Physics-Informed Neural Networks: Investigating the Potentials and Boundaries of DeepXDE in Dielectric Property Exploration

By

Reyhaneh Taj

Supervisors:

Olof Hjortstam
Tor Laneryd

Hitachi Energy Research Center

March 18, 2024

Abstract

Scientific machine learning (SciML) represents a notable advancement in merging machine learning (ML) with scientific methodologies. At the forefront of this progress are Physics-Informed Neural Networks (PINNs), offering a promising solution. Unlike conventional Neural Networks, PINNs leverage physical principles, reducing the reliance on extensive datasets. Automatic Differentiation further streamlines the integration of physical models into the network, enhancing its adaptability and effectiveness.

In this thesis, we explore the capabilities and constraints of DeepXDE, a powerful framework that leverages PINNs, in addressing both forward and inverse problems related to understanding dielectric properties.

Acknowledgments

I would like to express my deepest gratitude to my supervisors at Hitachi Energy, Olof Hjortstam and Tor Laneryd, for their invaluable guidance, continuous support, and insightful feedback throughout the entirety of my research work. Their expertise and dedication have been instrumental in shaping this thesis.

I am also indebted to Michele Luvisotto and Alireza Nami for their unwavering trust and encouragement. Their belief in my abilities has been a constant source of motivation.

Furthermore, I extend my sincere appreciation to all the colleagues, friends, and family members who have supported me with their encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The transition towards reducing greenhouse gas emissions globally necessitates significant changes in the electrical energy system. High Voltage Direct Current (HVDC) transmission is a powerful technology for efficiently transmitting electrical power across long distances. To ensure the reliability and optimization of HVDC insulation systems, it is essential to understand the dielectric properties of solid insulation materials. Most dangerous breakdowns are caused by the aging effects of HV insulation systems used within [HV] components, and there is still a lack of appropriate tools to diagnose such systems non-destructively and reliably in the field [1]. Dielectric spectroscopy[2] in the time domain plays a crucial role in providing key insights into these materials, particularly in applications such as HVDC converter transformers where liquid impregnated cellulose insulation is employed.

Machine learning (ML) has brought about a transformative paradigm shift in scientific practice. At the forefront of this revolution is scientific machine learning (SciML) [3], [4], [5]. The primary objective of SciML is to intricately integrate established scientific knowledge with ML methodologies, creating robust ML algorithms that leverage our prior understanding to achieve greater efficacy. In recent years, the field has witnessed the emergence of a method known as Physics-Informed Neural Networks (PINNs) within the realm of artificial intelligence (AI) [6].

Diverging from conventional Neural Networks (NNs), which often demand datasets to yield accurate results, PINNs capitalize on an inherent grasp of physical principles, mitigating the need for extensive training data while bolstering computational efficiency. At the heart of PINNs lies Automatic Differentiation (AD), a pivotal mechanism enabling the network to compute derivatives concerning neural network parameters, input data, and biases. This automated derivation is pivotal for navigating the intricate landscape of partial differential equations (PDEs)[7] and their associated boundary conditions, empowering the

seamless integration of diverse physical models within the network architecture.

Representing a cutting-edge fusion of advanced physics principles, typically described by partial differential equations, with the adaptability of neural networks, PINN is a promising approach to addressing the complexities inherent in HV insulation systems.

1.2 Project Objective

The objective of this project is to explore the capabilities and constraints of DeepXDE [8] , a framework based on Physics-Informed Neural Networks (PINNs), in addressing both forward and inverse problems. Additionally, we aim to analyze current measurements obtained during dielectric spectroscopy conducted in the time domain.

1.3 Outline

In Chapter 2, we provide a concise overview of Neural Networks (NN) followed by an in-depth description of Physics-Informed Neural Networks (PINNs), covering both the Forward Solution and Inverse Discovery Mode. Chapter 3 delves into the Description of our Physical Model Formulations, wherein we illustrate and formulate the simplified physical system under study. Moving forward to Chapter 4, our focus shifts to the Implementation of DeepXDE, a crucial step in our methodology. In Chapter 5, we present the Results and Analysis derived from our investigation, providing insights into the performance and implications of our approach. Finally, in Chapter 6, we draw Conclusions and outline Future Directions, highlighting areas for further research and development in the field.

Chapter 2

Physics-Informed Neural Networks (PINNs)

2.1 Introduction

In the era of scientific machine learning (SciML), the fusion of established scientific knowledge with machine learning techniques is revolutionizing research methodologies. This transformation is prominently exemplified by the application of physics-informed neural networks (PINNs) to tackle partial differential equations (PDEs). Differential equations (DEs) serve as fundamental tools for understanding how variables change in relation to each other over time. Enter physics-informed neural networks (PINNs): these innovative systems generate outputs that precisely match the behaviors prescribed by DEs, whether applied in physics, engineering, or other fields. Conversely, inverse physics-informed neural networks (iPINNs) operate in reverse, deducing the parameters of the underlying DEs based on observed responses. Both PINNs and iPINNs undergo training with a crucial constraint, ensuring that the neural network's input-output relationship precisely aligns with the DE being modeled, all achieved without the need for a large amount of data. This approach merges the power of neural networks with the precision of DEs, offering a versatile framework for modeling complex systems and phenomena across various disciplines.

This chapter is dedicated to explaining the concepts of Neural Networks as a preliminary step before delving into PINNs. Subsequently, it will elucidate the concepts of PINNs with a focus on both the forward model and inverse model.

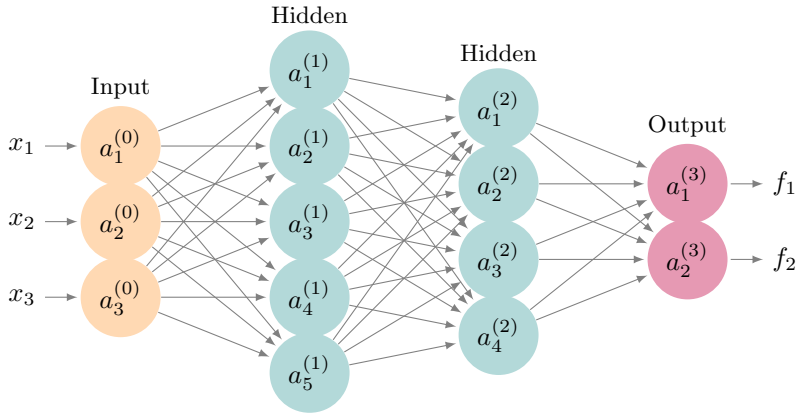


Figure 2.1: Neural Network Architecture with 3 input neurons, 5 neurons in the first hidden layer, 4 neurons in the second hidden layer, and 2 output neurons.

2.2 Feed-forward Neural Networks

Artificial Neural Networks (ANNs) [9] , as shown in Figure 2.1, are powerful models in machine learning, featuring a graph-like structure where nodes act as artificial neurons connected by weighted edges. Their ability to learn complex patterns and relationships in data makes them a valuable tool in solving real-world problems.

Feed-forward Structure

- **Input Layer:** The input layer comprises neurons that receive the initial data inputs. Each neuron in this layer represents a feature or attribute of the input data. In the provided neural network architecture, there are 3 input neurons.
- **Hidden Layers:** Hidden layers are intermediary layers between the input and output layers where computation occurs. Each hidden layer consists of neurons that transform the input data through weighted connections and apply activation functions to introduce non-linearity. In the the provided neural network architecture, there are two hidden layers with 5 neurons in the first hidden layer and 4 neurons in the second hidden layer.
- **Output Layer:** The output layer produces the final predictions or outputs of the neural network. Neurons in this layer represent the desired output variables. The number of neurons in the output layer depends on the nature of the task. In the illustrated architecture, there are 2 output neurons.

- **Activation Functions:**

  Activation functions introduce non-linearity into the network's computations, enabling the network to learn complex patterns in the data. Common activation functions include the sigmoid function, hyperbolic tangent function (tanh), Rectified Linear Units (ReLUs), Leaky ReLU, Parametric ReLU (PReLU), etc. However, the sigmoid function, though historically popular, suffers from issues like vanishing gradients and non-zero-centered outputs, making alternatives like tanh and ReLU variants more preferred in modern architectures.

The activation a_l^n of neuron n in layer l is computed as follows:

$$z_l^n = W_l \cdot a_{l-1} + b_l$$

$$a_l^n = \sigma(z_l^n)$$

where W_l is the weight matrix for layer l, b_l is the bias vector for layer l, a_{l-1} is the activation vector from the previous layer, and σ is the activation function.

The network processes input x through computations, generating activations a_l^n in each layer. The entire process represents the network as $f : X \to Y$, where X is the input space and Y is the output space.

Training Phase

During training, the aim is to minimize Mean Squared Error (MSE) loss function $L_{\text{data}}(\theta)$, which measures the difference between predicted and actual values, and θ is selected the set of parameters including weight matrices W_l and bias vectors b_l for each layer l.

The training set $D := \{(x_k, y_k)\}_{k=0}^K$ comprises observational data with $K+1$ tuples. In supervised learning, we train the network to learn a mapping $f : X \to Y$ consistent with observed data.

MSE Loss Function

The Mean Squared Error (MSE) loss function $L_{\text{data}}(\theta)$ measures the average squared difference between the predicted outputs $f(x_k; \theta)$ and the actual targets y_k for the training dataset.

$$L_{\text{data}}(\theta) = \frac{1}{K+1} \sum_{k=0}^{K} \|y_k - f(x_k; \theta)\|^2 \geq 0$$

The objective during training is to minimize this loss function, seeking parameters θ that lead to the smallest possible value of $L_{\text{data}}(\theta)$.

$$\min_{\theta} L_{\text{data}}(\theta)$$

However, due to the complexity of neural networks, obtaining an analytic solution ($\hat{\theta}$) is impractical. Instead, numerical optimization schemes like gradient descent are employed. Gradient descent iteratively updates the parameters θ in the direction that reduces the loss, using the gradient of the loss function with respect to θ.

$$\theta \leftarrow \theta - \eta \nabla L_{\text{data}}(\theta)$$

Here, η represents the learning rate, controlling the size of the steps taken in the parameter space during optimization. By repeatedly applying this update rule, the network gradually converges to a set of parameters that minimize the loss function.

Backpropagation

The backpropagation algorithm is a key component of training neural networks. It efficiently computes the gradient of the loss function with respect to the network's parameters. This process enables the network to update its parameters in a way that minimizes the loss.

Backpropagation works by propagating the error backward from the output layer to the input layer. It computes the gradient of the loss function using the chain rule of calculus, which allows for the efficient calculation of gradients across all layers of the network.

By iteratively updating the parameters based on the computed gradients, backpropagation guides the network towards better performance, ultimately improving its ability to make accurate predictions.

Overall, backpropagation plays a crucial role in training neural networks, allowing them to learn from data and adapt their parameters to improve their performance on a given task.

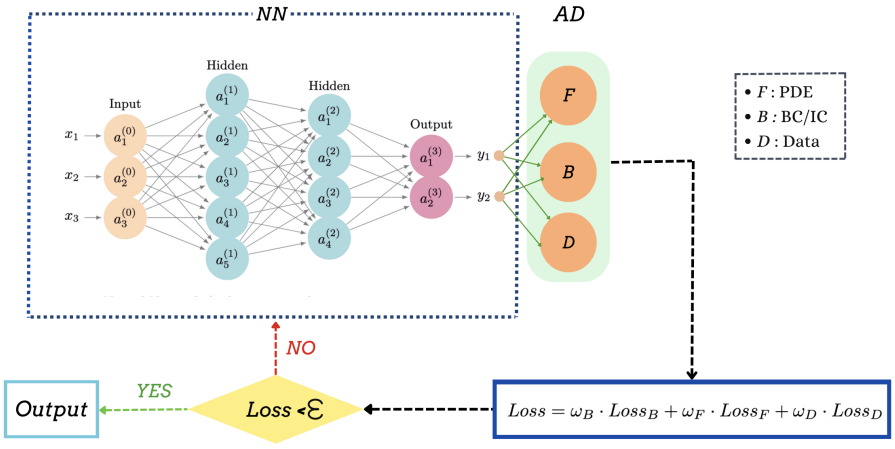


Figure 2.2: Overview of Physics-Informed Neural Networks (PINNs) architecture.

2.3 Physics-Informed Neural Networks (PINNs)

Simulating complex real-world systems presents significant computational hurdles, often requiring techniques like adaptive mesh refinement, subgrid parameterizations, and reduced-order modeling to manage costs. Deep neural networks excel in accuracy but demand extensive training data, a luxury often unavailable when analyzing intricate physical systems due to prohibitive data acquisition costs. Consequently, decisions must be made with partial information, necessitating a delicate balance between accuracy and cost. PINNs offer a solution by leveraging prior knowledge such as validated physical laws or domain expertise as a regularization mechanism. This prior knowledge constrains the solution space, reducing the need for vast amounts of training data and making PINNs an efficient alternative in scenarios where data scarcity is a concern.

Figure 2.2 provides a general overview of the PINNs architecture, comprising fundamental building blocks. These include residual terms from differential equations (loss terms), along with considerations for initial and boundary conditions. In the inverse mode, data loss is also considered. The network's inputs, representing variables, undergo a transformation to yield network outputs, essentially the field $u(x,t)$. The network is characterized by a set of parameters denoted by θ.

The second crucial aspect is the physics-informed network, which takes the output field $u(x,t)$ and computes its derivatives based on the governing equations. It also assesses boundary and initial conditions, unless they are already embedded as specific physics within the neural network. Moreover, if labeled data observations are accessible, they are integrated into the computations as

well.

The final step involves a feedback mechanism that minimizes the overall loss. This optimization process, facilitated by an optimizer with a specified learning rate, aims to adjust the neural network parameters (θ) to enhance the model's predictive capabilities.

In the following subsections, we will explore two key modes of PINNs: the forward mode and the inverse mode. It's worth noting that there are two distinct approaches in time domain modeling continuous and discrete. Here, we will focus on explaining the continuous approach.

2.3.1 The Forward Solution Mode

Performing forward simulation is a key task in predicting specific properties of a system based on input conditions. At a broad level, this can be expressed as the equation $b = F(a)$, where a represents the input conditions, F embodies a physical model of the system, and b encompasses the resulting properties derived from applying F to a. Depending on the problem at hand, a and b may involve scalars, vectors, tensors, functions, or other system descriptors.

Simulation involves solving a set of differential equations that describe the system. The general representation of the system often takes the form:

$$\mathcal{F}[u(z); \lambda] = f(z), \quad z \in \Omega, \quad \mathcal{B}[u(z)] = g(z), \quad z \in \partial\Omega,$$

where $\mathcal{F}$ is a differential operator, $\mathcal{B}$ represents boundary operators, $u \in R^{du}$ is the solution to the differential equation, $f(z)$ is a forcing function, $g(z)$ is a set of boundary functions, x is an input vector in the domain $\Omega \subset R^d$ (with $z := [x_1, \ldots, x_{d-1}; t]$), $\partial\Omega$ denotes the boundary of Ω, and λ is a set of parameters related to the physics. Many different differential equations can be represented in this form, including those with time-dependence or time-independence, and linear or nonlinear operators.

In the forward mode, the goal is to find the approximate $u(z)$ where λs are specified parameters, using a deep neural network, resulting in a PINN $f(z)$. Automatic differentiation(AD) facilitates the chain rule for differentiating compositions of functions. The loss function governing the forward solution is defined as:

In this context, the neural network (NN) as illustrated in Figure 2.2, is tasked with learning to approximate the differential equations for identified λ parameters that govern the behavior of the considered physical system. This involves minimizing a loss function intricately tied to the equation:

$$L(\theta) = \omega_B \cdot L_B(\theta) + \omega_F \cdot L_F(\theta) \quad \text{(Equation 4)}$$

where the ω_i, $i \in \{F, B\}$ are weights for the specific losses L_i, respectively defined for the PDE residual, the boundary/initial condition. ω_i determine the relative importance of each term in the overall loss function. Adjusting these weights allows for fine-tuning the neural network's learning process and can impact the optimization performance and convergence behavior. Assuming ω is

set to one means each term is equally weighted in the loss function. $L_B(\theta)$ corresponds to the discrepancy between the model predictions and initial/boundary data, while $L_F(\theta)$ enforces the structure imposed by Equation (2) at a finite set of collocation points. These collocation points are placed within the domain, serving as crucial locations where the PDE solution is enforced. Here, PINN can be viewed as an unsupervised learning approach when it is trained solely using physical equations and boundary conditions for forward problems.

Assuming ω_i equals one, the loss function $L(\theta)$ is defined as:

$$L(\theta) = \omega_B \cdot \frac{1}{N_B} \sum_{i=1}^{N_B} \left(B[u_{NN}(z_i;\theta)]\right)^2 \quad \text{(where } N_B \text{ is the number of boundary points)}$$

$$+\omega_F \cdot \frac{1}{N_C} \sum_{j=1}^{N_C} \left(D[u_{NN}(z_j;\theta)]\right)^2 \quad \text{(where } N_C \text{ is the number of collocation points)}$$

The objective during training is to minimize the loss function $L(\theta)$, seeking parameters θ that lead to the smallest possible value of $L(\theta)$:

$$\min_{\theta} L(\theta)$$

2.3.2 The Inverse Discovery Mode

Physics-Informed Neural Networks (PINNs) emerge as a powerful tool by amalgamating the strengths of physics-based modeling and machine learning, particularly in the inverse mode. Inverse problems, closely linked to simulation, play a critical role in real-world tasks where the objective is to estimate latent parameters of a system from observed outputs. Described within the framework $F(a) = b$, the task is to estimate a given b. In science and engineering, these problems are pervasive and involve the estimation of unknown parameters, properties, or inputs, encountering challenges like ill-posedness, non-linearity, and noisy data.

The general representation of the system mirrors the equation mentioned in the forward mode ($D[u(x,t;\lambda)] = f(x,t)$, $B_k[u(x,t;\lambda)] = g_k(x,t)$, x_k), the parameters of the differential operator λ turn into parameters of the PINNs f (t, x). Also synthetic or measured data is incorporated into the loss function. This addition leads to an extended loss function:

$$L(\theta,\lambda) = \omega_B \cdot L_B(\theta,\lambda) + \omega_F \cdot L_F(\theta,\lambda) + \omega_D \cdot L_D(\theta,\lambda) \quad \text{(Equation 4)}$$

Here, where the ω_i, $i \in \{F, B, D\}$ are weights for the specific losses L_i, respectively defined for the PDE residual, the boundary, and the data. Their expressions are detailed in the following section. $L_D(\theta,\lambda)$ corresponds to the discrepancy between the model predictions and the actual data, providing an

additional term that enhances the robustness of the inverse discovery mode. This adaptation allows for the incorporation of real-world data, making the estimation process more representative of the complexities inherent in practical applications. In the context of inverse problems or situations where physical properties are derived from potentially noisy data (measurement data), PINN can be regarded as a form of supervised learning methodology.

Additionally, in the inverse case, a parameter λ is introduced, transforming into parameters of the PINNs $f(z;\lambda)$. The collocation points are the number of training residual points sampled inside the domain contribute to the accuracy of the neural network approximation and enhance the reliability of the inverse solution.

$$L(\theta,\lambda) = \omega_B \cdot \frac{1}{N_B}\sum_{i=1}^{N_B}\left(B[u_{NN}(z_i;\theta,\lambda)] - u_{\mathrm{BC}}(z_i)\right)^2 \quad \text{(where } N_B \text{ is the number of boundary points)}$$

$$+\omega_F \cdot \frac{1}{N_C}\sum_{j=1}^{N_C}\left(D[u_{NN}(z_j;\theta,\lambda)]\right)^2 \quad \text{(where } N_C \text{ is collocation points)}$$

$$+\omega_D \cdot \frac{1}{N_D}\sum_{k=1}^{N_D}\left(u_{NN}(z_k;\theta,\lambda) - u_{\mathrm{true}}(z_k)\right)^2 \quad \text{(where } N_D \text{ is data points)}$$

The minimization of the loss function is conducted with respect to both the network parameters (θ) and the physical parameters (λ). It's important to note that the unknown parameter λ is a learnable parameter optimized in the same manner as the θ parameters:

$$\min_{\theta,\lambda} L(\theta,\lambda)$$

Chapter 3

Physical Model and Analytical Solution

3.1 Overview

This section explains the modeling of the dynamic behavior of dielectric materials in RC circuits. In the simplest case, we consider a series RC configuration (R_1, C_1) interacting with a DC voltage. The complexity gradually increases by adding parallel RC components to this case.

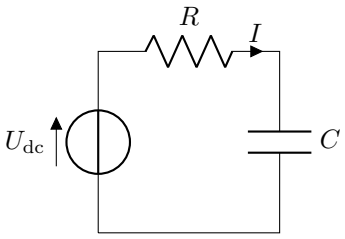
Our primary objective is to derive the governing current differential equations for each and elucidate their analytical solutions. This systematic approach aims to formulate the governing differential equation and its analytical solution for each case, which will be utilized for PINNs.

Given the simplicity of this model and a focus on reducing sensitivity, an alternate approach is considered. We propose rewriting the differential equations and analytical solutions in terms of $\ln(I)$. This modification offers insights into the behavior of the circuit from a different perspective, potentially aiding in the refinement of our model.

3.2 Differential Equations and Solutions for I

- **Case 0**

The simplest case involves a series RC circuit in contact with a DC voltage. Let's start by grounding ourselves in basic physics principles:



$$\begin{aligned} U_R &= R \cdot I && \text{(Ohm's Law)} \\ Q &= C \cdot U_C && \text{(Capacitor equation)} \end{aligned}$$

Now, applying Kirchhoff's voltage law:

$$U_{\text{dc}} = U_R + U_C$$

Upon substituting the expressions for U_R and U_C:

$$U_{\text{dc}} = R \cdot I + RC\frac{dU_C}{dt}$$

Reformulating using $U_C = U_{\text{dc}} - IR$ and $\frac{dU_C}{dt} = -R\frac{dI}{dt}$ (for constant U_{dc}):

$$U_{\text{dc}} = -RCR\frac{dI}{dt} + U_{\text{dc}} - RI$$

Simplifying the equation:

$$\begin{cases} \frac{dI}{dt} + \frac{I}{RC} = 0 & \text{Differential equation for } I \text{ in Case 0} \\ I(t=0) = \frac{U_{\text{dc}}}{R} & \text{Initial Condition for } I \end{cases}$$

The used initial condition assumes that the capacitor C is uncharged at time $t = 0$.

Analytical Solution:

The solution to the linear first-order differential equation $Y' + f(x)Y = g(x)$ is given by:

$$Y(x) = (Q(x) + K_0)\exp(-F(x))$$

In this equation, $F(x)$ represents the antiderivative of $f(x)$, while $Q(x)$ represents the antiderivative of $q(x)$, where $q(x) = g(x)\exp(F(x))$.

Applying this solution to our specific case (Case 0), where the differential equation is $\frac{dI}{dt} + \frac{I}{RC} = 0$, we find that $f(x) = \frac{t}{R \cdot C}$, $F(x) = \frac{t}{R \cdot C}$, $g(x) = 0$, and $q(x) = 0$.

Therefore, the solution for I is:

$$I = (K_0 + K_1) \cdot \exp\left(-\frac{t}{R \cdot C}\right)$$

Given the initial condition $I(t = 0) = \frac{U_{\text{dc}}}{R}$, we determine:

$$I = \frac{U_{\text{dc}}}{R} \cdot \exp\left(-\frac{t}{R \cdot C}\right)$$

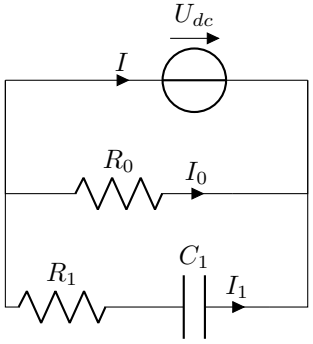
This solution describes the current I in the circuit over time.

So for the first case, we have:

$$\text{Case 0:} \begin{cases} \frac{dI}{dt} + \frac{I}{RC} = 0 & \text{Differential equation for } I \\ I(t = 0) = \frac{U_{\text{dc}}}{R} & \text{Initial Condition} \\ I = \frac{U_{\text{dc}}}{R} \cdot \exp\left(-\frac{t}{R \cdot C}\right) & \text{Analytical solution for } I \end{cases}$$

- **Case 1**

Moving on to Case 1, where the circuit includes an additional parallel branch with resistance R_0 connected to the DC voltage U_{dc}.



Starting with the differential equation for Case 1:

$$\begin{cases} I = I_0 + I_1 \\ I_0 = \dfrac{U_{\text{dc}}}{R_0} \\ I_1 = -R_1 C_1 \dfrac{dI_1}{dt} \quad = -R_1 C_1 \dfrac{dI}{dt} \end{cases}$$

so the total current is:

$$I = \frac{U_{\mathrm{dc}}}{R_0} - R_1 C_1 \frac{dI}{dt}$$

Rearranging and solving for $\frac{dI}{dt}$:

$$\frac{dI}{dt} = -\frac{1}{R_1 C_1}(I - \frac{U_{\mathrm{dc}}}{R_0})$$

Integrating both sides and Considering initial condition:

This simplifies to:

$$I = \frac{U_{\mathrm{dc}}}{R_0} + \left(\frac{U_{\mathrm{dc}}}{R_1}\right) exp(\frac{-t}{R_1 C_1})$$

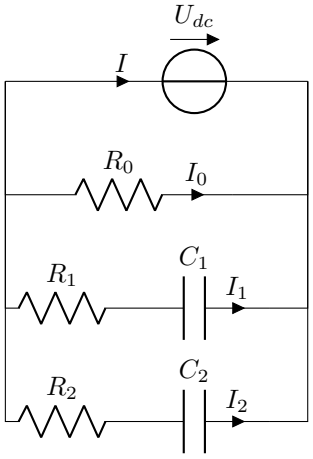
This equation describes the evolution of the current in the Case 1 RC circuit over time.

so for case1:

$$\text{Case1: } \begin{cases} \frac{dI}{dt} + \frac{1}{R_1 C_1}(I - \frac{U_{\mathrm{dc}}}{R_0}) = 0 & \text{Differential equation for } I \\ I(t=0) = \left(\frac{U_{\mathrm{dc}}}{R_0} + \frac{U_{\mathrm{dc}}}{R_1}\right) & \text{Initial Condition} \\ I = \frac{U_{\mathrm{dc}}}{R_0} + \left(\frac{U_{\mathrm{dc}}}{R_1}\right) exp(\frac{-t}{R_1 C_1}) & \text{Analytical solution for } I \end{cases}$$

- **Case 2**

  In this scenario, we extend our circuit by adding an R_2C_2 branch parallel to both R_1C_1 and R_0.



$$\begin{cases} I = I_0 + I_1 + I_2 \\ I_0 = \dfrac{U_{\text{dc}}}{R_0} \\ I_1 = -R_1C_1\dfrac{dI_1}{dt} \quad , I_1(t=0) = \dfrac{U_{\text{dc}}}{R_1} \\ I_2 = -R_2C_2\dfrac{dI_2}{dt} \quad , I_2(t=0) = \dfrac{U_{\text{dc}}}{R_2} \end{cases}$$

We can approach this case by considering the contributions from Case 0 and Case 1. Considering $I_{01} = I_0 + I_1$ we have

$$\frac{dI_{01}}{dt} = \frac{dI_1}{dt}$$

And we already have the analytical solution for I_{01} and I_2:

$$\text{Case2: } \begin{cases} \frac{dI_{01}}{dt} + \frac{1}{R_1C_1}(I_{01} - \frac{U_{\text{dc}}}{R_0}) = 0 & \text{Differential equation for } I_{01} \\ \frac{dI_2}{dt} + \frac{1}{R_2C_2}(I_2) = 0 & \text{Differential equation for } I_2 \\ I(t=0) = \left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1} + \frac{U_{\text{dc}}}{R_2}\right) & \text{Initial Condition} \\ I_{01} = \frac{U_{\text{dc}}}{R_0} + \left(\frac{U_{\text{dc}}}{R_1}\right) exp(\frac{-t}{R_1C_1}) & \text{Analytical solution for } I \\ I_2 = \left(\frac{U_{\text{dc}}}{R_2}\right) exp(\frac{-t}{R_2C_2}) & \text{Analytical solution for } I_2 \end{cases}$$

- **Case N (Generalized)**

  For Case N, where N, RC branches are connected in parallel with an R_0 the DC voltage source, the current I, can be expressed as:

$$
\text{CaseN:} \begin{cases}
\frac{dI_{01}}{dt} + \frac{1}{R_1 C_1}(I_{01} - \frac{U_{\text{dc}}}{R_0}) = 0 & \text{Differential equation for } I_{01} \\
\frac{dI_2}{dt} + \frac{1}{R_2 C_2}(I_2) = 0 & \text{Differential equation for } I_2 \\
. & \\
. & \\
. & \\
\frac{dI_N}{dt} + \frac{1}{R_N C_N}(I_N) = 0 & \text{Differential equation for } I_N \\
I(t=0) = \left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1} + ... + \frac{U_{\text{dc}}}{R_N}\right) & \text{Initial Condition} \\
I_{01} = \frac{U_{\text{dc}}}{R_0} + \left(\frac{U_{\text{dc}}}{R_1}\right) exp(\frac{-t}{R_1 C_1}) & \text{Analytical solution for } I \\
I_2 = \left(\frac{U_{\text{dc}}}{R_2}\right) exp(\frac{-t}{R_2 C_2}) & \text{Analytical solution for } I_2 \\
. & \\
. & \\
. & \\
I_N = \left(\frac{U_{\text{dc}}}{R_N}\right) exp(\frac{-t}{R_N C_N}) & \text{Analytical solution for } I_N
\end{cases}
$$

3.3 Differential Equations and Solutions for $\ln(I)$

Aiming to reduce sensitivity and achieve more consistency, in upcoming PINN implementation, we utilize the natural logarithm approach to redefine the differential equations and solutions.

- **Case 0**

  For Case 0, we derived:

$$\begin{cases} \frac{dI}{dt} + \frac{I}{RC} = 0 & \text{Differential equation for } I \\ I(t=0) = \frac{U_{\text{dc}}}{R} & \text{Initial Condition} \\ I = \frac{U_{\text{dc}}}{R} \cdot \exp\left(-\frac{t}{R \cdot C}\right) & \text{Analytical solution for } I \end{cases}$$

  To rewrite the differential equation for $\ln(I)$, we start from:

$$\frac{d}{dt}(\ln(I)) = \frac{dI/dt}{I}$$

  And from the above for Case 0, $dI/dt = -I/RC$. We will have:

$$\frac{d(\ln(I))}{dt} = \frac{-I/RC}{I}$$

  So the equation for $\ln(I)$ will be:

$$\text{Case 0: } \begin{cases} \frac{d\ln(I)}{dt} + \frac{1}{RC} = 0 & \text{Differential equation for } \ln(I) \\ \ln(I)(t=0) = \frac{U_{\text{dc}}}{R} & \text{Initial Condition} \\ \ln(I) = -\frac{t}{R \cdot C} + \ln\left(\frac{U_{\text{dc}}}{R}\right) & \text{Analytical solution for } \ln(I) \end{cases}$$

- **Case 1**

  For Case 1, we have:

$$\begin{cases} \frac{dI}{dt} + \frac{1}{R_1 C_1}(I - \frac{U_{\text{dc}}}{R_0}) = 0 & \text{Differential equation for } I \\ I(t=0) = \left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1}\right) & \text{Initial Condition} \\ I = \frac{U_{\text{dc}}}{R_0} + \left(\frac{U_{\text{dc}}}{R_1}\right) \exp\left(-\frac{t}{R_1 C_1}\right) & \text{Analytical solution for } I \end{cases}$$

To rewrite the differential equation for $\ln(I)$, we start from:

$$\frac{d}{dt}(\ln(I)) = \frac{dI/dt}{I}$$

And from the above for Case 1, $dI/dt = -\frac{1}{R_1 C_1}(I - \frac{U_{\text{dc}}}{R_0})$. We will have:

$$\frac{d(\ln(I))}{dt} = \frac{-\frac{1}{R_1 C_1}(I - \frac{U_{\text{dc}}}{R_0})}{I}$$

So the equation for $\ln(I)$ will be:

$$\text{Case 1:} \begin{cases} \frac{d\ln(I)}{dt} + \frac{1}{R_1 C_1}\left(\frac{U_{\text{dc}}}{R_0}\exp(-\ln(I)) + 1\right) = 0 & \text{Differential equation for } \ln(I) \\ \ln(I)(t=0) = \ln\left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1}\right) & \text{Initial Condition} \\ \ln(I) = \ln\left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1} \cdot \exp\left(-\frac{t}{R_1 C_1}\right)\right) & \end{cases}$$

- **Case 2**

  As Case 2 can be considered a combination of Case 0 and Case 1, the equations for $\ln(I)$ will be written as follows:

$$\text{Case 2:} \begin{cases} \frac{d\ln(I_{01})}{dt} + \frac{1}{R_1 C_1}\left(\frac{U_{\text{dc}}}{R_0}\exp(-\ln(I_{01})) + 1\right) = 0 & \text{Differential equation for } \ln(I_{01}) \\ \frac{d\ln(I_2)}{dt} + \frac{1}{R_2 C_2} = 0 & \text{Differential equation for } \ln(I_2) \\ \ln(I)(t=0) = \ln\left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1} + \frac{U_{\text{dc}}}{R_2}\right) & \text{Initial Condition} \\ \ln(I_{01}) = \ln\left(\frac{U_{\text{dc}}}{R_0} + \frac{U_{\text{dc}}}{R_1} \cdot \exp\left(-\frac{t}{R_1 C_1}\right)\right) & \text{Analytical solution for } \ln(I_{01}) \\ \ln(I_2) = -\frac{t}{R_2 \cdot C_2} + \ln\left(\frac{U_{\text{dc}}}{R_2}\right) & \text{Analytical solution for } \ln(I_2) \end{cases}$$

Additionally, the methods and equations developed for Case 2 can be generalized and extended to handle cases with more complex RC circuit configurations, denoted as CaseN:

$$
\text{CaseN:}\begin{cases}
\frac{d\ln(I_{01})}{dt} + \frac{1}{R_1C_1}\left(\frac{U_{\mathrm{dc}}}{R_0}\exp(-\ln(I_{01})) + 1\right) = 0 & \text{Differential equation for } \ln(I_{01}) \\
\frac{d\ln(I_2)}{dt} + \frac{1}{R_2C_2} = 0 & \text{Differential equation for } \ln(I_2) \\
. & \\
. & \\
. & \\
\frac{d\ln(I_2)}{dt} + \frac{1}{R_2C_2} = 0 & \text{Differential equation for } \ln(I_N) \\
\ln(I)(t=0) = \ln\left(\frac{U_{\mathrm{dc}}}{R_0} + ... + \frac{U_{\mathrm{dc}}}{R_N}\right) & \text{Initial Condition} \\
\ln(I_{01}) = \ln\left(\frac{U_{\mathrm{dc}}}{R_0} + \frac{U_{\mathrm{dc}}}{R_1}\cdot\exp\left(-\frac{t}{R_1C_1}\right)\right) & \text{Analytical solution for } \ln(I_{01}) \\
\ln(I_2) = -\frac{t}{R_2\cdot C_2} + \ln\left(\frac{U_{\mathrm{dc}}}{R_2}\right) & \text{Analytical solution for } \ln(I_2). \\
. & \\
. & \\
\ln(I_N) = -\frac{t}{R_N\cdot C_N} + \ln\left(\frac{U_{\mathrm{dc}}}{R_N}\right) & \text{Analytical solution for } \ln(I_N)
\end{cases}
$$

In this chapter, we have delved into the modeling of dielectric materials by equivalent RC circuits. Beginning with a simple series RC configuration, we gradually added complexity by incorporating parallel RC components. Through systematic derivation, we obtained the governing current differential equations for each case and provided their analytical solutions. Then, utilizing the natural logarithm approach, we explored how to rewrite the equations in terms of ln(I).

Moving forward, the derived equations and solutions will be utilized in PINNs, enabling us to develop predictive models for various RC circuit configurations.

In the subsequent chapters, we will build upon this foundation to explore applications of RC circuits in practical scenarios and investigate advanced modeling techniques to address real-world challenges.

Chapter 4

Implementation in DeepXDE

4.1 Introduction

Several frameworks exist for developing machine learning models, with TensorFlow, created by Google, being one of the most popular choices. TensorFlow offers numerous built-in modules and is particularly well-suited for deep learning tasks. However, while TensorFlow is a powerful tool, it does not provide specialized support for PINNs. Developing PINNs within TensorFlow typically requires creating every function, class, or module from scratch, which can be a time-consuming and error-prone process.

To address this challenge, the DeepXDE framework was introduced by Lu et al[]. Built on top of TensorFlow, DeepXDE serves as both an educational tool and a research platform for tackling problems in computational science and engineering using PINNs. It streamlines the development process by providing a comprehensive set of functionalities tailored specifically for solving forward and inverse problems. With DeepXDE, users can efficiently handle forward problems by specifying initial and boundary conditions, as well as tackle inverse problems using specific data.

DeepXDE is designed with simplicity and mathematical clarity in mind. The framework allows users to express mathematical formulations directly in code, resulting in concise and readable implementations. By leveraging DeepXDE, researchers and practitioners can focus on problem-solving and experimentation, rather than the intricate details of neural network implementation within TensorFlow.

This chapter provides an overview of our implementation in DeepXDE, focusing on both forward and inverse modes. We delve into the architecture and details of the implementation to predict current (I) in electrical circuits.

4.2 Forward Mode Architecture

This section presents the forward mode architecture. In the forward mode of DeepXDE, our aim is to predict the current I in electrical circuits. The forward mode begins by defining the problem domain, initial condition and constructing an exact solution.

- **Input Computational Domain**

  DeepXDE provides built-in geometries for defining computational domains. It discretize the time domain (t), treating it as our input variable. This involves dividing time into specific points, determining the temporal range for predictions. The accuracy and granularity of our forward mode predictions are influenced by this choice of time points.

- **Define our Problem Scope(specify the PDE)**

  Using Tensorflow syntax, the PDE can be specified within DeepXDE. This involves defining the differential equation,that constrain the NN in the PINN method, in terms of the desired variables and their derivatives , and formulate it as a contribution PDE's loss to the total loss function. This involves specifying the physical parameters, such as R_i, C_i which characterize the system under consideration.

- **Specify the boundary/initial condition**

  DeepXDE supports common boundary conditions such as Dirichlet, Neumann, Robin, and periodic conditions. For this study, we had just initial conditions(IC). These conditions are incorporated into the model specification.

- **Define the Exact Solution**

  The exact solution is defined based on the desired values of the parameters and the governing differential equation. In forward mode this analytical solution serves as a reference for comparison to the PINN solution.

- **Use a built-in function to create training and testing points**

  DeepXDE simplifies the generation of training and testing data by providing functions to combine the geometry, PDE, and boundary/initial conditions. Training points (collocation points) are strategically selected within the time domain, and specific initial conditions are considered. Also we can easily specify the distribution for sampling points. In the same command we incorporate analytical solutions.

- **Construct neural network**

  The construction of the neural network involves specifying its architecture, including the number of layers, types of activation functions, and neuron configurations. DeepXDE offers streamlined tools for constructing neural networks tailored to the specific problem at hand.

- **Define a model**

  Defining the model in DeepXDE is straightforward and typically requires just one line of code. This step combines the data generated with the neural network architecture defined in previous steps.

- **Set hyperparameters and compile the model**

  Hyperparameters such as optimizer choice, learning rate, and regularization techniques are set at this stage. These choices can significantly impact the training process and model performance.

- **Train the network**

  With the model defined and hyperparameters set, the network is trained. DeepXDE leverages Tensorflow's training functionality to optimize the model parameters iteratively. In our case the loss function is

  $$Loss_{PINNs} = Loss_{PDE} + Loss_{IC}$$

  Minimizing this loss is essential for accurate predictions.

- **Take Gradient Descent Step**

  Gradient descent is applied iteratively, adjusting the network parameters. This process refines the model's internal parameters, leading to convergence and improved accuracy.

- **Predict the PDE solution**

  Upon completion of training, the model is capable of predicting the desired PDE solution. This prediction can be evaluated against test data or analytical solutions if available, providing insights into the model's performance and accuracy.

4.3 Inverse Mode Architecture

In the inverse mode of DeepXDE, objective is to predict the parameters of the system. The governing physics is described by a differential equation that captures the dynamics of the system. The inverse problem involves inferring the parameters of interest, such as resistances (R)and capacitances (C), from the observed behavior of the current over time. This process requires training a neural network to learn the relationship between the input data (time domain) and the desired output (current values).

- **Specify the computational domain:**

  In the inverse mode, similar to the forward mode, we begin by specifying the computational domain and defining the input of the neural network. This involves determining the time domain over which the current (I) data is observed or measured. The accuracy and granularity of this domain significantly influence the model's ability to predict the parameters effectively.

- **Define the variables:**

  In the inverse mode, the problem scope involves specifying the parameters of interest that need to be inferred from the observed data. These parameters characterize the system under consideration. In our case, the variable parameters are R_i and C_i.

- **Define the inverse problem scope**

  Define the system function representing the differential equation system.

- **Data:**

  - **Generate Synthetic Data:** Synthetic data refers to artificially generated data that mimics real-world observations. In the inverse mode, synthetic data can be generated using the known or assumed parameter values and the governing differential equation. This synthetic data serves as training and testing data for the neural network, allowing it to learn the relationship between input parameters and observed data.
  - **Use Measured Data:** Instead of synthetic data, measured data refers to real-world observations obtained through experiments or other means. In the inverse mode, measured data can be directly used as training and testing data for the neural network. This real-world data provides more accurate and representative samples for the network to learn from, potentially improving its predictive performance.

- **Specify the boundary/initial condition**

In our considered physical model, we don't have boundary conditions. We deployed DeepXDE inverse mode in both with and without initial conditions, in order to explore what works best.

- **Use a built-in function to create training and testing points**

  Training and testing data are generated using a built-in function, incorporating the geometry, PDE, synthetic data and initial conditions(in case we considered it). Collocation points are selected within the time domain, and specific initial conditions are considered. The distribution for sampling points can also be specified, and analytical solutions can be incorporated if available.

- **Construct neural network**

  The neural network architecture is constructed, specifying the number of layers, types of activation functions, and neuron configurations. DeepXDE offers streamlined tools for constructing neural networks tailored to inverse problem settings.

- **Define a model**

  Defining the model involves combining the generated data with the neural network architecture. This step is typically accomplished with a single line of code, integrating the data and network components.

- **Set hyperparameters and compile the model**

  Hyperparameters such as optimizer choice, learning rate, and regularization techniques are set at this stage. These choices significantly impact the training process and model performance.

- **Train the network**

  The network is trained using the training data generated earlier. The loss function in the inverse mode typically involves minimizing the discrepancy between the observed data and the predictions made by the neural network. In this case, the loss function includes terms related to the PDE and initial conditions. The loss function is defined as:

$$Loss_{PINNs} = Loss_{PDE} + Loss_{Data} + (Loss_{IC})$$

  in the inverse mode minimizing the loss function involves adjusting both the parameters of the neural networks and the physical parameters.

- **Take Gradient Descent Step**

  Gradient descent in inversion mode is applied iteratively to adjust the PDE and network parameters, refining the model's internal learnable parameters of the physical system to achieve convergence and improved accuracy.

- **Predict the parameters**

  Upon completion of training, the model is capable of predicting the circuit parameters based on the observed data. These predictions is evaluated against analytical solutions, and its parameters (R_i) and (C_i), providing insights into the model's performance and accuracy.

Chapter 5

Results

5.1 Introduction

This chapter presents the outcomes of our investigation into the application of DeepXDE for analyzing a simplified model of dielectrics as electrical circuits. The results, obtained through both forward solution and inverse discovery modes, illuminate the model's performance across a spectrum of circuit scenarios.

In the forward mode, our focus on predicting the current (I) spans from simple configuration (Case0) to intricate scenarios (Case1, Case2, and Case3). Section 5.2 provides a detailed analysis, including comparisons between DeepXDE's predictions and analytical solutions. A hyperparameter optimization investigation is outlined, with a special emphasis on the efficacy of the natural logarithm approach in overcoming challenges. We discuss the model's strengths and limitations, offering a nuanced understanding of its predictive capabilities within varying time domains and circuit complexities.

Transitioning to the inverse mode, our attention turns to predicting circuit parameters, specifically resistance (R) and capacitance (C), based on anticipated current profiles. Section 5.3 explores the outcomes of the inverse mode analysis, delving into challenges and strategies deployed to enhance accuracy. This section also underscores the pivotal role of the natural logarithm approach in refining the precision and reliability of parameter predictions. As a post-processing step, a detailed analysis serves to evaluate the accuracy and efficiency of DeepXDE's inverse mode in predicting the parameters of a circuit.

In conclusion, this thesis systematically navigates the complexities of applying DeepXDE to electrical circuit analysis as a simplified model of dielectrics. The results signify an advancement in our understanding, shedding light on the strengths and limitations of DeepXDE. This exploration contributes to a comprehensive assessment of its capabilities, revealing avenues for future research and refinement.

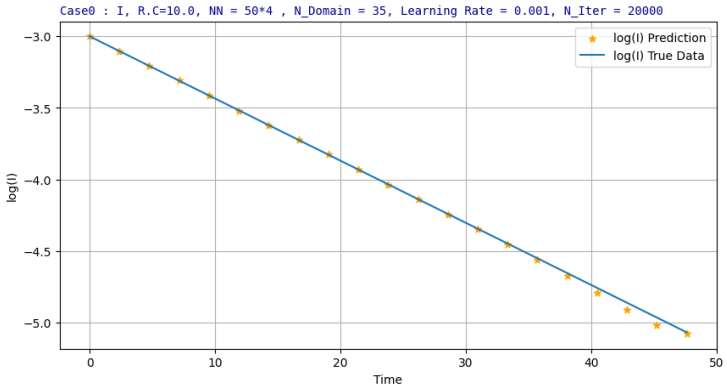


Figure 5.1: Case 0, Predicted Current in Forward Mode. 'True data' refers to the analytical solution.

Table 5.1: Circuit's Parameters

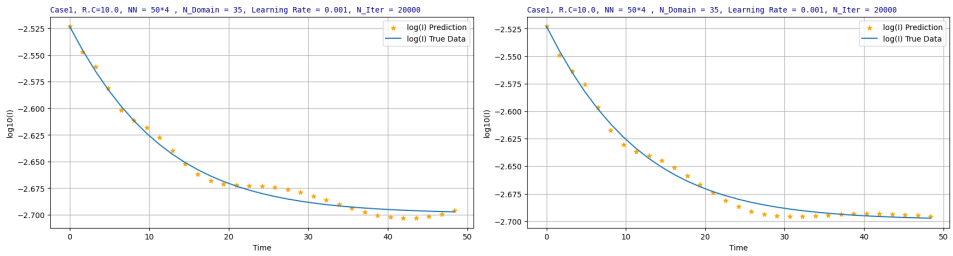
Parameter	**Value**
U_D	1.0
R_0	1000
R_1	500
C_1	0.02
R_2	1000
C_2	0.01
R_3	1000
C_3	0.01

5.2 Forward Mode: Results Analysis

5.2.1 Predicting Current (I)

In the initial phase, we employed DeepXDE's forward mode to predict solutions for the current differential equation across the range from Case0 to Case3 (as referenced in equations from the previous chapter). To enhance the resolution of diagrams, we opted to plot the base-10 logarithm of the current ($\log_{10}(I)$). This visualization approach aims to offer increased clarity, enabling a more nuanced evaluation of the predicted outcomes across various scenarios.

The model demonstrated accuracy and robustness for Case0 in predicting solutions within time domains under 40 seconds. However, as shown in Fig5.1, a distinct deviation became evident when the time domain extended beyond 40 seconds. This revealed a notable disparity between DeepXDE's predictions and the analytical solutions, bringing attention to limitations in the model's performance.



(a) (b)

Figure 5.2: Case 1: Predicted current in forward mode for two consecutive runs (a and b) of the same program, 'True data' refers to the analytical solution.

For cases other than Case0, the inconsistency in predicted current was pronounced. Figures 5.8a and 5.8b showcase the output for two consecutive runs of the same program for Case1, indicating that the predicted values for current lack both robustness and accuracy. This raises concerns about the reliability of the model across a broader range of scenarios.

Aiming to reduce sensitivity and enhance the overall robustness and precision of our findings, the subsequent section will focus on presenting the results of solving the differential equation for $\ln(I)$ and deploying DeepXDE specifically for predicting $\ln(I)$.

5.2.2 Predicting $\ln(I)$

In this step aimed at reducing the sensitivity of the output, we employ the Forward mode of DeepXDE to estimate ln(I) by solving the governing differential equation for each case. This process includes incorporating the initial conditions and analytical solutions for ln(I) while utilizing the constant parameters of the circuit outlined in Table 5.1. The subsequent section will provide a more detailed explanation of the optimized hyperparameters.

5.2.3 Hyperparameter Optimization for Enhanced Predictive Performance

In our quest to maximize DeepXDE's predictive capabilities, we explored and fine-tuned crucial hyperparameters. Our optimization efforts were dedicated to enhancing both accuracy and robustness, with the objective of achieving a configuration that not only improves predictive precision but also ensures the model's stability and reliability across various scenarios. The optimized hyperparameters are detailed in Table5.2.

- **Layer Size (40*3):** Through the optimized layer size of 40*3, determined through iterative experimentation, has demonstrated effectiveness in handling complex cases, including those investigated up to Case 6. This

choice of layer size proved crucial in preventing inaccurate predictions that were observed with smaller layer sizes.

- **Collocation Number (35):** Exploring different ranges of collocation points revealed that, for each 10 seconds, 35 collocation points achieved a balanced and accurate representation of the underlying system dynamics. The results for collocation numbers less than 35 lead to inaccurate results, and numbers exceeding 35 do not improve accuracy and convergence.

- **Learning Rate (0.01):** Extensive testing revealed that a learning rate of 0.01 struck the right balance for effective convergence during training while avoiding issues such as overshooting or slow convergence.

- **Number of Iterations (20,000):** Training the model for 20,000 iterations offered ample exposure to the training data, facilitating the learning of complex patterns and relationships within the differential equation. Considering fewer iterations, such as less than 20,000, leads to less accurate results and convergence issues for both Case 0 and Case 1. Notably, for Case 1, increasing the number of iterations to 40,000 led to a more significant deviation between the predicted and analytical solutions, indicating potential overfitting and convergence issues.

- **Optimizer (Adam):** The Adam optimizer, chosen for its default in Deepxde due to its adaptive learning rate capabilities, proved effective in optimizing convergence and mitigating issues associated with gradient descent.

- **Activation Function (tanh):** The hyperbolic tangent (tanh) activation function was identified as optimal for introducing non-linearity, enabling the model to capture intricate patterns within the differential equation.

- **Initializer (Glorot Uniform):** The Glorot uniform initializer was selected as the default in DeepXDE to set the initial weights of the network. This choice contributes to a well-balanced and stable initialization, enhancing the overall performance of the model.

  This optimization process involved manual tuning. Through iterative experimentation and careful consideration of the model's performance, these hyperparameters were identified as the most effective configuration for the given task.

Table 5.2: Forward Mode Optimized Hyper-parameters

Hyper-parameter	**Optimized**
Layer Size	40*3
Collocation Number	35/10(s)
Learning rate	0.01
Number of Iteration	20000
Optimizer	Adam
Activation Function	tanh
Initializer	Glorot uniform

The subsequent results and analyses reflect the success of this optimization approach, solidifying DeepXDE's reliability as a tool for precise and consistent $\ln(I)$ predictions across various scenarios. Figure 5.3 shows the predicted $\ln(I)$ for Case0 to Case3 in the 10-second time domain, considering the hyperparameters from Table 5.2

Table 5.3 illustrates the accuracy of DeepXDE predictions for various cases.

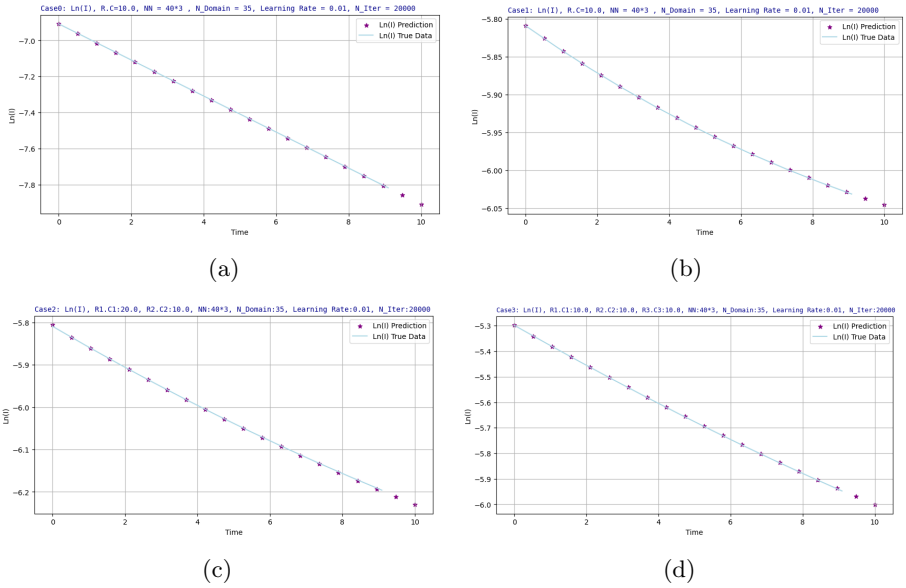


Figure 5.3: Forward Mode: The predicted $\ln(I)$ for Case0 (a), Case1 (b), Case2 (c), and Case3 (d). 'True data' refers to the analytical solution.

Table 5.3: Results for Different Cases in 10(s) Time Domain

(T=10(s))	Train Loss	Test Loss	Accuracy Metric	Train Time
Case0	1.56e-08	1.28e-08	9.40e-05	21.9 s
Case1	3.02e-09	1.65e-09	5.23e-06	22.8 s
Case2	3.22e-07	3.19e-07	1.38e-04	30.0 s
Case3	5.26e-07	5.08e-07	7.43e-04	31.6 s

An evaluation pertinent to the practicality of the physical model involves exploring different RC values, particularly in intricate cases where various branches in circuits have distinct time constants. Figure5.4 visually demonstrates the implications of selecting different RC values for Case 3. The predicted current is fitted to the analytical solution, referred to as "true data" in the figure, which is used as the test data in the Deepxde code. Consequently, there are no restrictions on choosing different RC values for different branches.

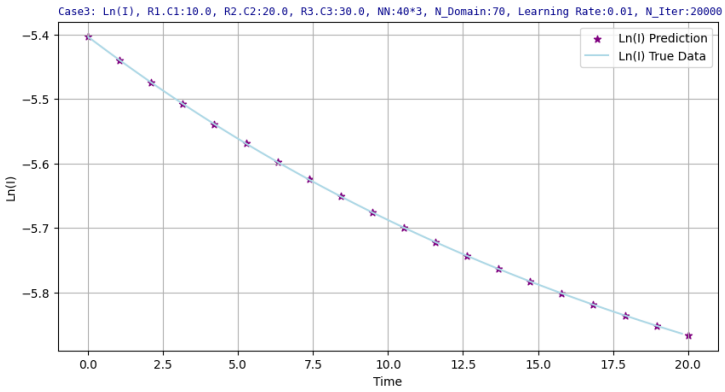


Figure 5.4: Case3: Impact of Varied Time Constants on DeepXDE Forward Mode Predictions. 'True data' refers to the analytical solution.

5.2.4 Limitations in Time Domain

To gain insights into the temporal limitations inherent in DeepXDE and its predictive capabilities, we conducted an examination of the accuracy and robustness of the predicted $\ln(I)$ across larger time domains—specifically, at intervals of 100 and 300 seconds. Thoroughly evaluating the model's performance across varying time scales, we aimed to assess its effectiveness and reliability in capturing temporal dynamics.

Table 5.4 presents the results for Case 2 in three different time domains. Achieving accurate and robust predictions when extending the time domain to 100 seconds requires an approximately tenfold increase in the

Table 5.4: Time Domain Limitations in DeepXDE's Forward Mode: Case 2 with RC = 10(s)

	collocation Number	**Train Loss**	**Test Loss**	**Accuracy Metric**	**Train Time**
T=10	35	3.22e-07	3.19e-07	1.38e-04	29.976110 s
T= 100	350	2.66e-07	1.38e-07	7.01e-04	44.765899 s
T=300	1050	1.48e-05	6.22e-07	1.30e+00	120.589814 s

Table 5.5: Time Constant Limitations in Deepxde's Forward Mode: Case2, T=10(s)

	collocation Number	**Train Loss**	**Test Loss**	**Accuracy Metric**	**Train Time**
R.C=10	35	1.86e-07	1.73e-07	3.44e-05	29.704234 s
R.C= 100	35	1.30e-08	1.02e-08	8.62e-05	29.455363 s
R.C=1000	35	1.08e-06	1.13e-06	1.08e-03	32.046105 s

number of points. However, for larger time domains like 300 seconds, accurate and consistent predictions prove challenging, even with an increased number of collocation.

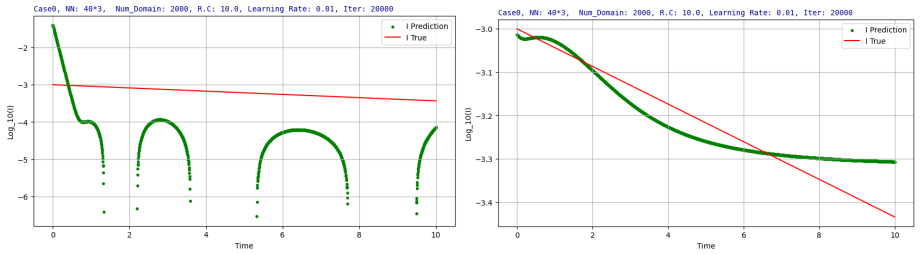
5.2.5 Limitations in Time Constant Range

Our investigation aims to uncover the boundaries of DeepXDE accuracy across various RC scales. The selected range provides valuable insights into the model's performance under distinct dielectric time constant regimes. Table 5.5 displays the prediction results of DeepXDE for Case 2 across three time constants (RC). The findings reveal that DeepXDE maintains accuracy and robustness for RC=10 and RC=100. However, as the scale increases to RC=1000, both accuracy and robustness diminish.

In summary, the performance of DeepXDE in Forward solution mode for predicting the current (I) is generally unsuccessful, except for the simplest case, Case0, which exhibits accurate and robust results. However, when employing the natural logarithm approach, DeepXDE, operating in Forward solution mode, demonstrates effective deployment across various scenarios. This success is particularly evident within a time domain ranging from 0 to 200, with time constant values spanning from 10 to 100. Moreover, It is important to note that the good performance is not limited to the same RC values, and various time constants can be selected.

5.3 Inverse Mode: Results Analysis

5.3.1 Results in inverse Mode for Current (I)



(a) (b)

Figure 5.5: Case0, Inverse Mode Analysis : (a) with , (b) without Initial Conditions.

During this phase, we utilized DeepXDE's inversion mode to predict circuit parameters R and C based on the anticipated current. To enhance diagram resolution, all plots were generated for $\log_{10}(I)$.

As illustrated in Fig. 5.5, the predictions for Case 0 within the 10-second time domain demonstrated a lack of accuracy and consistency, revealing evident challenges. Fig. 5.5(a) depicts the prediction considering initial conditions, while Fig. 5.5(b) represents the scenario without initial conditions. Removing initial conditions helped eliminate the discreteness in the current prediction, but it revealed clear inaccuracies and a lack of robustness.

To address these challenges and elevate the model's overall consistency and accuracy, we strategically rewrote the differential equation for Ln(I). Subsequently, DeepXDE's inversion mode was applied to predict R and C based on the reconstructed diagram for Ln(I). This adjustment is designed to mitigate sensitivity concerns and achieve reliable predictions, especially under complex conditions and for extended time domains.

5.3.2 Results in inverse Mode for Ln(I)

In our exploration of DeepXDE's inversion mode using the natural logarithm approach, we investigated two distinct strategies—one involving the consideration of initial conditions and the other without. For each strategy, we examined two scenarios: utilizing the analytical solution as the training set and employing measured data as the training set.

This analysis aimed to provide insights into the impact of different methodologies on the performance and accuracy of DeepXDE's inversion mode. By systematically evaluating these variations, we sought to understand

Table 5.6: Circuit's Parameters

Parameter	Value
U_D	1.0
R0	500
R1	1000
C1	0.01

how the choice of initial conditions and training data influences the model's ability to predict Ln(I) and, consequently, the circuit parameters R and C. Table 5.6 shows our considered values for Circuit's Parameters in all following results.

5.3.3 Hyperparameter Optimization for Enhanced Predictive Performance

Just as we have done in the forward mode, our optimization of hyperparameters involves an exploration focused on discovering the configuration that maximizes the model's performance. In our effort to enhance DeepXDE's predictive capabilities for the inverse discovery process, we intricately fine-tuned the following hyperparameters to achieve an optimized configuration:

- **Layer Size (40*3):** Through iterative experimentation, a layer size of 40*3 proved optimal, preventing inaccurate predictions that arose with smaller layer sizes, emphasizing the importance of an appropriately sized layer for the task's complexity.
- **Number of Domain (Case 0: 35, Case 1: 2000):** Exploring different ranges of Number of Domain revealed that, for each 10 seconds, 35 Domain points achieved a balanced and accurate representation of the underlying system dynamics for Case 0. In contrast, for Case 1, the optimal Number of Domain was determined to be 2000, providing the most accurate depiction of the system dynamics. The results for Number of Domain less than the optimized values lead to inaccurate outcomes, and numbers exceeding them do not improve accuracy and convergence.
- **Learning Rate (0.01):** Extensive testing revealed that a learning rate of 0.01 struck the right balance for effective convergence during training while avoiding issues such as overshooting or slow convergence.
- **Number of Iterations (20,000):** Training the model for 20,000 iterations offered ample exposure to the training data, facilitating the learning of complex patterns and relationships within the differential equation. When considering fewer iterations, such as less than 15,000, it leads to less accurate results and convergence issues.

Table 5.7: Inverse Mode Optimized Hyper-parameters

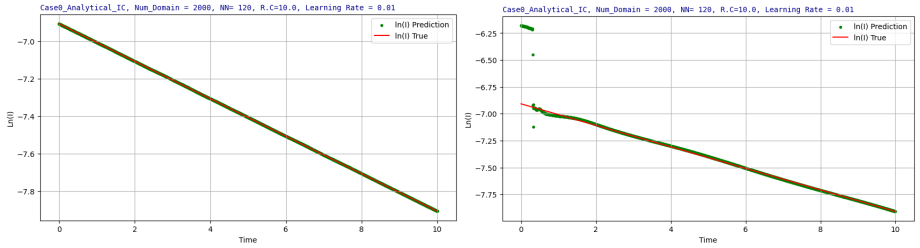
Hyper-parameter	**Optimized**
Layer Size	40*3
Number of Domain	Case0:35, Case1: 2000
Learning rate	0.01
Number of Iteration	20000
Optimizer	Adam, L-BFGS
Activation Function	tanh
Initializer	Glorot uniform

- **Initializer (Glorot Uniform):** The Glorot uniform initializer was selected as the default in DeepXDE to set the initial weights of the network. This choice contributes to a well-balanced and stable initialization, enhancing the overall performance of the model.
- **Optimizer (Adam, L-BFGS):** Employing both the Adam optimizer and the L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) optimizer.In sequence, starting with Adam that gives good starting parameters for L-BFGS. The use of L-BFGS consistently reduced the loss function by at least 10^{-2}, contributing to enhanced convergence and more accurate predictions.
- **Activation Function (tanh):** The hyperbolic tangent (tanh) activation function was identified as optimal for introducing non-linearity, enabling the model to capture intricate patterns within the differential equation.

The optimization process required manual tuning. By engaging in iterative experimentation and thoroughly evaluating the model's performance, we identified these hyperparameters as the most effective configuration for the specific task. The optimized hyperparameters are detailed in Table 5.7.

5.3.4 Considering Initial Condition: Utilizing the Analytical Solution as Training Data

In the case of the simplest scenario (Case 0),5.6a and 5.6b where we incorporated initial conditions and utilized the analytical solution as training data for two consecutive runs of the same program, the outcomes unveiled a deficiency in both robustness and accuracy. Notably, Figure 5.6b underscores challenges in accurately fitting the initial point of the current, signaling a lack of robustness. This observation prompted us to make a critical decision in our approach.



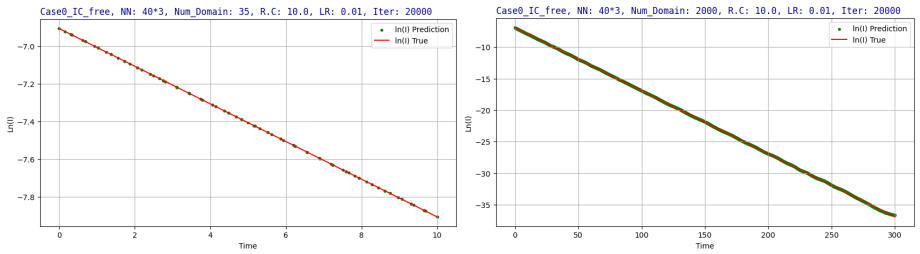
(a) (b)

Figure 5.6: Case 0: Logarithmic Approach, Inverse Mode Analysis with Initial Conditions and Synthetic Data,for two consecutive runs (a and b) of the same program, Ln(I) True: Analytical Solution.

Given the observed issues in accurately fitting the initial point of current, we have opted to remove the consideration of initial conditions from the model. This strategic decision is driven by the necessity for improved robustness and accuracy in the inversion mode output.

5.3.5 Removing Initial Condition: Utilizing the Analytical Solution as Training Data

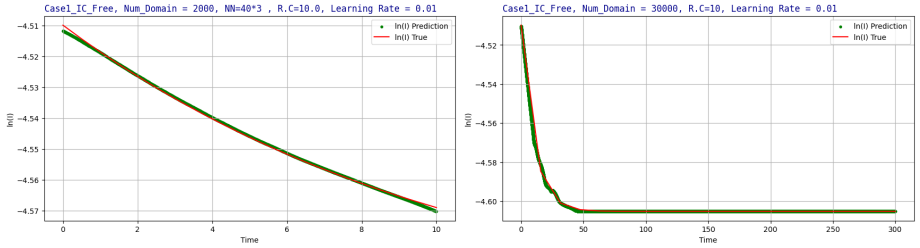
The removal of the initial condition significantly improved the accuracy and robustness of the model while eliminating issues associated with discrete points in the initial time domain. In the case of Case 0, illustrated in Figure 5.7a, for 10 seconds time domain we observe a precise and well-fitted prediction for the current, aligned with the analytical solution. Notably, there is no limitation in extending the time domain up to 300,Fig 5.7b.



(a) (b)

Figure 5.7: Case 0: Logarithmic Approach, Inverse Mode Analysis with Synthetic Data (No Initial Conditions), Ln(I) True: Analytical Solution.

For Case 1 for 10(s) time domain, as shown in Figure 5.8a, it is evident that in the beginning, there is some deviation between the predicted Ln(I) and the analytical solution, but it is well fit from 2(s) to the end of the time domain. We further investigated utilizing DeepXDE for a longer time domain of 300 (s), as depicted in Figure 5.8b, showing the best prediction with some deviation, acknowledging that the shown output is a result of optimized hyperparameters.



(a) (b)

Figure 5.8: Case 1: Logarithmic Approach, Inverse Mode Analysis with Synthetic Data (No Initial Conditions),Ln(I) True: Analytical Solution.

5.3.6 Post-Processing Analysis of Optimized Parameters

After completing the training of our DeepXDE model in inverse mode, utilizing the analytical solution as training data without considering initial conditions, a post-processing analysis was conducted to interpret the optimized parameters. This phase reveals insights into the model's behavior and its ability to capture the dynamics of electrical circuits.

The optimized values for the parameters R_r and RC were extracted from the model's output file. Furthermore, the model was utilized to predict the current at $t = 0$, opening up for additional analysis.

Parameter	**Training Assumption**	**DeepXDE Prediction**	**Error (%)**
$I(t = 0)$	0.001	0.0009999	0
R	1000	1000.06	0.006
C	0.01	0.009999	0

(a) Post-processing results for Case 0: $RC = 10(s)$

Hyper-parameter	**Optimized**
Layer Size	40*3
Collocation Number	35
Learning rate	0.01
Number of Iteration	20000
Optimizer	Adam, L-BFGS
Activation Function	tanh
Initializer	Glorot uniform

(b) Optimized Hyper-parameters

Figure 5.9: Post-processing Results for Case 0, $t = 10s$

For Case 0, as it is illustrated in Fig 5.9 the post-processing results are presented below, showcasing accurate predictions for the current at $t = 0$, calculated resistance (R), and the corresponding capacitance (C). The time domain is discretized with a duration of 10(s), and with RC set to 10(s), the associated error is minimal.

	Training Assumption	**DeepXDE Prediction**	**Error (%)**
$I(t=0)$	0.003	0.002	33.3
R	1000	1096.55	1.19
C	0.01	0.824	8140

(a) Post-processing results for Case 1: $RC = 10$, $R_r = 1000$.

Hyper-parameter	**Optimized**
Layer Size	40*3
Collocation Number	2000
Learning rate	0.01
Number of Iteration	20000
Optimizer	Adam, L-BFGS
Activation Function	tanh
Initializer	Glorot uniform

(b) Optimized Hyper-parameters

Figure 5.10: Post-processing Results for Case 1, $t = 10s$

For Case 1, the post-processing results are presented below, highlighting the predicted current at $t = 0$, calculated resistance (R), and the corresponding capacitance (C), with RC set to 10. As shown in Figure 5.11, the selected time domain for this analysis is 10 seconds. At $t = 0$, the predicted current (I) deviates from the analytical solution.

Figure 5.11(a) further highlights discrepancies in the predicted resistance (R) compared to the actual values, the associated noticeable error indicating a lack of accuracy in the prediction.

	Training Assumption	**DeepXDE Prediction**	**Error**
$I(t=0)$	0.003	0.003	0%
R	1000	1497.81	33.23%
C	0.01	0.26	90%

(a) Post-processing results for Case 1: $RC = 10$, $R_r = 500$.

Hyper-parameter	**Optimized**
Layer Size	40*3
Collocation Number	30000
Learning rate	0.01
Number of Iteration	20000
Optimizer	Adam, L-BFGS
Activation Function	tanh
Initializer	Glorot uniform

(b) Optimized Hyper-parameters

Figure 5.11: Post-processing Results for Case 1, $t = 300s$

Figure 5.11(a) presents the post-processing results for Case 1 in an extended time domain ($t = 300$(s)). Notably, a significant alignment is observed between the predicted and analytical solution currents across the majority of this prolonged time domain, as depicted in Fig 5.11((b). However, the error in predicted parameters, compared to the results obtained at $t = 10$(s), has increased to 20.54%.

In summary, the post-processing step has unveiled about the DeepXDE model's predictive accuracy in predicting R and C. Case 0's precision in predicting electrical circuit parameters contrasts with Case 1's noticeable errors, especially at $t = 300$(s). The observed discrepancies underscore the model's sensitivity to complicated scenarios and the challenges in maintaining accuracy over extended durations.

Chapter 6

Conclusion

6.1 Summary of Findings

The investigation began with the application of DeepXDE's forward mode to predict the current (I) across various circuit configurations, ranging from simple to more complicated scenarios. While the model demonstrated accuracy and robustness for Case 0 within certain time domains, deviations from analytical solutions became evident for more complex cases and extended time domains. Sensitivity to initial conditions and circuit parameters posed challenges, highlighting the need for innovative approaches to enhance predictive performance.

Transitioning to the inverse mode, our attention turned to predicting circuit parameters (R and C) based on anticipated current profiles. Despite initial challenges in accurately fitting initial conditions, strategic adjustments and hyperparameter optimization improved the model's accuracy and robustness, particularly when considering the natural logarithm approach. However, limitations persisted, especially in predicting parameters accurately for complex scenarios and extended time domains.

Further analysis revealed that the natural logarithm approach significantly improved the model's performance in both forward and inverse modes. By predicting the natural logarithm of the current ($\ln(I)$), sensitivity issues were mitigated, and the model's accuracy and robustness were enhanced across various scenarios.

Post-processing analyses provided valuable insights into the model's behavior and predictive accuracy, revealing discrepancies between predicted and actual parameters, particularly in intricate cases and prolonged time domains. However, the natural logarithm approach demonstrated notable improvements in addressing these discrepancies, underscoring its effectiveness in refining the precision and reliability of parameter predictions.

6.2 Future Directions

In conclusion, while this study represents a challenging step forward in the application of DeepXDE for dielectric modeling and electrical circuit analysis, further research and innovation are needed to unlock its full potential.

In conclusion, while this study represents a challenging step forward in the application of DeepXDE for dielectric modeling and electrical circuit analysis, further research and innovation are needed to unlock its full potential.

Sensitivity to initial conditions and circuit parameters highlighted the need for innovative approaches to enhance predictive performance.

Future investigations could focus on the following areas:

1. **Enhanced Modeling Techniques**

   Exploring optimized weighting for different terms of the loss function could significantly improve predictive accuracy and robustness. By fine-tuning the contributions of various components within the loss function, the model can better capture the nuances of complex circuit dynamics, leading to more accurate predictions.

2. **Exploration of Collocation Point Sampling Strategies**

   We also recommend exploring different collocation point sampling strategies. Experimenting with various sampling techniques, such as uniform sampling, random sampling, or adaptive sampling based on the solution dynamics, could help optimize the convergence and accuracy of the model. Additionally, investigating the influence of collocation point density and distribution across the domain could provide valuable insights into effectively capturing the underlying physics of the system.

3. **Initial Conditions and Circuit Parameters Sensitivity Exploration**
   Further investigation into the sensitivity of initial conditions and circuit parameters is warranted to enhance our understanding of their impact on the model's performance and predictive capabilities.

By addressing the identified limitations and exploring new avenues for advancement, we can pave the way for transformative developments in computational electromagnetics and engineering design.

Bibliography

[1] W.S. Zaenglaengl, "Dielectric spectroscopy in time and frequency domain for HV power equipment. I. Theoretical considerations", *IEEE Electrical Insulation Magazine*, Volume 19, Issue 5, Sept.-Oct. 2003.

[2] F. Kremer and A. Schönhals, *Broadband dielectric spectroscopy*, 2002.

[3] Nathan Baker, Frank Alexander, Timo Bremer, Aric Hagberg, Yannis Kevrekidis, Habib Najm, Manish Parashar, Abani Patra, James Sethian, Stefan Wild, Karen Willcox, and Steven Lee. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. Technical report, USDOE Office of Science (SC) (United States), feb 2019.

[4] Tony Hey, Keith Butler, Sam Jackson, and Jeyarajan Thiyagalingam. Machine learning and big scientific data. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 378(2166), mar 2020. ISSN 1364503X. doi: 10.1098/rsta.2019.0054.

[5] Salvatore Cuomo, Vincenzo Schiano di Cola, Fabio Giampaolo, Gianluigi Rozza, Maizar Raissi, and Francesco Piccialli. Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next. arXiv, jan 2022.

[6] M. Raissi et al., "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", *Journal of Computational Physics*, Volume 378, 2019, Pages 686-707.

[7] Blechschmidt,J.,Ernst,O.G.:Three ways to solve partial differential equations with neural networks– A review. GAMM-Mitteilungen 44(2), e202100,006 (2021).

[8] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis, "DeepXDE: A Deep Learning Library for Solving Differential Equations", SIAM ReviewVol. 63, Iss. 1 (2021)10.1137/19M1274067.

[9] Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.