Deep Learning Methods for Solving Some Partial Differential Equations

A Project Report Submitted

for the Course

MA498 Project I

by

Litesh Kumar

(Roll No. 210123037)

Upesh Jeengar

(Roll No. 210123067)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

November 2024

CERTIFICATE

This is to certify that the work contained in this project report entitled "Deep Learning Methods for Solving Some Partial Differential Equations" submitted by Litesh Kumar (Roll No.: 210123037) and Upesh Jeengar(Roll No.: 210123067) to the Department of Mathematics, Indian Institute of Technology Guwahati towards partial requirement of Bachelor of Technology in Mathematics and Computing has been carried out by him/her under my supervision.

It is also certified that this report is a survey work based on the references in the bibliography. It is expected to carry out some new works in the next phase of the project.

Guwahati - 781 039
November 2024

(Dr.Swaroop Nandan Bora)
Project Supervisor

Contents

Abstract

This document presents the implementation of deep learning techniques to solve multiple partial differential equations (PDEs) related to fluid dynamics, some other physical problems and the Effective Solute Removal (ESR) model using Physics-Informed Neural Networks (PINNs). We detail the theoretical background, initial and boundary conditions, neural network architectures, and the outcomes of the experiments. The ESR model addresses nutrient transfer and blood solute concentration, while other sections explore fundamental PDEs, including the wave and Euler beam equations.

Chapter 1

Machine Learning Methods

1.1 Introduction

This chapter provides a concise overview of various machine learning (ML) methods, focusing on their principles and common applications.

1.2 Supervised Learning

Supervised learning involves training a model on labeled data, where each input has a corresponding output label. The main goal is for the model to learn a mapping from inputs to outputs, enabling it to make predictions on unseen data. Common techniques in supervised learning include

- **Linear Regression**: A regression technique used to model the relationship between a dependent variable and one or more independent variables.
- **Logistic Regression**: Often used for binary classification tasks, logistic regression predicts the probability of a binary outcome.

- **Support Vector Machines (SVM)**: SVMs aim to find a hyperplane that best separates data points of different classes.
- **Decision Trees and Random Forests**: Decision trees split data based on feature values to make decisions, and random forests aggregate multiple trees to improve prediction accuracy.

1.3 Unsupervised Learning

In unsupervised learning, the model is trained on unlabeled data, with the goal of identifying patterns or structures within the data. Key methods include

- **Clustering (e.g., K-Means)**: Used to partition data into distinct groups based on similarity.
- **Principal Component Analysis (PCA)**: A dimensionality reduction technique that identifies the main components explaining data variance.
- **Anomaly Detection**: A technique used to identify rare or unusual data points within a dataset.

1.4 Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning where an agent interacts with an environment, learning a policy to maximize cumulative rewards over time. Popular RL algorithms include

- **Q-Learning**: A value-based algorithm that seeks to find the optimal action-selection policy.

- **Deep Q-Networks (DQN)**: A variant of Q-learning that utilizes neural networks to approximate Q-values.

1.5 Deep Learning

Deep learning uses neural networks with multiple layers to model complex patterns in data. Key architectures include

- **Convolutional Neural Networks (CNNs)**: Primarily used in image processing, CNNs extract spatial features through convolution operations.
- **Recurrent Neural Networks (RNNs)**: Ideal for sequence data, such as time series or text, RNNs have memory connections that capture temporal dependencies.

1.6 Conclusion

These methods represent core machine learning techniques used across a variety of fields. Each method has unique characteristics, making them suitable for different types of tasks, such as regression, classification, pattern recognition, and decision-making.

Chapter 2

Introduction to Deep Learning in PDEs

2.1 Introduction to Neural Networks

A neural network is a computational model inspired by the structure of the brain [1]. It is composed of **layers** of interconnected **neurons** (or nodes) where each neuron in one layer connects to every neuron in the next layer. The structure of a basic neural network can be summarized as:

$$\text{Input Layer} \rightarrow \text{Hidden Layers} \rightarrow \text{Output Layer}$$

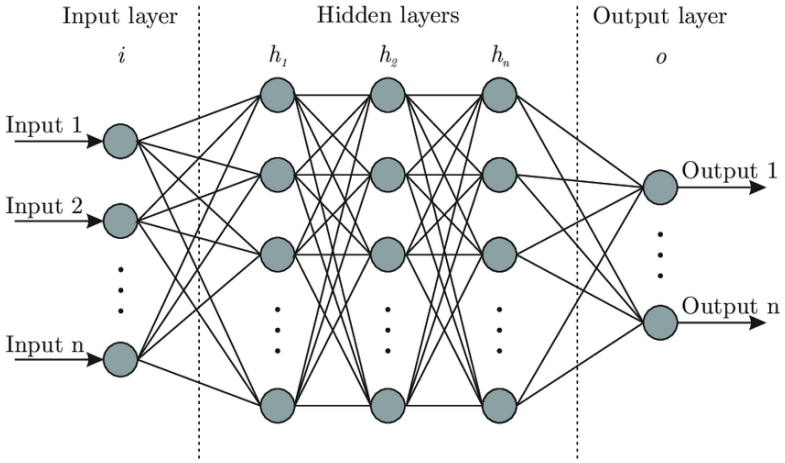

Figure 2.1: ANN Architecture

Each layer has weights and biases, which are adjusted during training to minimize error. Neural networks learn to approximate functions through optimization.

2.2 Forward Propagation

Forward propagation refers to the process of calculating the output of the network by passing input data through the network layer by layer. In each layer, the output of neuron i is computed as

$$z_i^{(l)} = \sum_{j=1}^{n_{l-1}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)},$$

where

- $z_i^{(l)}$ is the pre-activation value for neuron i in layer l,
- $w_{ij}^{(l)}$ is the weight connecting neuron j in layer $l-1$ to neuron i in layer l,

- $a_j^{(l-1)}$ is the output (activation) from the previous layer,
- $b_i^{(l)}$ is the bias for neuron i in layer l.

The activation $a_i^{(l)}$ is then calculated by applying an **activation function** f to $z_i^{(l)}$:

$$a_i^{(l)} = f(z_i^{(l)}),$$

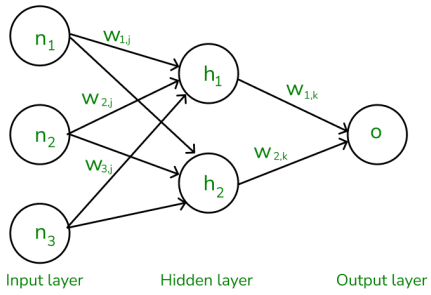

Figure 2.2: Forward Propagation

2.2.1 Activation Functions

Activation functions introduce non-linearity, allowing neural networks to approximate complex functions. Common activation functions include

- **Sigmoid**: $f(z) = \frac{1}{1+e^{-z}}$,
- **ReLU (Rectified Linear Unit)**: $f(z) = \max(0, z)$,
- **Tanh**: $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

Each function has specific properties that influence training. For example, ReLU helps avoid the vanishing gradient problem by keeping gradients larger for positive inputs.

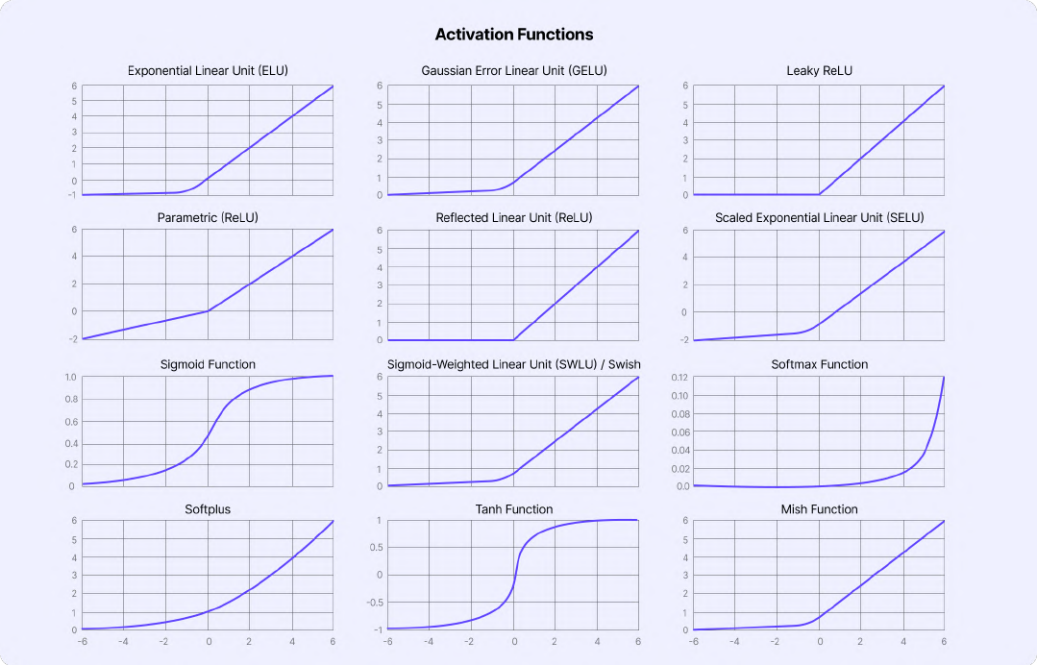

Figure 2.3: Different Activation Functions

2.3 Loss Functions

The loss function quantifies how well the model's predictions match the true labels. Common loss functions include

- **Mean Squared Error (MSE)**: Often used in regression tasks, it measures the average of squared differences between predicted and true values:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

- **Cross-Entropy Loss**: Used in classification tasks, it measures the error between predicted probabilities and actual labels:

$$\text{Cross-Entropy} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i).$$

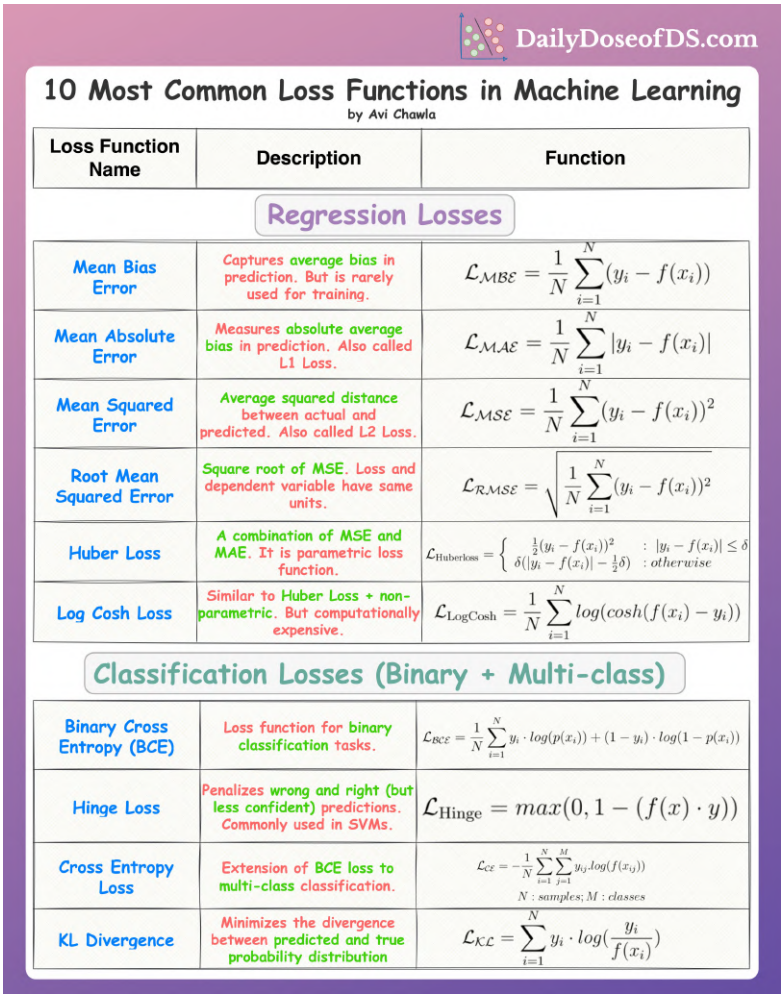

Figure 2.4: Loss Functions

The choice of the loss function depends on the task and influences how the network updates weights.

2.4 Backpropagation

Backpropagation is the process of calculating the gradient of the loss function with respect to each weight by applying the **chain rule**. This helps in updating the weights to minimize the loss.

The gradient of the loss with respect to a weight $w_{ij}^{(l)}$ in layer l can be computed as

$$\frac{\partial \text{Loss}}{\partial w_{ij}^{(l)}} = \frac{\partial \text{Loss}}{\partial a_i^{(l)}} \cdot \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}$$

where:

- $\frac{\partial \text{Loss}}{\partial a_i^{(l)}}$ is the derivative of the loss with respect to the activation,
- $\frac{\partial a_i^{(l)}}{\partial z_i^{(l)}}$ is the derivative of the activation function,
- $\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} = a_j^{(l-1)}$.

2.5 Optimization Algorithms

Optimizers update the weights to minimize the loss. Common optimizers include

- **Gradient Descent (GD)**: Updates weights by moving in the direction of the negative gradient:

  $$w \leftarrow w - \eta \frac{\partial \text{Loss}}{\partial w}$$

  where η is the learning rate.

- **Stochastic Gradient Descent (SGD)**: Computes gradients for each batch, which is computationally cheaper than full GD.

- **Adam (Adaptive Moment Estimation)**: Combines momentum and adaptive learning rates.

  $$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla \text{Loss},$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla \text{Loss})^2,$$

$$w \leftarrow w - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t.$$

where m_t and v_t are moving averages of the gradients and their squares, respectively.

2.6 Summary

This chapter outlines the key components of a neural network, including forward propagation, backpropagation, loss functions, activation functions, and optimizers. Deep learning methods are built on these foundational principles, enabling neural networks to solve complex tasks by approximating functions through optimization techniques.

Chapter 3

PINN Methods

3.1 Introduction

A Physics-Informed Neural Network (PINN) is a type of deep learning model that incorporates the principles of physics into the training of neural networks [2], providing a powerful tool for solving complex partial differential equations (PDEs) and other physics-based problems. Unlike traditional neural networks, which learn patterns from purely empirical data, PINNs are designed to "learn" solutions to problems governed by physical laws, making them especially useful in scientific and engineering applications. This is achieved by embedding the governing equations (often PDEs) directly into the loss function of the neural network. During training, the network minimizes the residuals of these equations across the domain of interest, ensuring that the solutions it produces satisfy the physics constraints (such as conservation laws) as closely as possible.

PINNs combine data-driven learning with domain knowledge, which allows them to generalize well even with sparse or noisy data, as the embedded physics serves as an additional source of "information" guiding the learn-

ing process. PINNs have shown great potential in fields like fluid dynamics, structural analysis, electromagnetism, and biophysics, where complex PDEs are often computationally expensive to solve with traditional methods. By leveraging automatic differentiation (a technique to calculate derivatives used in neural networks), PINNs can handle complex derivatives that appear in PDEs efficiently, without requiring discretization as in finite difference or finite element methods. Consequently, PINNs offer a promising alternative for solving high-dimensional, non-linear PDEs, providing a more computationally efficient and flexible approach to simulate real-world physics phenomena.

3.2 Solving PDEs Using PINN

3.2.1 Network Architecture

A PINN uses a fully connected neural network as a function approximator. The network takes spatial and temporal coordinates as inputs (e.g., x and t for a 1D space-time problem) and outputs the approximate solution $u(x, t)$ at each point.

The architecture typically consists of

- **Input Layer**: Represents spatial and temporal coordinates.
- **Hidden Layers**: Fully connected layers with non-linear activations (e.g., tanh) to capture complex solution behavior.
- **Output Layer**: Produces the approximate solution $u(x, t)$ to the PDE.

3.2.2 Formulating the Loss Function

The unique feature of PINNs lies in their custom loss function, which incorporates multiple terms to enforce the governing equations, boundary conditions, and initial conditions. This loss function is composed of

- **PDE Residual Loss**:
  - For a general PDE of the form $\mathcal{N}[u] = 0$, where $\mathcal{N}$ is a differential operator, the **PDE residual** is defined as

$$f = \mathcal{N}[u].$$

    For instance, for a wave equation $u_{tt} - c^2 u_{xx} = 0$, the residual becomes:

$$f = u_{tt} - c^2 u_{xx}.$$

    Ideally, f should be zero across the domain, ensuring that the network's solution satisfies the PDE. This residual is computed as the mean squared error of f, penalizing deviations from zero.

- **Boundary Condition Loss**
  - The **boundary loss** ensures that the network solution $u(x, t)$ adheres to boundary conditions at the edges of the spatial domain. For example, a Dirichlet boundary condition specifies fixed values for $u(x, t)$ at the domain boundaries.

- **Initial Condition Loss**
  - The **initial condition loss** ensures that the network's solution aligns with the initial conditions at the start of the time domain.

For example, if $u(x, 0) = \sin(x)$, this loss penalizes deviations between the network's output and $\sin(x)$ at $t = 0$.

The total loss function, which the network optimizes, is then given by

$$\text{Total Loss} = \text{PDE Loss} + \text{Boundary Loss} + \text{Initial Condition Loss}.$$

3.2.3 Automatic Differentiation for Derivatives

PINNs leverage automatic differentiation to compute the required derivatives for the PDE residual. Using automatic differentiation, we can compute gradients of any order with respect to the inputs of the neural network, such as u_x, u_{xx}, u_t, and u_{tt}. This capability allows for accurate and efficient calculation of the PDE terms, enabling a precise enforcement of the governing equations.

3.2.4 Optimization of the Network

The network's parameters are optimized by minimizing the total loss function, which combines the PDE residual, boundary conditions, and initial conditions. This optimization is generally performed using standard optimizers such as **Adam** [3] or **L-BFGS**, which iteratively adjust the weights to minimize the total loss:

$$\min_{\text{parameters}} \text{Total Loss}.$$

During training, the network learns a continuous approximation $u(x, t)$ that is consistent with both the physical laws and the specified initial/boundary conditions. By balancing these constraints, the trained PINN provides a high-quality approximation to the solution of the PDE.

3.2.5 Evaluating the Solution

Once trained, the PINN provides a continuous approximation of the solution across the domain. This continuous solution allows for efficient evaluation at any point in the spatial or temporal domain, making PINNs highly flexible and advantageous over traditional numerical methods, especially for:

- **High-dimensional PDEs**: Traditional methods face limitations due to the curse of dimensionality, but PINNs handle high dimensions without a significant increase in computational cost.
- **Complex Geometries**: PINNs can accommodate irregular domains and boundary conditions more easily than grid-based methods.

3.2.6 Benefits and Applications of PINNs

PINNs have several advantages over traditional numerical methods:

- **Flexibility**: PINNs can handle complex domains, irregular boundaries, and high-dimensional spaces without the need for domain discretization.
- **Generalization**: Once trained, a PINN provides a continuous solution that can be evaluated at any point in the domain.
- **Applications**: PINNs are applied in fields like fluid dynamics, electromagnetics, quantum mechanics, finance, and biomechanics, where they approximate complex physical phenomena governed by PDEs.

3.2.7 Conclusion

In summary, PINNs use a neural network to approximate the solution of PDEs by incorporating the governing equations, initial conditions, and bound-

ary conditions into the network's loss function. By optimizing this custom loss function, PINNs generate solutions that satisfy the physical laws, making them a powerful and flexible tool for solving PDEs in complex and high-dimensional settings.

Chapter 4

Solving Some Partial Differential Equations

As an initial step, we utilize deep learning in discussing some familiar PDEs.

4.1 Wave Equation

The wave equation is fundamental in fields like acoustics, electromagnetism, and fluid dynamics.

Problem Definition The wave equation is given by

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 < x < L, \quad t > 0.$$

where $c = 1$ represents the wave speed. Initial conditions are:

$$u(x, 0) = \sin(x), \quad u_t(x, 0) = \cos(x).$$

Boundary conditions are taken as

$$u(0,t) = 0 = u(L,t).$$

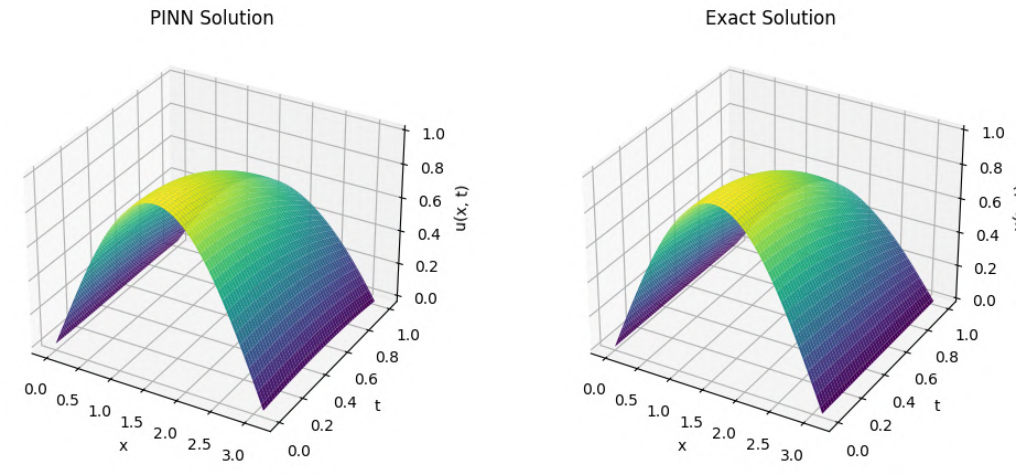

Figure 4.1: Wave Equation

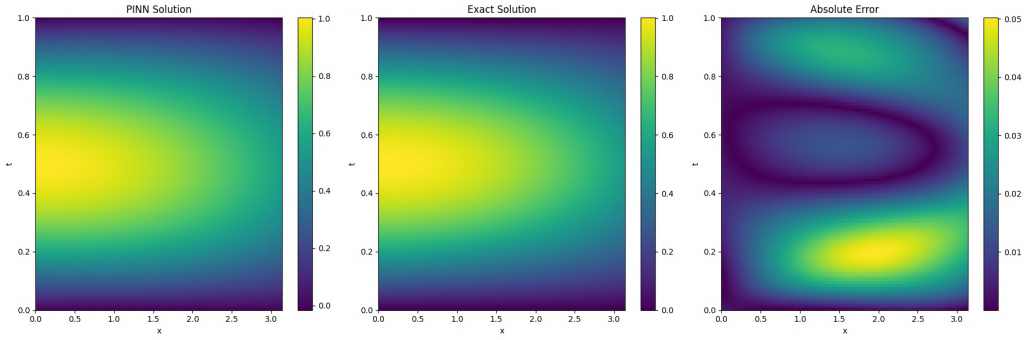

Figure 4.2: Heat Map Comparison of PINN solution with exact solution

4.2 Wave Equation with External Disturbances

The modified wave equation with an external force $F(x,t)$ is

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} + F(x,t).$$

The same initial and boundary conditions as in the basic wave equation are applied.

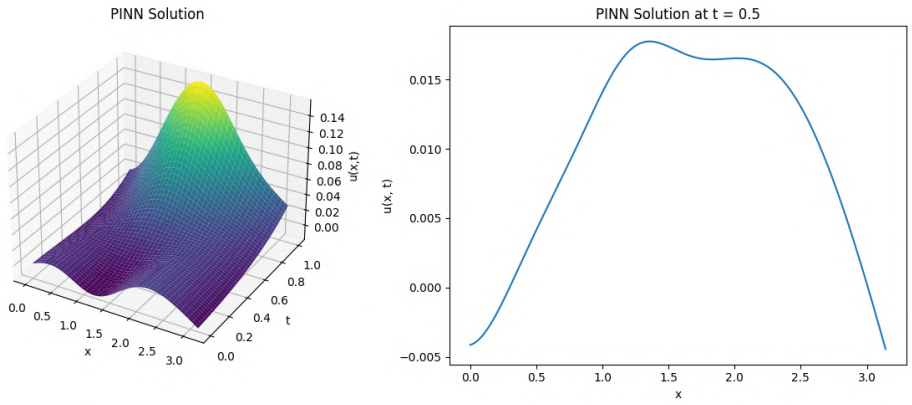

Figure 4.3: Wave Equation with external disturbances

4.3 Euler Beam Equation

The Euler Beam equation models the bending of beams under load in structural engineering.

Problem Definition: The Euler Beam equation is

$$\frac{\partial^2 u}{\partial t^2} + \frac{\partial^4 u}{\partial x^4} = 0, \quad 0 < x < L, \quad t > 0.$$

Boundary conditions for a beam pinned at both ends are:

$$u(0,t) = u(L,t) = 0, \quad \frac{\partial^2 u}{\partial x^2}(0,t) = \frac{\partial^2 u}{\partial x^2}(L,t) = 0.$$

Initial conditions simulate the beam being released from a deflected position.

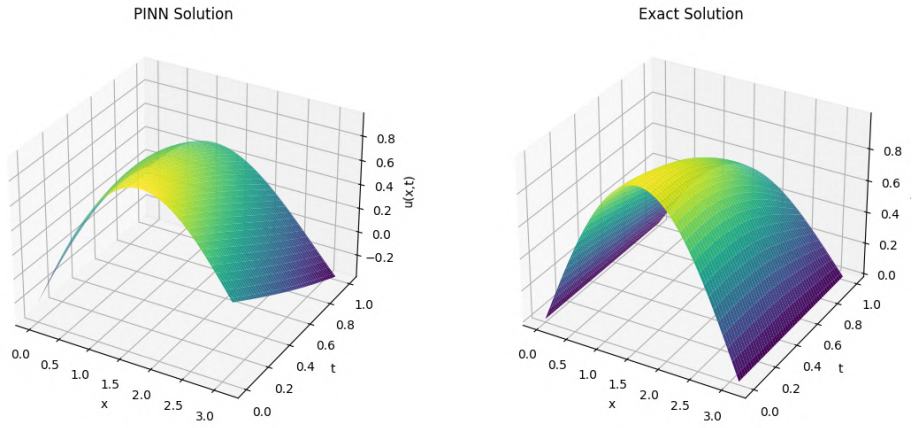

Figure 4.4: Euler Beam Equation

4.4 ESR Model

The ESR model [4] is used to represent solute concentration in blood, considering nutrient transfer rate and fluid dynamics.

Mathematical Model: The governing equation of the ESR model is

$$A\frac{\partial^2 C}{\partial x^2} - U\frac{\partial C}{\partial x} - \frac{\partial C}{\partial t} = \Phi(x,t),$$

where

- $C(x,t)$ is the blood concentration.
- A is the coefficient of axial dispersion.
- U is the average fluid velocity, set to zero in our simulations.
- $\Phi(x,t)$ represents the rate of nutrient transfer, modeled by

$$B\frac{\partial^2 \Phi}{\partial x^2} - k\Phi(x,t) - \frac{\partial \Phi}{\partial t} = 0.$$

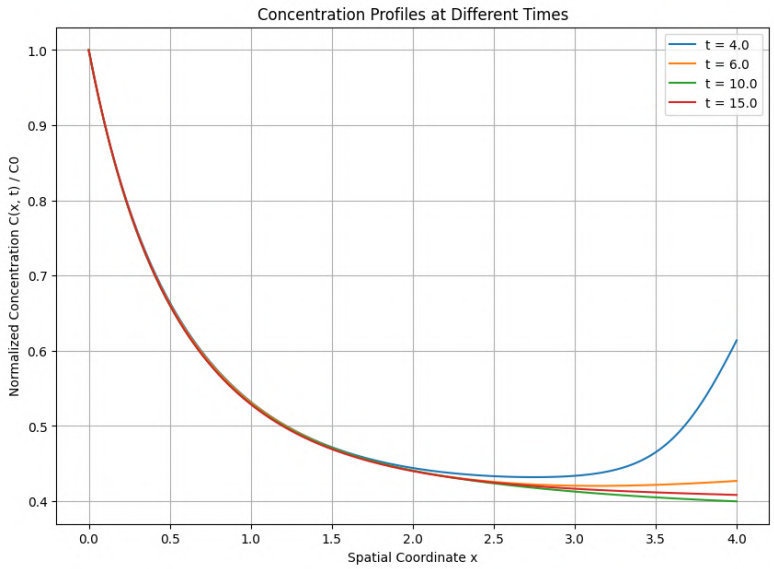

Figure 4.5: Fractional ESR Model

Boundary conditions for $\Phi(x,t)$ are

$$\Phi(x,0) = e^{-bx}, \quad \Phi(0,t) = e^{-at}, \quad \lim_{x\to\infty} \Phi(x,t) = 0.$$

Initial and boundary conditions for $C(x,t)$ include

$$C(x,0) = 0, \quad C(0,t) = 1, \quad \lim_{x\to\infty} C(x,t) = 0.$$

4.5 Neural Network Architecture and Implementation

The neural networks designed to solve these PDEs consisting of input layers, residual blocks with Swish activation functions, and output layers. The ESR model and other PDEs were implemented in PyTorch and TensorFlow,

using custom classes, Sobol sequences for domain coverage, and automatic differentiation for derivatives. Loss functions are constructed to minimize residuals in the PDE, boundary, and initial conditions.

4.6 Conclusion

The PINNs effectively simulate the wave equation, Euler beam equation and ESR model demonstrating a robust approach to solving complex PDEs in blood flow dynamics, fluid dynamics, and structural mechanics. Future work could extend these models to more complex conditions and PDE forms.

Chapter 5

Conclusion and future direction

In the Phase-I of our project, we the examine the possibility of initializing deep learning method for solving some problems depicting physical phenomena, we have introduced various methods concerning deep learning. We have considered some familiar partial differential equations such as the wave equation, Euler beam equation and ESR model for experiment.

In our next phase of project we wish to consider some other physical phenomena such as water wave propagation, flow discharge in open channels and blood flow in arteries, and apply the appropriate deep learning methods. Time permitting, we may also like to consider some important PDEs in mathematical finance.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] A. Shit and S. N. Bora, "Esr fractional model with non-zero uniform average blood velocity," *Computational and Applied Mathematics*, vol. 41, p. 354, 2022.