DEPARTMENT OF MATHEMATICS

Physics-Informed Neural Networks: A Deep Learning Framework for Solving Partial Differential Equations

Ali Uğur Kaymaz

20025021

Undergraduate Thesis

Advisor: Prof. Dr. Özgür YILDIRIM

May 2025

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. Dr. Özgür Yıldırım, for his valuable support and guidance throughout my thesis. His advice and encouragement helped me complete this work. I also want to thank all the professors and staff at the Mathematics Department of Yıldız Technical University for their help and for providing a good learning environment. Thanks to my friends who supported me and shared ideas during this process. Their support made the work easier and more enjoyable. Finally, I am very grateful to my family for their patience, love, and support. Without them, I could not have completed this thesis. This work would not have been possible without all of you. Thank you very much.

Contents

List Of Symbols

$\cdot$	Dot product or scalar multiplication
$\delta^{(l)}$	Error term at layer l in backpropagation
η	Learning rate
$\frac{\partial}{\partial x}$	Partial derivative with respect to x
$\hat{y}$	Predicted output
$\mathscr{L}$	Loss function
∇_θ	Gradient with respect to parameters θ
Ω	Spatial domain
$\sum$	Summation symbol
θ	Model parameters
$a^{(l)}$	Activation output of layer l
$f(x_i;\theta)$	Model prediction for input x_i
$L(y,\hat{y})$	Loss between prediction $\hat{y}$ and true label y
n	Number of training samples
$u(x,t)$	Unknown function of space and time
$W^{(l)}$	Weights of layer l in a neural network
x_i	Input feature vector for the i-th sample
y_i	True label for the i-th sample
$z^{(l)}$	Pre-activation output of layer l

List of Figures

ABSTRACT

This thesis focuses on the modeling of physical processes described by partial differential equations (PDEs) using Physics-Informed Neural Networks (PINNs). PINNs are hybrid models that incorporate both data-driven learning and prior physical knowledge into a unified neural network framework. This allows them to reduce common challenges of classical numerical methods such as mesh dependency, sensitivity to boundary conditions, and computational cost.

The study explains the fundamental concepts of PINNs and applies them to a representative problem, namely heat transfer. A custom loss function is constructed to enforce both data fidelity and adherence to the governing PDEs. The results demonstrate the theoretical and practical capabilities of PINNs as a powerful alternative to traditional approaches.

In this regard, PINNs offer a promising modeling tool for future studies in disciplines such as engineering, physics, and computational sciences.

Chapter 1

Introduction

1.1 Problem Definition

Partial Differential Equations (PDEs) have long served as fundamental tools for modeling complex and dynamic systems across various scientific disciplines for nearly 300 years. In recent times, systems encountered in fields such as heat transfer, fluid mechanics, and quantum mechanics often involve complex geometries and are defined in high-dimensional spaces. As a result, analytical solutions to such PDEs are rarely obtainable. This has led to a growing reliance on numerical methods for solving these equations.

Although numerical methods can provide accurate results, they often suffer from high computational costs, especially in high-dimensional solution spaces. Furthermore, complex boundary or initial conditions can make the implementation of these methods more difficult. Additionally, when external system conditions change, the need to recompute the solution limits the flexibility of traditional approaches.

In contrast, Physics-Informed Neural Networks (PINNs) have emerged in recent years as an alternative approach, driven by advances in machine learning and deep learning. PINNs integrate deep neural networks with physical laws, allowing the model to leverage both observational data and the underlying physics of the system. This dual focus helps the model converge toward physically consistent solutions. Compared to traditional methods, PINNs often demonstrate superior performance in solving parameterized PDEs. Moreover, the use of automatic differentiation enables the precise computation of complex derivative terms, significantly enhancing the model's accuracy.

In this thesis, the applicability of the PINN approach for solving a complex PDE problem is investigated. The theoretical and mathematical foundations of PINNs are explored in detail, and the method is evaluated through comparisons on a representative example problem.

1.2 Scope and Structure of the Thesis

The structure of the thesis is organized as follows:

- **Chapter 2 – Literature Review:** In this chapter, we provide a comprehensive overview of Artificial Neural Networks, key concepts in machine learning (such as why machines learn and how activation functions work), and the challenges of conventional deep learning when applied to PDEs. We also introduce the fundamentals of PDEs, their classifications, and common numerical solution techniques.
- **Chapter 3 – Physics-Informed Neural Networks (PINNs):** This chapter delves into the theoretical and mathematical underpinnings of the PINN framework. It describes the model architecture, mathematical background, loss functions, and optimization strategies used in training PINNs.
- **Chapter 4 – Case Study : Kuramoto-Sivashinsky Equation:** In this chapter, the Kuramoto–Sivashinsky (KS) equation is investigated as a case study to demonstrate the practical application of the Physics-Informed Neural Network (PINN) approach. The KS equation, known for its chaotic and nonlinear behavior, presents a challenging test case for numerical methods. The implementation details of the PINN model are provided, and the model's performance is evaluated through a comparison of the predicted and reference solutions. Numerical results, training dynamics, and error analysis are discussed to assess the accuracy and effectiveness of the proposed approach.
- **Chapter 5 – Future Work And Discussion :** The final chapter discusses the outcomes of the study, highlights the strengths and limitations of the approach, and outlines possible directions for future research and improvements.

Chapter 2

Literature Review

2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the biological brain. Although this analogy is loose—biological neurons are far more complex—ANNs rely on a conceptual architecture composed of interconnected units (neurons) that process information collectively. Each artificial neuron receives input signals, applies a transformation (typically a weighted sum followed by a nonlinear activation function), and transmits the result to the next layer.

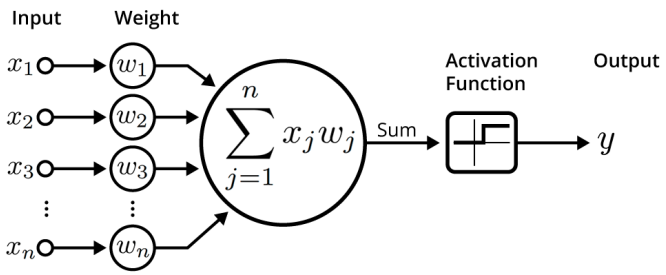


Figure 2.1: Single Neuron Architecture

These neurons are organized into layers: an input layer that receives raw data, one or more hidden layers that extract and transform features, and an output layer that produces predictions. The strength of the connections between neurons is encoded as trainable parameters called weights.

Deep learning refers to the use of multi-layer neural networks that perform hierarchical feature learning. Deep networks can learn increasingly complex patterns, from low-level features (e.g., edges, textures) to high-level abstract representations (e.g., objects, concepts). This hierarchical structure has led to groundbreaking successes in areas such as

image processing, natural language processing, and speech recognition.

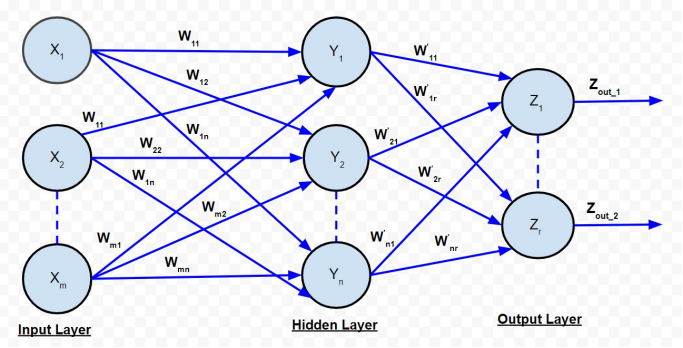


Figure 2.2: Deep Neural Network Architecture

The primary goal of Artificial Neural Networks (ANNs) is to learn meaningful patterns from input data and generalize this knowledge to new, unseen examples. This is achieved by optimizing the weights to minimize a predefined loss function that measures the discrepancy between predicted and actual outputs. During training, learning algorithms such as stochastic gradient descent iteratively adjust the weights to reduce this error, enabling the network to "learn" from experience.

Despite their remarkable achievements in tasks such as image classification, speech recognition, and natural language processing, traditional ANNs typically require large labeled datasets and lack mechanisms to incorporate domain-specific knowledge or physical constraints. These limitations have motivated the development of hybrid models such as Physics-Informed Neural Networks (PINNs) [4] , which aim to bridge the gap between data-driven learning and physical modeling.

In a feedforward neural network, data flows through layers by first undergoing a linear transformation followed by a non-linear activation. For layer l, this process is expressed as

$$\mathbf{z}^{(l+1)} = \mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)}, \quad \mathbf{a}^{(l+1)} = g(\mathbf{z}^{(l+1)}), \tag{2.1}$$

where $\mathbf{a}^{(l)}$ is the activation from the previous layer, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases, respectively, and g is a non-linear activation function such as sigmoid or ReLU. This process is repeated layer by layer, allowing the network to learn hierarchical representations.

2.1.1 Why Do Machines Learn?

Machine learning, and in particular deep learning, aims to model the relationships between inputs and outputs not through explicit rules, but by learning from data. But how

and why do machines actually "learn"?

Deep learning models are typically parameterized functions designed to map input data to desired outputs. These functions contain adjustable parameters, such as weights and biases. To measure how far off a model's prediction is from the true label, a loss function is defined:

$$\mathscr{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i;\theta)) \tag{2.2}$$

where θ represents the model parameters, which in a neural network correspond to the collection of all weight matrices $\mathbf{W} = \{\mathbf{W}^{(l)}\}$ and bias vectors $\mathbf{b} = \{\mathbf{b}^{(l)}\}$ across all layers l. Here, y_i is the model prediction, and L is a distance or divergence metric between prediction and truth. Commonly used loss functions include:

- **Mean Squared Error (MSE):** $L(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
- **Mean Absolute Error (MAE):** $L(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$
- **Binary Cross-Entropy:**

$$L(y_i, \hat{y}_i) = -\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- **Categorical Cross-Entropy:**

$$L(y_i, \hat{y}_i) = -\sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

Learning occurs by minimizing this loss function. This minimization is achieved through an optimization algorithm called **gradient descent** [7]. Gradient descent computes the derivative (gradient) of the loss function with respect to the model parameters and updates them in the direction that reduces the loss.

In deep neural networks, computing these gradients efficiently is accomplished through the **backpropagation algorithm**. Backpropagation applies the chain rule of calculus to systematically compute gradients layer by layer, starting from the output and propagating backward through the network. For a network with L layers, the algorithm first computes the error at the output layer, then uses this error to calculate gradients for the preceding layers through the recursive relationship:

$$\frac{\partial \mathscr{L}}{\partial W^{(l)}} = \frac{\partial \mathscr{L}}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}$$

where $W^{(l)}$ represents the weights of layer l, and $z^{(l)}$ is the pre-activation output of that layer. This backward pass allows the network to determine how much each parameter contributed to the final error, enabling precise parameter updates.

These updates are made iteratively using the computed gradients:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathscr{L}(\theta_t) \tag{2.3}$$

where η is the learning rate and $\nabla_\theta \mathscr{L}$ is the gradient of the loss function with respect to parameters θ. Over time, through this iterative process of forward propagation (computing predictions) and backpropagation (computing gradients), the model becomes better at making accurate predictions.

This process allows a model that initially behaves randomly to eventually capture meaningful patterns in the data, achieving better generalization and performance on unseen examples. In essence, machines "learn" by following a mathematical procedure that continuously improves their internal parameters to reduce error, allowing them to extract knowledge from data.

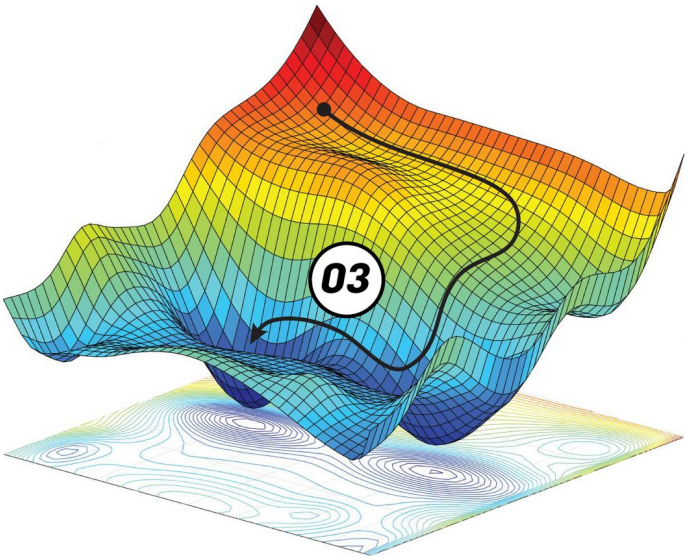


Figure 2.3: Gradient Decent Illustration In 3-dim Space

2.1.2 Activation Functions

Activation functions are critical components that enable neural networks to perform non-linear transformations. Without them, a multi-layer network would merely be a composition of linear transformations and would lose the ability to learn complex patterns.

- **Sigmoid Function**: $\sigma(x) = \frac{1}{1+e^{-x}}$ maps outputs to the range $[0,1]$, allowing probabilistic interpretation. However, it suffers from the vanishing gradient problem and

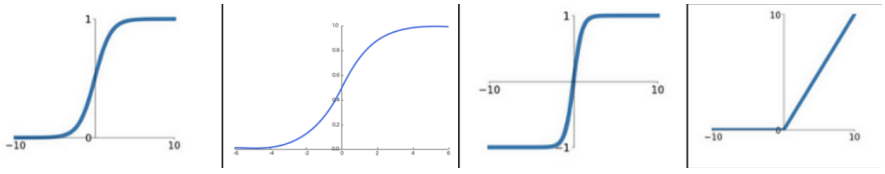
is computationally expensive, hence rarely used in modern architectures.

- **Softmax Function**: The softmax activation function is commonly used in the output layer of neural networks for multi-class classification tasks. It converts a vector of real-valued scores into a probability distribution, where each component lies in the interval $(0,1)$ and the sum of all components is equal to 1. Given a vector $\mathbf{z} = [z_1, z_2, \ldots, z_K]$, the softmax function for the i-th element is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad \text{for } i = 1, 2, \ldots, K$$

- **Tanh Function**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ maps inputs to $[-1,1]$ and provides better gradients compared to sigmoid.

- **ReLU (Rectified Linear Unit)**: $f(x) = \max(0,x)$ is widely used due to its computational efficiency and ability to mitigate the vanishing gradient problem.



Sigmoid Softmax Tanh ReLU

Figure 2.4: Activation Functions

2.1.3 Optimization Algorithms

The basic principle of gradient descent is given by the formula:

$$\theta_{t+1} = \theta_t - \eta \nabla L(\theta_t)$$

- **Stochastic Gradient Descent (SGD)** updates parameters using one example or mini-batch per iteration, providing computational efficiency but causing noisy updates.

- **Momentum** reduces oscillations and speeds up convergence by accumulating a velocity vector:

$$v_t = \beta v_{t-1} + \eta \nabla L(\theta_t), \quad \theta_{t+1} = \theta_t - v_t \tag{2.4}$$

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

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(\theta_t) \tag{2.5}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla L(\theta_t))^2 \tag{2.6}$$

2.1.3.1 Loss Functions and Performance Metrics

Loss functions guide the optimization by quantifying prediction error.

- **Mean Squared Error (MSE)**:

$$L = \frac{1}{n}\sum (y_i - \hat{y}_i)^2 \tag{2.7}$$

  Used in regression; penalizes large errors more.

- **Mean Absolute Error (MAE)**:

$$L = \frac{1}{n}\sum |y_i - \hat{y}_i| \tag{2.8}$$

  More robust to outliers.

- **Binary Cross-Entropy Loss**:

$$L = -\frac{1}{n}\sum [y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{2.9}$$

- **Categorical Cross-Entropy** for multi-class classification:

$$L = -\frac{1}{n}\sum\sum y_{ij}\log(\hat{y}_{ij}) \tag{2.10}$$

2.1.3.2 Performance Evaluation Metrics

In PINNs, conventional classification metrics are not applicable since the objective is to approximate continuous solutions. Instead, performance is typically evaluated using the Mean Squared Error (MSE) between the predicted and reference solutions, along with the residual error of the differential equation. These metrics indicate both the accuracy of the solution and the extent to which the physical laws are satisfied. Visualizations of solution fields and residual distributions are also used to assess performance qualitatively.

2.2 Partial Differential Equations (PDEs)

Partial Differential Equations (PDEs) are fundamental tools in mathematics used to model various physical, biological, and engineering phenomena involving continuous change

over space and time. These equations describe how physical quantities evolve and appear in fields such as fluid mechanics, thermodynamics, electromagnetism, and structural analysis. A PDE typically involves an unknown multivariable function and its partial derivatives, combined with initial and boundary conditions that define the system's constraints. A generalized form of a PDE can be written as:

$$u_t + \mathcal{N}[u] = 0, \quad x \in \Omega, \quad t \in [0,T] \tag{2.11}$$

Here, $u(x,t)$ represents the unknown function to be solved, $\mathcal{N}[\cdot]$ is a differential operator, Ω denotes the spatial domain, and $[0,T]$ is the time interval.

Despite their central role in scientific modeling, analytically solving PDEs is feasible only for a limited class of problems with idealized conditions. For most real-world applications, exact solutions do not exist and numerical methods such as the Finite Difference Method (FDM), Finite Element Method (FEM), and Finite Volume Method (FVM) [2] [9] are employed. These methods discretize the domain into grids or elements and approximate the solution at discrete points. Although effective, they face several challenges:

- **Computational cost:** High-resolution simulations require dense meshes, especially in high-dimensional domains, leading to excessive memory and computational demands.
- **Curse of dimensionality:** As the number of spatial dimensions increases, the computational burden grows exponentially, making traditional solvers infeasible.
- **Complex geometries and boundary conditions:** Irregular domains or non-standard boundary conditions are often difficult to handle accurately using mesh-based methods.

In recent years, neural networks have emerged as a promising alternative for solving PDEs. Unlike traditional methods that approximate solutions at discrete points, neural networks learn a continuous function that satisfies the governing equations and constraints in the domain. This mesh-free approach offers several advantages:

- **Generalization:** Once trained, the network can generalize over continuous domains and even adapt to parameter variations without retraining from scratch.
- **Smoothness:** Neural networks naturally produce smooth and differentiable solutions, which is especially useful for problems that require higher-order derivatives.
- **Scalability:** Deep learning architectures can, in some cases, scale more gracefully to higher dimensions, alleviating the curse of dimensionality.

- **Integration with data:** Neural approaches can incorporate both physical laws and observational data, enabling hybrid modeling where data is scarce or noisy.

These benefits motivate the growing interest in Physics-Informed Neural Networks (PINNs), which embed PDEs directly into the loss function during training, allowing the network to learn solutions that are both data-consistent and physically accurate.

2.2.1 Finite Difference Methods

Finite Element Methods (FEM) provide a powerful and flexible framework for solving PDEs, especially in problems involving complex geometries and heterogeneous material properties. The domain is divided into smaller, simpler elements (like triangles or tetrahedra), and the solution is approximated using basis functions over these elements. FEM excels in structural mechanics, heat transfer, and fluid dynamics, but requires a more elaborate setup and can become computationally expensive for large-scale simulations.

2.2.2 Finite Element Methods

Finite Element Methods (FEM) provide a powerful and flexible framework for solving PDEs, especially in problems involving complex geometries and heterogeneous material properties. The domain is divided into smaller, simpler elements (like triangles or tetrahedra), and the solution is approximated using basis functions over these elements. FEM excels in structural mechanics, heat transfer, and fluid dynamics, but requires a more elaborate setup and can become computationally expensive for large-scale simulations.

2.2.3 Spectral Methods

Spectral Methods are highly accurate numerical techniques that approximate solutions to PDEs using global basis functions, typically trigonometric polynomials or orthogonal polynomials. These methods are especially effective for smooth problems with periodic or well-behaved boundary conditions. However, spectral methods struggle with handling complex geometries and are often not the best choice for problems with discontinuities or sharp gradients. Additionally, their implementation demands a deep understanding of mathematical properties like orthogonality and convergence.

2.2.4 Challenges in Complex Geometries and High-Dimensional Problems

Traditional numerical methods face significant challenges when dealing with domains of irregular shape or problems in high dimensions. Mesh generation in complex geometries can be both time-consuming and error-prone, affecting solution accuracy. Furthermore, in high-dimensional PDEs (e.g., in financial modeling or quantum mechanics), computational costs grow exponentially—a phenomenon known as the "curse of dimensionality." These limitations motivate the exploration of alternative methods, such as Physics-Informed Neural Networks (PINNs), which promise mesh-free and dimension-agnostic solutions.

2.3 Physics-Based Modeling and Data-Driven Models

Physics-based modeling has long been the cornerstone of scientific inquiry and engineering analysis. These models are typically derived from first principles—laws of physics such as conservation of mass, momentum, and energy—and are represented using systems of equations, often in the form of partial differential equations (PDEs). They provide interpretable and theoretically grounded frameworks that can predict the behavior of complex systems with high accuracy, especially when the underlying mechanisms are well understood. However, physics-based models often require simplifying assumptions, may be computationally expensive to solve for large-scale or real-time problems, and can struggle when the system behavior is partially unknown or data is noisy and sparse.

In contrast, data-driven models—especially those powered by machine learning and deep learning—rely entirely on patterns learned from empirical data. These models do not assume prior knowledge of the underlying system but instead attempt to learn functional relationships directly from the data. As a result, they are particularly useful in situations where physical laws are too complex to model explicitly or where large volumes of data are available. While data-driven models can achieve remarkable predictive accuracy, especially with large datasets, they often lack interpretability, physical consistency, and extrapolation capabilities beyond the data distribution they were trained on.

Given the complementary strengths and weaknesses of these two paradigms, there is a growing interest in hybrid approaches that integrate physical knowledge with data-driven learning. Physics-informed models, such as Physics-Informed Neural Networks (PINNs), are a prominent example of this trend. These models incorporate physical laws into the loss function or architecture of neural networks, effectively constraining the model to respect known physics while still leveraging the flexibility and expressiveness of deep

learning. Such integration not only improves data efficiency and generalization but also enhances the robustness and trustworthiness of predictive models in scientific and engineering applications.

2.4 Evolution of the PINN Approach

The foundation of Physics-Informed Neural Networks (PINNs) was laid through a series of influential works by Raissi, Perdikaris, and Karniadakis between 2017 and 2019. [4] Their research demonstrated how neural networks can be constrained by physical laws, such as differential equations, to solve complex forward and inverse problems. These studies showed that incorporating the physics of the system into the learning process not only improves accuracy but also reduces the reliance on large datasets, making the approach especially attractive in fields where data is scarce or expensive to collect.

Physics-Informed Neural Networks (PINNs) emerged as a novel approach to combine the strengths of machine learning with the knowledge embedded in physical laws. Unlike traditional neural networks that rely heavily on large amounts of labeled data, PINNs incorporate differential equations and boundary conditions directly into the training process. This helps the model learn solutions that respect the underlying physics even when data is limited or incomplete.

Over time, researchers have worked on improving PINNs to make them more efficient and capable of handling complex problems. Enhancements include better ways to balance different parts of the training objective, techniques to divide large problems into smaller pieces, and network designs that better capture complicated solution patterns. PINNs have also been extended to work with various types of physical systems, including multiple interacting processes and uncertainty quantification. These developments have turned PINNs into a powerful tool that bridges data-driven learning with physics-based modeling for a wide range of scientific and engineering applications.

2.5 Summary of the Literature

This literature review has explored several foundational topics essential to understanding Physics-Informed Neural Networks (PINNs). We began by examining Artificial Neural Networks (ANNs), highlighting their biological inspiration and their core objective of learning meaningful patterns from data by minimizing error functions. Next, we discussed the challenges faced by traditional deep learning approaches, particularly their reliance on large datasets and limited robustness when data is scarce or incomplete.

We then reviewed the mathematical complexity of Partial Differential Equations (PDEs), which often model real-world physical phenomena but are notoriously difficult to solve analytically or numerically. The integration of physics-based modeling and data-driven methods was emphasized as a promising avenue, leveraging prior knowledge embedded in physical laws to guide learning algorithms.

Finally, we traced the evolution of the PINN approach, showing how it bridges machine learning and physics by embedding differential equations into neural network training. This progression reflects ongoing efforts to improve training efficiency, scalability, and applicability to complex multiphysics problems. Overall, the literature supports the growing potential of PINNs as a powerful tool for solving forward and inverse problems in scientific computing.

Chapter 3

Physics-Informed Neural Networks (PINNs)

3.1 Fundamental Principles and Motivation

Physics-Informed Neural Networks (PINNs) are a new approach in computational science. They combine traditional numerical methods with modern machine learning to solve problems with limited data, complex shapes, or multiple physical processes.

3.1.1 Traditional Numerical Methods: Strengths and Weaknesses

Traditional numerical methods such as the Finite Element Method (FEM), Finite Difference Method (FDM), and Spectral Methods have long been the foundation of scientific computing. FEM works by dividing the problem domain into small elements and solving approximate equations within each, making it highly versatile for complex geometries. However, this versatility comes at the cost of heavy computational demands, especially for very fine meshes. FDM approximates derivatives using difference formulas and is efficient for regular-shaped domains but struggles with complex or irregular boundaries. Spectral methods use global basis functions to represent solutions and can achieve very high accuracy for smooth problems, yet they face difficulties when the solution contains sharp discontinuities or irregularities.

These methods have some limitations:

- They require dividing the domain into a mesh, which can be difficult and slow.
- Computational cost increases rapidly as problem size grows.
- Hard to directly use experimental data.
- Difficult to handle multiple physical scales together.

A typical example where traditional numerical methods struggle is turbulent fluid flow around complex geometries. Finite element or finite difference methods require very fine meshes to capture small-scale eddies, which leads to extremely high computational cost and long simulation times. This makes them unsuitable for real-time or large-scale applications.These computational challenges are further complicated by the scarcity of high-quality data, which limits the effectiveness of traditional approaches.

3.1.2 The Problem of Data Scarcity

Modern science produces a lot of data using advanced sensors and measurement tools. However, this data is often not enough to solve complex problems. For example, fluid flow experiments might only measure speed or pressure at a few points, and climate data may cover only certain regions with gaps in time.

Physical laws and mathematical models help us understand how systems behave beyond the available data. Traditional numerical methods mostly rely on these physical models and don't directly use experimental data. On the other hand, purely data-driven methods ignore physical laws.

The main challenge is to combine limited data with physical laws to create models that use both experimental measurements and theoretical knowledge. This helps solve scientific problems more accurately and reliably.

3.1.3 Physics-Constrained Learning Approach

PINNs solve this problem by embedding physical laws directly into the neural network training. The network learns not only from data but also must satisfy the physical equations. This makes the solution respect both the observed data and the physics at the same time.

3.1.4 Key Conceptual Change

Traditional approach:

1. Write governing equations.
2. Discretize and solve numerically.
3. Compare results with data.

PINNs approach:

1. Write governing equations.
2. Design neural network.
3. Train network to satisfy equations and data simultaneously.
4. Get a solution consistent with physics and data.

This removes the gap between physics and data, creating a more complete solution.

3.1.5 Main Innovations of PINNs

- **Automatic Differentiation:** This technique allows the neural network to compute exact derivatives of its outputs with respect to inputs, without relying on approximate numerical methods. This ensures precise calculations and avoids discretization errors. We will discuss automatic differentiation in more detail in Section 3.2.
- **Universal Function Approximation:** Neural networks can represent many types of functions without being limited to fixed mathematical forms like polynomials. This flexibility helps the model learn complex physical behaviors directly from data and equations.
- **Multi-task Learning:** The model simultaneously learns to fit the available data, satisfy governing physical equations, and respect boundary and initial conditions, ensuring physically consistent and accurate solutions.
- **Mesh-free Solution:** Unlike traditional methods that require dividing the domain into small elements or meshes, PINNs represent the solution continuously and can make predictions at any point in the domain, offering greater flexibility.

3.2 Mathematical Background

3.2.1 Problem Definition and Data Representation

We consider a physical system governed by a partial differential equation (PDE) of the form:

$$\mathcal{N}[u](x,t) = 0, \quad (x,t) \in \Omega \times [0,T] \tag{3.1}$$

subject to initial and boundary conditions:

$$u(x,0) = g(x), \quad u(x,t) = h(x,t) \tag{3.2}$$

To approximate the solution, we define three sets of data points:

- **Initial/Boundary data:** Points where $g(x)$ and $h(x,t)$ are known.
- **Collocation points:** Points sampled from the domain where the PDE residual is enforced.
- **(Optional) Observational data:** Measured values of the solution, if available.

3.2.2 Neural Network Approximation

The solution $u(x,t)$ is approximated by a neural network $u_\theta(x,t)$, where θ represents the network parameters (weights and biases). The network is trained to minimize a loss function constructed to satisfy the initial and boundary conditions and to enforce the governing PDE.

$$u(x,t) \approx u_\theta(x,t)$$

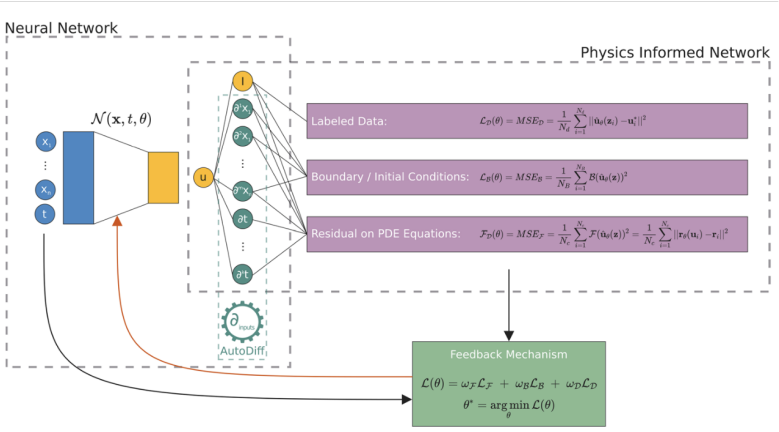


Figure 3.1: PINN Architecture

3.2.3 Loss Function Formulation

The total loss function $\mathscr{L}(\theta)$ is constructed as a weighted combination of three components:

$$\mathscr{L}(\theta) = w_d\mathscr{L}_d + w_b\mathscr{L}_b + w_f\mathscr{F}_d \tag{3.3}$$

where:

- $\mathscr{L}_d$: Loss on labelled (observational) data

- $\mathscr{L}_b$: Loss on initial and boundary conditions
- $\mathscr{F}_d$: Physics-based residual loss (PDE residual)
- w_d, w_b, w_f: Weighting coefficients for each term

Labelled Data Loss ($\mathscr{L}_d$): This term measures the discrepancy between the predicted solution and observed data values at N_D points:

$$\mathscr{L}_D = \frac{1}{N_D} \sum_{i=1}^{N_D} \left| u_\theta(x_i, t_i) - u^{(obs)}(x_i, t_i) \right|^2 \tag{3.4}$$

Boundary and Initial Condition Loss ($\mathscr{L}_B$): This term enforces the satisfaction of initial and boundary conditions. Let N_b denote the total number of such constraint points:

$$\mathscr{L}_B = \frac{1}{N_B} \sum_{i=1}^{N_B} \left| u_\theta(x_i, t_i) - u^{(bc)}(x_i, t_i) \right|^2 \tag{3.5}$$

Here, $u^{(bc)}(x_i, t_i)$ corresponds to the known value imposed by the boundary or initial condition.

PDE Residual Loss ($\mathscr{F}_d$): This term penalizes violations of the governing PDE at N_f collocation points sampled from the domain:

$$\mathscr{F}_D = \frac{1}{N_F} \sum_{i=1}^{N_F} |\mathscr{N}[u_\theta](x_i, t_i)|^2 \tag{3.6}$$

This term is also referred to as the physics-based loss or physics residual. The differential operator $\mathscr{N}$ may include temporal and spatial derivatives, which are obtained via automatic differentiation.

Summary: The total loss is minimized during training to enforce data consistency ($\mathscr{L}_D$), boundary/initial conditions ($\mathscr{L}_B$), and the underlying physical law ($\mathscr{F}_D$). This design enables the network to produce solutions that are both data-driven and physics-informed.

3.2.4 Automatic Differentiation and Optimization

To compute the PDE residual, we require spatial and temporal derivatives of the neural network output. These derivatives are obtained using automatic differentiation (AD),

which leverages the computational graph of the network.

For example, for a PDE containing second derivatives, we compute:

$$\frac{\partial u_\theta}{\partial x}, \quad \frac{\partial^2 u_\theta}{\partial x^2}, \quad \frac{\partial u_\theta}{\partial t}, \quad \text{etc.}$$

Modern deep learning frameworks (e.g., TensorFlow, PyTorch) enable the efficient calculation of these derivatives during training.

The total loss $\mathscr{L}(\theta)$ is minimized using gradient-based optimizers. Gradients are computed via backpropagation, leveraging AD:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta \mathscr{L}(\theta^{(k)}) \tag{3.7}$$

where η is the learning rate. Optimization algorithms such as Adam or L-BFGS are commonly used in practice and are detailed in Section 3.3.3.

3.3 Training Process and Loss Function of PINNs

3.3.1 Training Strategy

Physics-Informed Neural Networks (PINNs) are trained by minimizing a custom loss function that combines observational data, boundary/initial conditions, and the governing physical laws expressed as partial differential equations (PDEs). Unlike traditional supervised learning, PINNs leverage both data and physical constraints to guide the training process.

The network parameters θ are updated iteratively using gradient-based optimization algorithms (e.g., Adam, L-BFGS). During each iteration, the loss $\mathscr{L}(\theta)$ is computed and minimized with respect to θ.

The training process consists of the following steps:

1. **Sampling collocation points:** Random points are sampled within the domain Ω for evaluating the PDE residual $\mathscr{F}_d$.
2. **Sampling data and boundary/initial points:** Points where observational data ($\mathscr{L}_d$) and boundary/initial conditions ($\mathscr{L}_b$) are available are selected.
3. **Computing the total loss:** Using the previously defined weighted loss formulation:

$$\mathscr{L}(\theta) = w_d \mathscr{L}_d + w_b \mathscr{L}_b + w_f \mathscr{F}_d$$

4. **Backpropagation:** Gradients of $\mathscr{L}(\theta)$ with respect to the network parameters θ are computed using automatic differentiation.

5. **Parameter update:** Parameters are updated using an optimizer to reduce the total loss.

3.3.2 Use of Automatic Differentiation

PINNs rely heavily on automatic differentiation (AD) to compute the derivatives required by the PDE residual term. For example, in a PDE of the form

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

the network output $u_\theta(x,t)$ is differentiated with respect to x and t using AD to evaluate the residual $\mathscr{N}[u_\theta]$. This eliminates the need for discretization and allows for smooth and precise computation of derivatives.

3.3.3 Optimization Techniques

Depending on the problem and network size, different optimizers can be employed:

Adam Optimizer

Among various gradient-based optimization algorithms, **Adam (Adaptive Moment Estimation)** [1] is one of the most commonly used methods in training Physics-Informed Neural Networks (PINNs) due to its robustness and computational efficiency.

Adam combines the advantages of two popular optimizers: *AdaGrad* (adaptive learning rates) and *Momentum* (exponential moving averages of gradients). It maintains individual learning rates for each parameter and updates them adaptively during training.

At each iteration t, the Adam algorithm performs the following update steps:

1. Compute the gradient of the loss function:

$$g_t = \nabla_\theta \mathscr{L}(\theta_t)$$

2. Update biased first moment estimate (mean):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

3. Update biased second raw moment estimate (uncentered variance):

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

4. Compute bias-corrected moment estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

5. Update parameters:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

where:

- θ_t are the model parameters at iteration t,
- η is the learning rate,
- β_1 and β_2 are exponential decay rates (commonly 0.9 and 0.999, respectively),
- ε is a small constant (e.g., 10^{-8}) to avoid division by zero,
- m_t and v_t are the first and second moment estimates, respectively.

Interpretation

- m_t acts like momentum and helps smooth the gradient updates.
- v_t accounts for the magnitude of past gradients, enabling adaptive learning rates.
- Bias correction terms ($\hat{m}_t$, $\hat{v}_t$) adjust for initialization at zero in early steps.

L-BFGS Optimizer

The Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [5] is a quasi-Newton optimization method particularly effective for smooth, unconstrained problems. Unlike first-order methods such as Adam, L-BFGS approximates the inverse Hessian matrix to achieve faster convergence using curvature information, but without explicitly storing or computing the full matrix. This is crucial for large neural networks where full Hessian calculations are computationally infeasible.

The classical BFGS update for the inverse Hessian approximation H_k is given by:

$$H_{k+1} = \left(I - \rho_k s_k y_k^T\right) H_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T$$

where

- $s_k = \theta^{(k+1)} - \theta^{(k)}$ is the change in parameters,
- $y_k = \nabla_\theta L(\theta^{(k+1)}) - \nabla_\theta L(\theta^{(k)})$ is the change in gradients,
- $\rho_k = \frac{1}{y_k^T s_k}$.

L-BFGS uses a limited number m of previous (s_k, y_k) pairs to approximate H_k, drastically reducing memory usage and computational cost. The search direction is then computed as:

$$p_k = -H_k \nabla_\theta L(\theta^{(k)})$$

The parameters are updated iteratively using:

$$\theta^{(k+1)} = \theta^{(k)} + \alpha_k p_k$$

where α_k is determined through line search satisfying Wolfe conditions.

L-BFGS is particularly effective for PINNs in deterministic settings and often leads to more stable and accurate convergence compared to purely first-order methods.

3.3.4 Loss Weighting and Balancing

The choice of the weights w_d, w_b, and w_f is crucial for balancing the contributions of different components of the loss function. If w_f is too small, the physics constraints may not be enforced well. If w_d is too large, the model may overfit to noisy data and ignore the physics.

In some settings, these weights are chosen manually, while in others, adaptive methods such as *gradient balancing* or *uncertainty weighting* can be used to learn optimal weights during training.

3.3.5 Convergence Monitoring

During training, the convergence of each loss component is monitored to ensure that:

- The data loss $\mathscr{L}_d$ decreases — indicating that the model fits observed values.

- The physics loss $\mathscr{F}_d$ decreases — indicating that the model is satisfying the PDE.
- The boundary/initial loss $\mathscr{L}_b$ remains low — ensuring proper constraint satisfaction.

The training is stopped once all three components are sufficiently minimized, or based on early stopping criteria such as validation error.

Chapter 4

Case Study : Kuramoto-Sivashinsky Equation

4.1 Problem Definition

In this study, we focus on solving the Kuramoto–Sivashinsky (KS) 4.1 equation using a Physics-Informed Neural Network (PINN) approach. The KS equation is a nonlinear partial differential equation that arises in the study of instabilities in laminar flame fronts, reaction-diffusion systems, and other contexts exhibiting spatio-temporal chaos. Its characteristic nonlinear and higher-order derivative terms make it a suitable benchmark problem for testing the ability of PINNs to model complex physical dynamics.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} = 0 \tag{4.1}$$

The primary objective of this section is to demonstrate how a neural network, when trained with physics-based constraints derived from the governing PDE and initial/boundary conditions, can approximate the solution of the KS equation without requiring explicit labeled data.

By addressing this well-known chaotic system, we aim to validate the effectiveness and accuracy of the PINN framework in capturing both the temporal and spatial behaviors of nonlinear dynamical systems.

4.2 Dataset or Boundary Conditions

For the Kuramoto–Sivashinsky (KS) equation, the dataset does not consist of labeled observational data in the traditional sense. Instead, in accordance with the Physics-Informed Neural Networks (PINNs) framework, the training data is sampled over the spatio-temporal domain where the partial differential equation (PDE) is defined. This includes:

- **Collocation points** (x,t) sampled from the domain $[0,2\pi] \times [0,T]$ to enforce the PDE residual, as described in the physics loss (see Section 3.2.1).
- **Initial condition points** sampled at $t = 0$ to satisfy the initial profile $u(x,0) = \cos(x/16)(1+\sin(x/16))$, used in the initial condition loss (see Section 3.2.1).
- **Boundary points** at $x = 0$ and $x = 2\pi$ across the temporal domain to enforce periodic boundary conditions both on the function $u(x,t)$ and its spatial derivatives, incorporated in the boundary loss terms (see Section 3.2.1).

The overall loss function is formulated as a weighted sum of these individual components, balancing the contributions of the physics residual, initial condition, and boundary condition losses (see Equation 3.3) . This allows the neural network to learn a solution that satisfies the KS equation while respecting the physical constraints imposed by the initial and boundary conditions.

4.3 Model Architecture and Parameters

The neural network model used in this study follows a fully connected feedforward architecture. The network takes spatial position (x) and time (t) as inputs and outputs the solution function $u(x,t)$. The input layer has two neurons corresponding to x and t, and the output layer has one neuron for u. Between them, the model contains four hidden layers, each consisting of 128 neurons.

All hidden layers employ the hyperbolic tangent (`tanh`) activation function. This choice is motivated by the smoothness and differentiability of the `tanh` function, which is suitable for computing derivatives required in Physics-Informed Neural Networks (PINNs). This allows the model to produce solutions that satisfy not only the observed data but also the governing physical equations and constraints.

The model parameters are optimized using the Adam optimization algorithm with a learning rate of 0.001. Training was carried out for 10,000 steps, during which the model learns to approximate the solution of the differential equation while respecting the initial and boundary conditions.

The full implementation and source code of the model are available at: `https://github.com/ugurkaymaz-hub/PINN-.git`

4.4 Evaluation of Results

The effectiveness of the trained Physics-Informed Neural Network (PINN) is demonstrated through a series of visualizations and quantitative evaluations. The network's accuracy in approximating the solution to the Kuramoto–Sivashinsky (KS) equation is assessed based on both spatial and temporal domains.

Figure 4.1 presents a 2D visualization of the pointwise physics-based loss values over the spatiotemporal domain. This heatmap reveals regions where the model exhibits higher residuals, providing insights into areas of potential improvement.

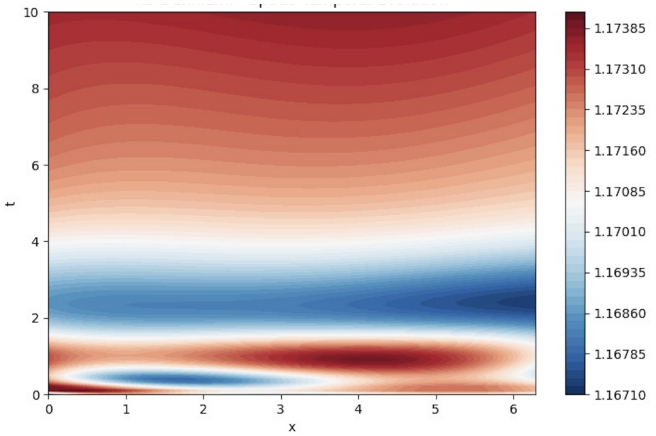


Figure 4.1: Kuramoto-Sivashinsky Equation - 2-Dim

Figure 4.2 shows the 3D surface plot of the same physics-based loss, offering a more detailed understanding of the residual landscape. Both visualizations confirm that the loss is generally minimized throughout the domain, with only minor fluctuations in specific areas. When viewed from above, the 3D surface effectively reduces to the 2D heatmap shown earlier, highlighting the consistency between both representations and providing complementary perspectives on the model's performance.

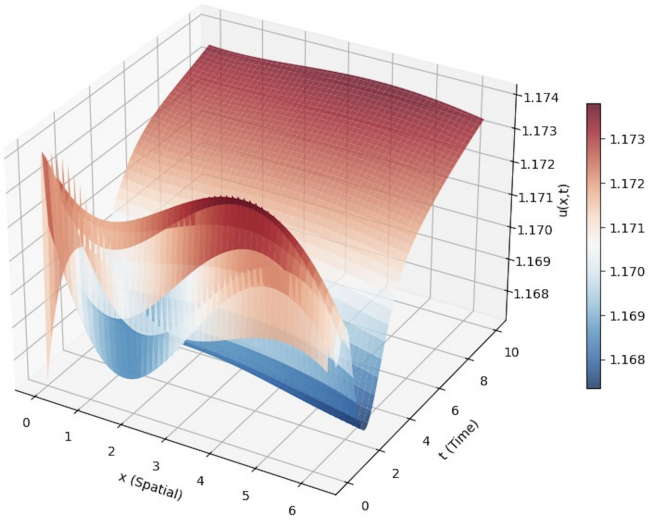


Figure 4.2: Kuramoto-Sivashinsky Equation - 3-Dim

Figure 4.3 illustrates the total loss over training epochs on a logarithmic scale. The use of a log scale highlights the rapid decrease in loss during the initial training phase. It is important to note that the model starts with a relatively high loss due to the randomly initialized weights and initial condition noise. As training progresses, the loss consistently decreases, demonstrating the model's successful convergence and its ability to align with the physical constraints imposed by the KS equation.

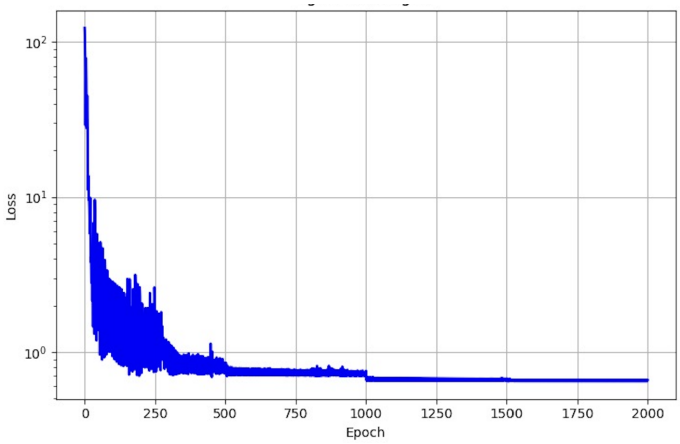


Figure 4.3: Total Loss

Chapter 5

Future Work And Discussion

Future work can focus on several improvements to enhance the model's accuracy and generalization capabilities. Firstly, optimizing training by experimenting with various hyperparameter combinations, such as learning rate, number of hidden layers, and activation functions, can lead to better performance. Additionally, implementing the model using alternative deep learning frameworks like PyTorch instead of TensorFlow can enable comparative studies across platforms.

Since the initial condition function directly influences the model's behavior, experiments with different initial conditions can provide a more comprehensive evaluation of the PINN approach. For example, it has been observed that choosing trigonometric functions as initial conditions yields better results compared to exponential and linear functions.

Moreover, adjusting the weights within the loss function or introducing new metrics can allow more precise control over the model's physical consistency.

Finally, the current neural network architecture can be enhanced with more complex structures, such as deeper networks, residual connections, or attention mechanisms, to improve solution quality. These modifications can significantly contribute to expanding the applicability of PINNs to diverse physical problems and obtaining more general solutions.

PINNs are expected to play a crucial role in the future for solving multidisciplinary problems, especially in modeling complex systems involving uncertainty and stochastic processes. Adaptive learning capabilities leveraging real-time data streams will be developed, enabling more flexible and accurate predictions in dynamic and variable systems. Furthermore, more efficient optimization algorithms and parallel computing techniques will be integrated to reduce computational costs. These advancements will facilitate the broad use of PINNs in both theoretical research and practical applications across engineering, physics, biology, finance, and many other fields.

Chapter 6

Conclusion

In this thesis, the concept and applications of Physics-Informed Neural Networks (PINNs) have been studied in detail. By combining data-driven approaches with the laws of physics, PINNs offer a powerful framework for solving partial differential equations (PDEs) and modeling complex physical systems.

We explored the theoretical foundations of PINNs and demonstrated their effectiveness through numerical experiments. The results show that PINNs can approximate solutions to differential equations with high accuracy, even in cases where data is sparse or noisy. This makes them a valuable tool in scientific computing and applied mathematics.

In addition, we discussed the advantages and limitations of PINNs compared to traditional numerical methods, highlighting their flexibility, generalizability, and potential for future research.

As a growing field, PINNs open many possibilities for solving inverse problems, modeling real-world systems, and integrating physical knowledge into machine learning models. Future studies can focus on improving training efficiency, handling more complex boundary conditions, and extending PINNs to multi-physics and high-dimensional problems.

In conclusion, this work contributes to the understanding and application of PINNs, and it lays the groundwork for further research in physics-informed machine learning.

Bibliography

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[2] Randall J. LeVeque. *Finite Difference Methods for Differential Equations*. University of Washington, 2005.

[3] Nazanin Ahmadi3 Maziar Raissi1, Paris Perdikaris2 and George Em Karniadakis4. Physics-informed neural networks and extensions. *arXiv preprint arXiv:2408.16806*, 2024.

[4] Paris Perdikaris2 Maziar Raissi1 and George Em Karniadakis1. Physics informed deep learning (part i): Data driven solutions of nonlinear partial differential equations. 1, 2017.

[5] Jorge Nocedal Hao-Jun Michael Shi Ping Tak Peter Tang Raghu Bollapragada, Dheevatsa Mudigere. A progressive batching l-bfgs method for machine learning. *arXiv preprint arXiv:1802.05374*, 2018.

[6] Keith Rudd. Solving partial dierential equations using artificial neural networks. *arXiv preprint arXiv:1609.04747*, 2013.

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2017.

[8] Tobias Würth, Constantin Krauß, Clemens Zimmerling, and Luise Kärger. Physics-informed neural networks for data-free surrogate modelling and engineering optimization – an example from composite manufacturing. *Materials Design*, 231:112034, 2023.

[9] O. C. Zienkiewicz and R. L. Taylor. Finite difference method for approximate solution of a boundary value problem with interior singular point. *Butterworth-Heinemann*, 1, 2021.