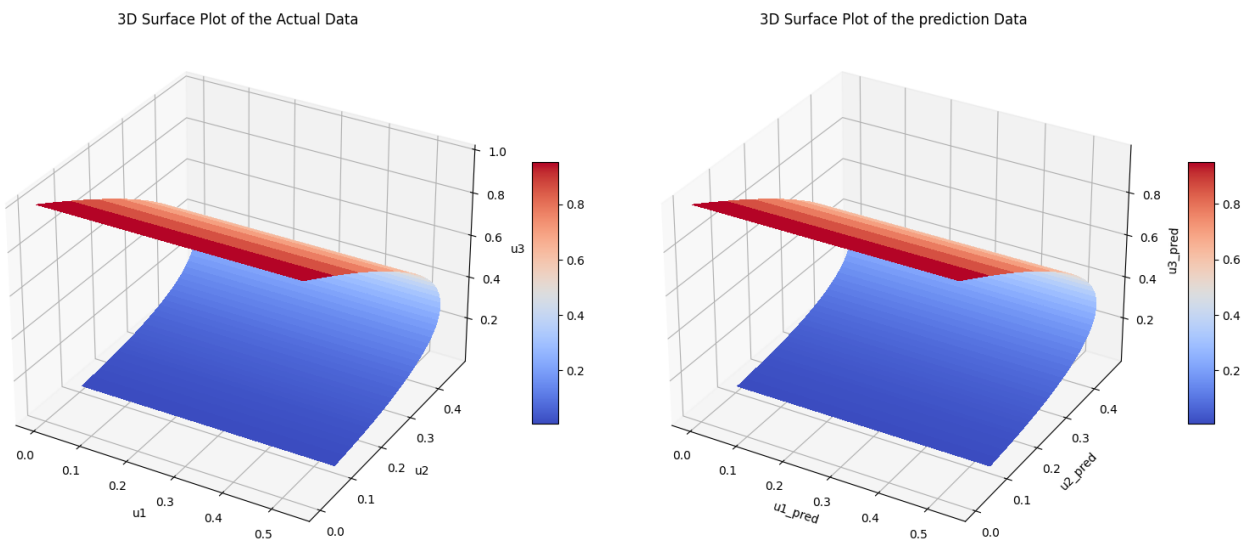
Figure 3.12: The biomass transfer exact solution and the predicted output of species u_1, u_2, u_3 against t_{data}.

Algorithm 7 OPINN algorithm for learning the parameters of the biomass transfer model

1: Construct OPINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize OPINN parameter: ω
   Output layer: $u_{1NN}(t_n;\omega)$, $u_{2NN}(t_n;\omega)$ and $u_{3NN}(t_n;\omega), n = 1, \ldots, X$
2: Construct neural network: $a(t)$
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize OPINN parameter: Φ
   Output layer: $a(t_n;\Phi), n = 1, \ldots, X$
3: Construct neural network: $b(t)$
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize OPINN parameter: Φ
   Output layer: $b(t_n;\Phi), n = 1, \ldots, X$
4: Construct neural network: $c(t)$
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize OPINN parameter: Φ
   Output layer: $c(t_n;\Phi), n = 1, \ldots, X$
5: Specify the training set
   $\kappa = \kappa_r \cup \kappa_t$ of the NN
6: Train the neural network
   Specify a loss function

$$\zeta(\omega;\kappa) = \zeta_t(\omega;\kappa_t) + \zeta_r(\omega;\kappa_r)$$
$$r_1(t_n) = \frac{du_{1NN}(t_n;\omega)}{dt} - \left(-a(t_n;\Phi)u_{1NN}(t_n;\omega) + b(t_n;\Phi)u_{2NN}(t_n;\omega) \right)$$
$$r_2(t_n) = \frac{du_{2NN}(t_n;\omega)}{dt} - \left(-b(t_n;\Phi)u_{2NN}(t_n;\omega) + c(t_n;\Phi)u_{3NN}(t_n;\omega) \right)$$
$$r_3(t_n) = \frac{du_{3NN}(t_n;\omega)}{dt} - \left(-c(t_n;\Phi)u_{3NN}(t_n;\omega) \right)$$

   Minimize $min_{\kappa}\zeta(\omega;\kappa)$ using Adam Optimizer.
7: Return OPINN solution
   $u_{1NN}(t_n;\omega)$, $u_{2NN}(t_n;\omega)$ and $u_{3NN}(t_n;\omega), n = 1, \ldots, X$
8: Return Parameter a:
   $a(t_n;\Phi), n = 1, \ldots, X$
9: Return Parameter b:
   $b(t_n;\Phi), n = 1, \ldots, X$
10: Return Parameter c:
   $c(t_n;\Phi), n = 1, \ldots, X$



(**a**) (**b**)

Figure 3.13: The true and the predicted values of species u_1, u_2 and u_3. (**a**) The true values of species U. (**b**) The predicted values of species U.

The neural network obtains the optimal values for the model parameters. The $u_1(t_n)$, $u_2(t_n)$ and $u_3(t_n)$ represent the loss function's training data. The training process minimizes the residual loss, which measures the discrepancy between the model predictions and the actual differential equations, and the training loss, which measures the discrepancy between the model predictions and the observed data. Finally, the biomass transfer model's time-varying parameters were obtained using OPINN with one layer of 10 neurons, 71,800 epochs, the tanh activation function, and a learning rate of 10^{-3}. The OPINN Algorithm (7) for learning the optimal parameters of the biomass transfer model (3.18) is shown below.

The time-varying parameters $a(t)$, $b(t)$, and $c(t)$ are obtained using the above two scenarios. The corresponding Equation (3.19) system was solved at $t^* = 0.26$, 0.65, and 0.59 for parameters a, b, and c, and Figure 3.12 shows the obtained exact solution and the predicted output of the biomass transfer model. The 3D surface plot of the actual output of species u_1, u_2 and u_3 and the predicted output of species u_1, u_2 and u_3 is shown in Figure 3.13. Figure 3.14 shows the learned time-varying parameter values of the biomass transfer model for different inputs of $a(t), b(t)$, and $c(t)$. The a, b, and c curves look like quadratic functions. The plot of the absolute error between

the target and the predictions is shown in Figure 3.15. Table 3.11 shows the parameters and error metrics obtained through the approaches described in Scenarios 1 and 2. We observe that Scenario 1 proved more accurate than Scenario 2. The results demonstrate that the OPINN approach yields accurate parameter estimates and predictions for the biomass transfer model. The obtained parameters are close to their true values, and the error metrics indicate high accuracy and performance.

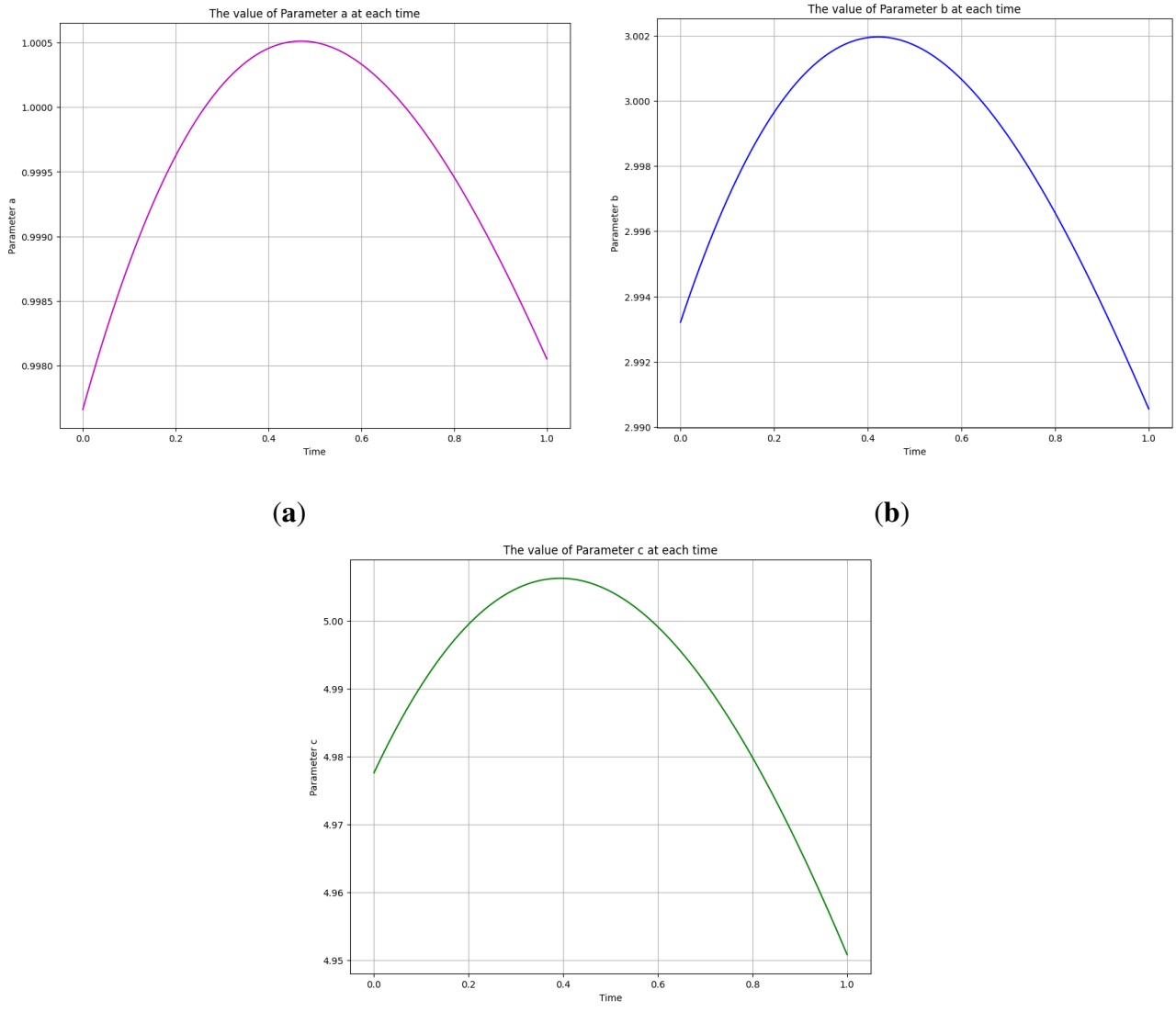


Figure 3.14: The learned time-varying parameter values of the biomass transfer model. (**a**) Time-varying parameter a. (**b**) Time-varying parameter b. (**c**) Time-varying parameter c.

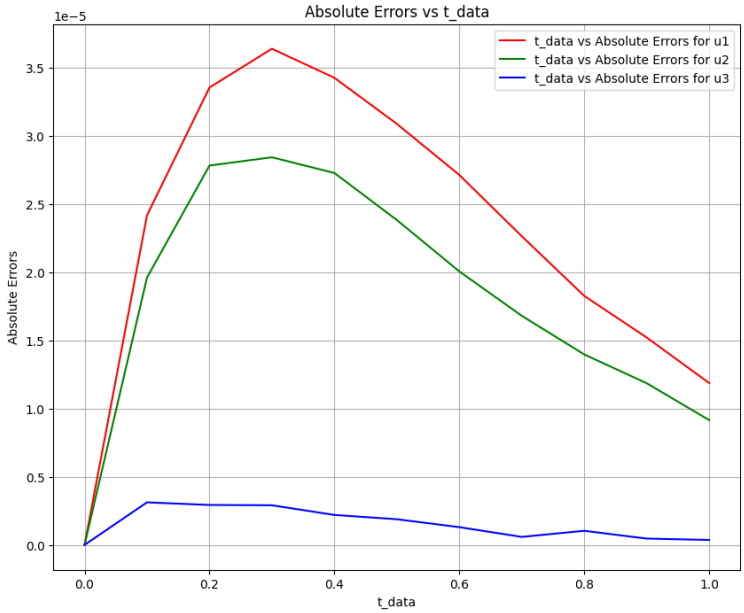


Figure 3.15: Absolute error plot between the data and OPINN solution.

In addition, the results obtained from the Optimized physics-informed neural networks approach are compared with the values and results reported in the literature using deep neural networks [56]. This comparative study was conducted with a configuration of 71,800 epochs and one hidden layer containing 10 neurons. The results are shown in Table 3.12. The OPINN approach outperforms the DNN approach in terms of accuracy and computational efficiency. Furthermore, Table 3.13 presents a comparative analysis between Optimized physics-informed neural networks and deep neural networks, illustrating the differences in their performance across shallow and deep layers. The CPU time of OPINN for one, two, and three layers with 40 neurons are 9, 9, and 6 s, while for DNN is 20, 19, and 16 s [56]. The data presented in Table 3.13 demonstrates that the OPINN approach outperforms the DNN approach in terms of accuracy and computational efficiency, as evidenced by comparing results and CPU times, especially in both shallow and deep-layer processing capabilities. Specifically, for a network with one layer of 40 neurons, the OPINN model was 55% more efficient than the DNN model. Similarly, for two layers of 40 neurons each, the efficiency improvement was 52.63%, and for three layers, it increased to 62.5%. These results highlight the potential of OPINNs in providing faster computational solutions, particularly

as the network complexity increases, making them highly suitable for real-time complex system modeling.

Table 3.11: Comparison of the two approaches using OPINN.

	a	b	c
True Values	1.0	3.0	5.0
Approach 1	0.99999785	2.999931	5.000046
Error of Approach 1	0.00000215	0.000069	0.000046
Approach 2	1.0000874	3.0001755	5.0010123
Error of Approach 1	0.0000874	0.0001755	0.0010123
	u_1	u_2	u_3
MAE of Approach 1	2.3118×10^{-5}	1.8068×10^{-5}	1.5237×10^{-6}
MAE of Approach 2	2.1915×10^{-5}	2.6433×10^{-5}	3.4797×10^{-5}
MSE of Approach 1	6.4652×10^{-10}	3.9821×10^{-10}	3.5008×10^{-12}
MSE of Approach 2	6.9882×10^{-10}	8.9255×10^{-10}	1.8445×10^{-9}
RMSE of Approach 1	2.5427×10^{-5}	1.9955×10^{-5}	1.8709×10^{-6}
RMSE of Approach 2	2.6435×10^{-5}	2.9876×10^{-5}	4.2948×10^{-5}

Table 3.12: Comparison of obtained results using OPINN vs. reported in the literature using DNN.

	True Values	**OPINN**	**DNN** [56]	**Epochs**
a	1.0	0.99999785	1.0024	
				71,800
b	3.0	2.999931	3.0026	
				71,800
c	5.0	5.000046	5.0150	

Table 3.13: Comparison of OPINN vs. reported in the literature using DNN [56] based on shallow vs. deep layers.

Layers	**Neurons**	a **(OPINN)**	a **(DNN)**	b **(OPINN)**	b **(DNN)**	c **(OPINN)**	c **(DNN)**	**Epochs**
1	40	1.0000379	1.0044	3.0000386	2.9936	5.0002103	4.9451	20,000
2	40, 40	1.0000603	1.0051	2.9999495	3.0125	4.9998903	5.0515	15,107
3	40, 40, 40	0.99996376	1.0044	3.000064	2.9936	4.999971	4.9451	9800

3.3.4 Lotka-Volterra Model

The Lotka-Volterra model is a model of the evolution of a prey-predator system. A prey-predator system is a completion where one species, called the predator, has more impact (winning), and the other species, less impact (losing), called the prey. James Lotka and Vito Volterra proposed the Lotka-Volterra model in 1925 and 1926 [4, 59]. If $P(t)$ is the prey population and $Q(t)$ the predator population at time t, then the ordinary differential equation of the Lotka-Volterra model is

$$\frac{dP(t)}{dt} = P(t)(a(t) - b(t)Q(t)) \tag{3.23}$$

$$\frac{dQ(t)}{dt} = Q(t)(c(t)P(t) - d(t)) \tag{3.24}$$

where variable $P(t), Q(t), a(t), b(t), c(t)$ and $d(t)$ represents

$P(t)$: the size of the prey population per time

$Q(t)$: the size of the predator population

$a(t)$: the prey per capital rate of increase per time

$b(t)$: the capture efficiency per time

$c(t)$: the conversion efficiency per time

$d(t)$: the mortality rate per time

The parameters $a(t), b(t), c(t), d(t)$ are non-negative.

The Lotka-Volterra model has an adequate system model up to some parameters that need to be determined. To solve the Lotka-Volterra model, you can use numerical methods or analytical techniques. A Optimized physics-informed neural network approach is proposed in this case. The OPINN combines neural networks with physical laws or equations to learn the model's time-varying parameters from data. Following the same OPINN architecture shown in Figure 3.1, we offer a OPINN algorithm with five networks to do this. The first network outputs are $P_{NN}(t_n; \omega)$

and $Q_{NN}(t_n;\omega)$, which admits t as the input data. The second, third, fourth, and fifth networks learn the parameters $a(t_n;\Phi)$, $b(t_n;\Phi)$, $c(t_n;\Phi)$ and $d(t_n;\Phi)$ from the data. We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows to quantify the discrepancy between the model predictions and the actual data. The OPINN algorithm minimizes the combined loss function to learn the optimal parameters and obtain the solution of the Lotka-Volterra model.

$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{2}\sum_{n=1}^{\kappa_r}|r_i(t_n)|^2. \tag{3.25}$$

where

$$\begin{aligned} r_1(t) &= \frac{dP_{NN}(t_n;\omega)}{dt} - \Big(P_{NN}(t_n;\omega)(a(t_n;\Phi) - b(t_n;\Phi)Q_{NN}(t_n;\omega))\Big) \\ r_2(t) &= \frac{dQ_{NN}(t_n;\omega)}{dt} - \Big(Q_{NN}(t_n;\omega)(c(t_n;\Phi)P_{NN}(t_n;\omega) - d(t_n;\Phi))\Big) \end{aligned} \tag{3.26}$$

and

$$\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t}|P_{NN}(t_n;\omega) - P(t_n)|^2 + \sum_{n=1}^{\kappa_t}|Q_{NN}(t_n;\omega) - Q(t_n)|^2\right) \tag{3.27}$$

To generate our measurement data, we numerically solve (3.23) and (3.24), utilizing an initial condition of $(P_0, Q_0) = (0.2, 0.3)$. We selected parameter values as follows: $a = 1, b = 2, c = 1$, and $d = 0.3$. The problem was resolved over a period delineated by the interval $[0, T]$, where T is set to 13, a duration that approximately encapsulates one cycle. We employ a $N_x = 1000$ points grid within this temporal scope. Our measurement data is procured by extracting the numerical solution vector at every alternate time step. As a result of this method, we obtain a dataset where $N = 50$ [1]. Finally, the Lotka-Volterra model parameters were obtained using OPINN and the approaches from Scenario 2 with three hidden layers, 64 neurons per layer, 50,000 epochs, the sigmoid activation function, and a learning rate of 10^{-3}. The OPINN Algorithm (8) for learning the optimal parameters of the Lotka-Volterra model is shown below.

After the parameters were learned using the same initial condition used to produce the data, the results that are obtained are then compared with the true values of the parameters and the actual data. Figures are provided to visualize the predicted prey and predator populations, the phase space plot, and the absolute error between the target and predicted values. The tables present the initial

and obtained parameters and error metrics such as MAE, MSE, and RMSE. Figure 3.16 displays the resulting solution of the actual and predicted output of the Lotka-Volterra model. Figure 3.17 shows the phase space plot of the anticipated output of species $P(t)$ versus species $Q(t)$ and the actual output of species $P(t)$ against species $Q(t)$. Figure 3.18 displays the Lotka-Volterra model's learned time-varying parameter values for various inputs of P_{NN} and Q_{NN}.

The learned time-varying parameters are $a(t)$, $b(t)$, $c(t)$, and $d(t)$. It was observed that the functional form of the curve of $a(t)$ is linear, while the functional forms of $b(t)$, $c(t)$, and $d(t)$ are

$$b(t) = B\ sin(\lambda t + \alpha) + k$$

$$c(t) = C\ sin(\lambda t + \alpha) + k$$

$$d(t) = D\ sin(\lambda t + \alpha) + k$$

where B, C and D are the amplitude, λ determines the frequency, α is the phase shift, and K is the vertical shift (which should be close to the initial values, given the curve's behavior).

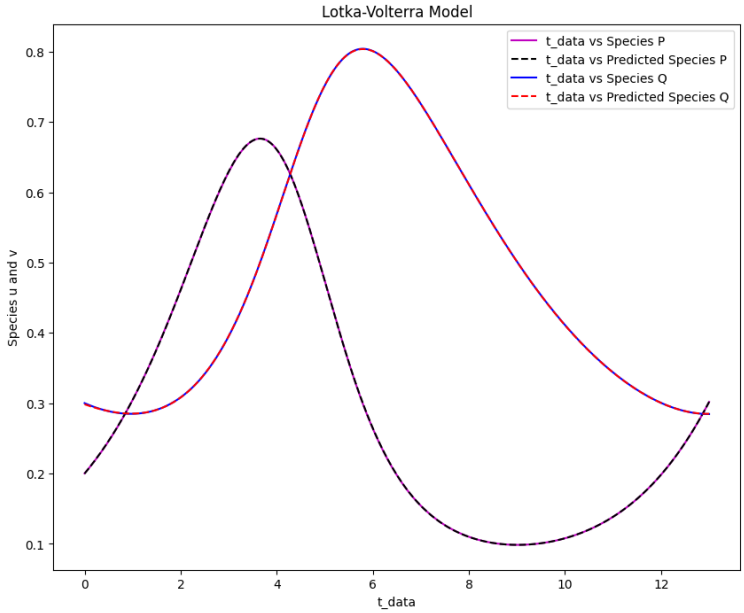


Figure 3.16: The Lotka-Volterra model solution for the real output of species P and Q against time data and the predicted output of species P and Q against time data.

Algorithm 8 OPINN algorithm for learning the parameters of Lotka-Volterra model

1: Construct OPINN
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize OPINN parameter: ω
   Output layer: $P_{NN}(t_n;\omega)$ and $Q_{NN}(t_n;\omega)$, $n = 1, \ldots, X$
2: Construct neural network: $a(t), b(t), c(t), d(t)$
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize OPINN parameter: Φ
   Output layer: $a(t_n;\Phi)$, $\mathrm{b}(t_n;\Phi)$, $\mathrm{c}(t_n;\Phi)$, $\mathrm{d}(t_n;\Phi)$, $n = 1, \ldots, X$
3: Specify the training set
   $\kappa = \kappa_r \cup \kappa_b$ of the NN
4: Train the neural network
   Specify a loss function

$$\zeta(\omega;\kappa) = \zeta_t(\omega;\kappa_t) + \zeta_r(\omega;\kappa_r)$$

$$r_1(t) = \frac{dP_{NN}(t_n;\omega)}{dt} - \left(P_{NN}(t_n;\omega)(a(t_n;\Phi) - b(t_n;\Phi)Q_{NN}(t_n;\omega))\right)$$

$$r_2(t) = \frac{dQ_{NN}(t_n;\omega)}{dt} - \left(Q_{NN}(t_n;\omega)(c(t_n;\Phi)P_{NN}(t_n;\omega) - d(t_n;\Phi))\right)$$

   Minimize $min_{\kappa}\zeta(\omega;\kappa)$ using Adam Optimizer.
5: Return OPINN solution
   $P_{NN}(t_n;\omega)$ and $Q_{NN}(t_n;\omega)$, $n = 1, \ldots, X$
6: Return Parameter a:
   $a(t_n;\Phi), n = 1, \ldots, X$
7: Return Parameter b:
   $b(t_n;\Phi), n = 1, \ldots, X$
8: Return Parameter c:
   $c(t_n;\Phi), n = 1, \ldots, X$
9: Return Parameter d:
   $d(t_n;\Phi), n = 1, \ldots, X$

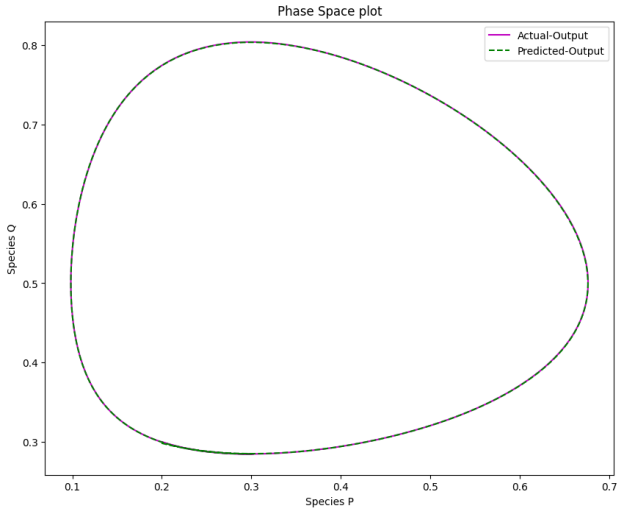


Figure 3.17: The phase space plot of the actual output of species P against species Q and the predicted output of species P against species Q of the Lotka-Volterra model.

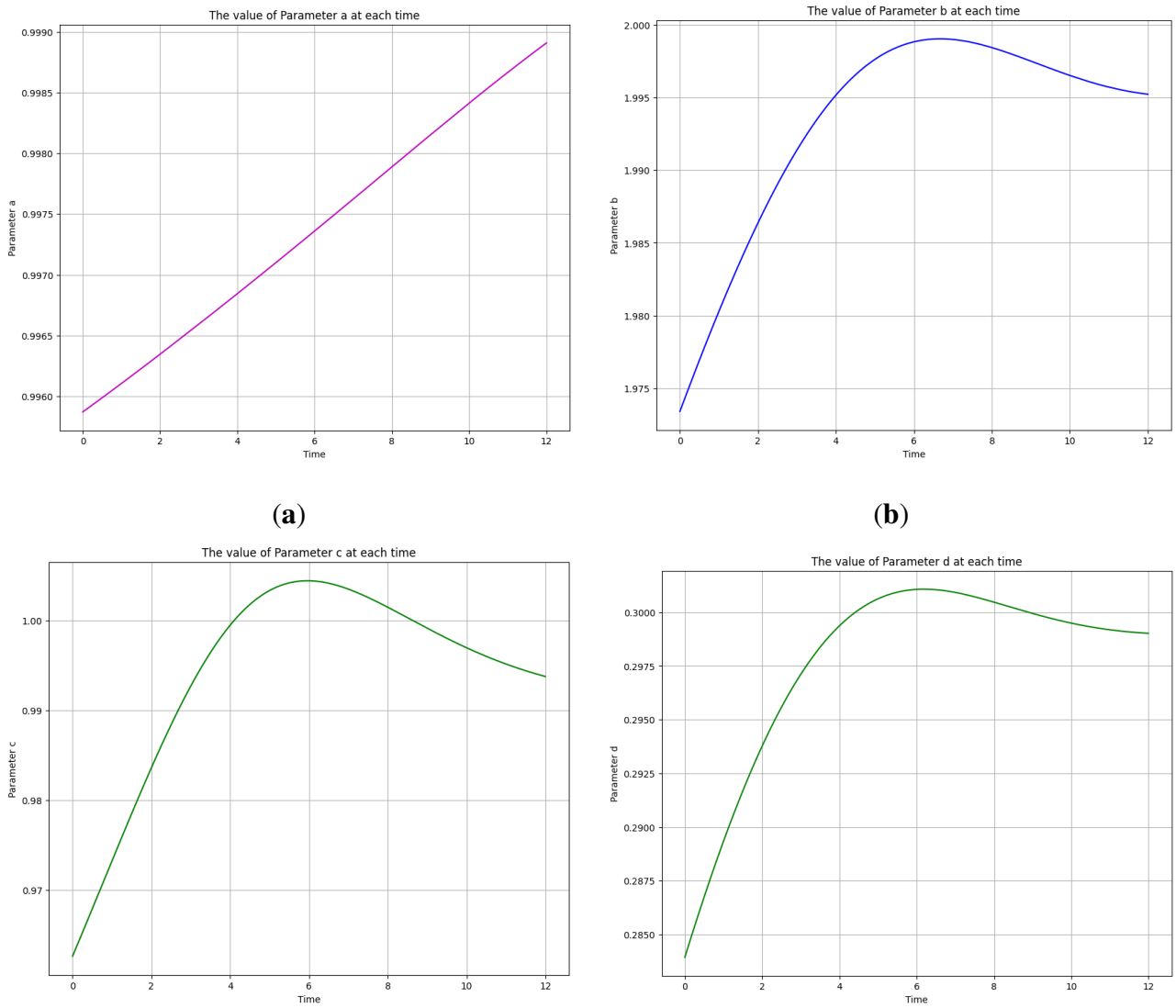


Figure 3.18: The learned time-varying parameter values of the Lotka-Volterra model. (**a**) Time-varying parameter a. (**b**) Time-varying parameter b. (**c**) Time-varying parameter c. (**d**) Time-varying parameter d.

Figure 3.19 plots the absolute error between the target and the Lotka-Volterra model's predictions. Tables 3.14 and 3.15 show the initial and the obtained parameters for the Lotka-Volterra model and the error metrics through the approaches described in Scenario 2.

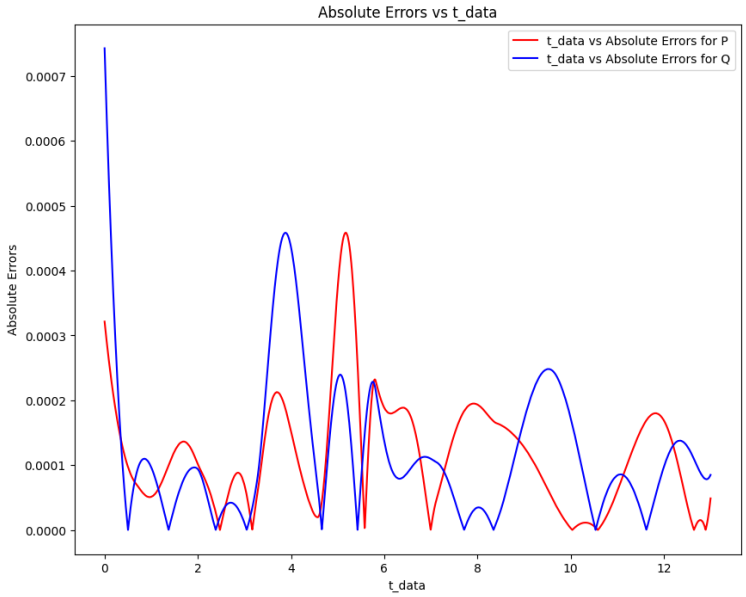


Figure 3.19: Absolute error plot between the data and OPINN solution of the Lotka-Volterra model.

Table 3.14: The parameter estimation of the Lotka-Volterra model.

	a	b	c	d
True Values	1.0	2.0	1.0	0.3
Approach 2	0.999136	1.9990387	1.000002	0.3
Error of Approach 2	0.000864	0.0009619	0.000002	0.00

Table 3.15: The error metrics of the Lotka-Volterra model.

	P	Q
MAE	2.3452×10^{-4}	2.4140×10^{-4}
MSE	8.5244×10^{-8}	1.0660×10^{-7}
RMSE	2.9197×10^{-4}	3.2649×10^{-4}

Table 3.16: Comparison of the obtained results using OPINN vs. those reported in the literature using DNN for the Lotka-Volterra model.

	True Values	**OPINN**	**DNN** [1]	**Epochs**
a	1.0	0.999876	0.9931	
				50,000
b	2.0	1.999771	1.9860	
				50,000
c	1.0	0.999869	0.9946	
				50,000
d	0.3	0.300004	0.2984	

3.3.5 SIR Model

The SIR model is a widely used mathematical model in epidemiology to understand the spread of infectious diseases within a population. It divides the population into three compartments: susceptible (S), infected (I), and recovered (R). In this model, individuals can transition between these compartments by interacting with infected others [62]. The SIR model assumes that the population size is constant, meaning no births, deaths, or migrations occur during the disease outbreak. Additionally, it assumes that individuals in the population mix randomly, and there is a homogeneous mixing pattern. A set of ordinary differential equations can describe the dynamics of the SIR model. Let us denote the number of susceptible individuals as $S(t)$, the number of infected individuals as $I(t)$, and the number of recovered individuals as $R(t)$. The following equations give the rates of change of these compartments over time:

$$\begin{aligned} \frac{dS}{dt} &= \frac{-\beta(t)S(t)I(t)}{N} \\ \frac{dI}{dt} &= \frac{\beta(t)S(t)I(t)}{N} - \gamma(t)I(t) \\ \frac{dR}{dt} &= \gamma(t)I(t). \end{aligned} \tag{3.28}$$

where variable $S(t)$, $I(t)$, $R(t)$, N, $\beta(t)$ and $\gamma(t)$ represents

$S(t)$: the numbers of susceptible individuals at time t

$I(t)$: the numbers of infected individuals at time t

$R(t)$: the numbers of individuals recovered at time t

N : the total population size.

$\beta(t)$: the transmission rate per time

$\gamma(t)$: the recovery rate per time.

The continuity equation is given by

$$N = S(t) + I(t) + R(t),\ t \geq t_0$$

where the initial conditions are denoted by $S(t_0) = S_0, I(t_0) = I_0$ and $R(t_0) = R_0$, where $t \geq t_0$ represents time in days and t_0 is the start date of the pandemic in the model. The SIR model provides insights into the dynamics of an infectious disease outbreak, such as the peak number of infected individuals, the duration of the epidemic, and the overall fraction of the infected population. These quantities depend on the model parameters β and γ values. It is important to note that the SIR model makes several simplifying assumptions. For instance, it assumes that the population is well-mixed, which may be false. We aim to learn the time-varying parameter $\beta(t)$ and $\gamma(t)$ of the SIR model from real-life data (COVID-19) using OPINN. Following the same procedure as in the previous application, we present OPINN Algorithm (9) with three networks to do this. The first network outputs are $S_{NN}(t_n;\omega)$, $I_{NN}(t_n;\omega)$ and $R_{NN}(t_n;\omega)$, which admits t as the input data. The second and third networks learn the parameters $\beta(t_n;\Phi)$ and $\gamma(t_n;\Phi)$ from the data, where ω and Φ represent the biases and weights of the network that minimize the loss function.

Algorithm 9 OPINN algorithm for learning the parameters of the SIR model

1: Construct OPINN

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: ω

Output layer: $S_{NN}(t_n;\omega)$, $I_{NN}(t_n;\omega)$ and $R_{NN}(t_n;\omega)$, $n = 1, \ldots, X$

2: Construct neural network: $\beta(t)$

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: Φ

Output layer: $\beta(t_n;\Phi), n = 1, \ldots, X$

3: Construct neural network: $\gamma(t)$

Specify the input: $t_n, n = 1, \ldots, X$

Initialize OPINN parameter: Φ

Output layer: $\gamma(t_n;\Phi), n = 1, \ldots, X$

4: Specify the training set

Training data: using cubic spline to generate $I(t_i), R(t_i); i = 1, \ldots, X$ given from the dataset.

5: Train the neural network

Specify a loss function

$$
\begin{aligned}
r_1(t) &= \frac{dS_{NN}(t)}{dt} - \left(\frac{-\beta(t_n;\Phi) S_{NN}(t_n;\omega) I_{NN}(t_n;\omega)}{N} \right) \\
r_2(t) &= \frac{dI_{NN}(t_n;\omega)}{dt} - \left(\frac{\beta(t_n;\Phi) S_{NN}(t_n;\omega) I_{NN}(t_n;\omega)}{N} - \gamma(t_n;\Phi) I_{NN}(t_n;\omega) \right) \\
r_3(t) &= \frac{dR_{NN}(t_n;\omega)}{dt} - \left(\gamma(t_n;\Phi) I_{NN}(t_n;\omega) \right)
\end{aligned}
$$

Minimize $min_{\kappa} \zeta(\omega;\kappa)$ using Adam Optimizer.

6: Return OPINN solution

$S_{NN}(t_n;\omega)$, $I_{NN}(t_n;\omega)$ and $R_{NN}(t_n;\omega)$, $n = 1, \ldots, X$

7: Return Parameter β:

$\beta(t_n;\Phi), n = 1, \ldots, X$

8: Return Parameter γ:

$\gamma(t_n;\Phi), n = 1, \ldots, X$

We define the residual loss $\zeta_r(\omega;\kappa_r)$ and training loss $\zeta_t(\omega;\kappa_t)$ as follows:

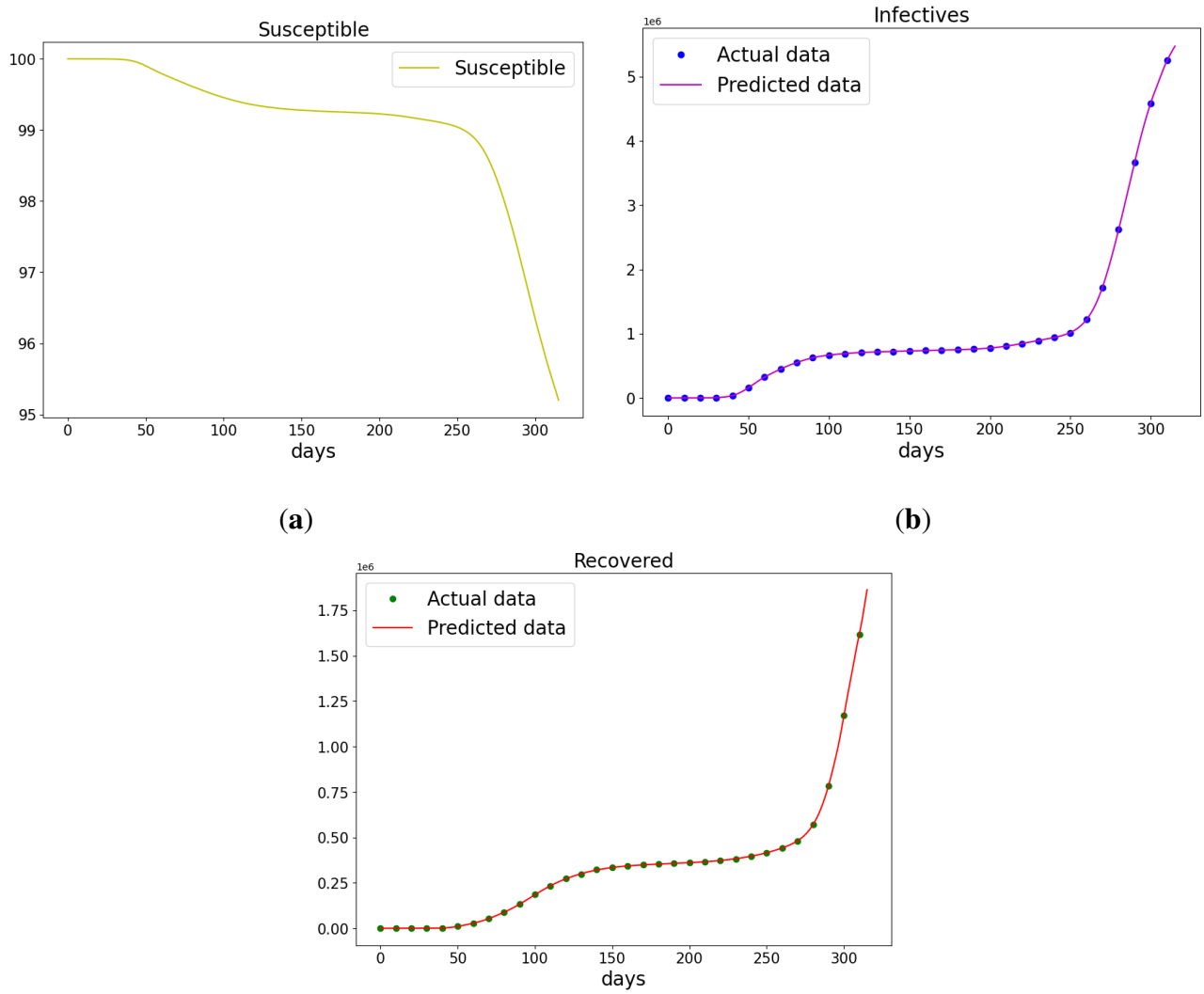
$$\zeta_r(\omega;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{3}\sum_{n=1}^{\kappa_r} ||r_i(t_n)||_2^2. \tag{3.29}$$

where

$$\begin{aligned}
r_1(t) &= \frac{dS_{NN}(t)}{dt} - \left(\frac{-\beta(t_n;\Phi)S_{NN}(t_n;\omega)I_{NN}(t_n;\omega)}{N}\right)\\
r_2(t) &= \frac{dI_{NN}(t)}{dt} - \left(\frac{\beta(t_n;\Phi)S_{NN}(t_n;\omega)I_{NN}(t_n;\omega)}{N} - \gamma(t_n;\Phi)I_{NN}(t_n;\omega)\right)\\
r_3(t) &= \frac{dR_{NN}(t)}{dt} - \left(\gamma(t_n;\Phi)I_{NN}(t_n;\omega)\right)
\end{aligned} \tag{3.30}$$

$$\begin{aligned}
\zeta_t(\omega;\kappa_t) = \frac{1}{\kappa_t}\Bigg(&\sum_{n=1}^{\kappa_t} ||S_{NN}(t_n;\omega) - S(t_n)||_2^2 + \sum_{n=1}^{\kappa_t} ||I_{NN}(t_n;\omega) - I(t_n)||_2^2\\
&+ \sum_{n=1}^{\kappa_t} ||R_{NN}(t_n;\omega) - R(t_n)||_2^2\Bigg).
\end{aligned} \tag{3.31}$$

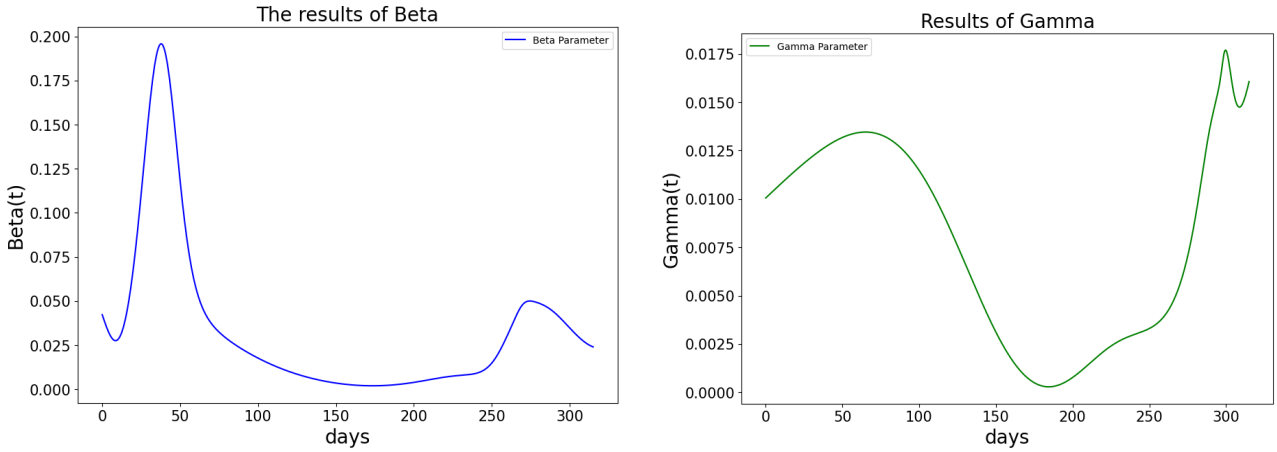
Here, we use data from Italy [**?**] starting from the date of the first reported cases to the day before vaccination data were reported, which is from 31st of January to 11 December 2020. We take the total population N to be 59.44 10^6 in Italy. Cubic spline interpolation generates 2000 training points from the cumulative infection and recovered data. The cumulative infections and recovered data are matched against the cumulative and recovered learned solutions. Algorithm 9 was implemented using publicly available COVID-19 data [**?**]. The parameters of the data using the SIR model and the learned cumulative infection and recovered data were obtained after using OPINN and the approach of Scenario 2 with five hidden layers, 64 neurons per layer, 100,000 epochs, the tanh activation function was used, and a learning rate of 10^{-3}. Figure 3.20 shows the learned solution of the SIR model, comparing the actual and predicted outputs of the infected and recovered populations. The cumulative data aligns closely with an exponential function. Figure 3.21 displays the learned values of $\beta(t)$ and $\gamma(t)$ using OPINN. The phase space plot in Figure 3.22 illustrates the relationship between the actual and predicted infected and recovered populations.



(**a**) (**b**)

(**c**)

Figure 3.20: The data and the learned SIR model using OPINN Algorithm 5 on COVID-19 data. (**a**) The susceptible graph. (**b**) The data and the learned infectives. (**c**) The data and the learned recovered population.



(**a**) (**b**)

Figure 3.21: The learned parameters of SIR model using OPINN Algorithm 5 on COVID-19 data. (**a**) The learned β. (**b**) The learned γ.

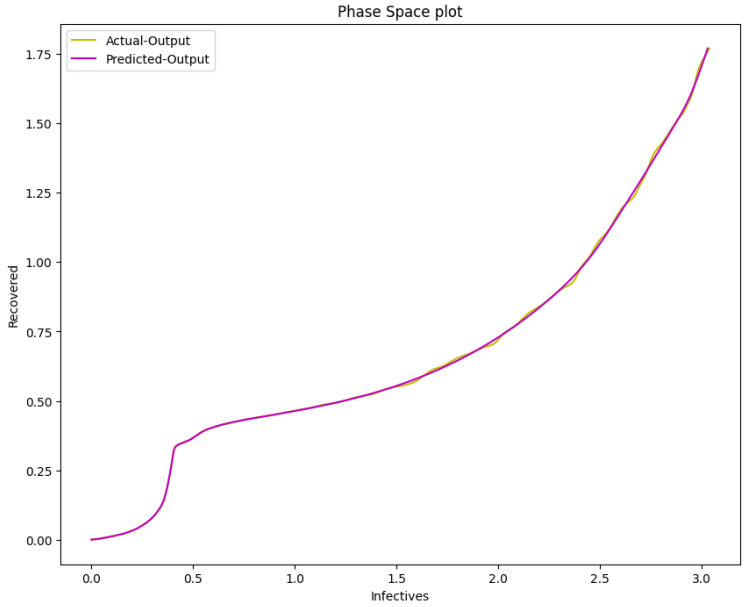


Figure 3.22: The phase space plot of the actual output of I against R and the predicted output of I against R of the SIR model from the COVID-19 data.

Figure 3.23 depicts the absolute error between the target and predicted values of the SIR model using OPINN. Table 3.17 summarizes the error metrics for the SIR model, indicating the accuracy of the OPINN solution. The graph displays the absolute errors for two datasets, "I" and "R" over time. After day 250, there is a noticeable spike in errors for both datasets. This surge might

be attributed to data inconsistencies, external factors impacting the measurements, or potential limitations in the predictive model if one was used. Additionally, increased variability in the data or unforeseen shifts in the underlying system around day 250 could also be factors. A thorough examination of the data collection process and any external events during that period is essential for a definitive conclusion. The metrics include mean absolute error (MAE), mean squared error (MSE), and root mean squared error (RMSE). These metrics show that the OPINN solution closely approximates the COVID-19 data, demonstrating the approach's effectiveness. Finally, learning time-varying parameters from real-life data using OPINN enhances our understanding of disease dynamics and can aid in making informed decisions regarding public health interventions. This approach can be applied to various infectious diseases, allowing for more accurate modeling and prediction of their spread. Combining the SIR model and OPINN presents a valuable tool for studying and managing epidemics.

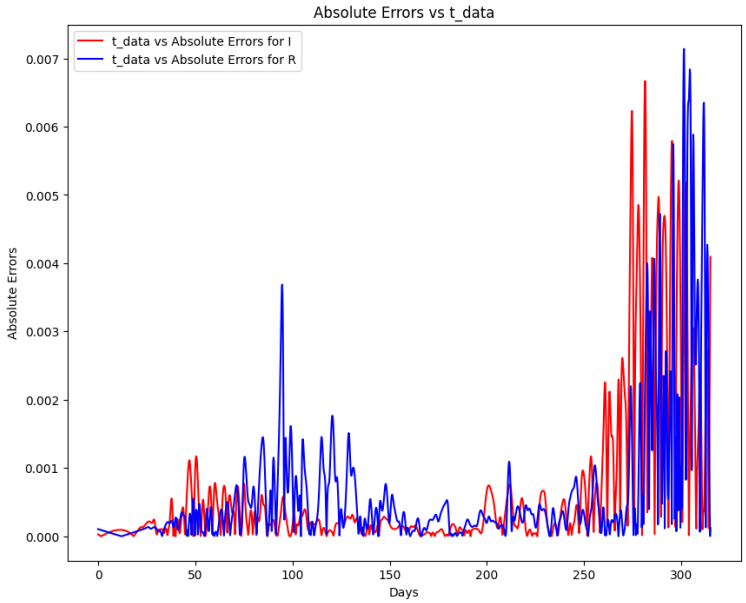


Figure 3.23: Absolute error plot between the COVID-19 data and the OPINN solution of the SIR model.

Table 3.17: The error metrics of the SIR model.

	I	R
MAE	1.3053×10^{-3}	1.2820×10^{-3}
MSE	6.9586×10^{-6}	6.0906×10^{-6}
RMSE	2.6379×10^{-3}	2.4679×10^{-4}

3.4 Conclusions

This study has systematically explored the capabilities of Optimized physics-informed neural networks in modeling dynamic systems, marking a significant contribution to the field. Our research has highlighted the enhanced computational efficiency and accuracy of Optimized physics-informed neural networks, especially when compared to traditional deep neural networks. Optimized physics-informed neural networks are very good at staying consistent with basic scientific laws because they use physical principles directly in the learning process. This is very important when working with complex dynamic systems. The novel contributions of this research lie in optimizing Optimized physics-informed neural networks for scenarios where the number of unknowns corresponds to the number of undetermined parameters. This approach has not only streamlined the process of dynamic system modeling but has also opened new avenues for research in this field. The application of our methodology to real-world scenarios, such as the COVID-19 pandemic model, further validates its practicality and relevance.

Quantitative comparisons have been a cornerstone of this study, providing clear evidence of the superiority of Optimized physics-informed neural networks over deep neural networks. We have demonstrated this through various error metrics. Additionally, computational efficiency has been quantitatively assessed, revealing that Optimized physics-informed neural networks significantly improve processing speed, an aspect critical for real-time applications. Finally, our results show that Optimized physics-informed neural networks have a lot of potential as strong, fast, and accurate scientific modeling tools that can deal with the complexities of dynamic systems. This

research opens up new possibilities for future studies. It shows how important it is to keep looking into and improving complex neural network designs for use in science. A key area for future work is to determine how to learn the time-dependent parameters of stiff dynamical systems.

CHAPTER 4

Learning the Time-varying Parameters of Stiff Dynamical Systems

4.1 Introduction

Stiff dynamical systems, characterized by equations with rapidly varying solutions over short time scales, play a crucial role in numerous scientific and engineering fields, from chemical kinetics to electrical circuits. However, these systems present significant challenges due to their sensitivity to numerical methods, requiring specialized algorithms to achieve stable and accurate solutions. Ordinary differential equations (ODEs) are fundamental mathematical modeling tools for studying and predicting dynamical systems in various scientific domains, including financial markets, biological models, chemical reactions, and climate models. Stiff ODEs are a family of equations in which some of the factors operate over different time scales or might cause changes in the behavior of the solution across short-term scales. The idea of stiffness in ODEs presents many difficulties for numerical techniques, particularly when the parameters change over time because of their sensitivity to step sizes. Accurately modeling and predicting the behavior of complex systems requires knowledge of these time-varying factors. Incorporating time-varying parameters when analyzing stiff ordinary differential equations has made it necessary to update traditional methods for solving stiff dynamical systems because they provide accurate predictions and adapt to any shifts in the system. Pioneering methods for parameter identification, as outlined by Bellman and strm[50], have been enhanced by advanced techniques such as Bayesian inference methods and Kalman filtering. These techniques enable the real-time adjustment of parameters using new data [33].

Machine learning (ML) and artificial intelligence (AL) methods have been increasingly used to solve ODE problems with non-stiff. For example, Oluwasakin et al. demonstrated how neural networks such as Optimizing Physics-Informed Neural Networks (OPINN) and Logistic-Informed Neural Networks (LINN) were able to learn the time-varying parameters of a dynamical system and that of omicron variant [43, 13] to provide accurate prediction and computational efficiency.

Extreme Learning Machine (ELM) by Yang et al. was used to solve ODEs where the weight and biases were chosen randomly, and the remaining unknown was computed by solving least squares with regularization [35]. Furthermore, physics-informed neural networks (PINN) have demonstrated a promising stride in integrating differential equations into neural networks [47]. This allows the networks to fit the data accurately, learn the parameters, and adhere to the equations. This approach employs neural networks to simulate nonlinear systems while minimizing the necessary data and limiting the model's search space with pre-existing knowledge, such as a system of differential equations. The fact that large-scale systems of ODEs lead to stiff ODEs arises in modeling an extremely wide range of dynamical system problems, and there is interest in developing new methods to solve the systems of stiff ODEs. Physics-informed Neural Networks, the above-mentioned neural networks, and more have difficulties solving problems when the dynamics are stiff. These failures occur because of the difficulties in solving gradient flow problems due to unbalanced back-propagated gradients during model training [75].

Many methods have been introduced to deal with stiff dynamical systems. Recently, Weiqi et al. used physics-informed neural networks for stiff chemical kinetics. This novel method of controlling stiffness in ODE systems uses the quasi-steady-state assumption (QSSA). This enables the PINN to handle either non- or slightly stiff systems by transforming them from their initial stiff condition [27]. Essential papers such as Chen et al. show how neural networks and machine-learning techniques can describe complicated, nonlinear systems and learn parameters from sparse or noisy data [23]. One benefit of these methods is that they can handle high-dimensional parameter spaces and the errors that come with real-world data. Hybrid techniques that use traditional number methods and machine learning with deep learning algorithms have become an interesting study area. These methods solve the systems of stiff equations because numerical methods are stable, and machine learning models are adaptable because they can learn from data and change based on time-varying parameters. Furthermore, Physics-informed neural networks with transfer learning have been used to predict nuclear reactor transients. The study shows that using physics-informed neural networks and transfer learning together makes modeling the dynamics of nuclear reactors

much more accurate [32]. The TL-PINN method uses pre-training on similar reactor transients to speed up and improve the accuracy of predictions for different transient scenarios. It shows up to two orders of magnitude faster prediction speeds than traditional methods.

Goswami et al. introduce neural network method for solving partial differential equations (PDEs) associated with complex problems such as Darcy flow and elasticity models[37]. Their approach uses deep transfer operator learning, which utilizes transfer learning to modify models trained on one set of conditions for another to address PDEs involving domain shifts. As a result, both computational efficiency and predictive accuracy improved in situations involving shifts in the domain. Also, Goswami et al. use extended deep neural operators to learn stiff chemical kinetics [6]. These methods offer a way to efficiently approximate the solution operator of differential equations, enabling the handling of complex, stiff chemical kinetics. However, learning time-varying parameters in stiff ordinary differential equations remains difficult despite great progress. Computational complexity and dimensionality give rise to issues, especially for high-dimensional systems with sparse or noisy data. It is also necessary to address philosophical and technical issues about the validity and reliability of ML and AL models to ensure their interpretability and integration into traditional scientific frameworks.

Many studies on the solution of stiff ODEs have demonstrated good numerical approximation accuracy of their proposed framework over short time intervals and grid sizes. However, no indication is given about learning time-varying parameters of the stiff ODEs or the computational time required by widely solving highly stiff problems. We propose a numerical framework based on Physics-Informed Transfer Learning Neural Networks (PITLNN) for solving initial value problems of ODEs and learning the time-varying parameters of stiff problems. PITLNNs are a family of networks, including physics-informed networks, deep operator networks (DeepONets), and transfer operator learning frameworks [47, 37, 60]. These networks have excelled in solving PDEs and ODEs, which offers a bridge between machine learning techniques and traditional numerical methods. Recently, Wang et al. revealed the failure of PINNs to solve problems with stiff dynamics [75]. The failure occurs during training, where the gradients become unevenly distributed,

which leads to potential failure in solving stiff problems accurately. Hence, we propose a multi-layer feedforward neural network with a framework that integrates neural network-based function approximation with physics law constraints and transfer learning with parameters that are properly uniformly initialized through the application of Xavier. The Xavier uniform initialization is designed to keep the scale of the gradients roughly the same in all layers. This helps prevent the gradient from vanishing in the networks and ensures that the parameters are spread evenly across a certain range, which is calculated to maintain a variance that supports an optimal gradient flow.

The PITLNN scheme constitutes specific functionalities to solve differential equations while incorporating physics laws, multi-network architecture, transfer learning, regularization, and adaptive optimization. Foundational principles and technological advances support the feasibility of schemes. Central to this is the universal approximation theorem, which establishes that neural networks, with adequate depth or width and suitable activation functions, can approximate any continuous function with high accuracy, making them capable of modeling the complex functions that underpin physical systems. In addition to this theoretical foundation, the physics-informed learning method directly incorporates recognized physics principles into the loss function to guarantee data-driven model predictions. This reduces the number of possible solutions to physically feasible ones, improving the model's generalizability. To strengthen this framework, transfer learning lets the model draw on data from similar jobs to improve learning efficiency. This is especially useful when there is limited data for the target activity. Furthermore, regularization helps avoid overfitting, maintains resilient models, and ensures reliable generalization to new data. We compute the derivatives by combining specific functions from the neural network. The Newton-Raphson method was used to solve nonlinear equation systems with no exact solutions. This method helps us tackle complex equations, like finding the best fit in a complicated puzzle. This strategy allows us to obtain the best way to calculate by collocating a neural network function to approximate the exact solution at any given point within the domain.

The presence of a gradient exploding and vanishing was handled by clipping and normalizing the gradient, which directly controls the gradient magnitude. In addition, we employ a mini-batch

gradient that uses a random choice sample of the data at the start of each iteration to improve the training process's stability and robustness, thereby computing losses for each batch, performing backpropagation, and updating the model's parameters. It also tracks and returns metrics like total loss, data loss, physics loss, and gradient norms. To demonstrate the efficiency of the proposed method, we choose three stiff ODE problems of different dimensions, namely the first-order irreversible chain reactions [58], in which an analytical solution exists, Brusselator [25, 43, 13], in which the exact solution does not exist. Lastly, biomass transfer, a system of three nonlinear ODEs, involves the movement of organic material from one organism to another within an ecosystem. The performance of the proposed models is assessed in terms of time-varying parameters, computational times, and approximation accuracy. Furthermore, our technique provides solutions directly as a function that can be evaluated at every domain point. Finally, we present new deep-learning numerical algorithms for solving stiff IVPs with solutions that may contain shape gradients.

The paper is organized as follows: Section 2, we introduce the system of stiff ordinary differential equations. In Section 3, we discuss the preliminary calculations that cover the foundation of our method. Section 4 presents our proposed methodology, detailing the development and implementation of PITLNNs for tackling stiff ODEs. Results and discussions are presented in Section 5, where we evaluate the performance of PITLNNs across several problems, demonstrating their efficacy and robustness in solving complex dynamical systems. Finally, Section 6 concludes the paper with a summary of our findings.

4.2 Stiff Ordinary Differential Equation

Dynamic system problems can be formulated as

$$\frac{d\Theta}{dt} = f(t,\Theta,\alpha), \quad and \quad t \in (t_0, t_{end}), \tag{4.1}$$

$$\Theta(t_0) = \Theta_0$$

where the function f and the initial values Θ_0 are known, and the function $\Theta(t)$ are the unknowns. Simplifying the notation, we group the functions $\Theta(t)$ in a vector function $\Theta(t) : \mathbb{R} \longrightarrow \mathbb{R}^m$ and the functions $f(t, \Theta_1, \Theta_2, ..., \Theta_m)$ in $f(t, \Theta)$. The ODE integrator can numerically solve (4.1). However, the unknown solution may exhibit shape gradients, including complex behavior, which may result in difficulty implementing numerical methods. These difficult problems are usually the ones where the so-called stiffness arises. When dealing with stiff ODEs using explicit methods, taking extremely small time steps is necessary, leading to a process requiring a lot of computation. Alternatively, one might use implicit ODE integration methods like the Backward Differentiation Formula for such problems.

Nonetheless, addressing stiff ODEs often takes a lot of time because the implicit approach typically requires Newton's method to solve nonlinear systems. As a result, finding efficient solutions for stiff ODE systems remains a significant area of ongoing research. Up to this point, "stiffness" lacks a clear, universally accepted definition. As Lambert [24] suggests, stiffness should be viewed more as a behavior displayed by the system rather than an inherent characteristic of it. Typically, an ODE problem is considered stiff if it includes solutions that change very slowly alongside solutions that change much more rapidly, necessitating many small steps by explicit numerical methods to achieve precise and trustworthy outcomes. It is challenging to define stiffness in terms of chemical kinetic models. Nevertheless, one way to consider it is by examining the different time scales of various species within the model. Using an implicit integrator requires less computational effort than using explicit ones, which implies that there is severe stiffness within the problem. The other approach would define stiffness as a function of constant parameters with a stiffness ratio. This can be demonstrated by considering a system with constant coefficients in a linear homogeneous context.

To highlight the stiffness, let's consider the linear dynamical system described as follows in $\mathbb{R}^m$:

$$\frac{d\Theta}{dt} = \gamma\Theta + f(t), \tag{4.2}$$

where $f : \mathbb{R} \to \mathbb{R}^m$ denotes a nonlinear function, and $\gamma \in \mathbb{R}^{m\times m}$ is a constant, diagonalizable matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ (assumed to be distinct) along with corresponding eigenvectors $c_1, c_2, \ldots, c_m$. Furthermore, it is assumed that:

$$\mathrm{Re}(\lambda_i) < 0, \quad \text{for} \quad i = 1, 2, \ldots, m. \tag{4.3}$$

This formulation sets the stage for discussing the system's stiffness by considering the rate at which solutions to the differential equation decay, as governed by the real parts of the eigenvalues of the matrix γ. The presence of distinct eigenvalues with negative real parts suggests that the system's solutions tend toward stability. However, the concept of stiffness emerges when considering the magnitude of the eigenvalues. Specifically, a system is deemed stiff if the ratio of the magnitudes of the largest to the smallest eigenvalues (in absolute value) is significantly large. This ratio, known as the stiffness ratio, is defined as:

$$\text{Stiffness Ratio} = \frac{|\mathrm{Re}(\lambda_{\max})|}{|\mathrm{Re}(\lambda_{\min})|}, \tag{4.4}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the eigenvalues of γ with the largest and smallest real parts in absolute value, respectively. It is difficult to determine all dynamics accurately when a large stiffness ratio means that there are components in the system solution that decay at very different rates.

Explicit methods face challenges of this kind during numerical integration because they will need extremely small time steps to be stable as well as accurate. Therefore, implicit methods that are usually more stable for stiff systems frequently involve solving a system of nonlinear equations at each step, making them more computationally expensive but able to take longer steps. As such, choosing an appropriate numerical method for solving stiff ODEs must balance the computational efficiency and accuracy of solutions.

4.3 Physics-Informed Transfer Learning Neural Network

Physics-Informed Transfer Learning Neural Network is a neural network that utilizes the concepts of learning from prior knowledge and embedding physics law directly into the learning process. In other words, PITLNN is based on the concept of transfer learning combined with PINNs to learn the time-varying parameters of a dynamical system. One fundamental element of this system is transfer learning, which enables the network to leverage knowledge obtained from one task to enhance the learning of another related task.

This means pre-training the network on simulated data produced by solving differential equations under different conditions before fine-tuning on real experimental/observational data. It starts by assimilating information in dataset D_s generated by simulators that solve differential equations:

$$D_s = \{(t_i, y_i) | t_i \in T, y_i = f(t_i)\}$$

where T refers to input variables, y_i stands for outputs of simulations given various values for different parameters, and $f(t_i)$ represents solutions to differential equations under several conditions.

This phase then utilizes the transfer learning approach as an application of knowledge gained from D_s towards improving studying using another new related dataset D_r obtained from actual experiments or observations:

$$D_r = \{(t_j, y_j) | t_j \in T, y_j \in Y\},$$

with T being the observed outputs. The network adjusts its parameters through both datasets to minimize the discrepancy between its predictions and the actual data:

$$Minimize : L(D_r, D_s) + \lambda P(\beta) + \mu\Gamma$$

where L is a loss function measuring the difference between the network's predictions and the actual data from both D_r and D_s, $P(\beta)$ represents the physics laws as part of the learning process,

λ is a weighting factor that balances the importance of adhering to physics laws against fitting the data. Therefore, the $\lambda P(\beta)$ is the physics loss (based on the differential equations governing the reaction kinetics), and $\mu\Gamma$ is the regularization loss (to maintain continuity with the pre-trained parameters) where μ is very small.

The objective of the PITLNN is to learn the time-varying parameters of $\alpha(t)$ that approximates the computational model, offering the capability to assess the solution $\Theta(t)$ at any given point within a continuous time frame. The solution of PITLNN for an input vector $t_n \in t$ is given as:

$$\Theta(t_n, \beta^*, \alpha) = \sum_{j=1}^{p} \Theta_j(t_n) \cdot \alpha_j(t_n), \tag{4.5}$$

where Θ_1, Θ_2, ..., Θ_p are the output solution of the network and α_1, α_2, ..., α_p are the output of the time-varying parameter. The transfer pre-training parameter β^* are optimized by minimizing the training, residual, and regularization loss.

4.3.1 Core Components

A novel framework that addresses the complexity of stiff differential equations is the PITLNN. A complex neural network with input, hidden, and output layers forms the basis of the PITLNN. For the network to successfully approximatively solve problems, these layers collaborate to abstract and gradually convert the incoming data into representations. The mathematical equation that captures the essence of this transformation could be written as $y = f(Wx + b)$, where x is the input, y is the output, W and b are the network's weights and biases, and f symbolizes the activation function. Residual loss is a key part of the PITLNN design because it directly uses physics laws in the learning process. This integration is shown mathematically by a loss term: $L_{physics} = ||P(y_{predicted}) - y_{true}||^2$; where P is the set of differential equations that govern the physical system and $y_{predicted}$ is the predicted value of y. This word ensures that the network's plans align with physics laws, making them more accurate and reliable.

The framework optimizes its performance by using a transfer learning technique. The network is first pre-trained on a task closely related to the target problem, often using synthetic or simu-

lated data generated by known NN models. Optimizing a pre-training loss function, denoted as $\mathrm{L}_{\mathrm{pretrain}} = ||y_{\mathrm{simulated}} - y_{\mathrm{predicted}}||^2$, governs this phase where the simulated dataset is the source of $y_{\mathrm{simulated}}$. Afterward, the target task is used to fine-tune the pre-trained biases and weights, using the previously learned information to enhance learning efficiency and results. A composite loss function, denoted as

$$L_{\mathrm{total}} = L_{\mathrm{data}} + \lambda L_{\mathrm{physics}} + \mu L_{\mathrm{regularization}} \tag{4.6}$$

where L_{data} measures the fit to the target data, L_{physics} enforces physics laws, $L_{\mathrm{regularization}}$ prevents overfitting, and λ and μ are balancing coefficients. A physics-informed layer enforces adherence to differential equations, a deep neural network can capture complex behaviors, and a transfer learning mechanism leverages pre-acquired knowledge; this architecture positions the PITLNN as a powerful tool for modeling and solving problems characterized by stiff differential equations.

4.3.2 Training Procedure

The PITLNN is trained through a very deliberate method, that starts with pre-training on a task closely related to the target problem. This first step typically uses a dataset generated from simulated scenarios with known exact solutions to the underlying physical models represented symbolically as optimizing.

$$L_{\mathrm{pretrain}} = \frac{1}{N}\sum_{i=1}^{N}(y_{\mathrm{sim},i} - \hat{y}_{\mathrm{sim},i})^2,$$

where $y_{\mathrm{sim},i}$ are the simulated outputs and $\hat{y}_{\mathrm{sim},i}$ are the network's predictions. Next, we perform fine-tuning of the network on the actual dataset corresponding to the target problem.

In this process, we adjust the network parameters to minimize L_{total}, which is a composite loss function. Adam or L-BFGS algorithms are employed for efficient navigation of complex loss landscape during optimization, especially in the search for network parameters that minimize L_{total}. The training process considers adaptive learning rate methods tailored towards dealing with sys-

tems that may be stiff while modeling using PITLNN since they help in controlling learning speed depending on problem dynamics and gradient clipping given by $min(\frac{values}{gradient}, 1))$ which solves exploding gradients issue when denoting. Also, the physics-informed loss component is emphasized more for stiff systems:

$$L_{\text{physics}} = \frac{1}{M} \sum_{j=1}^{M} \|P(x_j, \hat{y}(x_j))\|^2,$$

This ensures that our model's predictions satisfy the governing physical equations for different temporal scales, making the model more robust and accurate. To overcome the difficulties of modeling stiff differential equations, the PITLNN uses a comprehensive training strategy that includes pre-training, fine-tuning, and advanced optimization methods.

4.4 Handling Stiffness

To manage stiffness in differential equations effectively, PITLNN can incorporate adaptive time-stepping mechanisms within the training process. This lets the network adjust its time resolution dynamically so that computations are concentrated on intervals with rapid changes exhibited by systems while larger steps are used in more stable regions. For the PITLNN framework to navigate the challenges presented by stiff differential equations, it has a mechanism of adjusting time steps during training referred to as an adaptive time-stepping method. This novel approach enables the system to adapt its temporal resolution throughout learning by modifying t based on the local behavior of the solution.

$$\Delta t_{\text{new}} = \Delta t \cdot \min\left(\max\left(\frac{E}{\varepsilon}, r_{\min}\right), r_{\max}\right).$$

Here E refers to the estimated error at the current step; represents the desired accuracy level while $r_{\min}$ and $r_{\max}$ are predetermined factors which smoothens out abrupt changes when altering step size so that it does not vary too much from one iteration step to another. Through this technique, computation resources are intelligently distributed by taking smaller time steps t in areas

with rapid variations for better-capturing dynamics and larger ones near stability zones to enhance speed without sacrificing precision.

This kind of flexibility towards temporal resolutions plays a great role during the simulation of stiff systems since some processes may take different amounts of time depending on their nature. In other words, adaptive time stepping guarantees that all parts within the domain of interest can be simulated accurately without compromising too much on computational efficiency for rest periods in between oscillations being far apart from each other when viewed over long-term durations. This significantly widens the range over which network models physical behavior governed by stiff ODEs as it provides an effective solution to numerical analysis of dynamic systems which has always been difficult. At the same time, such an ability greatly improves the ability of PITLNNs to learn about physical systems responding to stiff differential equations because they are able to cope with them robustly and predict their behavior accurately.

4.4.1 Adaptive Scheme

The adaptive scheme within PITLNN for handling stiff systems of differential equations employs a dynamic adjustment of the time-stepping mechanism throughout the training process. This adaptive time-stepping is critical for efficiently modeling systems where the solution's behavior changes drastically over different time scales. Here is a detailed breakdown of the principles behind this scheme:

- **Adaptive Time-Stepping Mechanism:** The adaptive time-stepping mechanism dynamically adjusts the size of time steps Δt based on the local temporal behavior of the solution. The goal is to use smaller time steps when the solution changes rapidly (to capture the dynamics accurately) and larger steps when changes are gradual (to reduce computational cost). The adjustment can be mathematically represented as follows:

$$\Delta t_{\text{new}} = \Delta t \cdot \text{adjustment\_factor},$$

where the adjustment_factor is calculated based on the estimated local error E and a prede-

fined tolerance τ. The local error estimation could involve evaluating the difference between solutions obtained with two consecutive steps or using a higher-order method for comparison.

Error Estimation and Step Size Adjustment: The adjustment factor is determined through error estimation and control strategy, often guided by:

$$\text{adjustment\_factor} = \sqrt{\frac{E+\varepsilon}{\tau}},$$

where ε is a small number added to prevent division by zero. The tolerance τ reflects the desired accuracy, and E represents the estimated error at the current step. The square root is a conservative measure to ensure stability, but the exact formula might vary based on the stiffness characteristics and the specific solver's requirements.

- **Incorporating Physics-Informed Constraints:** Within the PITLNN framework, the adaptive time-stepping is seamlessly integrated with physics-informed constraints. This integration ensures that each time step adheres to computational efficiency criteria and respects the underlying physics laws modeled by the differential equations. The physics-informed loss component, crucial for embedding these constraints, is defined as:

$$L_{\text{physics}} = \frac{1}{N}\sum_{i=1}^{N} \|F(X_i,t_i;\theta) - S(X_i,t_i)\|^2,$$

where F represents the neural network's approximation of the solution, S denotes the source term or any additional terms from the differential equation, and θ includes the parameters (weights and biases) of the network.

- **Transfer Learning Adaptation:** The adaptive method works well with the transfer learning system. In this mechanism, pre-trained parameters provide initial approximation within solution space for stiff systems. PITLNN gains foundational knowledge through this pre-training executed on simpler or related tasks. This understanding is important as adaptive

time-stepping allows the network to be better fine-tuned. Transfer learning refers to moving a source models parameters represented by Θ_S into a target model represented by Θ_T. Each parameter in these models is denoted as θ_{S_i} for the source and θ_{T_i} for the target where i indexes them. The process can be written as:

$$\forall i, \theta_{T_i} \leftarrow \theta_{S_i}$$

This denotes that each parameter θ_{T_i} of the target model Θ_T is set to be equal to the corresponding parameter θ_{S_i} of the source model Θ_S, ensuring that the gradient computation history remains unaltered. The reason why this approach copies parameters strategically is that it initializes the target model near an optimal setting for the new task, thus speeding up the fine-tuning process. It saves training time and improves the model's performance on difficult tasks by starting optimization from more informed points in the solution space.

- **Optimization and Training:** The training process optimizes the combined loss, including the data loss, physics-informed loss, and a regularization term, under the adaptive time-stepping regime. Advanced optimization algorithms like Adam are employed to find the optimal parameters θ that minimize the total loss shown in (4.6).

This comprehensive, adaptive scheme enables PITLNN to tackle the challenges of stiff differential equations, ensuring high accuracy while maintaining computational feasibility.

4.4.2 Convergence

We explore the effectiveness of PITLNN in approximating complex functions governed by stiff differential equations. Leveraging the Universal Approximation Theorem (UAT), PITLNNs integrate physics-informed neural networks and transfer learning to enhance convergence. The architecture captures the underlying physics laws within its framework and utilizes transfer learning to quickly adapt pre-trained weights and biases, facilitating rapid convergence and robust function approximation. These features allow the network to fit the data approximately and learn the time-vary

parameter.

Universal Approximation Theorem (UAT): Given a continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ on a compact subset $K \subseteq \mathbb{R}^n$ and any desired accuracy $\varepsilon > 0$, there exists a output, denoted as Θ, such that for every input $x \in K$, the output $\Theta(x)$ approximates the function $f(x)$ within an error less than ε. This is mathematically represented as:

$$\exists \Theta : \sup_{x \in K} \|\Theta(x) - f(x)\| < \varepsilon.$$

Here, $f(x)$ represents the target function we aim to approximate.

Proof

Let $\Theta(x) = \sum_{j=1}^{N} c_j \sigma(w_j^T \cdot x + b_j)$, where N is the number of neurons, c_j are output weights, w_j are input weights, b_j are biases, and σ is a non-linear activation function.

Therefore, for σ satisfying certain conditions (e.g., non-constant, bounded, and continuous), it is possible to select N, c_j, w_j, and b_j such that:

$$\forall \varepsilon > 0, \exists \Theta : \sup_{x \in K} \|\Theta(x) - f(x)\| < \varepsilon.$$

This implies that for any continuous function f and any $\varepsilon > 0$, there exists a neural network g such that g approximates f within an error of ε over the domain of interest, demonstrating the universal approximation capability of neural networks with at least one hidden layer [16, 30].

Convergence Theorem for PITLNN: Given a continuous target function $f : \mathbb{R}^n \to \mathbb{R}$ represents the solution to a stiff differential equation system. Consider a PITLNN with output $F(X, \alpha(t); \beta)$ that combines physics-informed constraints and adaptive learning mechanisms. $\alpha(t)$ represents time-varying parameters influenced by differential equations, while β represents optimized weights and biases during pre-training and fine-tuning. A configuration of β exists for any series of adaptively determined parameters $\alpha(t)$ and a careful selection of β such that:

$$\lim_{m\to\infty} \|F(X,\alpha(t);\beta) - f(X)\| = 0 \text{ with probability 1},$$

where m represents network complexity (e.g., layers or neurons) and $\|\cdot\|$ represents a norm comparing PITLNN output to target function f.

Proof:

Given that $\Theta(x)$ is the output of the neural network, defined as

$$\Theta(x) = \sum_{j=1}^{N} \beta_j \sigma(W_j x + b_j),$$

where σ is the activation function and $f(x)$ is the target solution function. Then, the integration of adaptive learning and physics-informed learning such that

$$L_{\text{total}} = \|\Theta(x) - f(x)\| + \lambda \|P(\Theta(x)) - s(x)\|^2 + \mu \|\beta - \beta_{\text{pre}}\|^2,$$

where λ and μ are regularization parameters, P represents the physics-informed constraints, and $s(x)$ is the known solution or source term. The optimization process seeks to minimize the total loss:

$$\arg\min_{\beta,\alpha} L_{\text{total}}.$$

highlighting the adaptive adjustment of parameters to enhance model accuracy. Therefore, For any $\varepsilon > 0$ where β, is the pre-trained weights from a task related to the target problem and $\alpha(t)$, the time-varying parameters for adapting to the temporal dynamics of stiff systems there exists a network complexity m such that:

$$\forall \varepsilon > 0, \exists m : \sup_{X\in\mathbb{R}^n} \|\Theta(X,\alpha(t);\beta) - f(X)\| < \varepsilon,$$

ensuring the PITLNN can approximate any continuous function $f(x)$ within an error margin ε. Then as the network complexity m approaches infinity, the output of the PITLNN converges to the

target function $f(x)$ across the domain with probability 1:

$$\lim_{m\to\infty} \sup_{X\in\mathbb{R}^n} \|\Theta(X,\alpha(t);\beta) - f(X)\| < \varepsilon$$

Thus

$$\lim_{m\to\infty} \sup_{X\in\mathbb{R}^n} \|\Theta(X,\alpha(t);\beta) - f(X)\| = 0 \text{ with probability 1.}$$

This structured proof explains how the PITLNN framework, including architecture, physics-informed learning, adaptive time-stepping, and transfer learning, ensures convergence to the target solution f(x) using the UAT.

4.5 PITLNN Framework Architecture

The main objective is to earn the time-varying parameter of a stiff dynamical system (4.1) using a neural network framework called PITLNN. This can be conducted by finding the best network parameters, β^*, representing the biases and weights in the network that minimize the loss function. To do this, we offer two network architectures, as shown in Figure 4.1.

4.5.1 First Network Architectures

The first network (PITLNN pre-training) outputs pre-training weights and biases, denoted as $\beta^*(t_n)$ for $n = 1,\ldots,X$. These weights and biases are subsequently transferred to the second network. The first network receives t_n^*, comprising simple pre-training data. This data is generated by solving a system of ordinary differential equations that model reactions or species interactions.

The system parameters, represented by α, are varied slightly to generate multiple training datasets. To systematically vary α, we define a range for each parameter around their nominal values using linear spacing. For example, if α_0 denotes the nominal values, we introduce variations by adding an incremental value δ, sampled from a set of linearly spaced values centered at zero. The parameter α varied as follows:

$$\alpha = \alpha_0 + \alpha_k$$

where α_k is sampled from the set $\{-\gamma, 0, \gamma\}$. Here, γ represents a small deviation from the nominal value chosen to provide moderate variation and determine what happens to the model under slightly different physical condition alterations. For every combination of α_k, the differential equation system is solved to generate corresponding solution data sets. These datasets serve as training data for one instance of our model, thereby exposing it to different dynamics and thus widening its scope of learning.

After being trained on these diverse conditions, transfer learning is employed, where learned parameters enhance generalization ability through better fitting into the main model. This approach seeks to give more initial knowledge about dynamics so that the system can be trained accurately and efficiently. The training loop uses a custom loss function that combines data loss (4.10) and residual loss (4.8). The process ends by saving parameters ($\beta(t_n)^*$) after pretraining and then saving the main model that will be transferred to the second network (PITLNN main), which will enable it to learn the time-varying parameter and, at the same time provide the solution to the dynamical system. In other words, this approach employs transfer learning to refine the accuracy and robustness of the main model.

4.5.2 Second Network Architectures

The second network employs two networks, which are used to obtain the solution to the stiff ODE and learn the time-varying parameters of the dynamical system. Both networks admit the main data t with the transfer parameters β^* from the previous network (PINN) and output $\Theta(t_n, \beta^*)$, $n = 1, ..., X$, the solution to the systems, and $\alpha(t_n, \beta^*)$, $n = 1, ..., X$, which is the learned time-varying parameter of the dynamical system. Finally, the PITLNN outputs $\Theta(t_n, \alpha(t_n), \beta^*)$, $n = 1, ..., X$, which represents the solution to the dynamic system for each time, helping us understand how the reactions or systems change and behave over time. The training data (main data) is generated by

numerically solving the ODE system using the backward differentiation formula (BDF) method for stiff problems.

The solution from this numerical solver serves as the ground truth for training the PITLNN. The network is trained to minimize a custom loss function that incorporates both data loss and residual loss, along with a regularization loss, to prevent overfitting, which allows PITLNN to capture the interactions between the compartments. We define the residual loss, training loss, and regularization loss used to minimize the loss function as follows:

$$\xi(\beta^*;\kappa) = \xi_t(\beta^*;\kappa_t) + \theta\xi_r(\beta^*;\kappa_r) + \gamma\xi_r(\beta^*;\kappa_\omega) \tag{4.7}$$

$\xi_t(\beta^*;\kappa_t)$ is called the training loss, $\xi_r(\beta^*;\kappa_r)$ is called the residual loss and $\xi_r(\beta^*;\kappa_\omega)$ is called the regularization loss with $\theta,\gamma > 0$. $\kappa = \kappa_r \cup \kappa_t \cup \kappa_\omega$ is the total training dataset. We define the following:

$$\xi_r(\beta^*;\kappa_r) = \frac{1}{\kappa_r}\sum_{i=1}^{p}\sum_{n=1}^{\kappa_r} |r_i(t_n)|^2 \tag{4.8}$$

$$r_i(t_n) = \frac{d\Theta(t_n;\beta^*)}{dt} - f\Big(t_n, \Theta(t_n;\beta^*), \alpha(t_n;\beta^*)\Big) \tag{4.9}$$

$$\xi_t(\beta^*;\kappa_t) = \frac{1}{\kappa_t}\left(\sum_{n=1}^{\kappa_t} ||\Theta(t_n;\beta^*) - \Theta(t_n)||^2\right) \tag{4.10}$$

$$\xi_\omega(\beta^*;\kappa_\omega) = \frac{1}{\kappa_\omega}\left(\sum_{n=1}^{\kappa_\omega} ||\beta(t_n) - \beta^*(t_n)||^2\right) \tag{4.11}$$

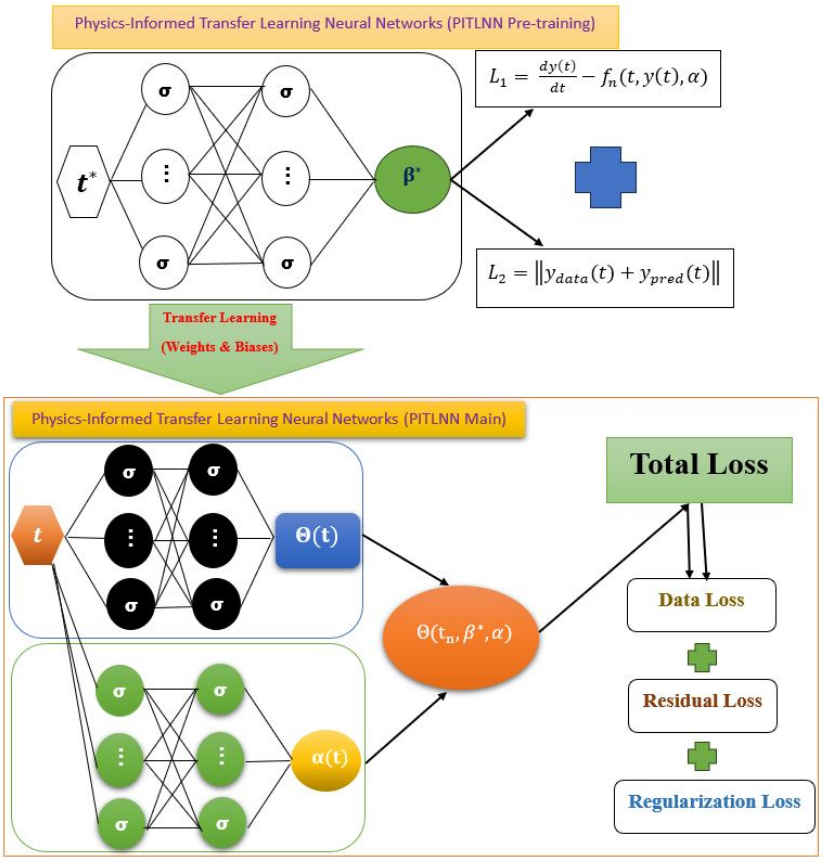


Figure 4.1: Schematic diagram of the PITLNN.

The network is trained to minimize (4.7); an optimizer such as Adams is the most optimal weight and bias that would minimize (4.7). The neural network also obtains the optimal values for the model parameters. The $\Theta(t_n)$ represents the loss function's training data. We identify the parameter α by solving (4.9). The PITLNN algorithm 11 is an excellent choice for learning the solution and the time-varying parameter of a stiff dynamic system of the ODE model, which is robust enough to adjust to whatever information is present in the data. We employed hidden layers with neurons each to achieve good accuracy in the neural network. As a result, the training loss was minimized using the number of epochs. The Gaussian error linear (GELU) function is the activation function for the hidden and output layers. PITLNNs were implemented using Python and Pytorch, a highly flexible and comprehensive tool for creating and training neural network models.

Algorithm 10 PITLNN Pre-training algorithm for learning the optimal parameters and dynamic system behavior.

1: Construct PITLNN Architecture
   Specify the input: $t_n, n = 1, \ldots, X$
   Initialize PITLNN parameters: layers for main network and parameter sub-networks
2: Define neural network architecture
   Construct main and parameter-specific sub-networks
   Use hidden layers with GELU activation and Xavier initialization
3: Generate training data for pretraining
   Define parameter ranges for α:
   $\alpha = \alpha_0 + \delta_c, \delta_c \in [\delta_{\alpha_{\min}}, \delta_{\alpha_{\max}}]$
   Solve ODE for each α pair using a numerical solver
   Train separate models on these datasets
   Save each model's weights
4: Initialize main model with transferred weights
   Transfer weights from pre-trained models with best validation performance
5: Train the main neural network
   Use a custom loss function that combines data loss (4.10) and residual loss (4.8)
   Train using Adam optimizer, update parameters using mini-batch gradient descent
6: Save the final trained main model
7: Return PITLNN solution and learned parameters
   Predicted dynamics $\Theta(t)$ and parameters $\alpha(t)^*$ based on learned models

Algorithm 10 outlines a procedure for optimizing learned parameters, denoted as $\beta(t_n)^*$. The algorithm begins by constructing the PITLNN Pre-training, initializing parameters β, the weight and biases, and setting up output layers for the system dynamics and the parameters. It then details the neural network's architecture, including hidden layers with GELU activation and separate networks for predicting parameters. Training data is generated through pre-training, where ranges for α are defined and varied to generate datasets via an ODE solver. Each dataset trains a model whose weights are saved. Training involves minimizing losses between network predictions about system dynamics and actual measurements using a suitable loss function that combines data loss and residual loss. This loss is minimized by an optimization algorithm such as Adam with batched updates. Finally, the trained model is saved by the algorithm along with predicted dynamics and parameters so that the neural network outputs align well with the true behavior of the ODE system.

Algorithm 11 PITLNN Main algorithm for learning the solutions and time-varying parameters of stiff dynamical systems

Initialize the neural network architecture with specific layers for Θ and α

Load the optimal PITLNN pre-trained model weights

Define the initial conditions and parameters for the ODE

Solve the ODE using a numerical solver to generate training data

Normalize the data for better neural network performance

Initialize the optimizer with learning rate and decay parameters

for each epoch from 1 to N_epochs **do**

Select a random batch from the normalized training data

Perform a forward pass through the network to predict Θ and α

Calculate the custom loss, which includes data loss (4.10), physics loss (4.8), and regularization loss (4.11)

Compute gradients and update the model parameters using backpropagation

Log and store loss metrics

If epoch % plot_frequency == 0 **then**

Generate plots for the predicted and actual data

Save plots and model checkpoints

endIf

end for

Save the final model and Output

Optionally, plot and analyze overall training losses and other metrics over all epochs

return The solution of the PITLNN model and the learned time-varying parameter

Algorithm 11 describes how PITLNN main can be used to learn time-varying parameters that approximate the solution of the stiff dynamical systems. The PITLNN is constructed by setting the input initialization parameters, defining the network layer, and loading weights from a list of pre-trained models. This sets up the network with a good starting point for training. The α neural

network is also constructed similarly, where inputs are specified and initialized to synchronize predictions for both the solution $\Theta(t)$ and the time-varying parameter $\alpha(t)$. Training is done by defining a custom loss function that combines the three losses (4.8), (4.10) and (4.11). The training process involves executing a loop using the Adam optimizer, which continuously updates weights and periodically saves the model state. During this process, training metrics such as loss values and gradient norms are recorded. Finally, the learned time-varying parameters $\alpha(t_n;\beta^*)$ and the approximated solutions $\Theta_{NN}(t_n;\beta^*)$ are outputted by the algorithm denoted over given time points. This comprehensive approach ensures that the PITLNN effectively captures the complex behavior of stiff dynamical systems through a combination of advanced neural network techniques.

4.6 Simulations and Discussion

In this section, we introduce the application of PITLNNs as a method for parameter identification and solving different types of stiff dynamical systems. Modeling stiff ordinary differential equations (ODEs) requires accurate parameter identification because each system has properties that largely determine its applicability in practice. These characteristics are responsible for such quantities as rate constants, initial conditions, and coefficients, which describe dynamics, rates of change, and stability. Learning these parameters contributes towards making accurate predictions and simulating dynamical behaviors effectively within a given system. To demonstrate the application of PITLNNs, our analysis involves four benchmark problems selected from established literature. For every case, we provide information regarding network architecture, including the number of neurons, hidden layers, activation functions used, and other relevant details. Two of these problems have exact analytical solutions; thus, we employ them to validate the accuracy of our model. Also, relative error metric that serves to show fitting accuracy for each system according to the PITLNN algorithm is obtained.

4.6.1 The Robertson Problem (ROBER)

The Robertson problem, also known as ROBER, is a classic example of a stiff system of ordinary differential equations (ODEs) derived from a set of autocatalytic chemical reactions. The system

involves three species A, B, and C, undergoing the following reactions:

$$\mathscr{A} \xrightarrow{k_1} \mathscr{B}$$

$$\mathscr{B} + \mathscr{B} \xrightarrow{k_2} \mathscr{C} + \mathscr{B}$$

$$\mathscr{B} + \mathscr{C} \xrightarrow{k_3} \mathscr{A} + \mathscr{C}$$

The rate constants for these reactions are $k_1 = 0.04$, $k_2 = 3 \times 10^7$, and $k_3 = 10^4$. The large disparity in these constants ($\frac{k2}{k1} \approx 10^9$) leads to the stiffness of the system. The system of ODEs modeling the dynamics of the species concentrations is given by:

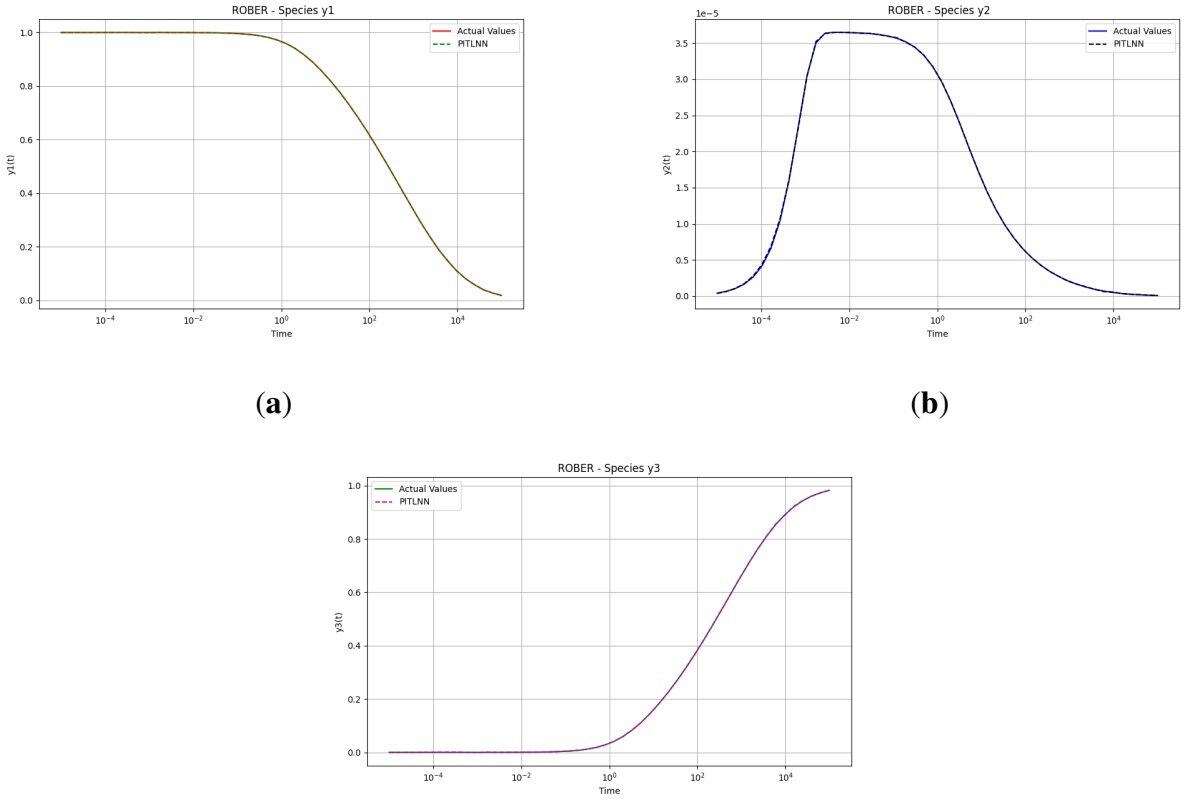
$$\begin{aligned} \frac{dy_1(t)}{dt} &= -k_1(t)y_1(t) + k_3(t)y_2(t)y_3(t) \\ \frac{dy_2(t)}{dt} &= k_1(t)y_1(t) - k_2(t)y_2^2(t) - k_3(t)y_2(t)y_3(t) \\ \frac{dy_3(t)}{dt} &= k_2(t)y_2^2(t) \end{aligned} \tag{4.12}$$

The system's initial conditions are $y_1(0) = 1$, $y_2(0) = 0$, and $y_3(0) = 0$, indicating that initially only species (A) is present. This problem is frequently used as a benchmark for testing numerical solvers designed to handle stiff ODE systems, emphasizing the need for specialized techniques such as implicit methods or adaptive time-stepping algorithms.

The schematic diagram of the PITLNN shown in Figure 4.1 was used to learn the time-varying parameters and the solution to the ROBER problem. We start by finding the optimal neural network parameter $\beta^*(t_n)$ (weight and biases) using algorithm 10, which will be transferred to algorithm 11 to minimize the loss function. This is done by generating pre-training data that are closer or mimic the main data using ranges for $k_1 = k_{1_0} + k_{1_k}$, $k_2 = k_{2_0} + k_{2_k}$ and $k_3 = k_{3_0} + k_{3_k}$ with numerical solver called Backward Differentiation Formula (BDF). To get the optimal neural network parameter $\beta^*(t_n)$, we use five hidden layers with 64 neurons each for the input and output and optimized Adam with a batch size of 50. The Gaussian Error Linear Unit (GELU) activation function is used with 5,000 epochs and 10^{-3} learning rate. We use (4.8) and (4.10), the residual and training losses

to minimize the loss function.

The algorithm saves the optimal model parameters so that the neural network outputs align well with the true behavior of the ODE system. The saved model parameters are now transferred to the proposed algorithm 11, which is used to solve and learn the time-varying parameters. The residual loss (4.8), training loss (4.10), and regularization loss (4.11) were use to minimize the loss function where $\omega = 10^{-7}$ and $\gamma = 10^{-6}$. Finally, the solution to ROBER problem and its time-varying parameters' was obtained using PITLNN with five layers of 64 neurons, 50000 epochs, GELU activation function, Adam, batch size = 50, learning rate of 10^{-4} and time span $[10^{-5}, 10^{5}]$ where 50 sample data points are uniformly spaced in a logarithmic time scale.



(**a**) (**b**)

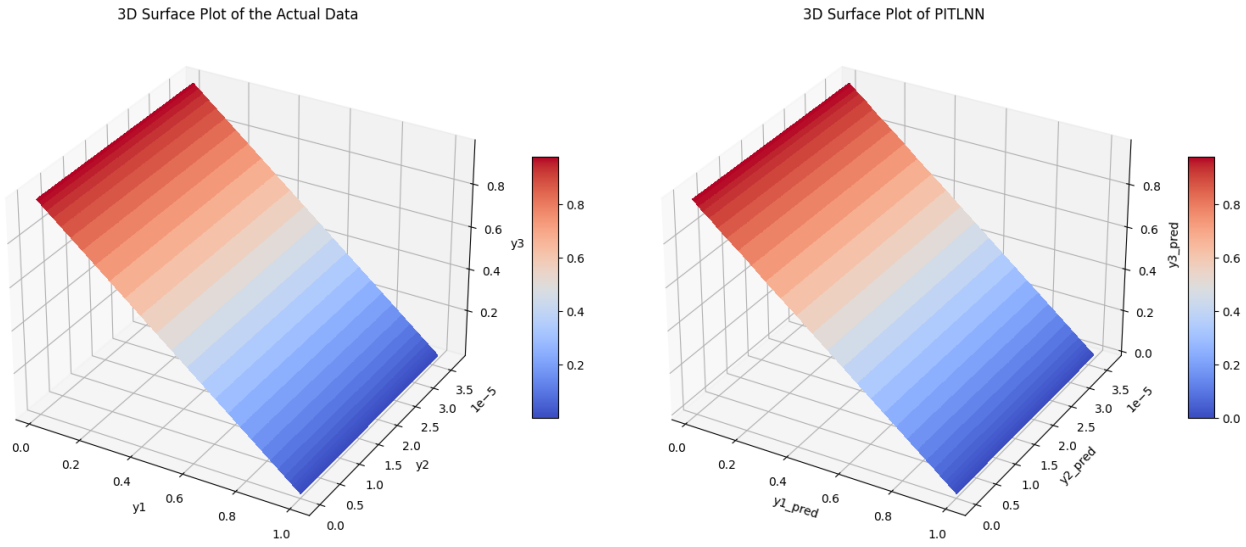
(**c**)

Figure 4.2: The actual solution and the PITLNN output solution for species y_1, y_2 and y_3 of the Robertson equations. (**a**) The actual and the PITLNN solution for y_1. (**b**) The actual and the PITLNN solution for y_2. (**c**) The actual and the PITLNN solution for y_3.

To obtain the best solution, let N denote the total number of training epochs with $N = 50,000$. The model saves checkpoints every 200 epochs, starting from the first checkpoint at epoch 200, represented as:

$$\text{Save checkpoint if } n\%200 = 0 \text{ and } n \neq 0, \quad \text{where } n \text{ is the epoch number.}$$

Therefore, the best model performance is achieved at epoch $n = 28,200$, which indicates that this particular epoch yields the most optimal solution based on the lowest total loss, which includes data loss, physics loss, and regularization loss along with error metrics such as MAE, MSE, RMSE, MAPE, L_2 norm and L_∞ norm used during training. During the training process, the loss functions steadily decreased, and the gradient norm remained stable, indicating effective learning dynamics and parameter adjustments. The training was conducted on a workstation using a single CPU processor, which took approximately 6 minutes and 45 seconds.



(**a**) (**b**)

Figure 4.3: The 3D phase space plot of the actual and PITLNN output of species y_1, y_2 and y_3. (**a**) The true values of species Y. (**b**) The predicted values of species Y.

Figure 4.2 presents the performance of the physics-informed transfer learning neural network in predicting the behavior of the three species over time as the Robertson equation describes. The

prediction follows the actual trajectory across the entire timescale. The 3D surface plot of the actual output of species y_1, y_2 and y_3 and the PITLNN output of species y_1, y_2 and y_3 are also shown in Figure 4.3. This shows that PITLNN effectively captures the dynamics of y_1, y_2, and y_3 across various time scales, including the stable state as time progresses. Figure 4.4 displays the time-varying parameters $k_1(t)$, $k_2(t)$, and $k_3(t)$, derived from the Robertson equations, capturing complex dynamics within a reactive system. The first plot shows a gradual increase in $k_1(t)$ over time, which indicates a shift in reaction dynamics. However, $k_2(t)$ decreases at first before stabilizing, suggesting a balance between competing chemical processes at different reaction stages. Initially, $k3(t)$ remains unchanged but then increases steeply, showing rapid change in a reaction.

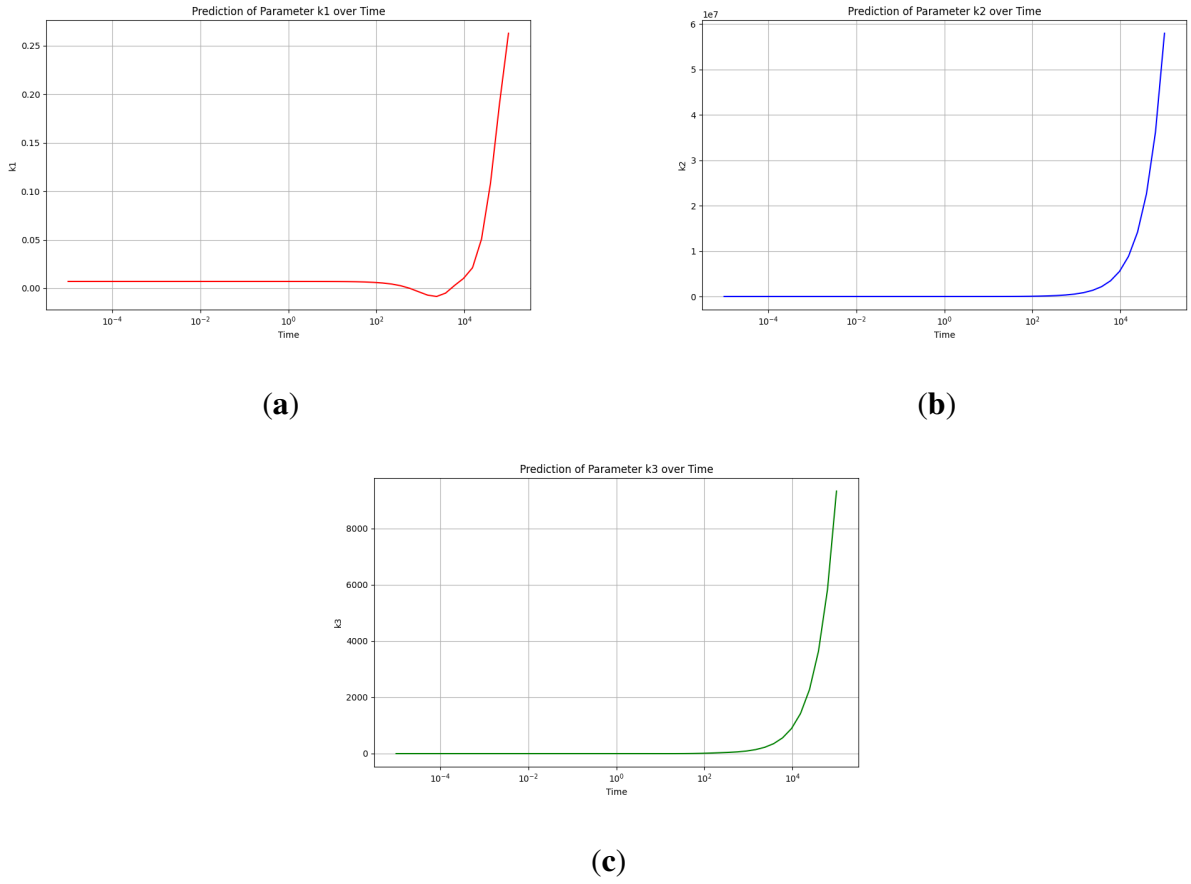
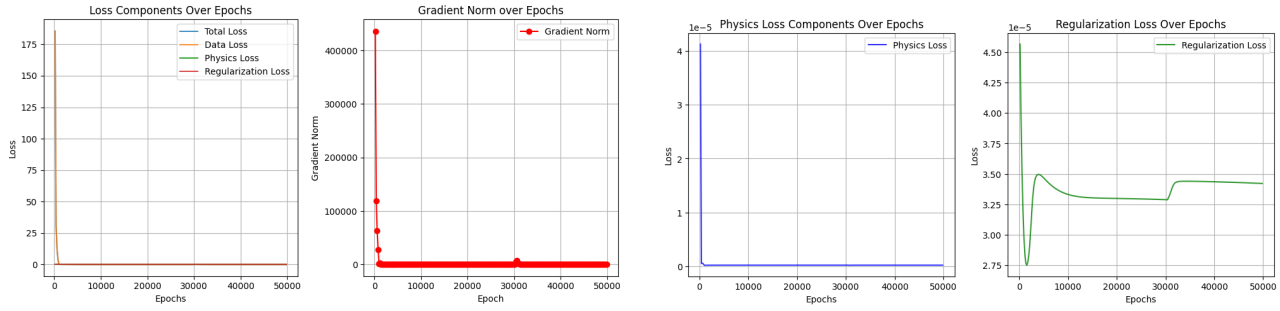


Figure 4.4: Temporal behavior of time-varying parameters k_1, k_2, and k_3 from the Robertson equations, highlighting complex dynamics in a reactive system. (**a**) Time-varying parameter $k_1(t)$. (**b**) Time-varying parameter $k_2(t)$. (**c**) Time-varying parameter $k_3(t)$.

These trends illustrate temporal shifts in chemical reaction mechanisms that support the efficiency of PITLNN in capturing and predicting multi-physics behavior in stiff reactions. Additionally, observing the training dynamics shown in Figure 4.5 could give an insight into the model's performance. A composition of loss across epochs that forms data loss, physics loss, and regularization loss similarly highlights that the integrity of the data set is essential for model training. The low value of regularization loss shows a properly restricted model that does not suffer from overfitting and remains generalizable. Variability in gradient norm, consistent with expectations during training, does not reveal any abnormal spikes or convergence problems; hence, stability and reliability are ensured in the training process.



(**a**) (**b**)

Figure 4.5: The loss components and gradient norm against epochs. (**a**) The loss functions and the gradient norm. (**b**) The physics loss and the regularization loss

Table 4.1: The error metrics for the Robertson equations using PITLNN

Error metrics	The best performance at epoch $28,200$	Error metrics at epoch 50000
MAE	6.7980×10^{-4}	8.3450×10^{-4}
MSE	1.5000×10^{-6}	2.0000×10^{-6}
RMSE	1.0402×10^{-3}	1.2402×10^{-3}
MAPE	308165.4541	2888316.9922
L_2	1.3758×10^{-2}	1.4979×10^{-2}
L_∞	6.2848×10^{-3}	7.5172×10^{-3}

Lastly, Table 4.1 shows a detailed assessment of the predictive accuracy of PITLNN for Robertson equations. To determine prediction errors, different error metrics such as mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), mean absolute percentage error (MAPE), and norm-based errors (L_2 and L_∞) have been employed at various training epochs.

The strongest predictions came from epoch $28,200$, which had a remarkable accuracy with extremely small errors in all metrics. At epoch 50,000, though there was a slight increment in error, it remained within a small range, showing how resilient the model is in maintaining prediction accuracy through extensive training periods. This implies that the model can accurately solve stiff problems, and it can also be applied in situations where the accuracy of long-term predictions is important. The effective prediction made across this broad range of time scales indicates strong training as well as a potential for adaptability to other stiff systems in chemical kinetics or related areas.

4.6.2 Damped Oscillator

A damped motion is an oscillatory movement with a damping force continuously reducing its amplitude. It is common in electrical and mechanical systems, and it is represented by the second-order ordinary differential equation below:

$$m\ddot{x} + c\dot{x} + kx = 0$$

Where variables $\ddot{x}$, $\dot{x}$, x, m, c and k represent

$\ddot{x}$: The acceleration

$\dot{x}$: The velocity

x : The displacement from equilibrium

m : The mass.

c : The damping coefficient

k : The spring constant.

Let $\dot{x} = \frac{dz_1(t)}{dt}$, and by setting $\frac{dz_1(t)}{dt} = z_2$, we obtain the system of first-order ODEs:

$$\begin{aligned} \frac{dz_1(t)}{dt} &= z_2(t) \\ \frac{dz_2(t)}{dt} &= -c(t)z_2 - k(t)z_1(t). \end{aligned} \tag{4.13}$$

Where $m = 1$, $c = 1001$ and $k = 1000$. This implies that since the damping coefficient c is very large, there will be one solution variable that decays much faster than any other. This marked discrepancy in decay rates among different components of the solution makes the system stiff. Given that the initial conditions is $z_1(0) = 1$ and $z_2(0) = 1$, the exact solution will be:

$$z_1(t) = -\frac{2}{999}e^{-t} + \frac{1001}{999}e^{-1000t}$$

$$z_2(t) = -\frac{2000}{999}e^{-t} - \frac{1001}{999}e^{-1000t}$$

In this study, we consider the model of a damped oscillator where $z_1(t)$ and $z_2(t)$ are the state variables of the system being acted upon by time-varying parameters $c(t)$ and $k(t)$. Although typically used to describe mechanical or electrical systems, this model can also be employed in other scientific domains to mimic some types of dynamical behavior. According to our investigation, depending on damping coefficient c and stiffness parameter k values, a system can manifest various dynamics: stable equilibrium states, oscillations, or chaos. Our main aim is to identify these damped oscillator models time-dependent parameters c(t) and k(t). We used synthetic data obtained by numerically solving (BDF) the differential equations governing a damped oscillator. The solver was performed for constant coefficients with given initial conditions over an interval from 0 to T where $T = 10$. The computational grid comprised $P_x = 1000$ points covering this interval to create observations or synthetic datasets.

We use the architecture shown in Figure 4.1 to learn the time-varying parameters of the damped oscillator model. The first network outputs $(\beta(t_n)^*)$ the network parameters from the input generating pre-training data that are closer or mimic the main data using ranges for $c = c_0 + c_k$ and $k = k_0 + k_k$ with numerical solver called Backward Differentiation Formula (BDF). We used five hidden layers with 64 neurons each for the input and output and optimized Adam with a batch size of 32. The Gaussian Error Linear Unit (GELU) activation function is used with 5,000 epochs, 10^{-3} learning rate, and the residual and training losses are used to minimize the loss function. The saved model parameters are then transferred to the second network that outputs the solution $z_{1NN}(t_n;\beta^*)$, $z_{2NN}(t_n;\beta^*)$ and the learned time-varying parameter $c(t_n;\beta^*)$, $k(t_n;\beta^*)$. The second network uses the same hyperparameters except for the learning rate 5 x 10^{-4} and the epoch 100000, where the model saves checkpoints every 10000 epochs. The residual loss, training loss, and regularization loss are used to minimize the loss function where $\omega = 10^{-6}$ and $\gamma = 10^{-4}$.

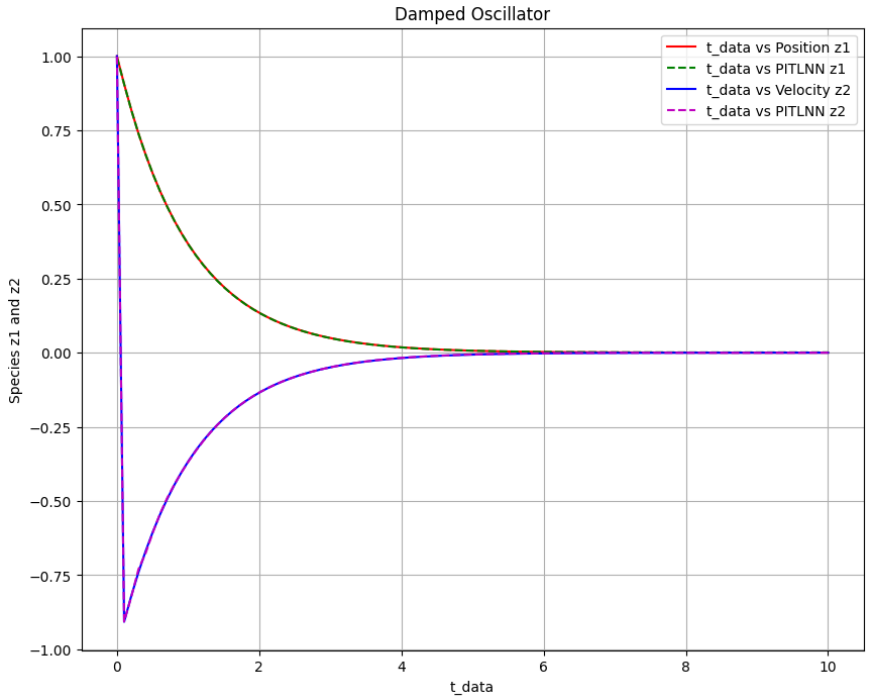


Figure 4.6: The actual solution and the PITLNN output solution for damped oscillator system

The best performance of the model was recorded at $n = 20000$ epochs, where the overall loss is the lowest. This loss includes data loss, physics loss, and regularization loss, measured using metrics such as MAE, MSE, RMSE, MAPE, L_2 norm, and L_∞ norm during training. There was a consistent drop in loss values with stable gradient norms regarding training dynamics, indicating that the models parameters were effectively adjusted throughout the training process. The training session took slightly over 12 minutes on a single CPU processor workstation. Figure 4.6 shows how the physics-informed transfer learning neural network predicts the evolution of three species modeled by a damped oscillator over time. The expected position curve is very close to the real data curve, and the predicted velocity curve is the same as its actual data curve.

This shows that the model has learned well and reproduced correctly what happens in a damped oscillator system. The close fit between the predicted and actual curves indicates a high level of accuracy in the model predictions. Additionally, Figure 4.7 demonstrates behavior exhibited by time-varying parameters $c(t)$ and $k(t)$ in the model as well as the comparisons of actual and predicted values for $z_1(t)$ and z_2, which reflects complex reactivity system dynamics. The parameter

$C(t)$, indicating the damping coefficient, decreases exponentially until it stabilizes at a lower magnitude. Such behavior can indicate a transitional phase in system dynamics when the damping effect weakens over time. The $k(t)$ also declines, resulting in exponential decay. This parameter eventually stabilizes at a lower value, implying that the systems stiffness diminishes, affecting its natural frequency and response to perturbations.

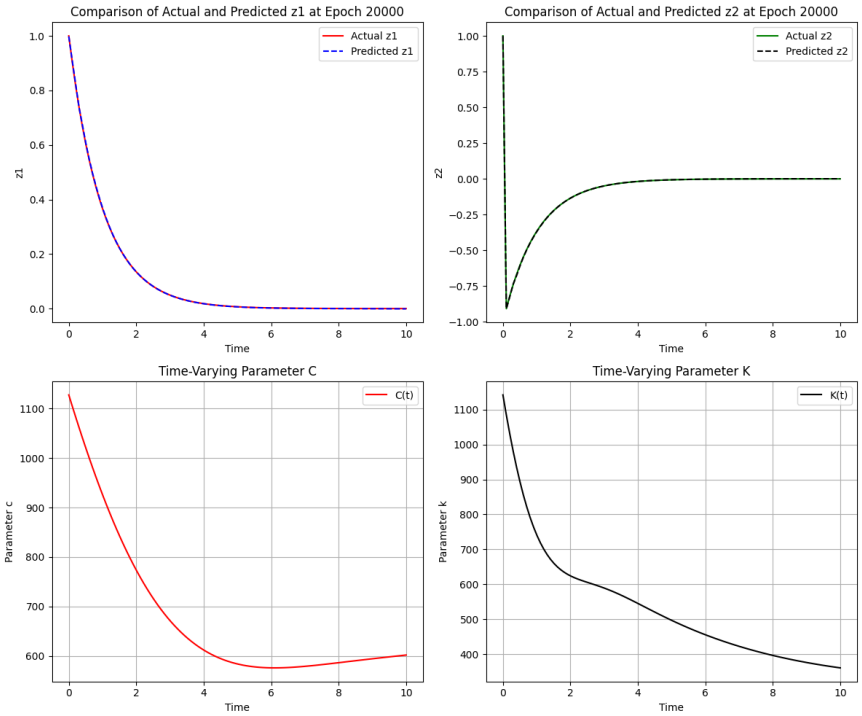
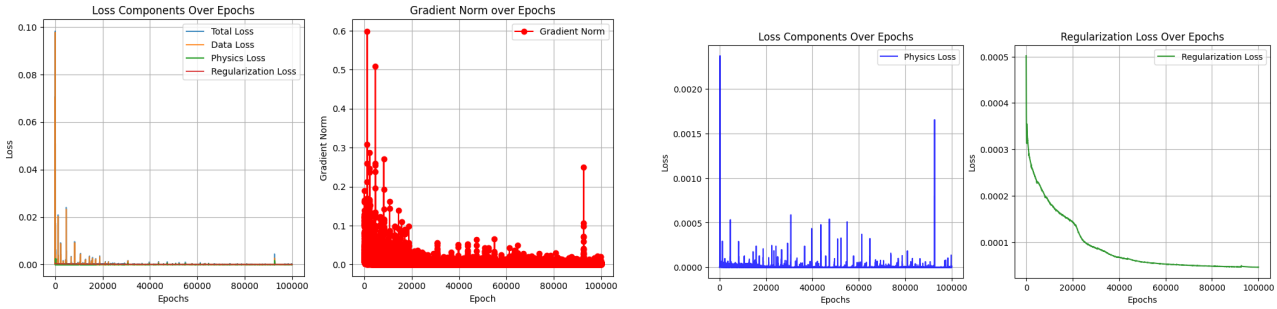


Figure 4.7: The combination of the actual solution, the predicted solution, and the time-varying parameter of the damped oscillator system

Figure 4.8 shows how the neural network learns over time. They do this by tracking changes in different loss components and gradient norms during 100,000 epochs. Initially, the overall loss declines quickly as it converges towards a minimum point. The convergence is mainly driven by reductions in data and physics losses, which are major components of the total loss function, indicating that the model has learned well. Regularization loss also drops and remains low throughout the process, implying no overfitting but good generalization. High variability and significant spikes can be observed within the gradient norm graph at early stages when parameters undergo large adjustments.



(**a**) (**b**)

Figure 4.8: The loss components and gradient norm against epochs. (**a**) The loss functions and the gradient norm. (**b**) The physics loss and the regularization loss

However, these spikes decrease later on as learning stabilizes and optimal parameters are reached, thus signifying such points in training where more accurate values were found out for weights or biases through error backpropagation algorithm stepwise iterations until they become approximately correct values according to certain thresholds defined based on some mathematical formulas depending on the algorithm used. This type of behavior seen here - rapid initial improvement followed by a period of little gradual change - should be expected from any successful neural network training since such models have to account both for data patterns and natural laws or principles underlying physical processes involved.

Table 4.2: The error metrics for the damped oscillator using PITLNN

Error metrics	The best performance at epoch $20,000$	Error metrics at epoch $100,000$
MAE	3.3260×10^{-4}	4.1980×10^{-4}
MSE	3.0000×10^{-7}	6.0000×10^{-6}
RMSE	5.7240×10^{-4}	7.4520×10^{-3}
MAPE	31.8271	75.0304
L_2	7.2200×10^{-3}	8.0666×10^{-3}
L_∞	5.2320×10^{-3}	7.8849×10^{-3}

The metrics of errors on the PITLNN model of a damped oscillator at two different epochs, i.e.,

20000 and 100000, are shown in Table 2. While the best performance is achieved by the model at epoch 20,000 with all error metrics being significantly lower than those recorded at 100,000 epochs, it is only Mean Absolute Error (MAE) and Mean Squared Error (MSE) that are much higher during later times which suggests that predictions made earlier were more precise. Predictive accuracy declines over time as Root Mean Squared Error (RMSE) and Maximum Absolute Percentage Error (MAPE) also rise sharply by this stage, indicating degradation. Furthermore, both L_2 and L_∞ norms exhibit slight increments, which may imply some overfitting or lack of generalization during further training stages. Still, they can slightly vary depending on what was learned from the data. This might indicate problems such as drifting of the models where their performances deteriorate, possibly due to changes in underlying data distribution or dynamics not captured towards the end of the training process.

4.6.3 Stiff First-Order Irreversible Chain Reactions

Stiff first-order irreversible chain reactions involve a series of chemical reactions where each step follows rapidly after the previous one. Once the reaction starts, it completes without any chance of reverting to the starting materials. The reaction is stiff because it involves a set of steps with very different speeds. In some cases, the speed can be extremely fast, but in others, it can be really slow, which makes it difficult to study and control these reactions. In mathematical models and simulations, this can cause problems because the equations that describe these reactions are hard to solve.

To mathematically describe stiff first-order irreversible chain reactions, let's consider a simple reaction chain involving three species: A, B, and C. The reactions proceed as follows:

$$\mathscr{A} \xrightarrow{k_1} \mathscr{B} \xrightarrow{k_2} \mathscr{C}$$

Here, k_1 and k_2 are the rate constants for each reaction step, with the time units. The reaction from A to B is assumed to be much faster than the reaction from B to C, which introduces stiffness into the system. The stiffness in this system comes from the difference in magnitudes of k_1 and k_2.

If $k_1 >> k_2$, the reaction A to B happens much faster than B to C, making the system stiff. The following are the characteristics of first-order reactions:

$$\begin{aligned} \frac{dz_1(t)}{dt} &= -k_1(t)z_1(t) \\ \frac{dz_2(t)}{dt} &= k_1(t)z_1(t) - k_2(t)z_2(t). \end{aligned} \tag{4.14}$$

The chemical species $\mathscr{A}$ and $\mathscr{B}$ are represented by the variables z_1 and z_2, respectively, while k_1 and k_2 are the rate constants that act as unknowns. In order to visualize this in equations for first-order reactions with stiffness, we draw graphs showing how these two quantities vary over time when different values of k_1 are used but keeping $k_2 = 1$ fixed. Letting $k_2 = 1$, there is a range of values that can be taken on by δ^i where $\delta = 5$ and $i = 2, ..., 4$, thus we will plot against each other such profiles for $k_1 = \delta^i$ with the initial condition being at time zero given by $z(t = 0) = [1, 0]$. Some differential equations are stiff and so require specified numerical techniques for integration into numerical schemes.

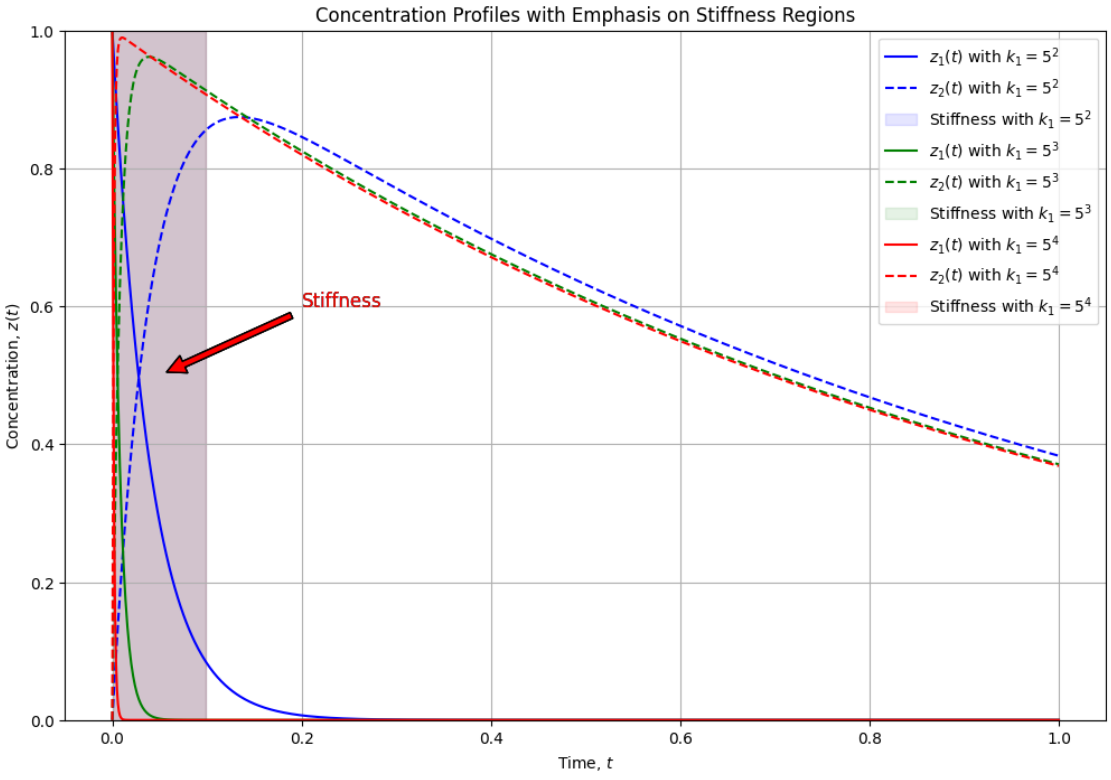


Figure 4.9: The Concentration Profiles Highlighting Stiffness Regions

In Figure 4.9 above, the concentrations of z_1 and z_2 over time for different values of k_1 are plotted to highlight the regions where stiffness is most pronounced. The colored areas at the beginning of each curve emphasize the stiffness regions for each value of k_1. The shaded areas under the curves in the early time segment (assumed here to be the first 10% of the timeline) visually represent the stiffness in the system. These regions show where the reactions are happening rapidly compared to the later stages. For each value of ($5^2, 5^3$, and 5^4), stiffness is indicated by a rapid change in concentration, which is more pronounced with higher k_1.

This illustrates how the system's response becomes increasingly stiff as the rate constant k_1 increases, leading to faster initial reactions that quickly reach a new equilibrium state. Furthermore, arrows have been added to point out the stiffness regions for each set of responses, corresponding to different values of k_1. These arrows aim to direct attention to the early part of the timeline, where the reaction rates change rapidly, indicating stiffness in the system. Finally, as k_1 increases, the rate at which $\mathscr{A}$ is consumed, and $\mathscr{B}$ is produced initially becomes faster, indicated by the steeper slopes at the beginning of each curve. This rapid change at the start, followed by a much slower approach to equilibrium, demonstrates the stiffness in the system. Stiffness is most pronounced for the highest k_1 value (5^4), where the initial change is so rapid that the concentrations adjust to their new values almost instantaneously before slowly approaching their final steady states.

The PITLNN approach to learning the solution and identifying the time-varying parameters of the stiff first-order irreversible chain reactions model is performed by finding the best network parameters, β^*, representing the biases and weights of the network that minimize the loss function. These optimal values, along with the system of ODEs generated from the chemical reactions using a numerical solver called BDF, are used to create pre-training data for network three, aiming to find the optimal neural network parameter β^* necessary to transfer to second networks to solve the dynamic systems $z_{1NN}(t_n;\beta^*)$ and $z_{2NN}(t_n;\beta^*)$. These networks take t as input data and learn the time-varying parameters $k_1(t_n;\beta^*)$ and $k_2(t_n;\beta^*)$ from the data. Finally, the mean value of the stiff first-order irreversible chain reaction model parameters was obtained using PILTNN with five hidden layers of 128 neurons for both input and output layers. The Gaussian Error Linear Unit

(GELU) activation function is used with 100,000 epochs and 10^{-5} learning rate where $\gamma = 10^{-6}$. To minimize the loss function, we define the residual, data, and regularization loss just as in the previous cases.

Figure 4.10 compares species' actual and predicted concentrations z_1 and z_2 over time. The lines for actual and predicted values overlap, indicating the high accuracy of the PITLNN model in simulating reaction kinetics for both species. The precise match between predicted and actual data showcases the PITLNN's effectiveness in capturing the dynamics of stiff chemical reactions. Figure 4.11 illustrates the time-dependent behavior of parameters k_1 and k_2. Parameter k_1 consistently increases over time, while k_2 initially decreases before stabilizing. These trends suggest complex dynamics within the reaction mechanism that the PITLNN model has successfully captured. Understanding the temporal evolution of these parameters is crucial for optimizing chemical processes involving stiff reactions.

Table 4.3: The results of the error metrics for the stiff first-order irreversible chain reactions using PITLNN

Error	$z_1(t)$	$z_2(t)$
MAE	1.9871 x 10^{-4}	1.3547 x 10^{-3}
MSE	5.2305 x 10^{-7}	2.2363 x 10^{-6}
RMSE	7.2322 x 10^{-4}	1.4954 x 10^{-3}
L_2	2.2870 x 10^{-2}	4.7289 x 10^{-2}
L_∞	1.9152 x 10^{-2}	1.8463 x 10^{-2}

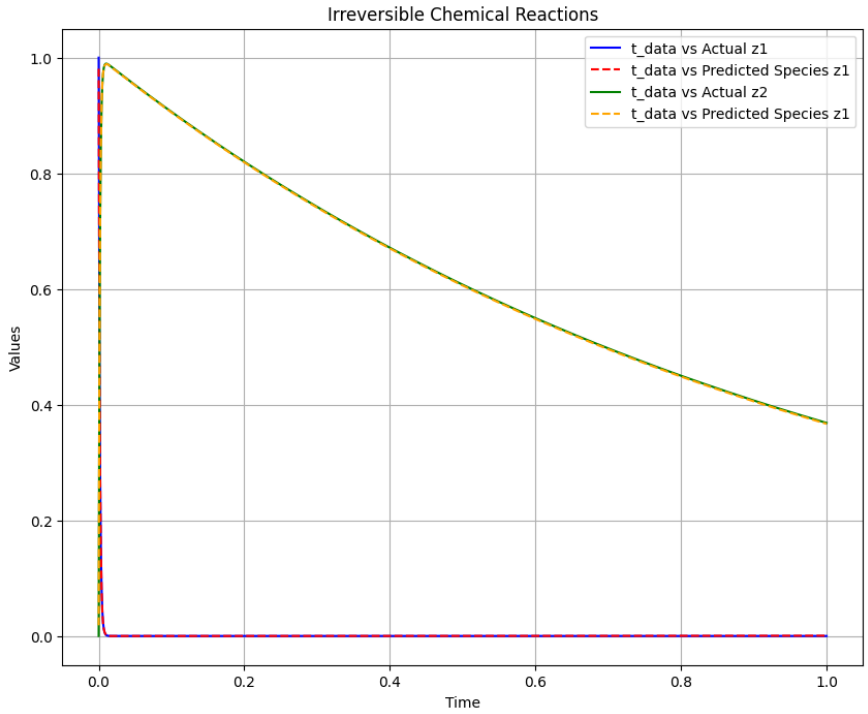
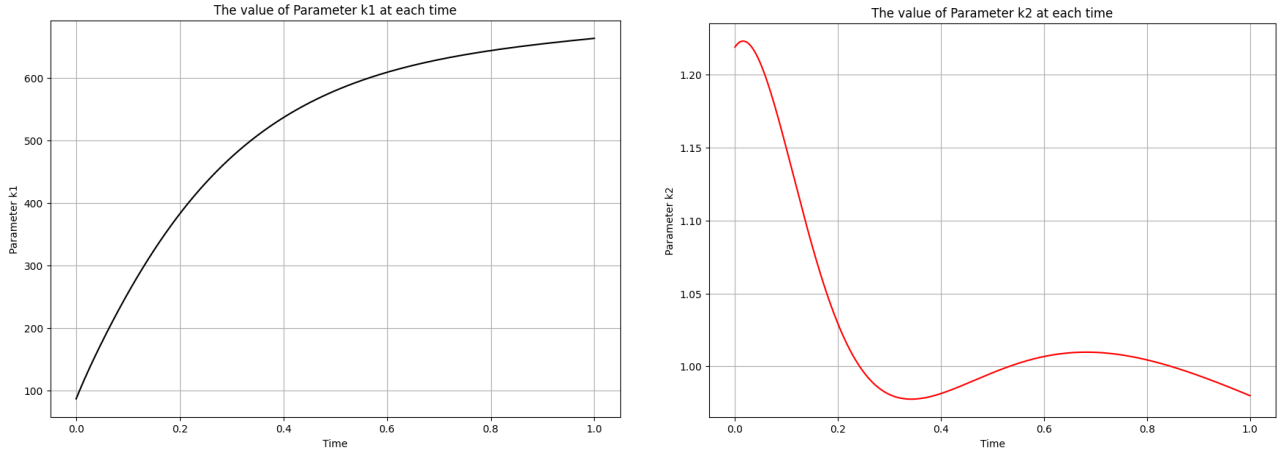


Figure 4.10: The Stiff first-order irreversible chain reactions solution of the actual output of species z_1, z_2 against t_{data} and the predicted output of species z_1, z_2 against t_{data} when $k_1 = 625$ and $k_2 = 1$.



(**a**) (**b**)

Figure 4.11: The learned time-varying parameter values of the Stiff first-order irreversible chain reactions. (**a**) Time-varying parameter k_1. (**b**) Time-varying parameter k_2.

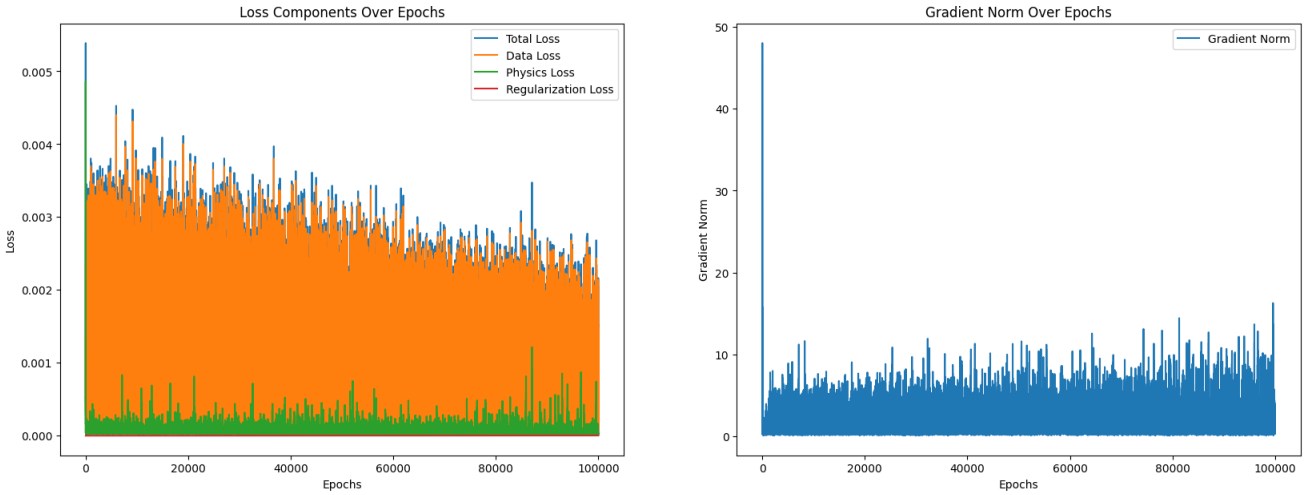


Figure 4.12: The results of the loss components and gradient norm over epochs

Figure 4.12 displays the loss components and gradient norm across multiple epochs. The total loss comprises data loss, physics loss, and regularization loss. Data loss significantly contributes to the total loss, emphasizing the importance of maintaining data fidelity during model training. The minimal regularization loss indicates that the model parameters are not overfitted. The table displays error metrics for a model predicting stiff first-order irreversible chain reactions for variables $z_1(t)$ and $z_2(t)$. The metrics indicate better accuracy for $z_1(t)$ with significantly lower error values across all measures than $z_2(t)$. This suggests that the model is more effective at predicting $z_1(t)$ than $z_2(t)$, potentially due to differences in the nature or complexity of the variables within the reaction system.

4.6.4 High Irradiance RESponse Problem of Schfer

The High Irradiance RESponse (HIRES) problem, presents a computational biology and chemical kinetics benchmark for testing numerical methods for stiff ordinary differential equations (ODEs). A model of the biochemical kinetics of a system exposed to high levels of irradiance gives rise to this problem. It thus mimics what happens when a chemical system responds dynamically under such circumstances. The HIRES model includes eight tightly connected nonlinear stiff ODEs representing various chemical species and reactions [**?**].

$$
\begin{aligned}
\frac{dy_1(t)}{dt} &= -r_1(t)y_1(t) + r_2(t)y_2(t) + r_3(t)y_3(t) + 0.0007,\\
\frac{dy_2(t)}{dt} &= r_1(t)y_1(t) - r_5(t)y_2(t),\\
\frac{dy_3(t)}{dt} &= -r_6(t)y_3(t) + r_2(t)y_4(t) + 0.035y_5(t),\\
\frac{dy_4(t)}{dt} &= r_3(t)y_2(t) + r_1(t)y_3(t) - r_{10}(t)y_4(t),\\
\frac{dy_5(t)}{dt} &= -1.745y_5(t) + r_2(t)y_6(t) + r_2(t)y_7(t),\\
\frac{dy_6(t)}{dt} &= -280y_6(t)y_8(t) + 0.69y_4(t) + r_1(t)y_5(t) - r_2(t)y_6(t) + 0.69y_7(t),\\
\frac{dy_7(t)}{dt} &= 280y_6(t)y_8(t) - 1.81y_7(t),\\
\frac{dy_8(t)}{dt} &= -280y_6(t)y_8(t) + 1.81y_7(t).
\end{aligned}
\tag{4.15}
$$

Where $r_1 = 1.17$, $r_2 = 0.43$, $r_3 = 8.32$, $r_5 = 8.75$, $r_6 = 10.03$, $r_{10} = 1.12$ and the initial condition are $y(0) = [1,0,0,0,0,0,0,0.0057]^T$ and the solution is sought in the interval $[0, 321.8122]$. This model contains parameters that can be changed to understand the dynamic nature of the system. We do not use all variables to keep it simple and comprehensible. To achieve this, we have chosen a few parameters from among many available with the HIRES system based on impact and sensitivity, role in the equation, diversity of effects, and computational efficiency. These equations are very dynamic and change over short periods, resulting in a stiff system that is difficult to solve numerically. The stiffness is caused by the fact that the steps involved occur at different speeds; therefore, a numerical solver with special capabilities for managing such discrepancies should be used effectively. Therefore, we use algorithms 10 and 11 to solve and learn the time-varying parameters of the HIRES problem.

This was done by inputting generating pre-training data that are closer or mimic the main data using ranges for $r_1 = r_{1_0} + r_{1_k}$, $r_2 = r_{2_0} + r_{2_k}$, $r_3 = r_{3_0} + r_{3_k}$, $r_5 = r_{5_0} + r_{5_k}$, $r_6 = r_{6_0} + r_{6_k}$ and $r_{10} = r_{10_0} + r_{10_k}$ with numerical solver called Backward Differentiation Formula (BDF) and outputting $(\beta(t_n)^*)$ the model parameter using algorithms 10. We used seven hidden layers with 64

neurons each for the input and output and optimized Adam with a batch size of 50. The Gaussian Error Linear Unit (GELU) activation function is used with 5,000 epochs and 10^{-4} learning rate with the residual and training loss used to minimize the loss function. The saved model parameters are then transferred to the second network that outputs the solution $y_{NN}(t_n;\beta^*)$, and the learned time-varying parameter $r(t_n;\beta^*)$. The second network uses the same hyperparameters except for the epoch that is 100000, where the model saves checkpoints every 200 epochs, $\gamma = 10^{-6}$ and $\omega = 10^{-7}$. We used the residual, data, and regularization losses to minimize the loss function, just as in the previous cases.

To obtain the best solution, let N denote the total number of training epochs with $N = 100,000$. The model saves checkpoints every 200 epochs, starting from the first checkpoint at epoch 200, represented as:

$$\text{Save checkpoint if } n\%200 = 0 \text{ and } n \neq 0, \quad \text{where } n \text{ is the epoch number.}$$

Therefore, the best model performance is achieved at epoch $n = 79,800$, which indicates that this particular epoch yields the most optimal solution based on the lowest total loss, which includes data loss, physics loss, and regularization loss along with error metrics such as MAE, MSE, RMSE, MAPE, L_2 norm and L_∞ norm used during training. During the training process, the loss functions steadily decreased, and the gradient norm remained stable, indicating effective learning dynamics and parameter adjustments. The training was conducted on a workstation using a single CPU processor, which took approximately 24 minutes and 13 seconds. Figure 4.13 displays the dynamics of eight species for the High Irradiance RESponse (HIRES) problem, showing their actual and predicted behaviors over time using a predictive model. All species show remarkable model performance, indicating the accuracy of handling the temporal dynamics of the system. For y_1 through y_8, the predicted values overlap with observed data points, showing that this model can capture concentrations changing quickly within the first few units of time.

After the initial period, y_1, y_2, y_3, and y_4 fall off steeply, indicating fast reactions or decays; such sharp transitions present difficulties to many methods, including some chemical kinetics models based on stiff equations, but they are still well handled by our approach which deals directly with them. Species five and six exhibit continuous decreases throughout time. These trends are mirrored closely by predictions made under our system since continuous decay is one situation that can be captured easily according to the design principles adopted. However, species y_7 and y_8 have a stable plateau at first but change at the end. For instance, the decline of y_7 is observed while y_8 displays a delayed exponential growth. These alterations are anticipated by the model, thus showing that it can handle situations where there is considerable change in system dynamics after extended stability periods. These results highlight how well-suited this model is to simulate any complex biological or chemical systems with different rates of reaction and stability.

Figure 4.14 show the dynamic behavior of parameters r_1, r_2, r_3, r_5, r_6, and r_{10} within a HIRES model. The time-varying parameters r_1, r_2, and r_3 display a steady increase over time, indicating their growing influence on the system's dynamics, with r_3 rising sharply towards the end, suggesting its critical role in later reaction stages. While r_5 has peaked before decreasing slightly in order to provide mainly temporary but significant assistance. A U-shaped curve has been traced by r_6, showing that it initially had a low influence on the process before rebounding and suggesting regulatory effects or competitive interactions. Finally, r_{10} fluctuates before descending as it undergoes a complex regulation within the network. These patterns help to learn about parameter roles within a chemical reaction mechanism that are important for successful optimization and control of chemical processes.

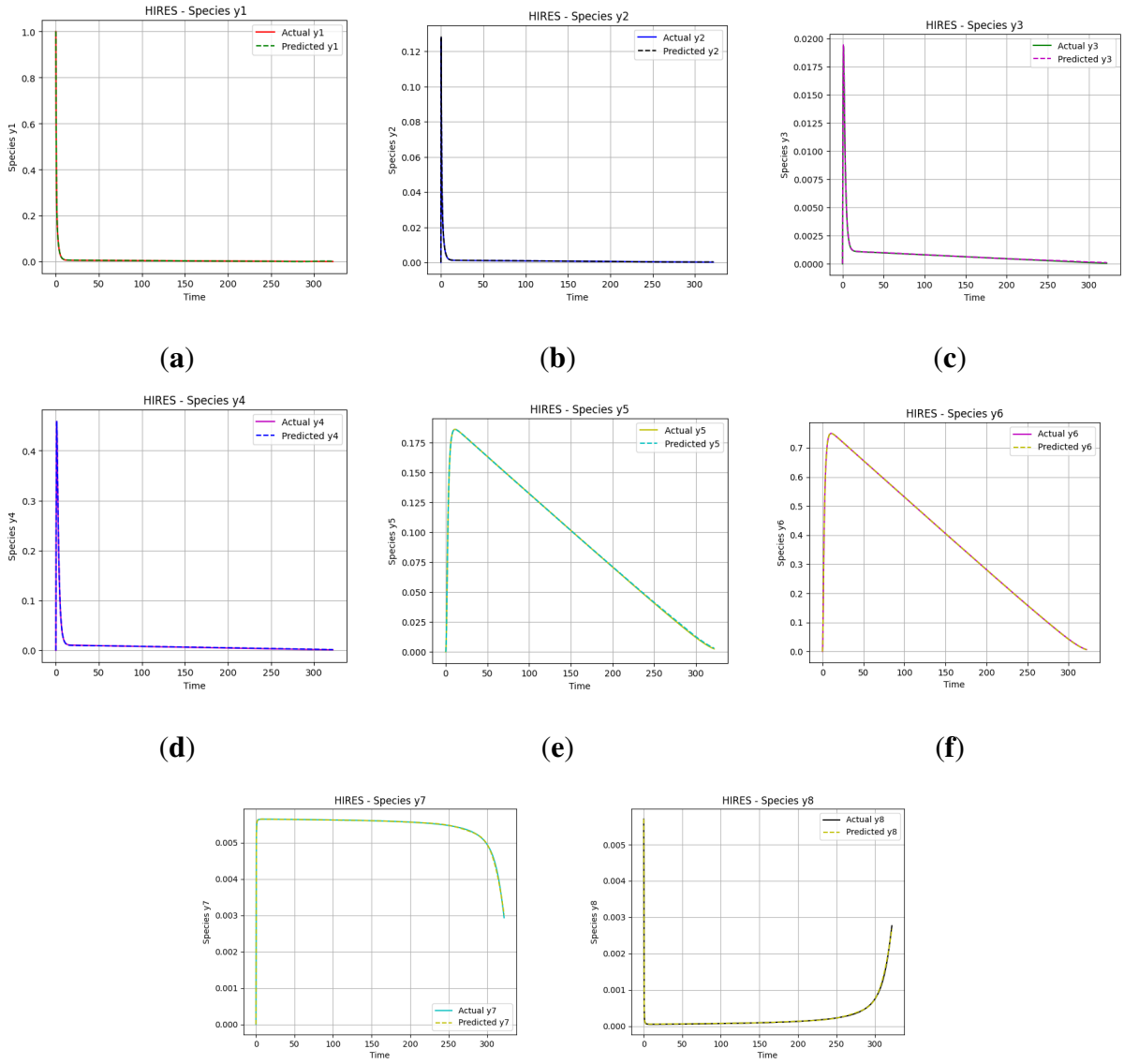


Figure 4.13: The actual solution and the PITLNN output solution for various species of the equations. (**a**) The actual and the PITLNN solution for y_1. (**b**) The actual and the PITLNN solution for y_2. (**c**) The actual and the PITLNN solution for y_3. (**d**) The actual and the PITLNN solution for y_4. (**e**) The actual and the PITLNN solution for y_5. (**f**) The actual and the PITLNN solution for y_6. (**g**) The actual and the PITLNN solution for y_7. (**h**) The actual and the PITLNN solution for y_8.

Table 4.4: The error metrics for the High Irradiance RESponse using PITLNN

Error metrics	The best performance at epoch 79,800	Error metrics at epoch 100000
MAE	1.0054 x 10^{-3}	2.3788 x 10^{-3}
MSE	2.9000 x 10^{-6}	8.5000 x 10^{-6}
RMSE	1.4060 x 10^{-3}	2.7935 x 10^{-3}
MAPE	6.63751	22.2062
L_2	1.3758 x 10^{-2}	4.4749 x 10^{-2}
L_∞	2.6819 x 10^{-2}	3.3342 x 10^{-2}

Performance metrics were compared in Table 4.4 for the High Irradiance RESponse model based on Physics-Informed Transfer Learning Neural Networks (PITLNN) at two different epochs, which demonstrated how well the model has been performing over time. At epoch 79,800, the lowest errors are shown as the model attains high performance during that epoch. Consequently, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are exceptionally low during this epoch, indicating that the model makes highly accurate predictions with MAE being approximately (1.0054×10^{-3}) and RMSE equal to (1.4060×10^{-3}). This is further confirmed by The Mean Square Error (MSE), which shows minimal error, indicating that the model is well-fitted at this stage. However, by epoch 100000, all error metrics have noticeably increased.

From previous performance, both MAE and RMSE have more than doubled, while MSE has nearly tripled. It means that overfitting or changes in learning rate may result in a decline in prediction accuracy over time. Moreover, Mean Absolute Percentage Error (MAPE) also increases from 6.63751 to 22.2062, implying reduced predictability of estimating real values. L_2 and L_∞ norm have higher errors with L_∞ showing a smaller relative increase, meaning that although there is an overall increase in errors spread by the model, maximum errors do not change as great as before. This clearly shows when the peak performance window for the model is and underscores its importance, particularly after particular epochs where training parameters need monitoring and possibly tweaking to maintain optimal performance.

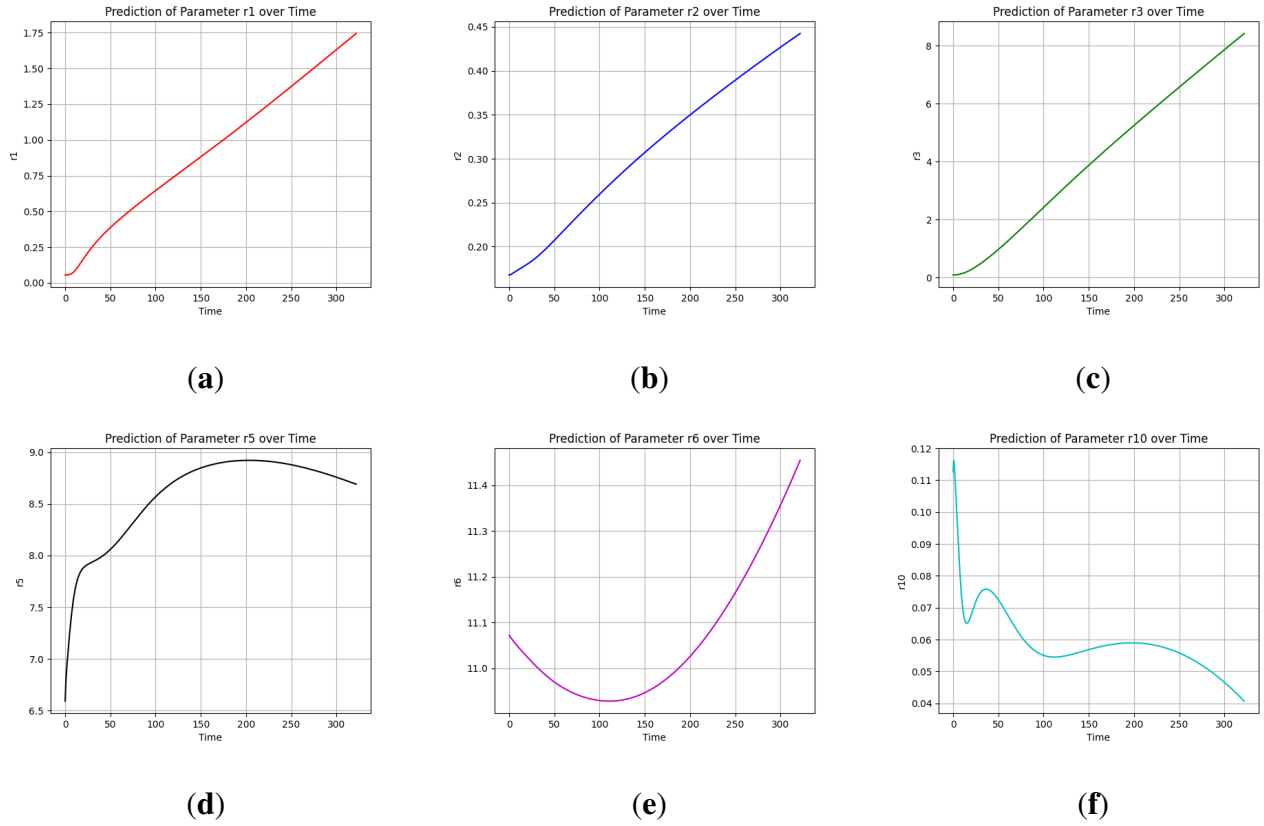
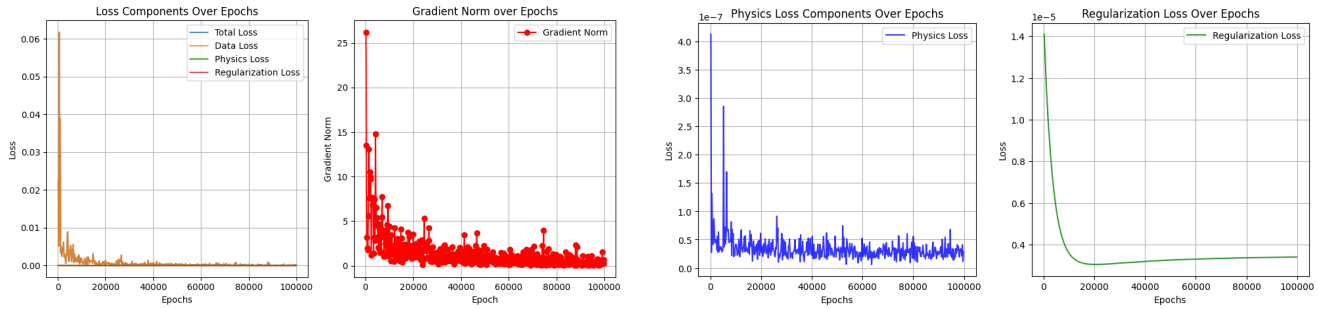


Figure 4.14: The learned time-varying parameter values of r_1, r_2, r_3, r_5, r_6, and r_{10} of High Irradiance Response.



(**a**) The loss functions and the gradient norm (**b**) The physics loss and the regularization loss

Figure 4.15: The loss components and gradient norm against epochs

Figure 4.15 demonstrates how well the PITLNN performs over 100,000 epochs with emphasis on loss components and gradient norms. Initially, all loss components, namely data, physics, and regularization, undergo sharp decreases before stabilizing very fast at low values. This demon-

strates that the model can adjust its parameters early in training to minimize errors while fitting data to physics laws. The gradient norm graph initially exhibits spikes, indicating significant weight adjustments. However, it quickly stabilizes, signaling a shift to a more reliable solution with fewer variations. The network shows its effective learning and stability by keeping consistently low values of gradient norm, which is important for continuous learning without steep changes in parameter updates. These observations reflect a well-configured training process where the network efficiently minimizes loss and handles complex dynamical learning features over long periods.

4.7 Conclusion

In this study, we utilized the Physics-Informed Transfer Learning Neural Networks to solve and learn the time-varying parameters of stiff differential equations within various scientific domains. The architecture improves learning efficiency using pre-trained models, reducing computational overhead while maintaining high accuracy. The algorithm successfully tackled stiff systems across four scenarios: the ROBER problem, a damped oscillator, stiff first-order irreversible chain reactions, and the HIRES problem. For the Robertson problem, the network excelled in capturing the complex kinetics of the system, with time-varying parameters accurately reflecting the underlying chemical reactions. Similarly, the damped oscillator model highlighted the neural network's capability to accurately predict the system's response under different damping conditions, providing deep insights into the system's dynamic behavior. The application to stiff first-order irreversible chain reactions showcased the algorithm's proficiency in handling rapid changes in reaction rates, a common challenge in chemical kinetics. The algorithm's predictions closely agreed with the expected outcomes, reinforcing its reliability in simulating chemical processes.

Moreover, the HIRES problem demonstrated the algorithm's robustness in dealing with a system of eight nonlinear stiff ordinary differential equations, a testament to its capability to handle highly complex systems. Throughout these applications, this neural network not only maintained high accuracy in predicting system behaviors but also effectively identified time-varying parameters, thus enriching our understanding of each system's dynamics. The adaptive capability of

Physics-Informed Transfer Learning Neural Networks to learn from pre-trained data and refine these learnings through exposure to new, related data sets highlights its potential as a powerful tool for predictive modeling. In summary, the implementation of the proposed in this research not only confirms its efficacy in handling a range of stiff systems but also opens avenues for its application in other complex systems across different scientific fields. The integration of physics-informed constraints and transfer learning within a neural network framework allows for a nuanced understanding of system behaviors, making Physics-Informed Transfer Learning Neural Networks a promising approach for future research and application in scientific and engineering problems characterized by complexity and stiffness.

CHAPTER 5

CONCLUSION

We developed three algorithms to solve and learn the time-varying parameters of dynamic systems. The first algorithm called the logistic-informed neural network, indicated efficiency and accuracy in predicting the number of individuals infected by the COVID-19 Omicron variant in the country where they practice strict and partial mitigation measures. The second algorithm is the optimized physics-informed neural network used to learn the time-varying parameters of different dynamic systems. Based on error metrics, our findings demonstrated that this algorithm performs more accurately, computationally, and efficiently than a fully connected deep neural network. Furthermore, we developed the third algorithm, called physics-informed transfer learning neural network, to solve and learn the time-varying parameters of stiff dynamic systems. The algorithm proved to be highly effective in handling stiff dynamic systems, showing great promise when applied to the dynamical systems.

Additionally, These algorithms collectively offer a robust framework for addressing complex, real-world problems where dynamic systems play a crucial role. For instance, the logistic-informed neural network could significantly aid public health officials in understanding and predicting the spread of infectious diseases, enabling the development of more effective containment strategies. Similarly, the optimized physics-informed neural network provides a powerful tool for scientists and engineers to model and understand various physical phenomena, from climate dynamics to materials science. The third method, leveraging physics-informed transfer learning, extends the applicability of neural networks to stiff dynamic systems, which are notoriously difficult to solve due to their sensitivity to initial conditions and parameter changes. This method's ability to efficiently handle such systems opens new paths for study in areas where these kinds of systems are common, like in the study of how chemicals react with each other, how nuclear reactors operate, and other similar topics.

Finally, our work indicates the potential of integrating deep learning with dynamic systems modeling to address some of the most pressing challenges society faces today. By considering the randomness in the model and the variability of human behavior in transmitting contagious diseases, we demonstrate how our algorithms can help public health officials develop more effective vaccination and mitigation strategies. This approach enhances our understanding of complex systems and provides a practical framework for decision-making in public health, environmental policy, and beyond, offering a pathway to more resilient and adaptable strategies in the face of uncertainty.

BIBLIOGRAPHY

[1] Borzi A. *Modelling with Ordinary Differential Equations: A Comprehensive Approach*, volume 1. Taylor and Francis Group, LLC, Abingdon, UK, 2020.

[2] Olumoyin K.; Khaliq A.; and Furati K. Data-driven deep-learning algorithm for asymptomatic COVID-19 model with varying mitigation measures and transmission rate. *Epidemiologia*, 2:471–489, 2021.

[3] Kartono A.; Wahyudi S.T.; Setiawan A.A.; and Sofian I. Predicting of the coronavirus disease 2019 (COVID-19) epidemic using estimation of parameters in the logistic growth model. *Infectious Disease Reports*, 13(2):465–485, 2021.

[4] Lotka A.J. *Elements of Physical Biology*. Williams and Wilkins Company, Philadelphia, PA, USA, 1925.

[5] Long J.; Khaliq A.Q.M.; and Furati K.M. Identification and prediction of time-varying parameters of COVID-19 model: A data-driven deep learning approach. *International Journal of Computer Mathematics*, 98(8):1617–1632, 2021.

[6] Goswami S.; Jagtap A.D.; Babaee H.; Susi B.T.; and Karniadakis G.E. Learning stiff chemical kinetics using extended deep neural operators. *Computer Methods in Applied Mechanics and Engineering*, 419:116674, 2024.

[7] Centers for Disease Control and Prevention. Omicron variant: What you need to know. https://www.cdc.gov/coronavirus/2019-ncov/variants/omicron-variant.html, 2022.

[8] Franois Chollet. *Deep Learning with Python*. Simon and Schuster, New York, NY, USA, 2017.

[9] Fanelli D. and Piazza F. Analysis and forecast of COVID-19 spreading in china, italy and france. *Chaos, Solitons and Fractals*, 134:109761, 2020.

[10] Radovic A.; Williams M.; Rousseau D.; et al. Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 560:41–48, 2018.

[11] Temesgen D.T. Deep neural network for system of ordinary differential equations: Vectorized algorithm and simulation. *Machine Learning and Applications*, 5:100058, 2021.

[12] Callaway E. Making sense of coronavirus mutations. *Nature*, 585(7824):174–177, 2020.

[13] Oluwasakin E.O.; and Khaliq A.Q.M. Data-driven deep learning neural networks for predicting the number of individuals infected by COVID-19 omicron variant. *Epidemiologia*, 4:420–453, 2023.

[14] European Centre for Disease Prevention and Control (ECDC). Data on SARS-2 variants in the eu/eea. https://www.ecdc.europa.eu/en/publications-data/data-virus-variants-{{{{COVID-19}}}}-eueea, 2022.

[15] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[16] Cybenko G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[17] Fokas A.S.; Dikaios N.; Kastis G.A. Mathematical models and deep learning for predicting the number of individuals reported to be infected with sars-cov-2. *The Royal Society Interface*, 17(169):20200494, 2020.

[18] Gustafson G.B. *Differential Equations and Linear Algebra, Undergraduate Mathematics Science and Engineering*. Amazon Kindle Direct Publishing, Amazon, 2022.

[19] Eyring H.; and Polanyi M.Z. Simple gas reactions. *Journal of Physical Chemistry B*, 12:279–311, 1931.

[20] Ge Y.; Zhang W.; Wu X.; Ruktanonchai C.W.; Liu H.; and Wang J. Untangling the changing impact of non-pharmaceutical interventions and vaccination on european COVID-19 trajectories. *Nature Communications*, 13:3106, 2022.

[21] Ying-Hen H. *Richards Model: A Simple Procedure for Real-time Prediction of Outbreak Severity*. 2009.

[22] Prigogine I.; and Lefever R. Symmetry breaking instabilities in dissipative systems ii. *Journal of Chemical Physics*, 48:1665–1700, 1968.

[23] Chen R.T.Q.; Rubanova Y.; Bettencourt J.; and Duvenaud D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[24] Lambert J. *Numerical Methods for Ordinary Differential Systems: The Initial Value Problem*. Wiley, 1992.

[25] Oluwasakin E.; Torku T.; Tingting S.; Yinusa A.; Hamdan S.; Poudel S.; Hasan N.; Vargas J.; and Poudel K. Minimization of high computational cost in data preprocessing and modeling using mpi4py. *Machine Learning with Applications*, 13:100483, 2023.

[26] Voss H.; Timmer J.; and Kurths J. Nonlinear dynamical system identification from uncertain and indirect measurements. *International Journal of Bifurcation and Chaos in Applied Sciences and Engineering*, 14:1905–1933, 2004.

[27] Weiqi j.; Weilun q.; Zhiyu S.; Shaowu p.; Deng S. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *Journal of Physical Chemistry A*, 125(36), 2021.

[28] Belet A.; Alahmadi.; Black S.; Cromer A.; Flegg D.; House J.A.; Jayasundara T.; Keith T.; McCaw J.M.; and Moss J.M. Influencing public health policy with data-informed mathematical models of infectious diseases: Recent developments and new challenges. *Epidemics*, 32:100393, 2020.

[29] Xue L.; Jing S.; Miller J.C.; Sun W.; Li H.; Estrada-Franco J.G.; Hyman J.M.; and H. Zhu. A data-driven network model for the emerging COVID-19 epidemics in wuhan, toronto, and italy. *Mathematical Biosciences*, 326:108391, 2020.

[30] Hornik K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[31] Jacobsen K.H. COVID-19 generates global preparedness lancet. *International Journal of Environmental Research and Public Health*, 395(10229):1013–1014, 2020.

[32] Prantikos K.; Chatzidakis S.; Tsoukalas L.H.; and Heifetz A. Physics-informed neural network with transfer learning (tl-pinn) based on domain similarity measure for prediction of nuclear reactor transients. *Scientific Reports*, 13:16840, 2023.

[33] Calderhead B.; Girolami M.; and Lawrence N. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. In *Advances in Neural Information Processing Systems*, volume 21, 2008.

[34] Torku K.; Khaliq A. Q. M.; and Furati K. M. Deep-data-driven neural networks for COVID-19 vaccine efficacy. *Epidemiologia*, 2(4):564–586, 2021.

[35] Yang Y.; Hou M.; and Luo J. 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, 2018.

[36] Goswami S.; Kontolati K.; Shields M.D.; and George E.K. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, 4:1155–1164, 2022.

[37] Goswami S.; Kontolati K.; Shields M.D.; and Karniadakis G.E. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, 4:1155–1164, 2022.

[38] Baden N.; and Villadsen J. A family of collocation-based methods for parameter estimation in differential equations. *Chemical Engineering Journal*, 23:1–13, 1982.

[39] Kalogerakis N.; and Luus R. Improvement of gauss-newton method for parameter estimation through the use of information index. *Industrial Engineering Chemistry Fundamentals*, 22:436–445, 1983.

[40] Minaee S.; Boykov Y.Y.; Porikl F.; Plaza A.J.; Kehtarnavaz N.; and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:3523–3542, 2022.

[41] National Health Commission of the Peoples Republic of China. Epidemic notification. http://www.nhc.gov.cn/xcs/yqtb/list_gzbd.shtml, 2022.

[42] National Institute of Health. Genetic diversity of the new coronavirus sars-cov-2 (COVID-19) in portugal. https://insaflu.insa.pt/covid19/relatorios/INSA_SARS_CoV_2_DIVERSIDADE_GENETICA_relatorio_situacao_2022-05-31.pdf, 2022.

[43] A.Q.M. Oluwasakin, E.O.; Khaliq. Optimizing physics-informed neural network in dynamic system simulation and learning of parameters. *Algorithms*, 16:547, 2023.

[44] Dixit P.; and S. Silakari. Deep learning algorithms for cybersecurity applications: A technological and status review. *Computer Science Review*, 39:100317, 2021.

[45] Domguia U.S.; Tchakui M.V.; Herve S.; Woafo P. Theoretical and experimental study of an electromechanical system actuated by a brusselator electronic circuit simulator. *Vibrations and Acoustics*, 139:061017, 2017.

[46] Dong S.; Wang P.; and Abbas K. A survey on deep learning and its applications. *Computer Science Review*, 40:100379, 2021.

[47] Raissi M.; Perdikaris P.; and Karniadakis G.E. Physics informed deep learning: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[48] Fokas A.S.; Cuevas-Maraver J.; Kevrekidis P.G. Two alternative scenarios for easing COVID-19 lockdown measures: one reasonable and one catastrophic. *Scientific Reports*, 11:5839, 2021.

[49] Varziri M.S.; Poyton A.A.; McAuley K.B.; McLellan P.J.; and Ramsay J.O. Selecting optimal weighting factors in ipda for parameter estimation in continuous-time dynamic models. *Computers Chemical Engineering*, 32:3011–3022, 2008.

[50] Bellman R.; and strm K. J. On structural identifiability. *Mathematical Biosciences*, 7(3-4):329–339, 1970.

[51] Field R.J.; and Noyes R.M. Oscillations in chemical systems. iv. limit cycle behavior in a model of a real chemical reaction. *Chemical Physics*, 60:1877–1884, 1974.

[52] Anderson R.M. and May R.M. The population dynamics of microparasites and their invertebrate hosts. *Philosophical Transactions of the Royal Society of London*, pages 451–524, 1981.

[53] Das A.; Sengupta P.; Dutta S.; Roychoudhury S.; Choudhury A.P.; Ahmed A.B.F.; Bhattacharjee S.; and Slama P.;. Viral pandemics of the last four decades: Pathophysiology, health impacts and perspectives. *International Journal of Environmental Research and Public Health*, 17(24):9411, 2020.

[54] Qureshi S.; and Yusuf A. Mathematical modeling for the impacts of deforestation on wildlife species using caputo differential operator. *Chaos, Solitons Fractals*, 126:32–40, 2019.

[55] Pan S.J.; and Yang Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[56] Temesgen D.T.; Chimdessa S.Y.; and Carlos F.P. Parameter estimation for dynamical systems using a deep neural network. *Applied Computational Intelligence and Soft Computing*, 2022:2014510, 2022.

[57] Viguerie A.; Lorenzo G.; Auricchio F.; Baroli D.; Hughes T.J.; Patton A.; Reali A.; Yankeelov T.E.; and Veneziani A. Simulating the spread of COVID-19 via a spatially-resolved susceptible-exposed-infected-recovered-deceased (seird) model with heterogeneous diffusion. *Applied Mathematics Letters*, 111:106617, 2021.

[58] Dua V. An artificial neural network approximation based decomposition approach for parameter estimation of system of ordinary differential equations. *Computers Chemical Engineering*, 35:545–553, 2011.

[59] Volterra V. Variazioni e fluttuazioni del numero dindividui in specie animali conviventi. *Memorie della R. Accademia Nazionale dei Lincei*, 2:31–113, 1926.

[60] Mathias S.; Ole W.; and Simon O. Implicit transfer operator learning: Multiple time-resolution surrogates for molecular dynamics. *arXiv*, 2023.

[61] Katare S.; Bhan A.; Caruthers J.M.; Delgass W.N.; and Venkatasubramanian V. A hybrid genetic algorithm for efficient parameter estimation of large kinetic models. *Computers Chemical Engineering*, 28:2569–2581, 2004.

[62] Kermack W.O.; and McKendrick A.G. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 115:700–721, 1927.

[63] World Health Organisation. Data available. https://covid19.who.int/data, Accessed 2023.

[64] World Health Organisation. Tracking SARS-CoV-2 variants. https://www.who.int/activities/tracking-SARS-CoV-2-variants, Accessed 2023.

[65] World Health Organization. Archived: WHO timeline-COVID-19. https://www.who.int/emergencies/diseases/novel-coronavirus-2019, 2021.

[66] Esposito W.R.; and Floudas C.A. Global optimization for the parameter estimation of differential-algebraic systems. *Industrial Engineering Chemistry Research*, 39:1291–1310, 2002.

[67] McCulloch W.S.; and Pitts W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[68] Jia L.; Li K.; Jiang Y.; Guo X.; and T. Zhao. Prediction and analysis of coronavirus disease 2019. *PLoS One*, 15(10):e0239960, 2020.

[69] Ning X.; Jia L.; Wei Y.; Li X.; and Chen F. Epi-dnns: Epidemiological priors informed deep neural networks for modeling COVID-19 dynamics. *Computers in Biology and Medicine*, 158:106693, 2023.

[70] Goodfellow I.; Bengio Y.; and Courville A. *Deep Learning*. MIT Press, Cambridge, 2016.

[71] LeCun Y.; Bengio Y.; and Hinton G. Deep learning. *Nature*, 521:436–444, 2015.

[72] Li X.; Zai J.; Zhao Q.; Nie Q.; Li Y.; and Foley B.T. Evolutionary history, potential intermediate animal host, and cross-species analyses of sars-cov-2. *Journal of Medical Virology*, 92(6):602–611, 2020.

[73] Lin Q.; Zhao S.; Gao D.; Lou Y.; Yang S.; Musa S.; Wang M.; Cai Y.; Wang W.; Yang Y.; and He D. A conceptual model for the coronavirus disease 2019 (COVID-19) outbreak in wuhan, china with individual reaction and governmental action. *International Journal of Infectious Diseases*, 94:211–216, 2020.

[74] Lv Y.; and Liu Z. Turing-hopf bifurcation analysis and normal form of a diffusive brusselator model with gene expression time delay. *Chaos, Solitons Fractals*, 152:111478, 2021.

[75] Wang S.; Teng Y.; and Perdikaris P. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[76] Yosinski J.; Clune J.; Bengio Y.; and Lipson H. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems*, 27, 2014.