ABSTRACT

Title of Dissertation: RESOLVING STEEP GRADIENTS FOR PHYSICS-INFORMED NEURAL NETWORKS: RICHARDS' EQUATION AND CONVECTION-DIFFUSION EQUATION

Jiajing Guan
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Howard Elman
Department of Computer Science, AMSC

Solving partial differential equations (PDEs) is fundamental to modeling physical, biological, and engineering systems. Traditional numerical methods, while accurate and robust, often struggle with high-dimensional problems, complex geometries, discontinuous boundary conditions, and steep solution gradients. Deep learning, and in particular Physics-Informed Neural Networks (PINNs), offers a promising mesh-free alternative by embedding physical laws directly into the training process of neural networks.

This dissertation investigates and improves the performance of PINNs for PDEs with sharp transitions and nonlinearities. We focus on two key equations: Richards' Equation, which models unsaturated flow in porous media, and the Convection-Diffusion Equation, which governs advective-diffusive transport processes. For Richards' Equation, we introduce a PINN framework that includes surface flux as an input and discrete residuals to enforce causality. For convection-diffusion problems, we address both known and unknown gradient layer locations. When the

location is known, we analyze PINN limitations and propose input transformations that focus model capacity on high-gradient regions. When the location is unknown or complex, we introduce the Auxiliary-Input PINN (AI-PINN), a novel architecture that adapts spatial transformations based on an auxiliary input. A successive training strategy is also used to learn solutions without prior knowledge of gradient positions.

The methods developed in this work offer general strategies for improving the accuracy and robustness of PINNs when applied to challenging PDE problems with steep gradients. Through a combination of architecture design, loss reformulation, and training strategies, this dissertation contributes toward making physics-informed learning a viable tool for complex real-world simulations.

RESOLVING STEEP GRADIENTS FOR PHYSICS-INFORMED NEURAL NETWORKS: RICHARDS' EQUATION AND CONVECTION-DIFFUSION EQUATION

by

Jiajing Guan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Howard Elman, Chair/Advisor
Professor Maria Cameron
Professor Haizhao Yang
Professor Deep Ray
Professor James Baeder

Table of Contents

Table of Contents ii

List of Tables iv

List of Figures v

List of Abbreviations viii

Chapter 1: Introduction 1
- 1.1 Deep Learning for PDEs . . . 1
- 1.2 Physics-Informed Neural Networks (PINNs) . . . 3
- 1.3 PDEs in Question . . . 7
  - 1.3.1 Richards' Equation . . . 7
  - 1.3.2 Convection-Diffusion Equation . . . 8
- 1.4 Contributions . . . 9

Chapter 2: Background 12
- 2.1 Governing Equations . . . 12
  - 2.1.1 Richards' equation . . . 13
  - 2.1.2 Convection-Diffusion Equation . . . 16
- 2.2 Classical Numerical Methods . . . 18
  - 2.2.1 Finite Difference Method (FDM) . . . 18
  - 2.2.2 Finite Element Method (FEM) . . . 19
  - 2.2.3 Finite Volume Method (FVM) . . . 20
  - 2.2.4 Challenges in Resolving Steep Gradients . . . 20
- 2.3 Neural Networks and Physics-Informed Neural Networks . . . 22
  - 2.3.1 Dense Neural Networks (DNNs) . . . 22
  - 2.3.2 Universal Approximation and Generalization . . . 26
  - 2.3.3 Physics-Informed Neural Networks (PINNs) . . . 27
  - 2.3.4 Training and Optimization . . . 29
- 2.4 Related Cooridinate-Based Enhancements on PINNs . . . 29
- 2.5 Summary and Outlook . . . 32

Chapter 3: PINNs for Richards' Equation 34
- 3.1 Introduction . . . 34
- 3.2 Problem Statement and Background . . . 35
  - 3.2.1 Monotonic Neural Networks . . . 36

3.2.2 Limitations of Existing PINNs for Richards' Equation 37
3.3 Improved Physics-Informed Neural Network Formulation 39
3.3.1 Flux as an Input 40
3.3.2 Discrete Residual (DR) 45
3.4 Results for Various Flux and Soil Conditions 49
3.5 Discussion 51

Chapter 4: PINNs for Convection-Diffusion Equation: Known Steep Gradient Layer Location 53
4.1 Introduction 53
4.2 Problem Statement 55
4.3 Correcting Oscillatory Discrete Solutions 57
4.4 Correcting Reduced Solutions: Reduced PINN (RPINN) 62
4.4.1 Necessary Conditions for Accurate Approximation: Correct Effective Shift 64
4.4.2 Necessary Conditions for Accurate Approximation: Steep First-Layer Weights 73
4.4.3 Neural Tangent Kernel (NTK) Analysis 84
4.5 The Two-Dimensional Convection-Diffusion Equation 91
4.6 Discussion 92

Chapter 5: PINNs for Convection-Diffusion Equation: Unknown Steep Gradient Layer Location 94
5.1 Introduction 94
5.2 Auxiliary-Input PINNs (AI-PINNs) 96
5.3 Successive Training 101
5.3.1 Successive Training Procedure 103
5.3.2 Network Comparison on 1D Convection-Diffusion Problem 104
5.3.3 Fixed Input Transformation 111
5.3.4 Networks without Bias 115
5.4 1D Reaction-Diffusion Problem 120
5.5 2D Convection-Diffusion Problems 124
5.5.1 Example 1: Vertical Wind, One Boundary Layer 126
5.5.2 Example 2: Vertical Wind, Characteristic Layers 129
5.5.3 Example 3: Constant 30° Wind, Multiple Boundary Layers 132
5.6 Discussion 137

Chapter 6: Conclusion and Future Work 140

List of Tables

3.1 The relative error of $\hat{\psi}$, $\hat{\theta}$ and $\hat{K}$ of a network trained with loss function in Eq. (3.4) for near-surface depths and the entire depth column. 42

3.2 The relative error of $\hat{\psi}$, $\hat{\theta}$ and $\hat{K}$ of a network trained with loss function in Eq. (3.6) for near-surface depths and the entire depth column. 47

3.3 The L_2 relative error of VWC for the out-of-sample time ($t \in [1.5, 3]$) and near-surface depths ($z \in [-20, 0]$). 49

5.1 Relative errors of RPINN, AIR-PINN, and TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ under successive training. 107

5.2 Relative mean errors of RPINN, AIR-PINN, and TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ under successive training. The transformation stretching factor is fixed to $a = 10$. 114

5.3 Relative mean errors of bias-free AIR-PINN and bias-free TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ with transformation stretching factor $a = \epsilon^{-1/2}$. 118

5.4 Relative mean errors of RPINN (non-successive) and bias-free AIR-PINN and TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$. Bias-free networks use fixed transformation $a = 10$. 120

5.5 Relative mean errors of AIR-PINN variants in the regions $[0, 0.1]$, $[0.1, 0.9]$, and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ under successive training with fixed input transformation. 123

5.6 Absolute errors (max and mean) of bias-free AIR-PINN and TS-PINN with fixed transformation, bias-free AIR-PINN and TS-PINN with $a = \epsilon^{-1/2}$ and standard AIR-PINN and TS-PINN with fixed transformation for various ϵ values and transformation settings. 128

5.7 Absolute difference (max and mean) between IFISS solution and bias-free AIR-PINN and TS-PINN with fixed transformation, bias-free AIR-PINN and TS-PINN with $a = \epsilon^{-1/2}$, and standard AIR-PINN and TS-PINN with fixed transformation for various ϵ values. 131

5.8 Absolute errors (max and mean) of bias-free AI-PINN with fixed transformation, bias-free AI-PINN with $a = \epsilon^{-1/2}$, and standard AI-PINN with fixed transformation for various ϵ values. 135

List of Figures

2.1 Illustration of the infiltration process in unsaturated porous media [1]. 13
2.2 Image of a fully-connected neural network. 22
2.3 Comparison of common activation functions: ReLU, Sigmoid, and Tanh. 24

3.1 The surface flux for $t \in [0, 3]$ days. 41
3.2 Predicted VWC (red) and simulated VWC (blue) at an in-sample time $t = 0.12$ days 3.2a and out-of-sample time $t = 2.16$ days 3.2b using the loss function in Eq. (3.4). 43
3.3 Predicted VWC (left), simulated VWC (middle) and absolute error of VWC (right) before adding discrete residual in the loss function. 44
3.4 The new PINN structure with the modified loss function in Eq. (3.6) 46
3.5 Predicted VWC (left), and simulated VWC (middle) and absolute error of VWC (right) after adding discrete residual in the loss function. 48
3.6 Predicted VWC (left) compared to simulated VWC (middle) for PINN trained with 3 depths and 0.02 noise level. 50

4.1 Exact solutions of Equation (4.1) for $\epsilon \in \{0.1, 0.01, 0.001, 0.0001\}$. 55
4.2 Corrections made by standard PINN on FDM oscillatory solutions for: $\epsilon = 0.01$ in Figure 4.2a and $\epsilon = 0.001$ in Figure 4.2b. 57
4.3 Corrections made by PINN with input transformation $T(x) = 100(x-1)$ on FDM oscillatory solutions for: $\epsilon = 0.01$ in Figure 4.3a and $\epsilon = 0.001$ Figure 4.3b. . . . 59
4.4 Different approximations generated using a standard PINNs under different random initialization seeds in 4.4a. Comparison of effects of transformation in 4.4b. Here the input is transformed from $x \in [0, 1]$ to $x - 1 \in [-1, 0]$. 62
4.5 Histograms of $|W_1^i|$ and shift values. 68
4.6 Histograms of $|W_1^i|$ and shift values for an *Untransformed* that output a line and a *Transformed* network that approximates the solution accurately. 69
4.7 For a network with one hidden layer of width 2, in the first hidden layer, the output after the affine transformation is $xW_1 + b_1$ (or $(x-1)W_1 + b_1$), a matrix of two columns. The two columns of the output of the affine transformation are shown in 4.7a. The two columns of the output of $\tanh(xW_1 + b_1)$ (or $\tanh((x-1)W_1 + b_1)$) are shown in 4.7b. Two inset figures show a zoomed-in look near $y = 1$ at $x = 0$ and $x = 1$. 71
4.8 Histograms of $|W_1^i|$ and shift values for networks with one hidden layer of width 2, 5, and 20. 76
4.9 Histograms of $|W_1^i|$ and shift values for networks with one hidden layer of width 20 for $\epsilon = 10^{-j}$, $j = 1, 2, 3, 4$. 77

4.10 Histograms of $|W_1^i|$ and shift values for networks with one, two, and three hidden layers of width 20. 80
4.11 Histogram of $|W_2^{i,j}|$ for networks with two and three hidden layers of width 20. . 81
4.12 Loss value during training for a network of one hidden layers of width 20 in Figure 4.12a. Corresponding approximations in Figure 4.12b. First 10000 epochs used Adam optimizer, followed by LBFGS optimizer. Here a and b are the linear transformation parameters in (4.21). 83
4.13 Eigenvalues of $\boldsymbol{K}$ for different network width and depth (4.13a), and for different transformation parameters a, b (4.13b). 84
4.14 Eigenvectors corresponding to the 6 largest eigenvalues of $\boldsymbol{K}$, for eigenvalues decreasing from left to right, top to bottom. Fig 4.14a: the eigenvectors for $a = 1, b = 0$. Fig 4.14b: the eigenvectors for $a = 1, b = -1$. Fig 4.14c: the eigenvectors for $a = 10, b = 0$. Fig 4.14d: the eigenvectors for $a = 10, b = -1$ . . 86
4.15 4.15a Eigenvalues of $\boldsymbol{K}$ for different residual sample sizes . 4.15b Eigenvectors of $\boldsymbol{K}$ corresponding to the six largest eigenvalues for sample size 513. 88
4.16 4.16a Approximations produced by trained networks of different size. 4.16b Zoom in view near $u = 1$. 90
4.17 Eigenvalues and eigenvectors corresponding to the 8 largest eigenvalues of $\boldsymbol{K}$ for $a = 1, b = -1$ after training. Eigenvalues decrease from left to right, top to bottom. Fig 4.17a: the eigenvalues of NTK for networks of 1 hidden layer with width 5 and 100, respectively; Fig 4.17b: the eigenvectors for a network of 1 hidden layer with width 5, Fig 4.17c: the eigenvectors for a network of 1 hidden layer with width 100. 90
4.18 Solution of (4.30) for $\epsilon = 1e-4$ in 4.18a. Approximations by PINN in 4.18b and 4.18c. The input transformations are for 4.18b: no transformation; and for 4.18c: $(x, y) \to (x, 10(y-1))$. 91
4.19 Eigenvectors corresponding to the 6 largest eigenvalues of $\boldsymbol{K}_{rr}$. Eigenvalues decrease from left to right, top to bottom. Fig 4.19a: the eigenvectors for no transformation. Fig 4.19b: the eigenvectors for transformation $(x, y) \to (x, 10(y-1))$. 92

5.1 Comparison of approximations for $\epsilon = 0.1$ and $\epsilon = 1e-5$ using RPINN, AI-PINN, and TS-PINN following 1. 105
5.2 Absolute and relative errors of RPINN, AI-PINN, and TS-PINN $\epsilon = 1e-5$ following 1. 106
5.3 Comparison of initial predictions after loading trained weights and biases from $\epsilon = 0.1$ and top four NTK eigenvectors for RPINN, AI-PINN, and TS-PINN at $\epsilon = 1e-5$. 110
5.4 Comparison of approximations for $\epsilon = 0.1$ and $\epsilon = 1e-5$ using RPINN, AI-PINN, and TS-PINN following 1. The transformation stretching factor is fixed to $a = 10$. 112
5.5 Absolute and relative errors of RPINN, AIR-PINN, and TS-PINN $\epsilon = 1e-5$ following 1. The transformation stretching factor is fixed to $a = 10$. 113
5.6 Comparison of approximations for $\epsilon = 0.1$ and $\epsilon = 1e-5$ using bias-free AI-PINN and bias-free TS-PINN following 1. The transformation stretching factor is $a = \epsilon^{-1/2}$. 117

5.7 Absolute and relative errors of bias-free AIR-PINN and bias-free TS-PINN $\epsilon = 1e-5$ following 1. The transformation stretching factor is $a = \epsilon^{-1/2}$. 117
5.8 Top four NTK eigenvectors for bias-free AIR-PINN and bias-free TS-PINN at $\epsilon = 1e-5$ at zero epoch. 118
5.9 Comparison of approximations produced for $\epsilon = 0.1$ and $\epsilon = 1e-5$ of (5.17). . . 122
5.10 Absolute and relative errors of approximations for $\epsilon = 1e-5$ using AI-PINN. . . 123
5.11 Comparison of exact solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.1$ in Example 1. 127
5.12 Comparison of exact solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.0001$ in Example 1. 127
5.13 Comparison of IFISS solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.1$ in Example 2. 130
5.14 Comparison of IFISS solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.0001$ in Example 2. 130
5.15 Comparison at $\epsilon = 10^{-1}$ using AI-PINN with constant a. 136
5.16 Comparison at $\epsilon = 10^{-2}$ using AI-PINN with constant a. 136
5.17 Comparison at $\epsilon = 10^{-3}$ using AI-PINN with constant a. 137
5.18 AI-PINN with constant a prediction at $\epsilon = 10^{-4}$; no IFISS reference due to mesh limitations. 138

List of Abbreviations

AI-PINN	Auxiliary-Input Physics-Informed Neural Network
AIR-PINN	Auxiliary-Input Reduced Physics-Informed Neural Network
DNN	Dense Neural Network
FDM	Finite Difference Method
FEM	Finite Element Method
FFE-PINN	Fourier-Feature-Embedded Physics-Informed Neural Network
FVM	Finite Volume Method
HFC	Hydraulic Conductivity Function
LWIR	Longwave Infrared
MS-PINN	Multi-Scale Physics-Informed Neural Network
MWIR	Midwave Infrared
PDE	Partial Differential Equation
PINN	Physics-Informed Neural Network
R-PINN	Reduced Physics-Informed Neural Network
ReLU	Rectified Linear Unit
TS-PINN	Two-Scale Physics-Informed Neural Network
UAT	Universal Approximation Theorem
VWC	Volumetric Water Content
WRC	Water Retention Curve

Chapter 1: Introduction

1.1 Deep Learning for PDEs

Deep learning has transformed many fields by enabling data-driven approaches to solving complex problems. From computer vision [2, 3] and natural language processing [4, 5, 6, 7, 8, 9] to physics simulations [10, 11, 12] and engineering applications [13, 14, 15, 16], deep neural networks have demonstrated remarkable success in modeling highly nonlinear functions and extracting hidden patterns from large datasets [17, 18]. One of the emerging areas where deep neural networks have shown promise is solving Partial Differential Equations (PDEs)—a fundamental class of equations that describe a wide range of physical and engineering systems. The governing equations of many physical phenomena are described in the form of PDEs. These equations can be generally written as

$$u_t + \mathcal{N}[u] = 0 \tag{1.1}$$

where $\mathcal{N}$ denotes a (possibly nonlinear) differential operator. In most cases, the models are far too complicated to find explicit solutions. Thus, a surrogate function $\hat{u}$ that approximates the solution u would be needed. Traditional numerical methods such as the finite difference method (FDM), finite element method (FEM), and spectral methods have long been the standard approaches

for solving PDEs [19, 20, 21]. While these approaches are well-established and accurate, they require discretization in space and time, which results in high computational and memory costs. These limitations are exacerbated in high-dimensional settings, irregular geometries, and problems involving steep gradients or complex boundary conditions. Moreover, integrating data into simulations, as required for inverse problems or uncertainty quantification, often falls outside the capabilities of classical solvers.

The Universal Approximation Theorem [22, 23] guarantees that a deep neural network whose layers containing a sufficient number of neurons can approximate any continuous function and its derivatives on a compact domain, under mild assumptions on the activation function. This theoretical guarantee motivates the use of neural networks to approximate PDE solutions, particularly in cases where classical methods require fine discretizations.

One of the primary advantages of using deep neural networks for solving PDEs is their mesh-free approximation capability [24, 25]. Unlike classical solvers that require explicit grid discretization, neural networks can adapt to different problem domains without the need for a predefined mesh. This flexibility allows them to handle complex geometries and irregular domains more effectively [26, 27, 28].

Another significant benefit is the scalability in high dimensions. Traditional numerical methods often struggle with the curse of dimensionality, where the computational cost increases exponentially with the number of dimensions. In contrast, neural networks can efficiently learn mappings in high-dimensional spaces [29, 30], making them well-suited for problems involving multiple spatial and temporal dimensions.

Deep neural networks also excel in data-driven and physics-guided learning. They can integrate real-world data into their training process, enabling hybrid approaches that combine

data and physics for solving PDEs [31, 32, 33]. This capability is particularly useful for modern scientific computing applications, such as inverse problems and uncertainty quantification, where incorporating data into simulations is crucial.

Lastly, neural networks offer generalization across parameters. Once trained, a neural network can infer solutions for multiple parameter settings without the need to re-solve the system from scratch [34, 35]. This property significantly reduces the computational cost and makes neural networks a powerful tool for solving PDEs in various fields.

Given these properties, deep neural networks have emerged as a promising tool for solving PDEs in various fields, including fluid mechanics [36, 37], bio-engineering [38, 39], meta-material design [40, 41], free boundary problems [42]. Among different deep learning approaches, Physics-Informed Neural Networks (PINNs) have gained significant attention for their ability to encode physical laws directly into the learning process, making them particularly effective for solving PDEs without requiring large labeled datasets.

1.2 Physics-Informed Neural Networks (PINNs)

PINNs have demonstrated significant successes across various scientific and engineering disciplines by integrating physical laws into neural network architectures. Notable achievements include simulations and solving inverse problems, where internal system parameters are inferred from observable data [43, 44, 45, 46]. They have been utilized in areas such as nano-optics [47], topology optimization [48], multiphase flow in porous media [49], and high-speed fluid dynamics [50]. Notably, PINNs exhibit robustness when handling noisy and uncertain datasets, enhancing their reliability in practical applications [51].

These successes highlight the versatility and efficacy of PINNs in integrating domain-specific knowledge with machine learning, leading to more accurate and efficient solutions to complex problems.

Despite their strengths, PINNs face several challenges. One of the primary challenges in PINNs is spectral bias, which refers to the tendency of neural networks to favor low-frequency components while struggling to capture high-frequency details [52, 53]. This limitation is particularly problematic when solving wave propagation, turbulence modeling, and multi-scale physics problems, where fine-scale structures must be accurately represented. Traditional neural networks often fail to approximate high-frequency oscillations efficiently, leading to poor solution accuracy in these cases [54].

To address this issue, several approaches have been proposed. Fourier feature networks and sinusoidal activations have been introduced to explicitly encode high-frequency information into the network, reducing spectral bias [55, 56]. Additionally, multi-scale deep neural networks (MscaleDNNs) decompose the solution into different frequency bands, allowing the network to learn fine and coarse-scale features separately [57, 58, 59] . Furthermore, phase-shifted networks have been applied to wave-based PDEs, improving approximation accuracy [60]. Collectively, these methods enhance the ability of PINNs to resolve high-frequency physics and complex oscillatory phenomena.

Another significant limitation of PINNs is their slow convergence rate and high computational cost. Training deep neural networks for PDE solutions often requires large-scale optimization across the entire problem domain, making convergence inefficient and computationally expensive [61, 62, 63].

Additionally, PINNs often struggle with high-dimensional function spaces, leading to

excessive training times [49]. Solving high-dimensional PDEs remains a formidable challenge for traditional numerical methods and PINNs alike. As the number of spatial and temporal dimensions increases, the computational cost of solving PDEs scales exponentially, leading to the curse of dimensionality [62, 64].

Several deep learning strategies have been proposed to address this issue. Dimension-augmented PINNs enhance network representations by explicitly incorporating extra dimensions into the learning process, allowing for better generalization [30]. Additionally, self-paced learning strategies such as SelectNet gradually increase the complexity of training samples, improving training efficiency and convergence rates [65]. These advancements help PINNs scale to complex, high-dimensional physics problems, making them viable alternatives to traditional solvers.

A fundamental challenge in training PINNs is ensuring balanced and stable loss functions [53]. Since PINNs incorporate both data-driven loss terms and physics-based constraints, improper weighting of these components can lead to instability or inaccurate solutions. If physics loss terms dominate, PINNs may fail to capture important data-driven features. Conversely, if the data-driven terms are too dominant, the learned solution may not satisfy the governing physical laws accurately.

To mitigate this problem, residual-based attention mechanisms have been introduced to dynamically balance loss function weights throughout training [66]. Additionally, Sobolev training helps improve gradient flow and stabilization, allowing for more effective optimization of PINNs [67, 68]. Self-adaptive loss balancing techniques dynamically adjust loss weights based on the relative contribution of each term, ensuring stable convergence and improved accuracy [56, 69]. These strategies have significantly enhanced the reliability and robustness of PINNs in complex physics applications.

Unlike traditional numerical solvers that enforce time evolution step-by-step, PINNs often solve PDEs across the entire time-space domain simultaneously, leading to causality violations in time-dependent problems. These violations result in non-physical solutions, where future states influence past predictions, degrading overall model performance.

To address this, researchers have developed causality-aware training techniques, ensuring that information flows in the correct temporal order [70]. Additionally, hierarchical learning structures have been introduced to enforce correct sequential dependencies in PINNs [71]. These methods preserve the physical validity of solutions while improving model accuracy in time-dependent PDEs.

The efficiency of PINNs is highly dependent on the selection of training points, as poor sample selection can lead to wasted computation and slow convergence. Traditional training methods often use uniformly spaced collocation points, which may not be optimal for capturing localized features or high-gradient regions.

To improve sampling efficiency, self-paced training strategies such as SelectNet allow networks to dynamically adjust sample point selection based on learning progress [65]. Additionally, uncertainty-aware sequential sampling prioritizes regions with high prediction uncertainty, improving computational efficiency [72]. These adaptive strategies ensure that PINNs focus training efforts where they are most needed, accelerating convergence and reducing computational cost.

PINNs remain a rapidly evolving research area with numerous open questions. Improving their efficiency, stability, and generalization capabilities will be key to their widespread adoption in scientific computing. With ongoing advancements, they have the potential to revolutionize the way PDEs are solved, bridging the gap between data-driven learning and physics-based modeling.

1.3 PDEs in Question

In this section, we introduce two PDEs that we focus on in this dissertation: Richards' equation and the convection-diffusion equation.

1.3.1 Richards' Equation

The ability to predict soil moisture content is valuable for many applications, such as agriculture, water resource management, and for understanding the performance of sensors impacted by soil moisture. Richards' equation is a nonlinear PDE that is used to model the unsaturated flow of water within soil and can be used to accurately approximate soil moisture given known soil properties and precipitation influx. In the application of remote sensing, the performance of sensors, such as, longwave infrared (LWIR) [73, 74, 75], midwave infrared (MWIR)[76], radar [77**?**], and terahertz[78] are impacted by soil moisture content[79]. Thus, approximating soil moisture without in-situ sensor measurements will enable the ability to understand sensor performance prior to deployment of the sensors for remote sensing applications. For example, LWIR imagery can be used to detect subsurface foreign objects by detecting the presence of a thermal anomaly. However, the degree of thermal contrast between object and background is reduced by high levels of soil moisture and can lead to false negatives in the detection[79]. For radar, the one-way propagation loss is determined by the attenuation factor, which is directly impacted by soil characteristics, such as soil moisture. Clearly, it is important to have an understanding of soil moisture content to inform sensor performance when using sensors such as LWIR and radar for remote sensing applications. Satellite data can be used to approximate soil moisture on a large spatial and temporal scale[2], however, the timescale of this data may be insufficient for a given

remote sensing operation.

Richards' equation is a widely used PDE that describes the liquid flow in an unsaturated porous homogeneous rigid soil [80, 81, 82, 83]. It is widely used in hydrology, soil science, and agriculture to model infiltration, evaporation, and water retention in soil. The equation is given by:

$$\frac{\partial \theta}{\partial t} = \nabla \cdot (K(\theta)\nabla h) + S, \tag{1.2}$$

where θ is the volumetric water content, $K(\theta)$ is the hydraulic conductivity, h is the pressure head, and S is a source or sink term. The nonlinearity in Richards' equation arises from the dependence of $K(\theta)$ on the moisture content, making it challenging to solve using traditional numerical methods.

The complexity of solving Richards' equation stems from sharp wetting fronts, heterogeneous soil properties, and nonlinear boundary conditions. Numerical approaches such as finite difference and finite element methods often require fine spatial discretization to capture these effects accurately, leading to high computational costs. Furthermore, standard numerical methods struggle with mass conservation and stability issues in heterogeneous or anisotropic media.

1.3.2 Convection-Diffusion Equation

The convection-diffusion equation describes the transport of a scalar quantity such as heat, mass, or momentum within a medium due to both convection and diffusion. It plays a crucial role in fluid mechanics, environmental modeling, and heat transfer applications. The general form of the equation is given by:

$$\frac{\partial u}{\partial t} - \epsilon \nabla^2 u + \vec{w} \cdot \nabla u = f, \tag{1.3}$$

where u represents the transported scalar field, $\vec{w}$ is the velocity field, ϵ is the diffusion coefficient, and f is a source or sink term. The interplay between convection and diffusion determines the solution behavior, with convection dominating when transport is advection-driven, and diffusion dominating when molecular or turbulent mixing is prevalent.

Challenges arise in convection-dominated regimes, where sharp gradients and boundary layers lead to numerical instability in traditional solvers. Classical numerical approaches such as finite volume and finite element methods often require artificial diffusion and often stabilization techniques to mitigate oscillations.

1.4 Contributions

This dissertation advances the development and application of PINNs for solving nonlinear and advection-dominated PDEs with challenging features such as steep solution gradients, discontinuous boundary conditions, and unknown internal layer structures. The key contributions are as follows:

- **Richards' Equation with discontinuous surface fluxes:** We propose treating the surface flux as an explicit input to the PINN to account for sudden changes in volumetric water content. We also introduce discrete residual terms to the loss function to enforce temporal causality and improve solution consistency in time-dependent problems.

- **Convection-Diffusion Equation with known steep gradient location:** We conduct an

in-depth analysis of the conditions under which PINNs succeed or fail in resolving steep internal layers. To enhance accuracy in such cases, we introduce an input transformation method that adjusts spatial coordinates to sharpen the network's focus around the gradient region.

- **Convection-Diffusion Equation with unknown steep gradient location:** We develop a new architecture, the *Auxiliary-Input PINN (AI-PINN)*, in which the network receives an auxiliary input controlling the degree of coordinate transformation. This allows the PINN to better describe the location of the steep gradient regions. Additionally, we utilize a *successive training strategy* that learns the location of the steep gradient layer for larger diffusion coefficient, and transfer this knowledge for more convection-dominated cases, enabling accurate approximations without prior knowledge on the characteristics of the solution.

These contributions offer generalizable improvements to PINN design and training strategies, with applications beyond the two PDE types studied here. The proposed methods demonstrate how physics-informed learning can be adapted to handle complex nonlinear phenomena and sharp solution features.

The remainder of this dissertation is organized as follows:

- **Chapter 2** provides the foundational knowledge required for the dissertation. It introduces dense neural networks (DNNs), the PINN framework, Richards' Equation, the Convection-Diffusion Equation, and a review of relevant PINN literature.

- **Chapter 3** applies PINNs to Richards' Equation, which models unsaturated flow in porous media. This chapter introduces a method of using surface flux as an additional input

to capture discontinuities in volumetric water content and proposes a discrete residual formulation to enforce causality.

- **Chapter 4** focuses on the convection-diffusion equation when the location of the steep gradient layer is known. We study the necessary conditions for PINNs to accurately resolve sharp internal layers and introduce input transformations to improve performance in these cases.

- **Chapter 5** extends the work to convection-diffusion problems where the steep gradient location is unknown or complex. We propose the Auxiliary-Input PINN (AI-PINN) and a successive training strategy that enables the network to adaptively resolve steep features without prior localization.

- **Chapter 6** summarizes the key findings of the dissertation and discusses future research directions.

Chapter 2: Background

This chapter provides the mathematical and algorithmic background that motivates the developments in subsequent chapters. We begin by introducing the governing equations of interest in Section 2.1, Richards' equation and the convection-diffusion equation, both of which are known to exhibit steep gradients or internal layers that challenge traditional numerical solvers. We then briefly review classical discretization methods in Section 2.2, and discuss their limitations in resolving such features efficiently. Next, we present the structure and function of dense neural networks in Section 2.3, followed by an introduction to Physics-Informed Neural Networks (PINNs), a mesh-free learning framework for solving partial differential equations (PDEs). Finally, we review recent architectural enhancements to PINNs designed to improve performance in multiscale and singularly perturbed regimes in Section 2.4. This chapter lays the foundation for the new methods proposed in this dissertation.

2.1 Governing Equations

This work focuses on two representative PDEs that exhibit sharp layers and pose challenges for standard numerical and learning-based solution methods: the Richards' equation and the Convection-Diffusion Equation. These equations serve as benchmark problems for evaluating the performance of PINNs and their extensions.

2.1.1 Richards' equation

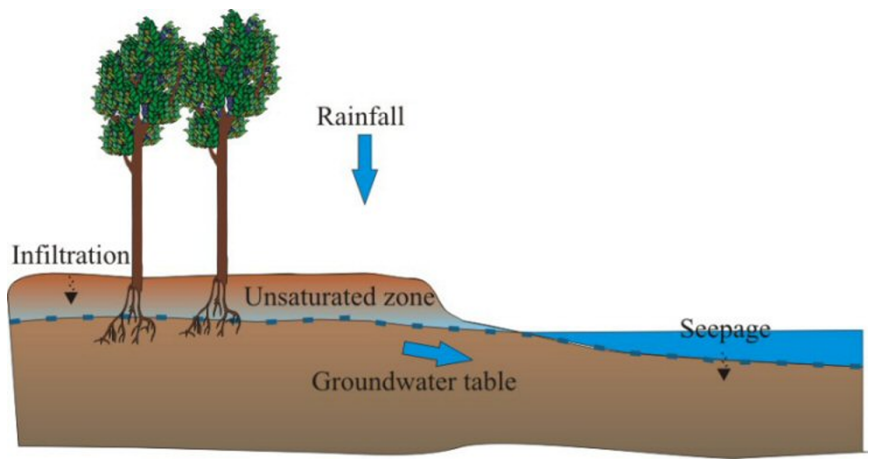


Figure 2.1: Illustration of the infiltration process in unsaturated porous media [1].

Richards' equation is a widely used PDE that describes the liquid flow in an unsaturated porous homogeneous rigid soil [80, 81, 82, 83], which is crucial for numerous environmental, agricultural, and hydrological applications. Physically, it models how water infiltrates and redistributes within the vadose zone—the partially saturated region between the land surface and groundwater table. An illustration of said process is shown in Figure 2.1. Accurate modeling of unsaturated flow is essential for predicting soil moisture distribution, optimizing irrigation strategies, managing groundwater resources, and preventing groundwater contamination from pollutants.

In agricultural settings, Richards' equation helps determine optimal irrigation schedules by predicting how water moves through soil profiles, thereby maximizing water efficiency and crop yields [84]. Environmentally, it aids in understanding infiltration processes, crucial for predicting groundwater recharge rates and assessing the potential for contaminant transport from the surface to aquifers [85]. Moreover, accurate simulations using Richards' equation are important for

mitigating risks associated with slope instability and landslides, which can occur due to changes in moisture content within hillslopes [86]. Due to its comprehensive representation of complex, nonlinear soil-water interactions, Richards' equation remains a cornerstone tool for researchers and practitioners dealing with subsurface hydrology and environmental engineering challenges.

The equation is based on Darcy's law, which describes the fluid flux as follows

$$q(\theta) = -K(\theta)\left(\frac{\partial \psi(\theta)}{\partial z} + 1\right), \tag{2.1}$$

where $\theta(z,t)$ is the volumetric water content (VWC) [-], $q(\theta)$ is the volume flux [LT^{-1}], $\psi(\theta)$ is the hydraulic head [L], $K(\theta)$ is the empirical unsaturated hydraulic conductivity function [LT^{-1}], and z is the depth in vertical direction [L]. Richards' equation is obtained by substituting (2.1) into the law of mass conservation for an incompressible porous medium, which yields

$$\frac{\partial \theta}{\partial t} = \frac{\partial}{\partial z}\left[K(\theta)\left(\frac{\partial \psi(\theta)}{\partial z} + 1\right)\right]. \tag{2.2}$$

Equation (2.2) is referred to as the mixed water content form of Richards' equation because the water content θ is the dependent variable and the capillary head ψ is a function of θ [80]. It is common to use the capillary head ψ as the dependent variable since it is continuous because the water content θ is typically not continuous in layered soils. Then the equation becomes

$$\frac{\partial \theta(\psi)}{\partial t} = \frac{\partial}{\partial z}\left[K(\psi)\left(\frac{\partial \psi}{\partial z} + 1\right)\right], \tag{2.3}$$

which is the mixed form of Richards' equation that ensures mass-conservation [87]. The Van

Genuchten–Mualem constitutive relationships between θ, K, and ψ are defined in [88] as

$$\begin{aligned} \theta(\psi) &= \frac{\theta_s - \theta_r}{(1 + (-\alpha\psi)^n)^m} + \theta_r, \\ K(\theta) &= K_s S_e^l \left(1 - \left(1 - S_e^{1/m}\right)^m\right)^2, \end{aligned} \tag{2.4}$$

where $m = 1 - \frac{1}{n}$, $S_e = \frac{\theta - \theta_r}{\theta_s - \theta_r}$. The relationship between θ and ψ is described by the water retention curve (WRC), and the relationship between K and θ is called the hydraulic conductivity function (HCF). The Van Genuchten-Mualem constitutive relationships are commonly used in physical modeling of the infiltration process.

Richards' equation can be solved under various types of boundary conditions, each representing different physical scenarios. The Dirichlet boundary condition, also known as the prescribed head condition, specifies the pressure head (or sometimes VWC) at the boundary. The Neumann boundary condition, or prescribed flux condition, specifies the water flux across the boundary. This is given by:

$$q = -K(h)\left(\frac{\partial h}{\partial x} + 1\right) = q_0(t), \quad \text{on } \partial\Omega_N$$

This condition is used to model evaporation from soil or infiltration through membranes. Other boundary conditions include zero-flux boundary condition, mixed boundary condition, and sign-dependent boundary condition, which models bedrock or sealed boundaries, semi-permeable surfaces, and rainfall infiltration with runoff, respectively. The boundary conditions we consider in Chapter 3 are Dirichlet conditions at the bottom depth and a time-dependent flux condition at the surface. Such boundary conditions are commonly used to model unsaturated soil columns with a constant head at the bottom and a time-dependent flux at the surface, such as rainfall or evaporation.

The nonlinear and potentially degenerate nature of Richards' equation leads to solutions with sharp wetting fronts from discontinuous influx, posing difficulties for traditional discretization schemes and neural network-based solvers alike.

2.1.2 Convection-Diffusion Equation

Singularly perturbed problems arise in PDEs when a small parameter $\epsilon \ll 1$ multiplies the highest derivative term. Unlike regular perturbation problems, singularly perturbed PDEs exhibit solutions that are not uniformly valid as $\epsilon \to 0$; setting $\epsilon = 0$ often leads to a qualitatively different equation or loss of boundary conditions. This mismatch gives rise to boundary or interior layers—narrow regions of rapid variation, requiring specialized numerical or asymptotic techniques for accurate resolution. Standard numerical methods often fail to resolve these sharp gradients unless extremely fine meshes are used.

The convection-diffusion equation is an example of singularly perturbed problems. It arises in a variety of transport phenomena, including heat transfer and pollutant dispersion. The steady-state convection-diffusion equation is given by

$$-\epsilon \nabla^2 u + \vec{w} \cdot \nabla u = f, \quad x \in \Omega \subseteq \mathbb{R}^d \tag{2.5}$$

where $\epsilon > 0$, $\nabla^2 u$ corresponds to diffusion, ∇u represents convection, $\vec{w}$ denotes the advection velocity and f is the source term.

The behavior of the solution can be characterized by the Péclet number, defined as

$$\text{Pe} = \frac{wL}{\epsilon}, \tag{2.6}$$

where L is a characteristic length scale. For high Péclet numbers, i.e. when $\epsilon \to 0$, the solution exhibits boundary or internal layers with steep gradients. The Péclet number is also used in numerical schemes to determine the stability of numerical methods.

The specification of boundary conditions plays a critical role in determining the location and nature of steep gradient layers in the convection-diffusion equation. In particular, when Dirichlet boundary conditions are imposed, sharp boundary layers tend to form near the outflow boundary, where the solution rapidly transitions to match the specified value. This steep transition occurs because the convective transport dominates the diffusion (Pe $\gg 1$), preventing the solution from gradually adjusting over the domain.

Such boundary-induced layers are a well-known difficulty in numerical simulation. Uniform discretization schemes struggle to resolve the rapid variation within the narrow region, requiring either excessive mesh refinement or specialized methods such as layer-adapted meshes, upwinding, or stabilization techniques.

Mathematically, the thickness of boundary layers in convection-dominated regimes is often proportional to the small parameter ϵ for Dirichlet boundary conditions ($\sqrt{\epsilon}$ for Neumann boundary conditions). For example, in a one-dimensional steady-state convection-diffusion equation with constant coefficients, the analytical solution typically involves exponential terms of the form $u(x) \sim 1 - e^{-(1-x)/\epsilon}$, which rapidly decay over a narrow region near the outflow boundary. This implies that the boundary layer width is of order $\mathcal{O}(\epsilon)$. Consequently, resolving the solution accurately requires mesh spacing significantly smaller than ϵ in the layer region, highlighting the stiff nature of the problem.

Both Richards' equation and convection-diffusion equation serve as canonical examples of

PDEs with localized features that motivate the development of more adaptive and expressive solver frameworks. The following sections review conventional numerical methods and learning-based approaches used to approximate their solutions.

2.2 Classical Numerical Methods

Numerical solutions of PDEs are traditionally obtained by discretizing the underlying domain and approximating derivatives using algebraic expressions. Common approaches include the Finite Difference Method (FDM), Finite Element Method (FEM), and Finite Volume Method (FVM), each offering distinct advantages depending on the structure of the PDE and the geometry of the domain. Since traditional methods are not the focus of this dissertation; basics of these methods are briefly described in this section.

2.2.1 Finite Difference Method (FDM)

The FDM replaces derivatives with finite difference approximations on structured grids. For a scalar function $u(x, t)$, spatial derivatives are typically approximated using central or upwind schemes. For example, the second-order central difference for the second derivative is given by:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}, \tag{2.7}$$

while the first derivative can be approximated using:

$$\text{Central difference:} \quad \left.\frac{\partial u}{\partial x}\right|_{x_i} \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x}, \tag{2.8}$$

$$\text{Upwind difference:} \quad \left.\frac{\partial u}{\partial x}\right|_{x_i} \approx \frac{u_i - u_{i-1}}{\Delta x} \quad \text{(for positive advection).} \tag{2.9}$$

For time integration, implicit or explicit schemes such as forward Euler, backward Euler, and Crank–Nicolson are commonly used. For instance, the forward Euler method approximates the time derivative as:

$$\left.\frac{\partial u}{\partial t}\right|^{n} \approx \frac{u^{n+1} - u^{n}}{\Delta t}. \tag{2.10}$$

These schemes are combined to discretize PDEs in space and time, resulting in systems of algebraic equations that can be solved directly or iteratively.

2.2.2 Finite Element Method (FEM)

The FEM reformulates the PDE as a weak (variational) problem. Given a domain Ω and test space V, the goal is to find $u \in V$ such that:

$$a(u, v) = f(v) \quad \forall v \in V, \tag{2.11}$$

where $a(\cdot, \cdot)$ is a bilinear form derived from the PDE, and $f(\cdot)$ is a linear functional representing forcing terms and boundary conditions.

The solution space is approximated by a finite-dimensional subspace $V_h \subset V$, typically

spanned by piecewise polynomial basis functions. This leads to a sparse linear system:

$$\mathbf{A}\mathbf{u} = \mathbf{f}, \tag{2.12}$$

where $\mathbf{A}$ is the stiffness matrix and $\mathbf{f}$ is the load vector.

FEM is well-suited for complex geometries and offers systematic convergence analysis but requires mesh generation and assembly of variational forms.

2.2.3 Finite Volume Method (FVM)

The FVM is based on the integral form of conservation laws. The computational domain is partitioned into control volumes, and the governing PDE is integrated over each volume:

$$\int_{V_i} \frac{\partial u}{\partial t}\, dx + \int_{\partial V_i} \mathbf{F}(u) \cdot \mathbf{n}\, ds = \int_{V_i} S(u)\, dx, \tag{2.13}$$

where $\mathbf{F}(u)$ is the flux and $S(u)$ is a source term. Numerical flux functions are used to approximate fluxes across boundaries, preserving local conservation.

FVM is widely used in fluid dynamics and transport due to its conservative properties, but implementation complexity increases for unstructured meshes.

2.2.4 Challenges in Resolving Steep Gradients

Problems like the Richards' equation and the convection-diffusion equation with high Péclet number give rise to solutions with steep gradients or narrow layers. These features demand fine spatial resolution to maintain accuracy and stability. For example, in convection-diffusion

equation, central difference schemes, though simple and widely used, often introduce nonphysical oscillations near steep gradients due to lack of numerical damping. Stable methods like upwind differencing may excessively smear the solution, leading to loss of accuracy in capturing sharp boundary layers. Finite element methods can also suffer from spurious oscillations unless stabilized formulations such as streamline upwind/Petrov–Galerkin (SUPG) or discontinuous Galerkin (DG) methods are used [89]. Furthermore, resolving narrow layers with conventional meshing requires extremely fine grids, significantly increasing computational cost. In higher dimensions or complex geometries, generating suitable meshes and maintaining stability becomes even more challenging. These issues underscore the need for more adaptive or data-driven approaches that can represent steep features without relying on uniform grid refinement. In Richards' equation, nonlinearities in $\theta(h)$ and $K(h)$ exacerbate stiffness, requiring adaptive strategies or very small time steps. Computational costs quickly increase as a result.

For the conveciton-diffusion equation, in numerical computations, a quantity known as the numerical Péclet number provides a local measure of the ratio between convective and diffusive effects at the grid or element level. It is typically defined as

$$\mathrm{Pe}_h = \frac{\|\vec{w}\| h}{2\epsilon}, \tag{2.14}$$

where h denotes the local mesh size and $\|\vec{w}\|$ is the magnitude of the velocity field. When $\mathrm{Pe}_h \gg 1$, numerical schemes such as central differencing become unstable or inaccurate, requiring stabilization techniques like upwinding or artificial diffusion. Conversely, for $\mathrm{Pe}_h \ll 1$, diffusion dominates and central schemes typically perform well. The numerical Péclet number thus guides the selection of discretization strategies to balance accuracy and stability. We use Pe_h as an

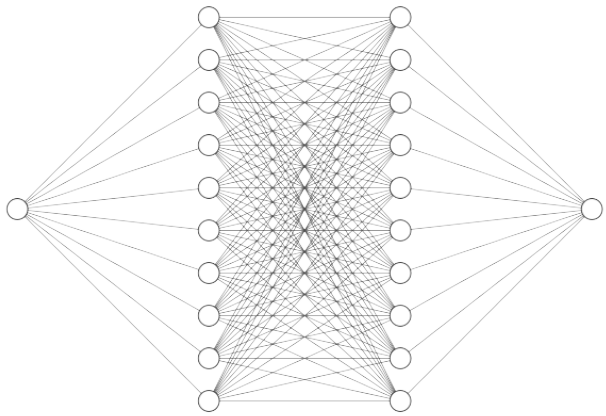


Figure 2.2: Image of a fully-connected neural network.

indicator for how accurate approximations generated using traditional methods are in Chapter 5.

2.3 Neural Networks and Physics-Informed Neural Networks

Neural networks have emerged as powerful tools for approximating complex, high-dimensional functions, and they form the computational backbone of many machine learning and scientific computing applications. Among various architectures, dense neural networks (DNNs) [90], also known as fully-connected neural networks, are the most basic and widely used. Their simple structure makes them a natural starting point for function approximation tasks, including those that arise in solving PDEs. In this section, we introduce the key components, mathematical formulation, and theoretical foundations of DNNs, setting the stage for their role in physics-informed learning frameworks.

2.3.1 Dense Neural Networks (DNNs)

A DNN consists of an input layer, multiple hidden layers, and an output layer. Each layer processes information by applying a linear transformation followed by a nonlinear activation $\sigma(\cdot)$.

Specifically, if the input to layer l is denoted Y_l, then the output is given by:

$$Y_{l+1} = \sigma(Y_l W_l + b_l), \tag{2.15}$$

where $W_l \in \mathbb{R}^{d_l \times d_{l+1}}$ is the weight matrix, $b_l \in \mathbb{R}^{1 \times d_{l+1}}$ is the bias vector, and $\sigma(\cdot)$ is a nonlinear activation function applied element-wise.

Let $X_{\text{in}} \in \mathbb{R}^{N \times d_0}$ be a batch of N input vectors with dimension d_0. For a network with L hidden layers, the complete forward pass can be written recursively as:

$$\begin{aligned} Y_1 &= \sigma(X_{\text{in}} W_1 + \mathbb{1}_{\text{in}} b_1), \\ Y_{l+1} &= \sigma(Y_l W_l + \mathbb{1}_{\text{in}} b_l), \quad \text{for } l = 1, \ldots, L-1, \\ Y_{\text{out}} &= Y_L W_{L+1} + \mathbb{1}_{\text{in}} b_{L+1}, \end{aligned} \tag{2.16}$$

where $\mathbb{1}_{\text{in}} \in \mathbb{R}^{N \times 1}$ is a column vector of ones used to broadcast the bias term across all inputs.

The full set of trainable parameters $\boldsymbol{\theta}$ includes all weight matrices W_l and bias vectors b_l for $l = 1, \ldots, L+1$. The choice of activation function σ affects the expressiveness and differentiability of the network.

Several activation functions are commonly used in practice, each with distinct characteristics and applications. The Rectified Linear Unit (ReLU) is defined as $\text{ReLU}(x) = \max(0, x)$. ReLU is widely used due to its simplicity and effectiveness in deep networks. It avoids saturation for positive inputs and helps mitigate the vanishing gradient problem. However, its non-differentiability at zero and zero gradients for negative inputs can limit its applicability in physics-informed applications.

The sigmoid function, given by $\sigma(x) = \frac{1}{1+e^{-x}}$, is smooth and bounded in the range $(0, 1)$, making it particularly useful for probability outputs. Despite its advantages, the sigmoid function

suffers from vanishing gradients for large input magnitudes, which can slow down the training process.

Another commonly used activation function is the hyperbolic tangent ($\tanh$), defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Like the sigmoid function, $\tanh$ is smooth and differentiable, but it maps inputs to the interval $(-1, 1)$. This property, combined with its stronger gradients near zero, often leads to faster convergence during training compared to sigmoid.

In the context of solving partial differential equations with neural networks, the choice of activation function is particularly important. Since the learning process often involves computing higher-order derivatives of the network output with respect to input variables, smooth and differentiable activations such as $\tanh$ or sigmoid are typically preferred over piecewise-linear functions like ReLU. These choices help ensure accurate gradients when using automatic differentiation.

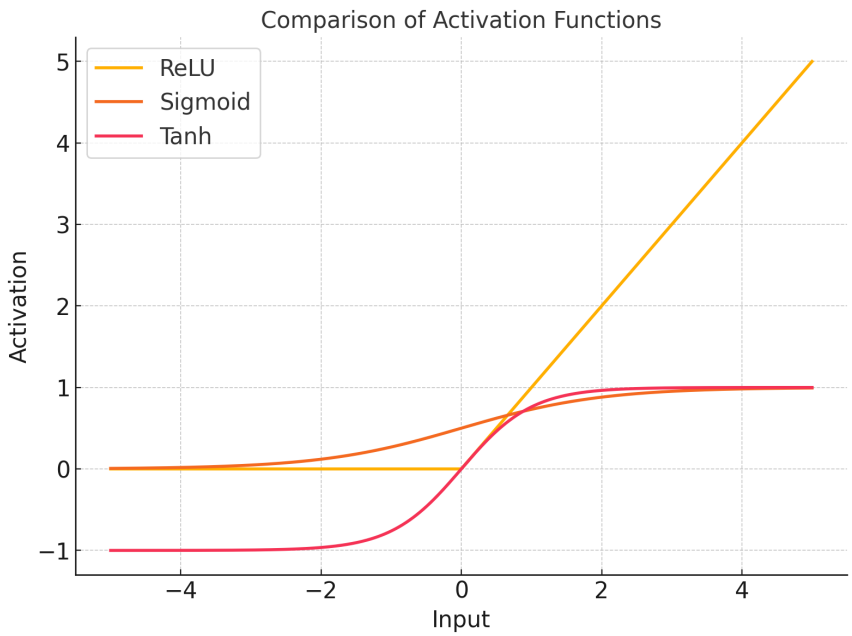


Figure 2.3: Comparison of common activation functions: ReLU, Sigmoid, and Tanh.

Figure 2.3 illustrates the qualitative differences between the ReLU, sigmoid, and $\tanh$ functions. Smooth activations such as $\tanh$ enable continuous and differentiable function approxi-

mations, which are critical in physics-informed settings.

Proper initialization of network weights plays a crucial role in ensuring stable and efficient training of neural networks. One commonly used method is the *Xavier initialization* (also known as Glorot initialization) [91], which aims to maintain the variance of activations across layers and prevent gradients from vanishing or exploding during backpropagation.

In Xavier initialization, the weights W_l of the l-th layer are drawn from a distribution with zero mean and variance:

$$\text{Var}(W_l) = \frac{2}{d_{l-1} + d_l},$$

where d_{l-1} and d_l are the input and output dimensions of the layer, respectively. In practice, the entries of W_l are often sampled from a normal distribution:

$$W_l \sim \mathcal{N}\left(0, \frac{2}{d_{l-1} + d_l}\right),$$

or a uniform distribution with similar variance.

Xavier initialization is particularly effective when using symmetric activation functions such as $\tanh$, as it helps keep the signal from shrinking or exploding as it propagates through the network. In the experiments presented in this work, all networks are initialized using the Xavier scheme unless stated otherwise. Bias terms b_l are initialized to zero.

In the context of PINNs, this initialization strategy helps maintain numerical stability when computing PDE residuals and their derivatives using automatic differentiation. Because PINNs rely on minimizing a composite loss function that includes PDE residuals, boundary conditions, and initial conditions, poor initialization can lead to imbalanced gradients or stagnation during training.

Xavier initialization provides a balanced starting point that facilitates effective optimization, especially when smooth activation functions like $\tanh$ are used.

2.3.2 Universal Approximation and Generalization

By the Universal Approximation Theorem [23], a DNN with two hidden layers of infinite width is capable of approximating any function. In actuality, infinite width is unachievable. Networks with more hidden layers of large width are used as surrogate functions for approximations.

The Universal Approximation Theorem (UAT) is a fundamental result in the theory of neural networks, establishing that a feedforward neural network with at least one hidden layer and a non-linear activation function can approximate any continuous function to an arbitrary degree of accuracy, provided it has a sufficient number of neurons. This theorem, originally proven by Cybenko [22] and further extended by Hornik [23], highlights the immense expressive power of neural networks in function approximation.

Mathematically, the UAT states that for any k-times continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ and any arbitrarily small error tolerance $\delta > 0$, there exists a neural network function $\hat{f}$ of a finite number of neurons using any non-polynomial, bounded and measurable activation function such that $\hat{f}(x)$ approximates f and all its partial derivatives up to order k on any compact subset of $\mathbb{R}^n$ with uniform accuracy less than δ.

This implies that neural networks can approximate any function with arbitrary precision, making them particularly well-suited for solving PDEs where complex relationships between input and output variables exist. The original versions of the theorem focused on shallow networks with a single hidden layer, but later studies demonstrated that deep networks can achieve the

same approximation accuracy with significantly fewer neurons per layer, leveraging hierarchical feature learning [92]. For PDEs, the UAT justifies the use of deep neural networks as solution approximators.

2.3.3 Physics-Informed Neural Networks (PINNs)

PINNs are a class of deep learning methods designed to approximate solutions of PDEs by embedding the governing physical laws directly into the loss function of a neural network [43]. Unlike traditional supervised learning models that rely solely on labeled input-output pairs, PINNs incorporate the underlying physics as soft constraints, enabling the network to learn solutions from sparse data—or even in the absence of data—by leveraging knowledge of the differential equation itself.

To formally define PINNs, consider a general PDE of the form:

$$\begin{aligned} \partial_t u(t,x) + \mathcal{N}[u](t,x) &= f(t,x), \quad (t,x) \in [0,T] \times \Omega, \\ \mathcal{G}[u](t,x) &= g(t,x), \quad (t,x) \in [0,T] \times \partial\Omega, \\ u(0,x) &= h(x), \quad x \in \Omega, \end{aligned} \tag{2.17}$$

where $\mathcal{N}$ and $\mathcal{G}$ are (possibly nonlinear) differential operators, and $u(t,x) : \bar{\Omega} \to \mathbb{R}$ is the solution defined over the spatial domain $\Omega \subset \mathbb{R}^{d_0}$ and time interval $[0,T]$.

Define the PDE residual as:

$$r(t,x) = \partial_t u(t,x) + \mathcal{N}[u](t,x) - f(t,x). \tag{2.18}$$

In a PINN, the unknown solution $u(t, x)$ is approximated by a neural network $u_{\boldsymbol{\theta}}(t, x)$ with trainable parameters $\boldsymbol{\theta}$. By leveraging automatic differentiation [93], one can compute the required derivatives of $u_{\boldsymbol{\theta}}$ with respect to t and x, and evaluate the residual $r_{\boldsymbol{\theta}}(t, x)$ at any point in the domain—even where no data is provided.

The network is trained to minimize a composite loss function that penalizes deviations from both the physical laws and the boundary/initial conditions. This loss function is typically defined as:

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_{\text{res}}\mathcal{L}_{\text{res}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}}, \tag{2.19}$$

where:

$$\mathcal{L}_{\text{res}}(\boldsymbol{\theta}) = \frac{1}{N_r}\sum_{i=1}^{N_r}\left(r_{\boldsymbol{\theta}}(t_r^{(i)}, x_r^{(i)})\right)^2, \tag{2.20}$$

$$\mathcal{L}_{\text{BC}}(\boldsymbol{\theta}) = \frac{1}{N_b}\sum_{i=1}^{N_b}\left(\mathcal{G}[u_{\boldsymbol{\theta}}](t_b^{(i)}, x_b^{(i)}) - g(t_b^{(i)}, x_b^{(i)})\right)^2, \tag{2.21}$$

$$\mathcal{L}_{\text{IC}}(\boldsymbol{\theta}) = \frac{1}{N_0}\sum_{i=1}^{N_0}\left(u_{\boldsymbol{\theta}}(0, x_0^{(i)}) - h(x_0^{(i)})\right)^2. \tag{2.22}$$

Here:

- $\{(t_r^{(i)}, x_r^{(i)})\}_{i=1}^{N_r}$ are sample points in the interior of the domain,
- $\{(t_b^{(i)}, x_b^{(i)})\}_{i=1}^{N_b}$ are sampled on the spatial boundary $\partial\Omega$,
- $\{x_0^{(i)}\}_{i=1}^{N_0}$ are points in Ω at initial time $t = 0$.

This framework allows PINNs to learn physically consistent solutions by integrating domain knowledge directly into the training process. In this work, we use PINNs to approximate solutions

to nonlinear PDEs such as Richards' equation and the convection-diffusion equation, where sharp gradients and strong nonlinearities pose challenges for conventional numerical methods.

2.3.4 Training and Optimization

The parameters θ are optimized to minimize the total loss $\mathcal{L}(\theta)$ using gradient-based optimizers. A common training strategy involves first using Adam [94], a stochastic first-order method, followed by L-BFGS [95], a quasi-Newton method with fast convergence in the later stages of training. Usually the training process is divided into two phases: the first phase uses Adam to quickly reduce the loss, while the second phase switches to L-BFGS for fine-tuning. This two-phase approach helps balance exploration and exploitation in the optimization landscape.

2.4 Related Cooridinate-Based Enhancements on PINNs

To improve the ability of PINNs to resolve steep gradients and internal layers, recent work has focused on augmenting the input domain through coordinate transformations. The underlying idea is to redistribute spatial resolution toward regions with large variations in the solution, thereby enabling the neural network to better represent high-frequency or localized features.

In this section, we introduce three PINN architectures improvements that are based on coordinate transformationswith three alternative architectures: Multi-Scale PINNs (MS-PINNs) [96], Two-Scale PINNs (TS-PINNs) [97], and Fourier-Feature-Embeded PINNs (FFE-PINNs) [98]. These architectures are designed to alleviate issues associated with spectral bias, and has a similar structure as our input-transformed PINN proposed in Chapter 4. We discuss their structures, mathematical formulations, and how they address challenges in modeling steep gradients and

boundary layers.

MS-PINNs are designed to capture features at various scales by incorporating hierarchical structures within the network. A notable example is the multiscale neural network based on hierarchical matrices ($\mathcal{H}$-matrices) [96]. The architecture leverages the hierarchical decomposition of matrices to efficiently represent interactions at multiple scales.

Mathematically, the application of an $\mathcal{H}$-matrix to a vector $\mathbf{v}$ can be interpreted as:

$$\mathbf{u} = \mathcal{H}\mathbf{v}, \tag{2.23}$$

where $\mathcal{H}$ represents the hierarchical matrix capturing multi-scale interactions [96].

While these networks effectively handle multi-scale features through input stretching mechanisms, they often lack explicit input shifting strategies. Consequently, they may struggle with accurately capturing steep boundary layers, potentially leading to incorrect flow solutions.

TS-PINNs are specifically designed to address problems with distinct scales by incorporating separate components or subnetworks dedicated to capturing coarse and fine features. This architecture is particularly useful for solving PDEs with small parameters, as demonstrated in [97].

The TS-PINNs can be expressed as:

$$u(x) \approx u_{\text{NN}}\left(x, \frac{x - x_c}{\epsilon}; \theta\right), \tag{2.24}$$

where u_{NN} is a neural network parameterized by θ, x_c is the midpoint of the domain, and ϵ is a small parameter indicating the presence of multiple scales [97].

This approach can be viewed as a reduced version of MS-PINN, where it only handles two scales of the input: x and $\frac{1}{\epsilon}x$. The performance is similar to MS-PINN while requiring fewer number of parameters.

FFE-PINNs are an extension of standard physics-informed neural networks that aim to improve the network's ability to approximate high-frequency or multi-scale solutions. The key idea is to transform the input coordinates using a randomized Fourier feature embedding before feeding them into the neural network, thereby enhancing the representation capacity of the model [98].

Let $u_{\boldsymbol{\theta}}(x)$ be the neural network approximation of the PDE solution, where $x \in \mathbb{R}^d$ represents the input coordinates. In FFE-PINNs, the network is modified to operate on a transformed input:

$$u_{\boldsymbol{\theta}}(x) := \text{NN}_{\boldsymbol{\theta}}(\gamma(x)),$$

where the Fourier feature mapping $\gamma(x)$ is defined as:

$$\gamma(x) = [\sin(2\pi Bx), \cos(2\pi Bx)] \,.$$

In this approach, $B \in \mathbb{R}^{m \times d}$ is a trainable matrix whose entries are sampled from a Gaussian distribution, $B_{ij} \sim \mathcal{N}(0, \sigma^2)$, where m represents the number of Fourier modes used in the embedding. The transformed feature vector $\gamma(x) \in \mathbb{R}^{2m}$ is then passed through a standard dense neural network $\text{NN}_{\boldsymbol{\theta}}$, parameterized by $\boldsymbol{\theta}$. This transformation enhances the network's ability to capture high-frequency components and localized features in the solution space.

This input transformation allows the network to more easily capture oscillatory or localized

features in the solution, which are often difficult to resolve using standard coordinate representations. However, similar to MS-PINNs, FFE-PINNs do not explicitly incorporate input shifts, which are crucial for aligning the network's sensitivity with the location of steep gradients or boundary layers. As a result, while F-PINNs can improve the representation of high-frequency components, they may still struggle to accurately capture steep gradients in singularly perturbed problems.

2.5 Summary and Outlook

This chapter has presented the mathematical and computational foundations necessary for understanding the limitations of traditional numerical solvers and the motivation behind using neural network-based approaches for PDE approximation. Two representative PDEs were introduced: Richards' equation, which models unsaturated flow in porous media, and the convection-diffusion equation, which captures transport phenomena with sharp layers under high Péclet regimes. Both equations exhibit localized steep gradients that are difficult to resolve using standard finite difference, finite element, or finite volume methods without fine meshes or prior knowledge of gradient locations.

We then introduced the architecture of feedforward neural networks, emphasizing design considerations critical for PDE approximation, such as smooth activation functions, weight initialization, and universal approximation capabilities. Building upon these foundations, PINNs were presented as a mesh-free framework that enforces physical constraints through the loss function. While PINNs offer appealing flexibility, they face well-documented challenges when dealing with solutions that contain sharp features or are governed by singular perturbations.

To address these challenges, recent developments have focused on architectural enhancements such as domain decomposition, input transformation, and multi-scale embedding. We discussed several such methods—MS-PINNs, TS-PINNs, and FFE-PINNs—that redistribute resolution or extend expressiveness to improve performance in challenging regimes. However, each of these approaches relies on some level of a priori knowledge or introduces added complexity, such as larger network size or carefully tuned embeddings.

The insights and tools introduced in this chapter set the stage for the novel contributions in the following chapters. We aim to build upon the strengths of existing approaches while addressing their limitations, introducing architectures and training strategies that improve PINN performance on problems with unknown gradient locations and strong multiscale behavior, without requiring explicit domain knowledge or handcrafted transformations.

Chapter 3: PINNs for Richards' Equation

3.1 Introduction

In Chapter 2, we introduced Physics-Informed Neural Networks (PINNs) as a mesh-free framework for approximating solutions to partial differential equations (PDEs), particularly in cases where data are limited or the solution exhibits complex dynamics. A key challenge arises when applying PINNs to time-dependent nonlinear PDEs, such as Richards' Equation, which governs water movement in unsaturated soils. Under realistic precipitation scenarios, the surface flux boundary condition can be highly discontinuous and variable, posing difficulties for traditional numerical solvers and standard PINNs alike. In particular, standard PINNs tend to violate causality in forward-time predictions and struggle to maintain physical consistency near flux discontinuities.

This chapter addresses a specific instance of this broader challenge: the prediction of volumetric water content (VWC) in unsaturated soils under discontinuous surface flux conditions using improved PINNs. We aim to develop a modified PINN framework that incorporates a flux-aware architecture and a causality-aware loss function to enhance prediction accuracy and physical fidelity. This work builds on the foundations laid in the previous chapter by extending PINNs to handle discontinuities in boundary conditions and ensuring temporally consistent predictions, which are critical in hydrological modeling.

The key contributions of this chapter include the development of a modified PINN structure

that generates steep gradients where the flux is discontinuous, thereby improving the model's ability to capture rapid changes in VWC. We also introduce a novel loss function that explicitly enforces causality through the incorporation of a discrete residual term, ensuring temporally consistent predictions. Finally, we comprehensively evaluate the proposed method under varying precipitation scenarios and soil types, demonstrating its effectiveness in achieving accurate forward-in-time predictions. This work is adapted from publication [99].

The remainder of this chapter is structured as follows: Section 3.2 reviews related work on PINNs and monotonic neural networks for solving Richards' Equation. Section 3.2 introduces our improved PINN formulation, including network architecture, training strategies, and loss design. Section 3.4 presents numerical experiments and validation results across different soil types and flux conditions. Finally, Section 3.5 summarizes key insights, limitations, and avenues for future research.

3.2 Problem Statement and Background

The central problem addressed in this chapter is the forward-time prediction of VWC $\theta(z,t)$ in unsaturated soils, where the dynamics are governed by the nonlinear Richards' Equation. We consider the one-dimensional vertical form of Richards' equation for unsaturated flow in soils:

$$\frac{\partial \theta(\psi)}{\partial t} = \frac{\partial}{\partial z}\left[K(\psi)\left(\frac{\partial \psi}{\partial z} + 1\right)\right], \tag{3.1}$$

where $\theta(z,t)$ is the volumetric water content [–], $\psi(z,t)$ is the pressure head $[L]$, $K(\psi)$ is the hydraulic conductivity function $[L, T^{-1}]$, and z denotes depth $[L]$.

Our goal is to construct a PINN, denoted $u_{\boldsymbol{\theta}}(z,t)$, that approximates $\theta(z,t)$ throughout a

spatiotemporal domain. This approximation leverages sparse measurements of $\theta(z,t)$ at fixed spatial depths, a known initial condition $\theta(z,0)$, a flux boundary condition at the soil surface $q(0,t)$ that may be discontinuous in time, and a constant pressure head boundary condition at the bottom of the domain. The network is trained by minimizing a loss function that includes both data fidelity and physics-based residual terms derived from Richards' equation.

However, this task is especially challenging due to two major factors. First, the flux boundary condition $q(0,t)$ may be discontinuous in time, such as when modeling precipitation or evaporation events as step functions. This introduces steep gradients in ψ and θ near the soil surface, making accurate predictions difficult. Second, standard PINN formulations lack mechanisms to enforce temporal causality. As a result, solutions for $\theta(z,t)$ may fail to depend on prior time states, violating the evolution implied by Richards' Equation.

To address these issues, we propose enhancements to the existing PINN framework for solving Richards'equation, including architectural change in including flux as an input and a modified loss function that enforces temporal consistency through discrete residuals. Next we will discuss the existing PINN architecture and its limitations in detail.

3.2.1 Monotonic Neural Networks

First, we introduce Monotonic Neural Networks (MNNs) and their significance in approximating monotonic functions. Monotonicity is a crucial property in many applications, particularly in soil hydrology, where the relationships between hydraulic head, VWC, and hydraulic conductivity are inherently monotonic.

Dense neural networks (DNNs) are universal approximators, meaning they can approximate

any continuous function to arbitrary accuracy given sufficient data and network capacity. However, DNNs do not guarantee that the approximated function will be monotonic, even if the target function is monotonic. This can lead to physically unrealistic predictions in applications where monotonicity is a requirement.

When a DNN is initialized with nonnegative weights, the network is called a monotonic neural network (MNN). It was proven in [100] that networks of this type guarantee monotonicity between inputs and outputs. Such a property can be utilized when approximating solutions to the Richards' equation because the relationships between the hydraulic head. The relations between hydraulic head ψ and conductivity K, ψ and VWC θ are monotonically increasing, as seen in (2.4). It is appropriate to use MNNs to represent functions between ψ and K, ψ and θ, respectively. Thus, employing MNNs to represent these functions naturally enforces physically consistent behavior, enhancing model stability and predictive accuracy.

3.2.2 Limitations of Existing PINNs for Richards' Equation

Bandai and Ghezzehei [101] introduced a PINN architecture for approximating Richards' equation solutions based on sparse VWC measurements. Their model employed separate neural networks for the pressure head $\hat{\psi}(z,t)$, volumetric water content $\hat{\theta}$, and hydraulic conductivity $\hat{K}$, with the latter two conditioned on the log-transformed pressure head $\hat{\psi}_{\log} = -\ln(-\hat{\psi})$.

Specifically, the PINN architecture is set up in the followin way: let $\hat{\psi}$, $\hat{\theta}$ and $\hat{K}$ be DNNs that approximate ψ, θ and K, respectively. The capillary head $\hat{\psi}$ is constructed first. Let $\hat{\psi}(z,t)$ be a DNN that takes in coordinate pairs of depth z and time t, and outputs approximations to $\psi(z,t)$. Due to the monotonic relationships between ψ and θ, K, MNNs are used for $\hat{\theta}$ and $\hat{K}$.

The constitutive relationships in Eq. (2.4) become simpler if ψ is in logarithmic scale. Thus, one can take $\hat{\psi}$ and convert it into logarithmic scale by $\hat{\psi}_{\log} = -\ln(-\hat{\psi})$. Then $\hat{\theta}(\hat{\psi}_{\log})$ and $\hat{K}(\hat{\psi}_{\log})$ take $\hat{\psi}_{\log}$ as the input and output approximations to $\theta(\psi)$ and $K(\psi)$, respectively.

Using automatic differentiation, we can obtain $\frac{\partial \hat{\psi}}{\partial z}$ and $\frac{\partial \hat{\theta}}{\partial t}$. Then we can calculate the approximated Darcian flux and residual:

$$\hat{q} = -\hat{K}(\frac{\partial \hat{\psi}}{\partial z} + 1),$$
$$\hat{r} = \frac{\partial \hat{\theta}}{\partial t} + \frac{\partial \hat{q}}{\partial z}.$$

Let $\{\mathbf{W}_\psi, \mathbf{b}_\psi\}$, $\{\mathbf{W}_\theta, \mathbf{b}_\theta\}$ and $\{\mathbf{W}_K, \mathbf{b}_K\}$ be the collections of weights and biases of $\hat{\psi}$, $\hat{\theta}$ and $\hat{K}$, respectively. Let $\{\mathbf{W}, \mathbf{b}\}$ be the collection of all weights and biases, i.e., $\mathbf{W} = \{\mathbf{W}_\psi, \mathbf{W}_\theta, \mathbf{W}_K\}$ and $\mathbf{b} = \{\mathbf{b}_\psi, \mathbf{b}_\theta, \mathbf{b}_K\}$. Then the loss function of the network in [101] is defined as

$$\mathcal{L}(\mathbf{W}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{\theta}^{(i)} - \theta^{(i)})^2 + \frac{1}{N}\sum_{i=1}^{N}(\hat{r}^{(i)})^2. \tag{3.2}$$

where $\{t^{(i)}, z^{(i)}, \theta^{(i)}\}_{i=1}^{N}$ is a set of volumetric water content time-series data measured at fixed different depths in soils, where N is the number of measurement data.

However, their initial loss function in Eq. (3.2) did not explicitly account for flux boundary conditions. As a consequence, this formulation failed to accurately predict VWC at time steps for which no observational data were available during training.

To address these shortcomings, Bandai and Ghezzehei later introduced an improved architecture where Van Genuchten-Mualem functions are used in place of $\hat{\theta}$ and $\hat{K}$ networks, and an improved loss function in [102]. Their refined loss function explicitly included boundary

conditions and weighting schedules:

$$
\begin{aligned}
\mathcal{L}(\mathbf{W}, \mathbf{b}) = & \frac{W_\theta}{N_\theta} \sum_{i=1}^{N_\theta} (\hat{\theta}^{(i)} - \theta^{(i)})^2 + \frac{W_r}{N_r} \sum_{i=1}^{N_r} (\hat{r}^{(i)})^2 \\
& + \frac{W_\psi}{N_\psi} \sum_{i=1}^{N_\psi} (\hat{\psi}^{(i)} - \psi^{(i)})^2 + \frac{W_q}{N_q} \sum_{i=1}^{N_q} (\hat{q}^{(i)} - q^{(i)})^2 \\
& + \frac{W_{ic}}{N_{ic}} \sum_{i=1}^{N_{ic}} (\hat{\theta}^{(i)}(t=0) - \theta_{ic}^{(i)})^2.
\end{aligned}
\tag{3.3}
$$

While this improved performance under steady boundary conditions, two significant limitations persisted. First, the use of Van Genuchten–Mualem functions frequently resulted in unstable training processes, often causing the loss function to diverge and making the network difficult to train effectively. Second, the model struggled to generalize under discontinuous or rapidly varying flux boundary conditions, which are common in realistic precipitation scenarios. These limitations highlight the need for further enhancements to the PINN framework to ensure stability and robustness in diverse and challenging conditions. This motivates the development of enhanced architectures presented in the following sections.

3.3 Improved Physics-Informed Neural Network Formulation

The limitations discussed in the previous section—namely, the inability of existing PINNs to handle discontinuous flux boundary conditions or preserve temporal causality—motivate the development of an improved framework. In what follows, we introduce a modified PINN architecture tailored to the Richards' Equation setting. This includes both structural and algorithmic enhancements: (i) incorporating surface flux as an explicit input to the network and (ii) modifying the loss function to capture temporal dynamics more faithfully. We begin by describing the

architecture and training configuration, followed by numerical experiments that highlight the strengths and limitations of this design under realistic soil moisture scenarios.

3.3.1 Flux as an Input

In this section, we introduce our modified PINN framework, extending the model architecture from [101] to improve performance under highly variable flux boundary conditions.

In our initial attempt, we found that including the surface flux $q(0,t)$ as a third input to the network—alongside depth z and time t—significantly improves the model's ability to approximate sharp transitions in $\theta(z,t)$. Large-magnitude fluxes can induce steep gradients in the pressure head and VWC near the surface. When the flux is encoded explicitly as an input, the network can learn to associate high-gradient zones with flux magnitude more effectively, resulting in better localized approximation capacity. A more detailed theoretical justification for this phenomenon is presented in Chapter 4, where the impact of input-driven gradients on network sensitivity is analyzed.

Following [101], we adopt the same network widths and depths: $\hat{\psi}(z,t,q)$ is represented by a DNN with 6 hidden layers of width 40; $\hat{\theta}(\hat{\psi})$ is modeled by a MNN with 1 hidden layer of width 40; and $\hat{K}(\hat{\psi})$ is an MNN with 3 hidden layers of the same width.

To address the shortcomings of the original loss function, which lacked explicit enforcement of physical quantities like hydraulic head and flux, we define an enhanced loss function:

$$\begin{aligned}\mathcal{L}(\mathbf{W},\mathbf{b}) = &\frac{W_\theta}{N_\theta}\sum_{i=1}^{N_\theta}(\hat{\theta}^{(i)} - \theta^{(i)})^2 + \frac{W_r}{N_r}\sum_{i=1}^{N_r}((\hat{r}^{(i)})^2) \\ &+ \frac{W_\psi}{N_\psi}\sum_{i=1}^{N_\psi}(\hat{\psi}^{(i)} - \psi^{(i)})^2 + \frac{W_q}{N_q}\sum_{i=1}^{N_q}(\hat{q}^{(i)} - q^{(i)})^2,\end{aligned} \tag{3.4}$$

where the additional terms for $\hat{\psi}$ and $\hat{q}$ promote adherence to physical state and flux conditions.

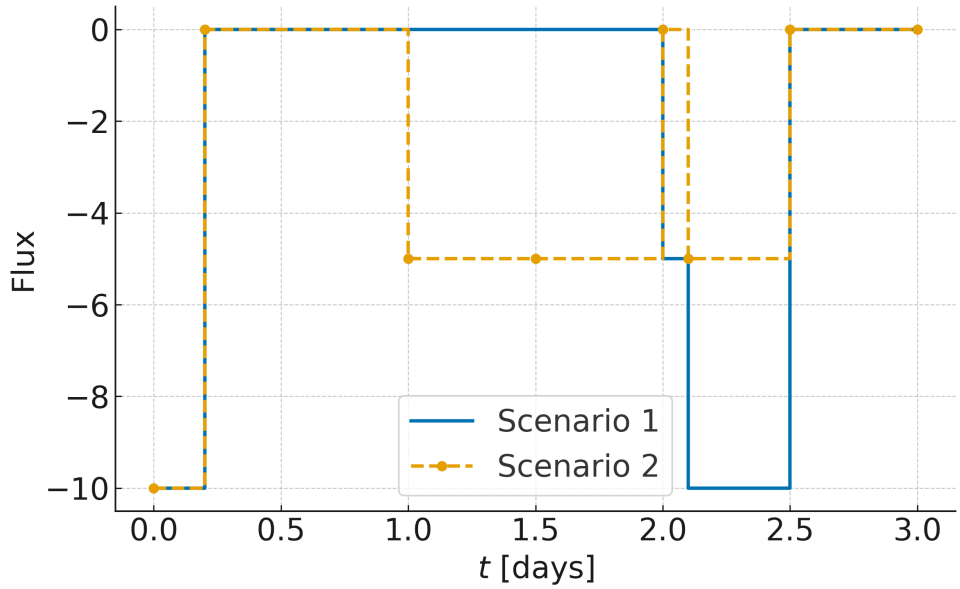


Figure 3.1: The surface flux for $t \in [0, 3]$ days.

This loss is inspired by the formulation in [102] but adapted for use with Van Genuchten–Mualem soil models and non-constant flux inputs.

To evaluate the effectiveness of our previously suggested architecture, we simulate the dynamics of unsaturated flow in a homogeneous sandy loam soil. The simulation domain spans $z \in [-100, 0]$ cm (depth below the surface), and the temporal domain covers $t \in [0, 3]$ days. The hydraulic and retention properties follow the Van Genuchten–Mualem model (Eq. (2.4)) with parameters: $\theta_r = 0.065$, $\theta_s = 0.41$, $\alpha = 0.075$, $n = 1.89$, $K_s = 106.1$ cm/day, and $l = 0.5$.

To emulate realistic boundary conditions, the top surface is subjected to alternating precipitation and evaporation events via a step-function flux:

$$q(0, t) = q_{\text{top}}(t),$$

$$\psi(-100\,\text{cm}, t) = \psi_{\text{bottom}} = -1000\,\text{cm}.$$

Table 3.1: The relative error of $\hat{\psi}$, $\hat{\theta}$ and $\hat{K}$ of a network trained with loss function in Eq. (3.4) for near-surface depths and the entire depth column.

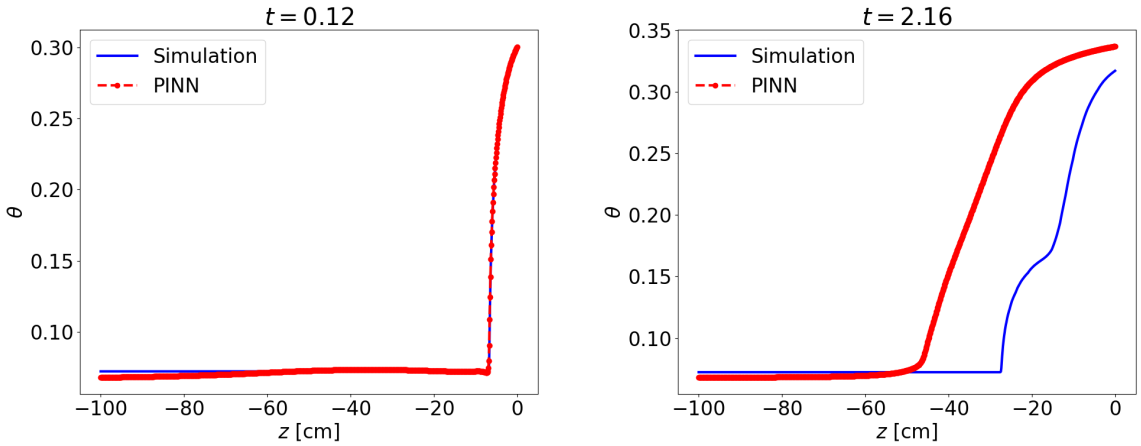
	Relative Error of (for $z \in [-20, 0]$)		
	$\hat{\theta}$	$\hat{\psi}$	$\hat{K}$
$t \in [0, 1.5)$	6.33614e-05	4.20471e-02	4.15087e-02
$t \in [1.5, 3]$	1.17356e-01	5.86184e-01	3.30945
	Relative Error of (for $z \in [-100, 0]$)		
	$\hat{\theta}$	$\hat{\psi}$	$\hat{K}$
$t \in [0, 1.5)$	2.11832e-03	1.66644e-02	4.28022e-02
$t \in [1.5, 3]$	1.37454e-01	6.93174e-02	3.94688

The surface flux profile $q_{\text{top}}(t)$ is shown in Fig. 3.1, designed to capture sharp transitions in infiltration and evaporation magnitudes.

We frame the prediction task as follows: VWC measurements are assumed to be available for the first half of the simulation period ($t \in [0, 1.5)$), and the network is trained to extrapolate predictions for the second half ($t \in [1.5, 3]$). Measurements are provided at 10 uniformly spaced depths: $z \in \{-1, -3, -5, -7, -9, -11, -13, -15, -17, -19\}$ cm. Prediction accuracy is evaluated primarily within the extrapolation window, with a particular focus on the near-surface region ($z \in [-20, 0]$), which is most sensitive to boundary flux variations and relevant for sensing applications.

The simulation data are generated using HYDRUS-1D [103], with a temporal step size $\Delta t = 0.012$ days and spatial discretization $\Delta z = 0.1$ cm. The PINN is trained on collocation points and boundary condition samples spanning the full domain.

We employ a two-stage optimization strategy consistent with [101]: an initial phase using the Adam optimizer [94] for 150,000 epochs, followed by 50,000 epochs of L-BFGS optimization [104]. Default parameter settings from TensorFlow [105] and SciPy [106] are used.



(a) (b)

Figure 3.2: Predicted VWC (red) and simulated VWC (blue) at an in-sample time $t = 0.12$ days 3.2a and out-of-sample time $t = 2.16$ days 3.2b using the loss function in Eq. (3.4).

The weighting coefficients in Eq. (3.4) are chosen to balance the influence of different loss components: $W_\theta = 100$, $W_\psi = 1$, $W_q = 10$, and W_r is scheduled to increase exponentially from 10^{-3} to 10^2 over 100,000 epochs, with updates every 500 epochs.

Relative prediction errors are reported in Table 3.1, and visualizations of network performance are presented in Fig. 3.2.

As shown in Fig. 3.2, the network produces accurate approximations at in-sample times (e.g., $t = 0.12$ days), closely matching the HYDRUS simulation. However, the performance deteriorates at out-of-sample times, particularly near flux discontinuities. At $t = 2.16$ days, a heavy precipitation event ($q = -10$) occurs, but the network incorrectly predicts that its effects penetrate deeply into the soil rather than remaining localized near the surface—an unphysical behavior under unsaturated conditions. Figure 3.3 illustrates the predicted VWC, simulated VWC, and absolute error of the current network setup. The network's predictions are accurate in the training interval but exhibit significant errors starting in the extrapolation interval, particularly near the surface.

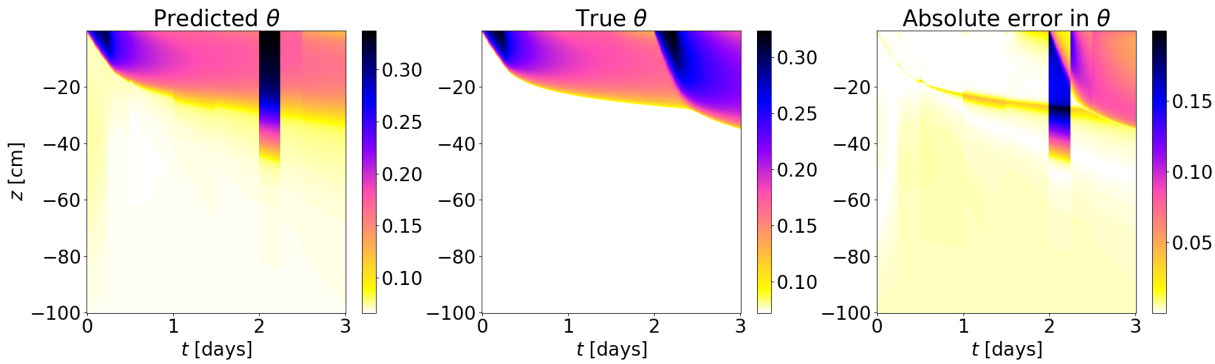


Figure 3.3: Predicted VWC (left), simulated VWC (middle) and absolute error of VWC (right) before adding discrete residual in the loss function.

The results in Table 3.1 confirms the ovbservations from Figure 3.3. First, in the training interval $t \in [0, 1.5)$, the network achieves low relative errors across all variables, particularly for $\hat{\theta}$, where the error is on the order of 10^{-5} near the surface and 10^{-3} across the entire domain. However, in the extrapolation interval $t \in [1.5, 3]$, prediction errors increase substantially—most prominently near the surface. For example, the relative error in $\hat{\theta}$ rises by over three orders of magnitude, and the error in $\hat{K}$ exceeds 3.3 near the surface and 3.9 over the full depth. This degradation reflects the network's inability to maintain physical consistency and temporal dependency during forward prediction. The substantial rise in $\hat{\psi}$ and $\hat{K}$ errors suggests that flux boundary information is not properly propagated through the network during extrapolation, leading to unphysical moisture dynamics.

This discrepancy highlights a fundamental limitation of standard PINNs: their approximations are localized in time. That is, predictions at time t are not inherently conditioned on prior predictions at earlier time steps, despite the underlying PDE being causal in nature. This breakdown becomes especially problematic in extrapolation settings, where the model must simulate evolving processes based on past inputs.

Wang, Sankaran, and Perdikaris [53] identified similar causality violations in time-dependent PINNs and proposed weighting schemes that bias training toward earlier time points. However, their approach relies on embedding boundary information directly into the network, which is not feasible in our case due to the time-varying and discontinuous nature of the flux boundary condition.

To overcome this limitation, we propose a principled modification to the PINN loss function by introducing a time-discrete residual term that enforces consistency between predictions at successive time steps. This approach explicitly couples the approximation at time t to that at $t - \tau$, introducing a notion of causal dependence directly into the training objective. The details of this formulation are presented in the next subsection.

3.3.2 Discrete Residual (DR)

To mitigate the temporal inconsistency observed in standard PINNs, we introduce a discrete residual term into the loss function that enforces causal dependence between successive time steps. Specifically, we define the discrete residual $\hat{r}_d(z, t)$ at each spatial-temporal point (z, t) as:

$$\hat{r}_d := \hat{\theta}(z, t) - \tau \frac{\partial \hat{\theta}(z, t)}{\partial t} - \hat{\theta}(z, t - \tau) \tag{3.5}$$

where τ is the discrete time step size used in the simulation. This expression is derived from a first-order backward finite difference approximation of the temporal derivative:

$$\frac{\partial \hat{\theta}}{\partial t}(z, t) \approx \frac{\hat{\theta}(z, t) - \hat{\theta}(z, t - \tau)}{\tau}.$$

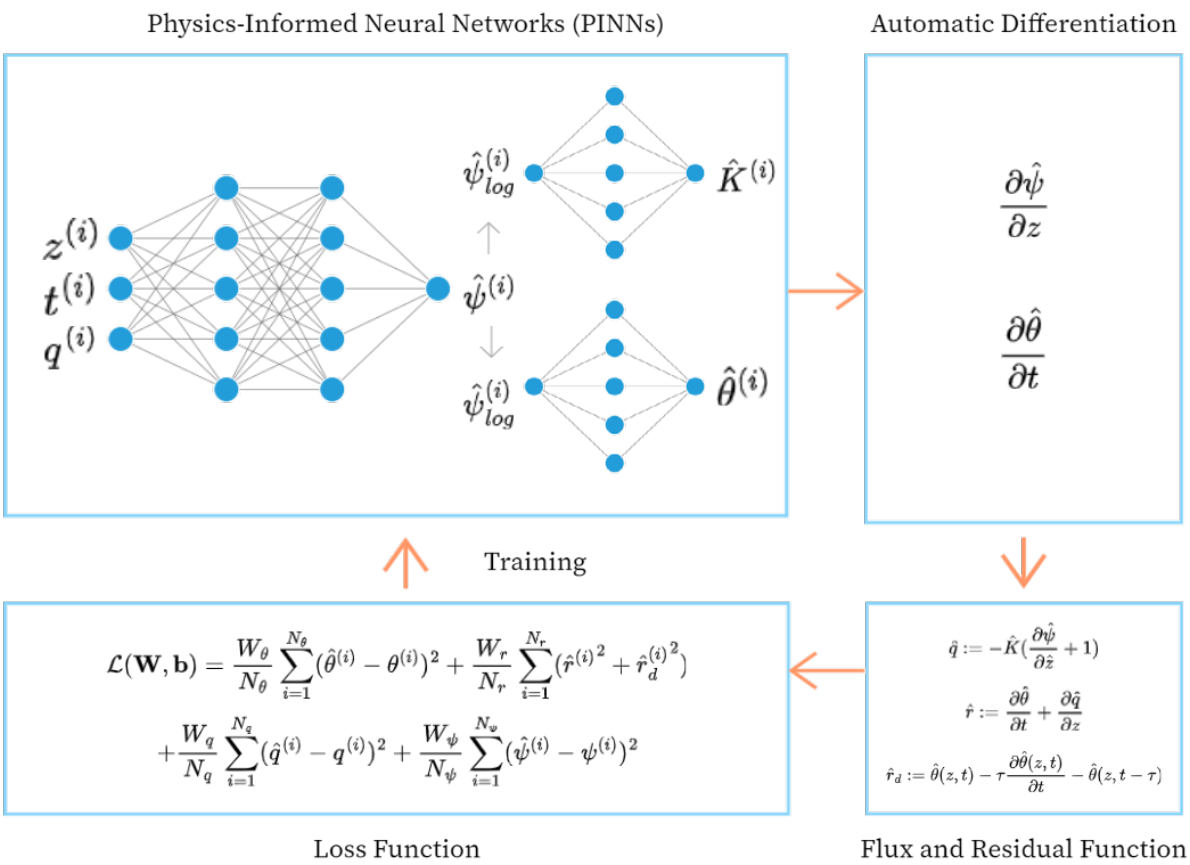


Figure 3.4: The new PINN structure with the modified loss function in Eq. (3.6)

.

Intuitively, $\hat{r}_d(z,t)$ measures the discrepancy between the predicted time evolution of $\hat{\theta}$ and its finite difference approximation. By penalizing this discrepancy in the loss function, we promote temporal smoothness and causal dependence in the learned solution. Other time discretization schemes, such as the Crank-Nicolson method, could also be employed, but we found that higher-order scheme increases computation overhead exponentially, and the backward finite difference approach suffices for our purposes.

The final loss function, which incorporates this time-discrete residual, is given by:

$$\begin{aligned}\mathcal{L}(\mathbf{W},\mathbf{b}) = \frac{W_\theta}{N_\theta}\sum_{i=1}^{N_\theta}(\hat{\theta}^{(i)}-\theta^{(i)})^2 + \frac{W_r}{N_r}\sum_{i=1}^{N_r}((\hat{r}^{(i)})^2+(\hat{r}_d^{(i)})^2) \\ + \frac{W_\psi}{N_\psi}\sum_{i=1}^{N_\psi}(\hat{\psi}^{(i)}-\psi^{(i)})^2 + \frac{W_q}{N_q}\sum_{i=1}^{N_q}(\hat{q}^{(i)}-q^{(i)})^2.\end{aligned} \tag{3.6}$$

This formulation introduces a time-dependent coupling between predictions, encouraging the network to generate temporally consistent approximations of VWC even in extrapolation windows. The structure of this causality-aware loss function is illustrated in Fig. 3.4, which reflects the final architecture used throughout the rest of this chapter.

Table 3.2: The relative error of $\hat{\psi}$, $\hat{\theta}$ and $\hat{K}$ of a network trained with loss function in Eq. (3.6) for near-surface depths and the entire depth column.

	Relative Error of (for $z \in [-20, 0]$)		
	$\hat{\theta}$	$\hat{\psi}$	$\hat{K}$
$t \in [0, 1.5]$	3.25071e-04	5.00523e-02	1.76608e-02
$t \in [1.5, 3]$	2.94460e-04	2.37570e-02	1.44628e-02
	Relative Error of (for $z \in [-100, 0]$)		
	$\hat{\theta}$	$\hat{\psi}$	$\hat{K}$
$t \in [0, 1.5]$	1.28913e-03	1.50194e-02	1.77080e-02
$t \in [1.5, 3]$	2.41329e-03	3.92990e-02	1.48857e-02

With the time-dependent loss function defined in Eq. (3.6), we retain the same training setup and weighting scheme described previously. The network is trained using residual samples and boundary condition data spanning the full time domain. The weighting coefficients are kept consistent: $W_\theta = 100$, $W_\psi = 1$, $W_q = 10$, and W_r is exponentially increased from 10^{-3} to 10^2 over the first 100,000 epochs, with updates applied every 500 epochs. The optimization procedure is identical to earlier experiments: 150,000 epochs using the Adam optimizer, followed by 50,000

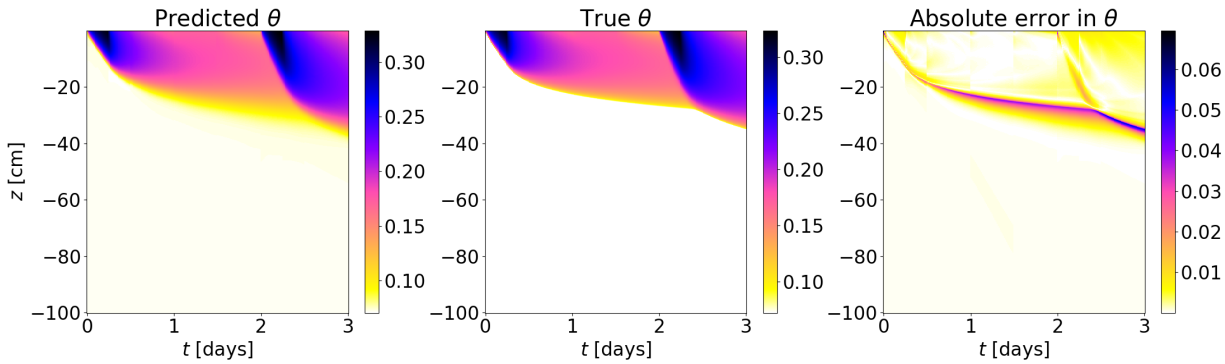


Figure 3.5: Predicted VWC (left), and simulated VWC (middle) and absolute error of VWC (right) after adding discrete residual in the loss function.

epochs of L-BFGS.

The predictive performance of the improved model is summarized in Table 3.2. Comparing Table 3.1 and Table 3.2, we observe a significant improvement in predictive accuracy after incorporating the discrete residual term into the loss function. In the original model (Table 3.1), the relative error for $\hat{\theta}$ in the extrapolation window ($t \in [1.5, 3]$) near the surface was as high as 0.117, while the corresponding error in the revised model (Table 3.2) dropped to 2.94×10^{-4}. This represents a three-orders-of-magnitude improvement in out-of-sample prediction accuracy for $\hat{\theta}$. Similar improvements are observed for $\hat{K}$ and $\hat{\psi}$ in the same interval: the error in $\hat{K}$ decreased from 3.31 to 0.014, and the error in $\hat{\psi}$ dropped from 0.586 to 0.0238.

These improvements are further visualized in Fig. 3.5. The predicted VWC values remain accurate across both the training and extrapolation windows, and the model is able to correctly localize the effects of high-magnitude flux events near the surface. Some discrepancies remain near regions of steep moisture gradients. Nonetheless, these results confirm that enforcing temporal consistency via discrete residuals enhances the physical realism and robustness of PINN-based solutions for Richards' Equation under challenging boundary conditions.

3.4 Results for Various Flux and Soil Conditions

In this section, we evaluate the predictive performance of the time-dependent PINN model under varying soil textures, measurement densities, noise levels, and flux conditions. Specifically, we consider three USDA soil types—sandy loam, loam, and silt loam—and two surface flux boundary conditions (shown in Fig. 3.1). These tests assess the model's robustness under realistic and challenging hydrological scenarios.

Table 3.3: The L_2 relative error of VWC for the out-of-sample time ($t \in [1.5, 3]$) and near-surface depths ($z \in [-20, 0]$).

Noise Level	Soil Type								
	Loam	Sandy Loam	Silt Loam	Loam	Sandy Loam	Silt Loam	Loam	Sandy Loam	Silt Loam
Number of Measurements	3			5			10		
	Flux Scenario 1								
0	1.50e-3	2.14e-3	1.16e-3	1.62e-3	1.77e-3	1.08e-3	1.59e-3	1.92e-3	1.40e-3
0.005	1.86e-3	2.01e-3	1.28e-3	1.47e-3	1.68e-3	1.10e-3	1.51e-3	2.05e-3	1.38e-3
0.01	1.22e-3	2.31e-3	1.21e-3	1.43e-3	1.74e-3	9.91e-4	1.94e-3	2.23e-3	1.59e-3
0.02	1.76e-3	1.71e-3	1.58e-3	1.27e-3	1.35e-3	9.64e-4	1.77e-3	2.03e-3	1.62e-3
	Flux Scenario 2								
0	2.49e-4	7.62e-4	3.01e-4	1.01e-3	5.99e-4	6.60e-4	6.56e-4	9.44e-4	3.43e-4
0.005	4.84e-4	7.67e-4	1.31e-4	7.43e-4	8.11e-4	3.06e-4	3.84e-4	1.06e-3	8.75e-5
0.01	2.59e-4	7.80e-4	3.29e-4	6.82e-4	4.83e-4	1.09e-4	8.23e-4	8.72e-4	3.04e-4
0.02	3.58e-4	3.29e-4	2.72e-4	1.43e-3	3.50e-4	1.16e-4	5.68e-4	1.03e-3	2.24e-4

Volumetric water content (VWC) measurements are synthetically generated at either 3, 5, or 10 depths. The 3-depth configuration includes $z = \{-1, -9, -19\}$ cm; the 5-depth setup samples $z = \{-1, -5, -9, -15, -19\}$ cm; and the 10-depth case covers $z = \{-1, -3, -5, -7, -9, -11, -13, -15, -17, -19\}$ cm. To emulate sensor uncertainty, we inject additive Gaussian noise into the measurements, with zero mean and standard deviations of 0, 0.005, 0.01, and 0.02.

We emphasize evaluation of near-surface prediction accuracy ($z \in [-20, 0]$ cm). Table 3.3 reports the L_2 relative errors of VWC predictions during the extrapolation window ($t \in [1.5, 3]$ days) across all combinations of noise level, soil type, measurement count, and flux condition. The

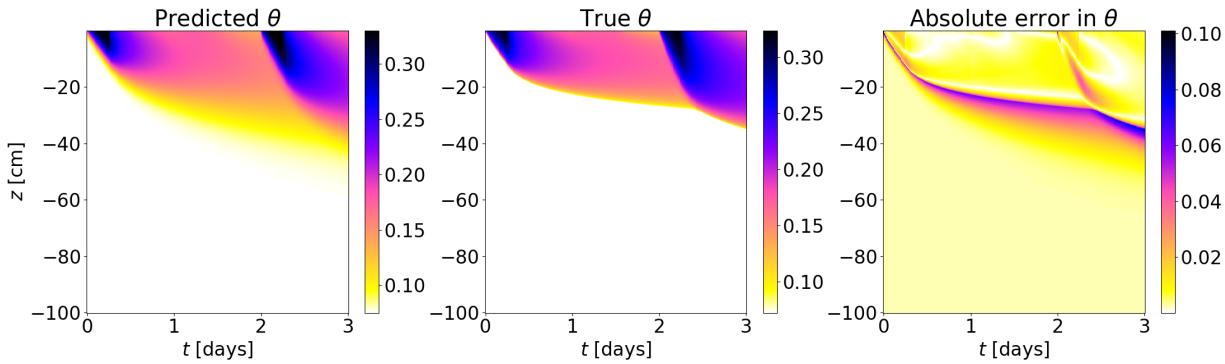


Figure 3.6: Predicted VWC (left) compared to simulated VWC (middle) for PINN trained with 3 depths and 0.02 noise level.

results demonstrate that: the model consistently achieves out-of-sample errors on the order of 10^{-3}, even when limited to only three measurement depths and moderate noise levels. It demonstrates robustness to input noise, as increasing variance does not significantly degrade performance. Additionally, prediction accuracy improves with greater spatial measurement density; however, the marginal gains diminish beyond five measurement depths.

These trends are visually confirmed in Fig. 3.6, which compares predicted and simulated VWC profiles for sandy loam soil, using 3 measurement depths and the highest noise level ($\sigma = 0.02$). The model captures the overall dynamics of VWC evolution, although it struggles to accurately predict values near zero in deeper soil layers. This is likely due to the inherent difficulty of modeling low moisture content with limited data, as well as the influence of noise on the training process.

Overall, the time-dependent PINN formulation with discrete residual augmentation demonstrates strong generalization capabilities, handling complex flux patterns, heterogeneous soil types, and sparse, noisy observations with minimal loss in predictive accuracy.

3.5 Discussion

This chapter introduced a causality-aware PINN framework for predicting volumetric water content (VWC) in unsaturated soils governed by Richards' Equation under discontinuous flux boundary conditions. The proposed model improves upon prior PINN formulations by: (i) incorporating the surface flux as a network input and (ii) introducing a discrete residual term to impose temporal consistency.

Simulation results across multiple soil types, flux patterns, and noise levels demonstrate that the time-dependent PINN achieves accurate out-of-sample predictions, with relative errors on the order of 10^{-3} in near-surface regions. Importantly, the network maintains robustness even under sparse measurements and moderate observational noise. These results validate the effectiveness of the discrete residual formulation in enforcing causal structure, enabling PINNs to better generalize beyond the training window.

While promising, the model is subject to several limitations. First, the assumption of a homogeneous soil column excludes spatial heterogeneity commonly encountered in field applications. Second, the model does not account for sink terms such as plant root uptake or surface runoff that occur when precipitation exceeds the soil's saturated hydraulic capacity. Finally, temperature-dependent evaporation and lateral fluxes are omitted, which may affect model fidelity under certain environmental conditions.

Despite these simplifications, the proposed framework provides a fast and accurate alternative to traditional solvers (e.g., HYDRUS-1D). The ability to predict VWC solely from soil type and surface flux inputs has important implications for remote sensing and hydrological modeling applications. The discrete-residual loss design introduced here is also readily extensible to other

time-dependent PDE problems with evolving boundary conditions.

Future work will focus on relaxing current modeling assumptions by incorporating heterogeneous media, dynamic sink terms, and more realistic surface-atmosphere interactions. Additionally, we aim to quantify uncertainty in predictions and explore adaptive sampling strategies that optimize sensor placement based on model sensitivity.

In the next chapter, we shift our focus to the challenge of resolving steep gradients in PINN approximations, using the convection–diffusion equation as a testbed. Due to its simpler structure and the possibility of reducing it to an ordinary differential equation (ODE) in certain cases, the convection–diffusion equation provides a tractable setting for investigating numerical stiffness and solution sharpness in PDE learning. This allows us to isolate and analyze the core mechanisms behind gradient under-resolution and evaluate techniques that enhance PINN performance in the presence of sharp features—insights that will ultimately inform more robust designs for solving Richards' Equation and other nonlinear transport PDEs.

Chapter 4: PINNs for Convection-Diffusion Equation: Known Steep Gradient Layer Location

4.1 Introduction

Building upon the previous chapter, which introduced the Richards' equation and demonstrated how causality-aware loss functions and modified input can improve physics-informed neural network (PINN) performance in the presence of discontinuous boundary fluxes, this chapter shifts focus to convection-diffusion problems characterized by sharp boundary layers whose locations are known. In the convection-dominated regime, traditional numerical schemes such as finite difference or finite volume methods often produce spurious oscillations unless sophisticated stabilization or adaptive mesh refinement techniques are employed. Physics-Informed Neural Networks (PINNs), which embed the PDE residual into the loss function, offer an alternative solution framework. However, we will show that standard PINNs can also fail to approximate solutions accurately in the presence of sharp boundary layers. These difficulties arise partly due to the weak gradient signals provided by the loss, which can fail to guide the network toward accurately resolving steep gradients, especially when using standard input coordinates.

In this chapter, we consider convection-diffusion problems with known boundary layer locations and show how PINNs can be used to correct inaccurate or reduced solutions. Specifically,

we propose Reduced PINNs (RPINNs), which aim to learn corrections to a reduced solution obtained in the convective limit. By removing the diffusion term, the reduced solution captures the structure of the solution away from the boundary layer, allowing the PINN to focus its capacity on the localized region near the boundary. We analyze the improved training behavior of RPINNs using Neural Tangent Kernel (NTK) theory and identify key factors influencing convergence, such as the alignment of network sensitivity with the boundary layer and the impact of input transformations like linear shifts and scalings.

We also demonstrate the effectiveness of RPINNs in both one-dimensional and two-dimensional convection-diffusion settings. Finally, this chapter sets the stage for the next, in which we address the more challenging case where the location of the boundary layer is not known a priori.

The remainder of this chapter is structured as follows. Section 4.2 introduces the one-dimensional convection-diffusion test problem and highlights the challenges associated with steep boundary layers. Section 4.3 investigates the use of standard Physics-Informed Neural Networks (PINNs) as correctors for oscillatory finite difference solutions, demonstrating their limitations and sensitivity to input representation. Section 4.4 proposes Reduced PINNs (RPINNs), which focus on correcting the reduced (convective-limit) solution instead of the discrete numerical solution, leading to a more stable learning task. Section 4.4.1 provides an in-depth analysis of RPINNs, identifying two necessary conditions for accurate approximation: alignment of network sensitivity with the boundary layer and sufficiently steep first-layer weights. Section 4.4.3 explores input transformations, such as linear shifts and scalings, and analyzes their effect on convergence using both empirical results and NTK theory. Section 4.5 extends the approach to a two-dimensional problem, demonstrating the applicability of transformed RPINNs to higher-dimensional settings.

Finally, Section 4.6 summarizes the key findings and prepares the foundation for the following chapter, which addresses the case where the location of the boundary layer is unknown.

4.2 Problem Statement

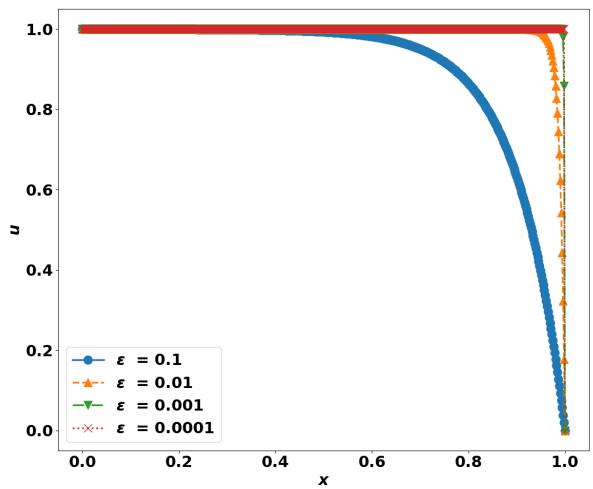


Figure 4.1: Exact solutions of Equation (4.1) for $\epsilon \in \{0.1, 0.01, 0.001, 0.0001\}$.

In this section, we describe the class of convection-diffusion equations used as test cases throughout the chapter. We consider one-dimensional problems with Dirichlet boundary conditions. A representative example is the simplified form of Equation (2.5):

$$\begin{aligned} -\epsilon u''(x) + u'(x) = 0, \quad x \in (0,1), \\ u(0) = 1 - e^{-1/\epsilon}, \quad u(1) = 0, \end{aligned} \tag{4.1}$$

which admits the analytic solution

$$u_{ex}(x, \epsilon) = 1 - e^{(x-1)/\epsilon}. \tag{4.2}$$

Figure 4.1 shows the exact solutions for several values of ϵ. As $\epsilon \to 0$, the solution develops increasingly sharp boundary layers near $x = 1$, a hallmark of convection-dominated problems.

These boundary layers are known to pose significant challenges for traditional numerical solvers, and, as we will show, they also impact the performance of PINNs. In some cases, the trained network approximates a function that more closely resembles the solution to a *different* convection-diffusion equation. This misbehavior motivates our investigation into the underlying causes in Sections 4.4.1.

To facilitate later discussion, we denote by $c_{\boldsymbol{\theta}}(x)$ the PINN surrogate using a dense neural network (DNN), with inputs denoted by X_{in}. The physical interpretation of x will be clarified subsequently. We adopt standard initialization schemes shown in Chapter 2: the weights W_l in the l-th hidden layer are initialized using the Xavier method [91], with entries drawn i.i.d. from $\mathcal{N}(0, \frac{2}{d_l+d_{l+1}})$, and biases are set to zero.

Since input scaling plays a key role in improving neural network training, it is common practice to normalize input coordinates [107]. Motivated by this, we study the effect of applying explicit transformations to the spatial input. While the training samples and corresponding targets remain fixed, transforming the input coordinates changes the structure of the loss function. This transformation may implicitly ease the optimization process and yield more accurate approximations.

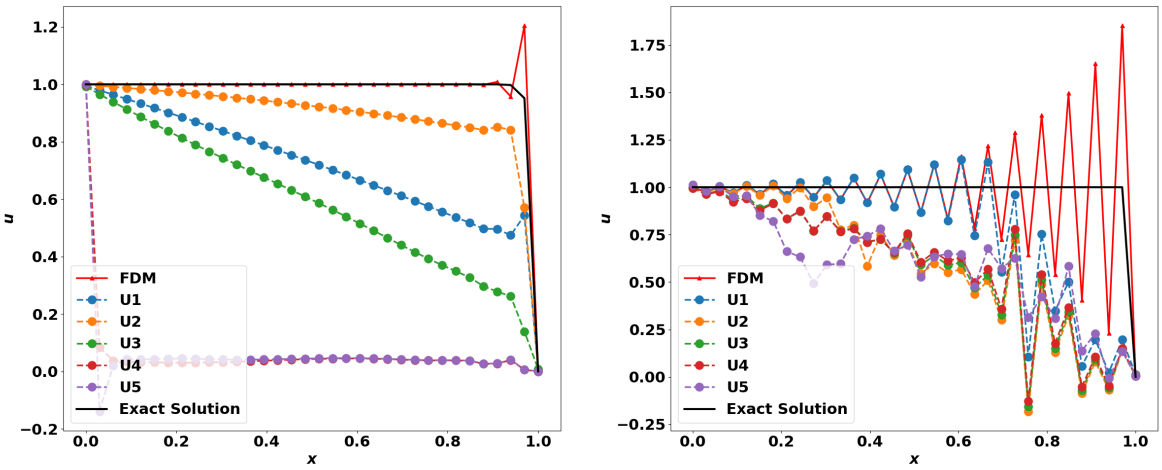
Specifically, we propose an affine transformation of the form:

$$T(x) = a(x + b), \tag{4.3}$$

which is applied to the spatial coordinate x prior to being fed into the network. The implications of

this transformation for loss minimization and PINN accuracy will be examined in the subsequent sections.

4.3 Correcting Oscillatory Discrete Solutions



(a) (b)

Figure 4.2: Corrections made by standard PINN on FDM oscillatory solutions for: $\epsilon = 0.01$ in Figure 4.2a and $\epsilon = 0.001$ in Figure 4.2b.

Since the introduction of PINNs, there have been several efforts to use them in resolving boundary layers [108, 109, 110]. Although the exact solution (4.2) to the convection-diffusion problem is continuous, it exhibits steep gradients for small ϵ values, rendering the solution effectively close to discontinuous. This presents a well-known challenge for neural networks, which are notoriously poor at approximating discontinuities [108, 111]. As such, it becomes essential to understand how standard PINNs behave when faced with steep boundary layers.

This section presents a first attempt to use PINNs to correct oscillatory discrete solutions obtained from finite difference methods (FDMs). Instead of using a neural network directly to approximate the solution, we explore a correction-based strategy inspired by iterative solvers. In

each iteration, a neural network is trained to correct the error in the discrete solution, producing a successively improved approximation. This approach is inspired by multi-level PINNs [112, 113, 114], where residuals are minimized iteratively using different networks.

Consider solving Equation (4.1) using a central difference FDM on an equi-spaced mesh $\{x_j\}_{j=0}^{N}$ with mesh size h. The discrete system becomes:

$$\Big(-\frac{\epsilon}{h^2}-\frac{1}{2h}\Big)U_{i-1}+\Big(2\frac{\epsilon}{h^2}\Big)U_i+\Big(\frac{1}{2h}-\frac{\epsilon}{h^2}\Big)U_{i+1}=0, \tag{4.4}$$

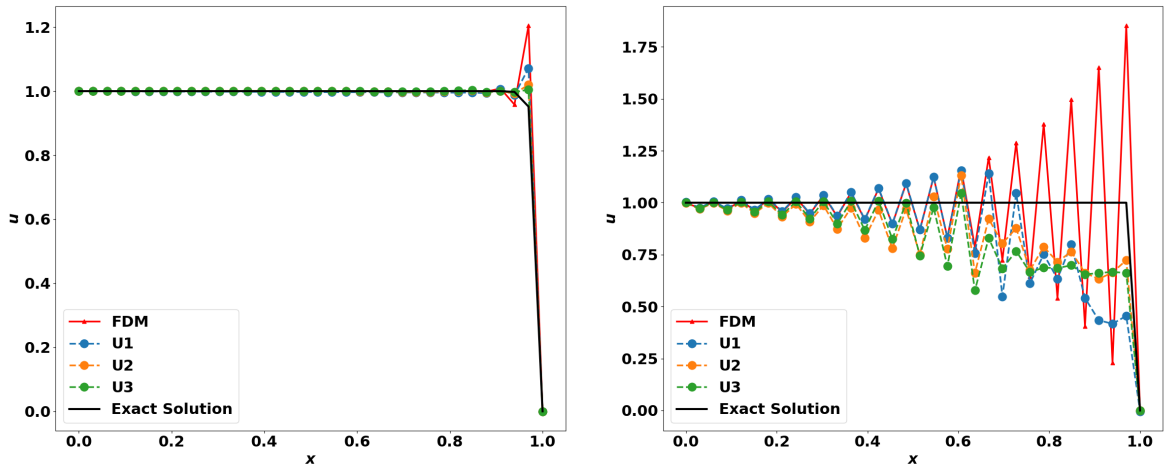
with boundary conditions $U(0) = 1 - e^{-1/\epsilon}$ and $U(1) = 0$. When the mesh Peclet number $\mathcal{P}_h = h/(2\epsilon)$ exceeds one, these discrete solutions become oscillatory and unphysical [89], failing to resolve the boundary layer. Figure 4.2 shows representative examples.

To improve these discrete solutions, we define a correction network $c_{\boldsymbol{\theta}}$ such that the surrogate approximation is $u_s(x) = \hat{u}(x) + c_{\boldsymbol{\theta}}(x)$, where $\hat{u}$ is the piecewise linear interpolant of the FDM solution. Substituting u_s into Equation (4.1), and noting that $\hat{u}'' = 0$ between grid points, we obtain the modified residual:

$$\begin{aligned} -\epsilon c''_{\boldsymbol{\theta}}(x) + c'_{\boldsymbol{\theta}}(x) &= -\hat{u}'(x), \quad x \in (0,1), \\ u_s(0) = (\hat{u} + c_{\boldsymbol{\theta}})(0) &= 1 - e^{-1/\epsilon}, \\ u_s(1) = (\hat{u} + c_{\boldsymbol{\theta}})(1) &= 0. \end{aligned} \tag{4.5}$$

Residual samples are chosen to avoid the grid points $\{x_i\}$ to circumvent non-differentiability in $\hat{u}$. The corresponding loss function follows Equation (2.19), with the residual term:

$$r_{\boldsymbol{\theta}}(x_r^{(i)}) = -\epsilon c''_{\boldsymbol{\theta}}(x_r^{(i)}) + c'_{\boldsymbol{\theta}}(x_r^{(i)}) + \hat{u}'(x_r^{(i)}). \tag{4.6}$$



(a) (b)

Figure 4.3: Corrections made by PINN with input transformation $T(x) = 100(x-1)$ on FDM oscillatory solutions for: $\epsilon = 0.01$ in Figure 4.3a and $\epsilon = 0.001$ Figure 4.3b.

This iterative correction process continues by training new networks $c_{\boldsymbol{\theta}}^j$ to update the approximation:

$$(U^j)(x_i) = (U^{j-1} + c_{\boldsymbol{\theta}}^j)(x_i), \quad j = 1, 2, \ldots, \; i = 0, \ldots, N, \tag{4.7}$$

where U^j denotes the updated discrete approximation at step j. With given initial approximations $\{U_i^0\}_{i=0}^N$, we can train $c_{\boldsymbol{\theta}}^j$ for correction iteration $j = 1, 2, \ldots$. Each subsequent $c_{\boldsymbol{\theta}}^{j+1}$ will be a correction for $\hat{u} + \sum_{k=1}^j c_{\boldsymbol{\theta}}^k$. These correction steps can be performed until $(\hat{u} + \sum_{k=1}^j c_{\boldsymbol{\theta}}^k)(x)$ is smooth and accurately approximates $u(x)$.

We implemented this iterative PINN-based correction using 3-layer networks of width 40, trained with the Adam optimizer for 10,000 epochs per correction step. For $\epsilon = 0.01$ and 0.001, using a uniform grid of $N = 32$ points, the initial FDM solutions are oscillatory. Figure 4.2 shows that after five correction steps, the results still deviate significantly from the true solution—particularly for $\epsilon = 0.001$, the corrected approximation fails to converge and exhibits

nonphysical behavior.

To further investigate whether a transformation of the input space might improve convergence, we applied the affine transformation $T(x) = 100(x - 1)$ before feeding inputs into the network. This effectively shifts the training domain, resulting in a new surrogate $u_s(T(x)) = \hat{u}(x) + c_{\boldsymbol{\theta}}(T(x))$. With this setup, we observed notable improvements for $\epsilon = 0.01$, where the corrected approximation converged smoothly toward the exact solution by the third iteration (Figure 4.3). However, for $\epsilon = 0.001$, the transformation did not yield improvements; the corrected function still failed to capture the steep boundary layer and introduced incorrect structures.

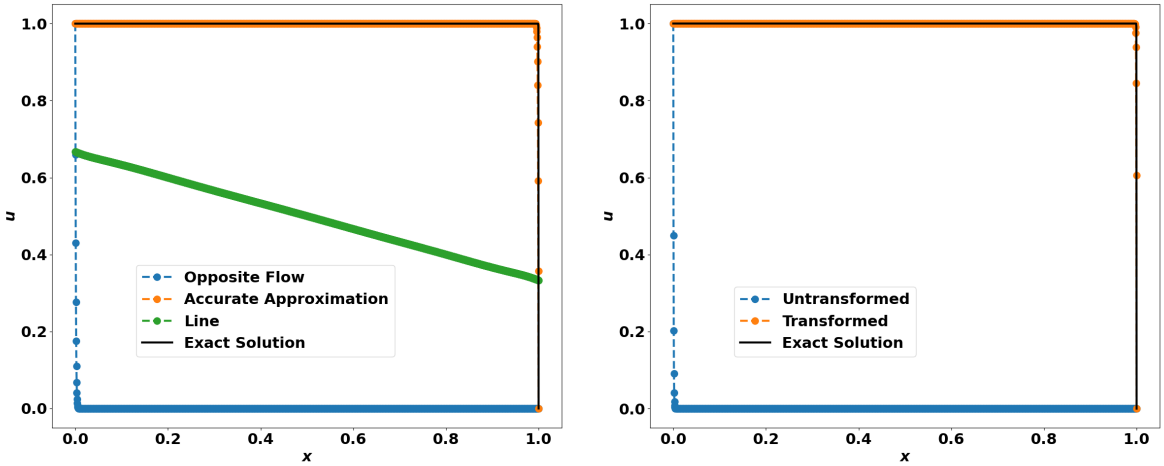
These experiments reveal that directly correcting oscillatory FDM solutions using PINNs is not a robust strategy—particularly for small ϵ, where the corrections may converge to inaccurate or entirely incorrect profiles. This motivates the development of a new approach in the next section: instead of correcting an oscillatory numerical solution, we will train PINNs to correct a smoother *reduced* model solution, whose behavior is more amenable to learning.

Representative examples of this process are shown in Figure 4.2 for $\epsilon = 0.01, 0.001$, on a uniform grid of size $N = 32$ over $x \in [0, 1]$. The mesh Peclet numbers are larger than unity for both values of ϵ, and the FDM solutions are oscillatory. An equi-spaced collection of points over $[0, 1]$ with stepsize $1/128$ was used as the training residual sample collection. The uniform grid points used to generate FDM approximations were not used as sample points, which rendered 96 points in the residual training samples. We train individual DNNs of 3 hidden layers with width 40 per layer using 10,000 epochs of the Adam optimization scheme for 5 correction iterations. For $\epsilon = 0.001$, by the fifth correction, the approximations U^5 are still far from the exact solution. The corrections do not converge monotonically to an accurate surrogate function. Indeed, for $\epsilon = 0.01$,

approximations of U^5 present a flow opposite to that of the exact solution.

As discussed in Section 4.1, input transformations result in different loss functions. We explore the impact of transformations on this process, using the transformation $T(x) = 100(x-1)$ on inputs of the network. The resulting corrected approximation becomes $u_s(T(x)) = \hat{u}(x) + c_{\boldsymbol{\theta}}(T(x))$. Using the same network and training settings as in the previous tests, we found that $c_{\boldsymbol{\theta}}(T(x))$ performs better than $c_{\boldsymbol{\theta}}(x)$ for $\epsilon = 0.01$. This can be seen from Figure 4.3 where the corrected solution approaches the exact solution with each subsequent iteration for $\epsilon = 0.01$. By the third correction iteration, the corrected solution U^3 closely follows the exact solution. Unfortunately, this transformation is not effective for $\epsilon = 0.001$. The corrected solutions failed to capture the steep boundary layer. Instead of correcting to a steep boundary layer near $x = 1$ around $y = 1$, the corrected approximations develops an incorrect flat region near $y = 0.75$. These results are representative of our experience for correcting oscillatory solutions, and we found that using such transformations to correct oscillatory discrete solutions does not lead to a robust solution strategy.

In the next section, we explore an alternative approach, where PINNs are used to correct a simpler solution, derived from the reduced equation.



(a) (b)

Figure 4.4: Different approximations generated using a standard PINNs under different random initialization seeds in 4.4a. Comparison of effects of transformation in 4.4b. Here the input is transformed from $x \in [0, 1]$ to $x - 1 \in [-1, 0]$.

4.4 Correcting Reduced Solutions: Reduced PINN (RPINN)

In this section, we consider an alternative approach for beginning the PINN procedure. Instead of the test problem in (4.1), consider the following *reduced problem*

$$\begin{aligned} u_0' &= 0, \\ u_0(0) &= 1, \end{aligned} \tag{4.8}$$

which corresponds to the hyperbolic problem arising from $\epsilon = 0$. The reduced solution is $u_0(x) = 1$ for $x \in [0, 1]$, and there is no boundary condition (or layer) at the outflow boundary $x = 1$. We take a different point of view and apply PINNs as corrections to the reduced solutions of the convection-diffusion equations, i.e., the solution to the pure convection equation for $\epsilon = 0$.

We set $\hat{u}(x) = u_0(x)$. The governing equation that $c_{\boldsymbol{\theta}}$ needs to satisfy is

$$-\epsilon c_{\boldsymbol{\theta}}'' + c_{\boldsymbol{\theta}}' = 0 \tag{4.9}$$

We would like to train $c_{\boldsymbol{\theta}}$ such that $u_0 + c_{\boldsymbol{\theta}}$ approximates the solution to (4.1). The boundary conditions for $c_{\boldsymbol{\theta}}$ are $c_{\boldsymbol{\theta}}(0) + u_0 = 1 - e^{-1/\epsilon}$ and $c_{\boldsymbol{\theta}}(1) + u_0 = 0$, which leads to resulting boundary conditions as $c_{\boldsymbol{\theta}}(0) = 0$ and $c_{\boldsymbol{\theta}}(1) = -1$ The loss function for $c_{\boldsymbol{\theta}}$ is the same as in (4.6) except the residual and boundary condtions are updated.

We call this approach as *Reduced PINN* (RPINN). RPINN possesses the benefit that the residual functions are the similar across different convection-diffusion equation with various wind and forcing functions.[1] The residual functions all have a zero right hand side. Therefore, the analysis that we perform on RPINN in the following sections, where we utilize the fact that the forcing term is zero, applies to all convection-diffusion equations with constant wind under the framework of RPINN.

To explore the performance of RPINN, we use RPINN to approximate the correction to the reduced solution for $\epsilon = 0.0001$. For residual samples, we create an equi-spaced mesh over $x \in (0, 1)$ with stepsize $h = 1/128$, as we did for correcting FDM solutions. The boundary samples are $\{(0, -e^{-1/\epsilon}), (1, -1)\}$. The loss function is the standard PINN loss function in (2.19). We train a network with one hidden layer of width 20 using the Adam optimizer [94] for

[1]Different one-dimensional convection-diffusion equations may need similar correction of different reduced solutions. An example is

$$\begin{aligned} -\epsilon u'' + u' &= 1, \\ u(0) = u(1) &= 0. \end{aligned} \tag{4.10}$$

The reduced solution of (4.10) is x. But the correction needed satisfies the residual function of $-\epsilon c_{\boldsymbol{\theta}}'' + c_{\boldsymbol{\theta}}' = 0$ with boundary conditions $c_{\boldsymbol{\theta}}(0) = 0$ and $c_{\boldsymbol{\theta}}(1) = -1$. The correction needed for (4.10) has similar characteristics to the one for (4.1). Both corrections contain a steep gradient layer near $x = 1$.

10,000 epochs, followed by another 10,000 epochs using LBFGS [104]. Weights and biases of the networks were initialized following the Xavier initialization using different random seeds. Using 100 different random seeds, we found that only seven trained networks produced good approximations that closely followed the exact solution. Eighty-six of the trained networks produced approximations with a steep boundary layer located near $x = 0$, corresponding to a flow in the direction opposite to that of the solution, as for the solution of $-\epsilon u'' - u' = 0$ with the same boundary conditions as (4.1). The remaining seven of the trained networks produced lines with negative slopes. The three types of approximations are shown in Figure 4.4a. We also found that increasing the depth or width of the network does not improve the chance for PINNs obtaining an accurate approximation. This points to the questions of whether it is possible for a network to reliably obtain good approximations and the cause of such different results.

While these results indicate that RPINN offers a promising framework, the success rate remains inconsistent without additional adjustments to the network or input. This inconsistency raises fundamental questions about the internal mechanisms of PINNs when applied to problems with steep boundary layers.

In the next section, we examine the necessary conditions for PINNs to reliably approximate these steep gradients, drawing insights from both empirical patterns and theoretical analysis.

4.4.1 Necessary Conditions for Accurate Approximation: Correct Effective Shift

The results from the previous section reveal a critical limitation: even when PINNs are applied in the simpler RPINN framework, the networks struggle to consistently learn the correct

steep boundary layer near $x = 1$. Instead, standard PINNs often misplace the boundary layer near $x = 0$ or fail to develop any steep gradient at all, frequently converging to incorrect linear profiles.

These outcomes suggest that there are deeper structural factors at play that influence whether a network successfully approximates steep gradients in singularly perturbed problems like the convection-diffusion equation.

Two key questions emerge: First, what intrinsic properties of the network and its initialization govern the location and formation of steep boundary layers? Secondly, can we identify specific conditions under which PINNs are more likely to capture the correct solution behavior?

Informed by our empirical observations, we hypothesize that the issue may stem from a mismatch between the activation function's sensitivity region and the physical location of the steep gradient. Our intuition was that inaccurate surrogates can be attributed to the properties of the activation function used, which was $\tanh(\cdot)$. The hyperbolic tangent function plateaus as the independent variable approaches $\pm\infty$, and its first derivative $\mathrm{sech}^2(\cdot)$ attains the largest value at the origin. It seemed beneficial to align the zero-crossing point of $xW_1 + b_1$ with the location of the boundary layer $x = 1$ at the initialization of the network. As described in Chapter 2, weights are initialized following a normal distribution of mean zero and biases are initialized to be zeros. With this convention, $xW_1 + b_1 = xW_1$. The root of $xW_1 + b_1$ at network initialization is expected to be $x = 0$, which does not naturally align with the location of the boundary layer $x = 1$. Shifting the input to the right by 1 shifts the root of $(x - 1)W_1 + b_1$ to $x = 1$, i.e. the location of the boundary layer.

To investigate our hypodissertation, we begin by empirically examining how a simple input shift impacts the network's ability to learn steep gradients.

Specifically, we apply the transformation $T(x) = x - 1$, which re-centers the domain so that

the outflow boundary layer at $x = 1$ aligns with $T(x) = 0$, the region where the $\tanh$ activation function exhibits its highest sensitivity. We then repeat the same RPINN training procedure described previously, using 100 random seeds with networks of one hidden layer and width 20. With the shifted inputs, only three approximations corresponded to the flow in the wrong direction. The rest of the ninety-seven approximations are close to the exact solution. To summarize, without input shifting, fewer than 10% of the networks produced accurate approximations, with the majority converging to boundary layers near $x = 0$ or incorrect linear profiles. However, after applying the input transformation, more than 95% of the networks successfully captured the correct boundary layer near $x = 1$, closely matching the exact solution.

These results provide compelling initial evidence that input shifting plays a crucial role in guiding PINNs to the correct solution. They also suggest that aligning the network's sensitivity with the location of the steep gradient is key to improving training robustness. Next, we explore the internal network dynamics—focusing on weights and biases after training—to further explain why this simple shift has such a significant effect.

Note that we perform input shifting with prior knowledge about where the boundary layer is. For a simple one-dimensional case, the location of the boundary layer always appears near the outflow boundary. Such prior information is not difficult to obtain. More complex cases where the locations of the boundary layers are not known a priori are discussed in Chapter 5.

We denote networks trained with unshifted inputs as *Untransformed* networks and the ones trained with shifted inputs as *Transformed* networks. The opposite flow approximation produced by an *Untransformed* network is shown in Figure 4.4b along with the accurate approximation produced by a *Transformed* network. To better understand why input shifting leads to markedly improved approximations, we analyze the trained weights and biases of both the *Untransformed*

and *Transformed* networks.

As a reminder, an *Untransformed* network with one hidden layer of width 20 can be written as

$$c_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{20} W_2^i \tanh(W_1^i x + b_1^i) + b_2. \tag{4.11}$$

We can arrange the network and rewrite (4.11) as

$$c_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{20} W_2^i \tanh(W_1^i (x + \frac{b_1^i}{W_1^i})) + b_2. \tag{4.12}$$

The Transformed network with the same depth and width can be written as

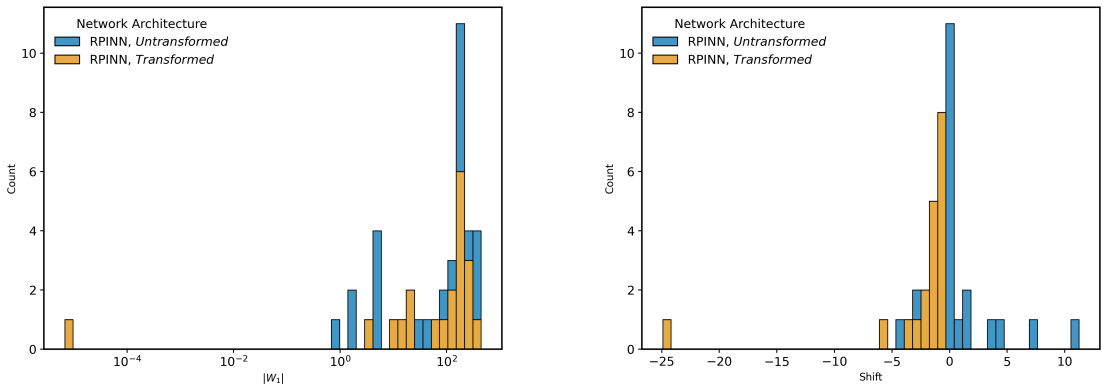
$$c_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{20} W_2^i \tanh(W_1^i (x + (-1 + \frac{b_1^i}{W_1^i}))) + b_2. \tag{4.13}$$

From (4.12) and (4.13), we can see that the only difference between the two networks is the input to the activation function. More precisely, the difference resides in the linear shift in the inputs. We focus on networks with one hidden layer and width 20 and examine two key metrics:

- The magnitude of the first layer weights, $|W_1^i|$, for $i = 1, \ldots, 20$.
- The *effective shift* defined as $\frac{b_1^i}{W_1^i}$ for *Untransformed* networks and $-1 + \frac{b_1^i}{W_1^i}$ for *Transformed* networks.

After training, these affine transformations govern how the activation functions are excited across the domain. We examine the distribution of $|W_i|$ and *effective shift* for the two types of networks in Figure 4.5.

Figure 4.5 shows histograms of these two quantities across all neurons and training seeds.



(a) Histogram of $|W_1^i|$

(b) Histogram of effective shift

Figure 4.5: Histograms of $|W_1^i|$ and shift values.

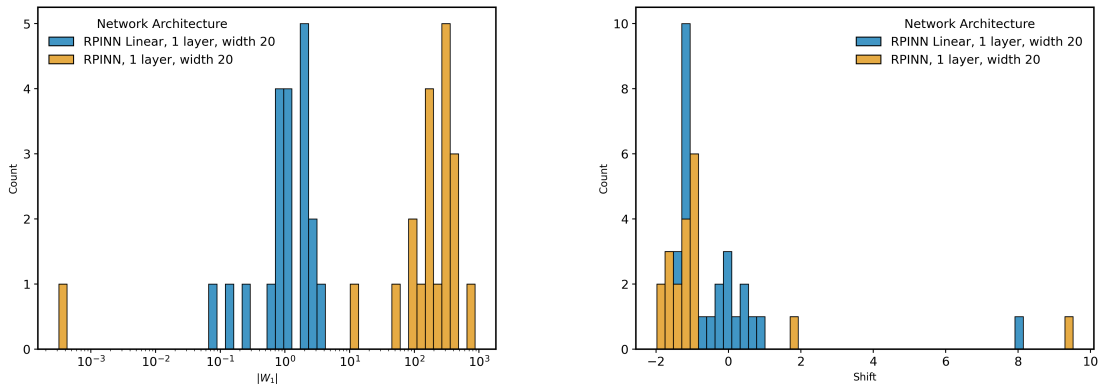
Two trends immediately emerge:

1. The weights $|W_1^i|$ tend to distribute around large values (centered near 10^2–10^3) for both network types.

2. The shift distribution for the *Transformed* networks is centered around -1, aligning the activation function sensitivity near $x = 1$, whereas for *Untransformed* networks it is centered near 0, focusing sensitivity incorrectly near $x = 0$. Keeping in mind that the *Shift* for the *Transformed* network comes with an inherent shift of -1 from the input transformation, $\frac{b_1^i}{W_1^i}$ should center around 0 for the *Transformed* network. This realization tells us that the trained bias values of the *Transformed* network is close to their initializations, which are 0.

These findings corroborate our intuition: the shift induced by $T(x) = x - 1$ successfully relocates the network's most sensitive region (around $\tanh(0)$) to the outflow boundary at $x = 1$, where the boundary layer forms.

Moreover, the analysis suggests that for *Untransformed* networks to succeed, they would require non-trivial bias values b_1^i to compensate for the natural centering at $x = 0$. However,



(a) Histogram of $|W_1^i|$

(b) Histogram of effective shift

Figure 4.6: Histograms of $|W_1^i|$ and shift values for an *Untransformed* that output a line and a *Transformed* network that approximates the solution accurately.

standard training initialization schemes (e.g., Xavier) typically begin with $b_1^i = 0$, making it less likely for the network to explore and settle into the desired parameter region during optimization.

We compare two distinct outcomes of the trained network shown in Figure 4.4a. The first is *Linear*, where the network converges to an incorrect linear solution. In this scenario, the network simply connects the boundary conditions with a straight line, failing to form the steep boundary layer characteristic of the exact solution. The second is *Accurate Approximation*, where the network successfully captures the steep boundary layer near $x = 1$ and closely matches the exact solution. These two cases highlight the variability in network performance and the importance of initialization and input transformations in guiding the network toward the correct solution.

Figure 4.6 contrasts the weight magnitudes and effective shift values between a representative *Untransformed* network stuck in the linear solution and a *Transformed* network achieving the correct solution.

Key differences between the two cases can be summarized as follows. In the linear case, the values of $|W_1^i|$ cluster around small magnitudes, closer to 1, indicating that the network

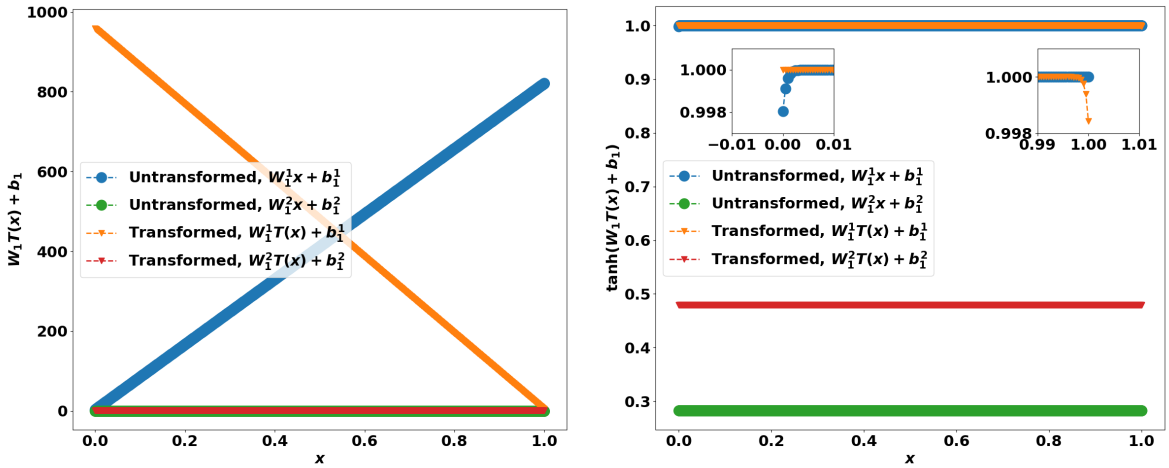
failed to develop sufficiently large weights to model the steep gradient. In contrast, the accurate approximation case exhibits at least one $|W_1^i|$ in the range of 10^2–10^3, which aligns with the requirement derived previously for resolving sharp boundary layers effectively.

Additionally, the effective shift values differ significantly between the two cases. For the linear case, the shifts are widely dispersed and centered closer to 0, reflecting the network's inability to align its sensitivity region with the location of the steep boundary layer. On the other hand, in the accurate case, the shifts are tightly centered around -1, which corresponds to the transformed input and ensures that the network's activation function is most sensitive near the boundary layer at $x = 1$.

This comparison suggests that networks converging to linear solutions fall into a "local minimum" where both the weights and shifts remain too small to enable steep gradient formation. In contrast, networks that escape this local minimum—via input shifting or other mechanisms—develop weight and shift distributions more aligned with the necessary conditions derived earlier. This evidence further supports the hypodissertation that input transformations can steer the network away from undesirable parameter regions and promote convergence to accurate approximations.

4.4.1.1 A Closer Look: Affine Transformations in a Width-2 Network

To further elucidate how the distribution of weights and biases affects boundary layer approximation, we narrow our attention to a simple yet illustrative case: a network with a single hidden layer and width 2. This minimalist setup allows us to directly visualize the effect of the affine transformation ($xW_1 + b_1$ or $(x-1)W_1 + b_1$) and its interaction with the activation function. The *Transformed* network produced an accurate approximation, while the *Untransformed* network



(a) Components of $T(x)W_1 + b_1$

(b) Components of $\tanh(T(x)W_1 + b_1)$

Figure 4.7: For a network with one hidden layer of width 2, in the first hidden layer, the output after the affine transformation is $xW_1 + b_1$ (or $(x-1)W_1 + b_1$), a matrix of two columns. The two columns of the output of the affine transformation are shown in 4.7a. The two columns of the output of $\tanh(xW_1 + b_1)$ (or $\tanh((x-1)W_1 + b_1)$) are shown in 4.7b. Two inset figures show a zoomed-in look near $y = 1$ at $x = 0$ and $x = 1$.

.

converged to an approximation into the wrong flow direction.

For a network of one hidden layer of width 2, we have $W_1 \in \mathbb{R}^{1\times 2}$, $b_1 \in \mathbb{R}^{1\times 2}$, $W_2 \in \mathbb{R}^{2\times 1}$, $b_2 \in \mathbb{R}$. The network can be written as

$$c_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{2} W_2^i \tanh(xW_1^i + b_1^i) + b_2. \tag{4.14}$$

Figure 4.7 shows the output of the first layer of the trained networks, i.e. $xW_1 + b_1$ for the *Untransformed* network and $T(x)W_1 + b_1$ for the *Transformed network*.

After training, we observe that in both the *Transformed* and *Untransformed* cases, one column of W_1 tends to have a large magnitude (e.g., $\sim 10^2$ or larger) while the other remains close to zero. This pattern leads to an affine transformation where one neuron generates a steep linear function, while the other contributes a nearly constant value across $x \in [0, 1]$. For example, for

the transformed case, $W_1^1 x + b_1^1$ is a line with a negative slope that varies from 0 to around 900 in a span of $x \in [0, 1]$, while $W_1^2 x + b_1^2$ is a line with a slope near zero that stays close to the x axis.

Figure 4.7a shows the output of $xW_1 + b_1$ or $(x-1)W_1 + b_1$ before activation. The neuron associated with the large weight creates a steeply sloped line that crosses zero either near $x = 0$ (for untransformed inputs) or $x = 1$ (for shifted inputs). This zero-crossing location is crucial, as $\tanh(\cdot)$'s sensitivity is highest near zero.

After applying the activation, as seen in Figure 4.7b, the steep line translates into a saturated $\tanh$ output across most of the domain around 1, except near the zero-crossing where a localized steep gradient emerges. This mechanism allows the network to form a sharp boundary layer aligned with the boundary where the zero-crossing occurs. The two inset figures in Figure 4.7b show a zoomed-in look of $\tanh(xW_1 + b_1)$ and $\tanh(T(x)W_1 + b_1)$ near the largest values on the y axis at $x = 0$ and $x = 1$. The gradients near the two boundaries are stretched into steep gradients shown in Figure 4.4b after passing through the affine transformation in the output layer. Note that the affine transformation in the output layer does not change the character of the first layer output. Steep boundary layers cannot form if gradients are not seen in the output of the first layer. These observations explained the formation of distributions shown in Figure 4.5.

Thus, the *Transformed* network, by shifting the input, ensures that this steep activation gradient appears near $x = 1$, the correct location of the boundary layer. Conversely, the *Untransformed* network often places this sensitive region near $x = 0$, where no boundary layer exists, leading to spurious solutions.

4.4.2 Necessary Conditions for Accurate Approximation: Steep First-Layer Weights

We now aim to investigate the necessary conditions on weights and biases that enable RPINNs to approximate solutions of the convection-diffusion equation, connecting this analysis to our previous discussion of input shifting and network sensitivity. While earlier sections revealed that input transformations help networks resolve steep boundary layers, we now shift focus to understanding how network size and structure contribute to this capability.

Recall that the residual of $c_{\boldsymbol{\theta}}$ is

$$-\epsilon c_{\boldsymbol{\theta}}'' + c_{\boldsymbol{\theta}}' = r. \tag{4.15}$$

When the residual $r = 0$, $c_{\boldsymbol{\theta}}$ satisfies

$$\frac{c_{\boldsymbol{\theta}}''}{c_{\boldsymbol{\theta}}'} = \frac{1}{\epsilon}. \tag{4.16}$$

We begin by analyzing the simplest case: a network with one hidden layer and width one ($d = 1$) in (4.14). The derivatives of $c_{\boldsymbol{\theta}}$ with respect to x are straightforward in this case, for $x \in [0, 1]$,

$$c_{\boldsymbol{\theta}}' = W_2 W_1 \operatorname{sech}^2 (x W_1 + b_1),$$

$$c_{\boldsymbol{\theta}}'' = -2 W_2 (W_1)^2 \tanh(x W_1 + b_1) \operatorname{sech}^2(x W_1 + b_1).$$

If $c'_{\boldsymbol{\theta}} \neq 0$, then

$$\frac{c''_{\boldsymbol{\theta}}}{c'_{\boldsymbol{\theta}}}| = -2W_1 \tanh(xW_1 + b_1).$$

Setting this equal to $\frac{1}{\epsilon}$ from (4.16), we find

$$-2W_1 \tanh(xW_1 + b_1) = \frac{1}{\epsilon}.$$

Since $|\tanh(x)| < 1$ for all $x \in \mathbb{R}$, we immediately obtain $|-2W_1 \tanh(xW_1 + b_1)| < 2|W_1|$. By triangle inequality,

$$2|W_1| > \frac{1}{\epsilon}, \tag{4.17}$$

as $\epsilon > 0$, which indicates that $|W_1|$ must grow large as $\epsilon \to 0$. This aligns with our earlier observation from Figure 4.7, where $|W_1|$ is approximately 10^3 when $\epsilon = 10^{-4}$.

Extending to a one-hidden-layer network with width $d_1 > 1$, for $x \in [0, 1]$,

$$c_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{d_1} W_2^i \tanh(xW_1^i + b_1^i) + b_2,$$

where the subscripts indicate the layer index and the superscripts indicate the entry index. The network derivatives are

$$\begin{aligned} c'_{\boldsymbol{\theta}} &= \sum_{i=1}^{d_1} W_2^i \operatorname{sech}^2(xW_1^i + b_1^i)(W_1^i), \\ c''_{\boldsymbol{\theta}} &= -2\sum_{i=1}^{d_1} W_2^i \tanh(xW_1^i + b_1^i) \operatorname{sech}^2(xW_1^i + b_1^i)(W_1^i)^2. \end{aligned}$$

Thus,

$$\frac{c''_{\boldsymbol{\theta}}}{c'_{\boldsymbol{\theta}}} = \frac{-2\sum_{i=1}^{d_1} W_2^i \tanh(xW_1^i + b_1^i)\,\mathrm{sech}^2(xW_1^i + b_1^i)(W_1^i)^2}{\sum_{i=1}^{d_1} W_2^i\,\mathrm{sech}^2(xW_1^i + b_1^i)(W_1^i)}$$

We recognize this as a weighted average:

$$\frac{c''_{\boldsymbol{\theta}}}{c'_{\boldsymbol{\theta}}} = -2 \cdot \frac{\sum_{i=1}^{d_1} W_1^i \tanh(xW_1^i + b_1^i)\alpha_i}{\sum_{i=1}^{d_1} \alpha_i}, \quad \text{with} \quad \alpha_i = W_2^i W_1^i\,\mathrm{sech}^2(xW_1^i + b_1^i). \tag{4.18}$$

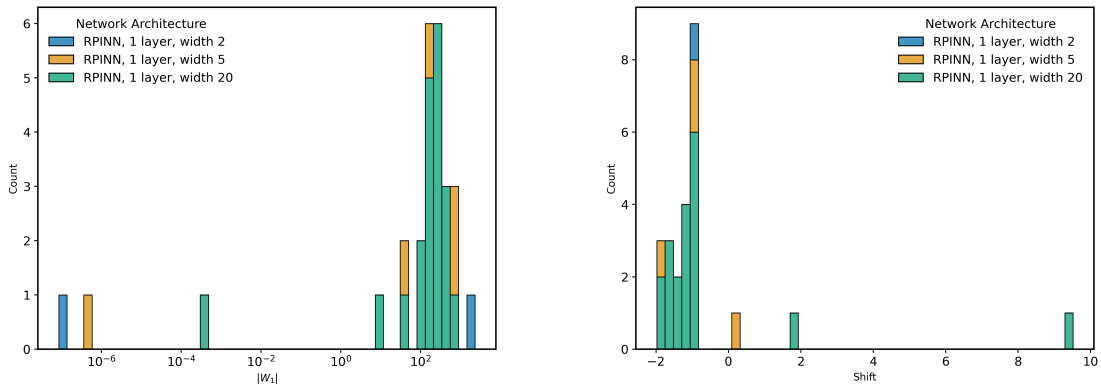
The weighted average should also satisfy (4.16), i.e.,

$$\frac{1}{\epsilon} = \left|\frac{c''_{\boldsymbol{\theta}}}{c'_{\boldsymbol{\theta}}}\right| = 2\left|\frac{\sum_{i=1}^{d_1} W_1^i \tanh(xW_1^i + b_1^i)\alpha_i}{\sum_{i=1}^{d_1} \alpha_i}\right| \tag{4.19}$$

From (4.19), we can infer that the magnitude of W_1^i must collectively be large. Since both $\tanh$ and sech^2 are bounded, the only way to scale the weighted average with $\mathcal{O}(1/\epsilon)$ is through scaling W_1. The weighted average also suggests that larger networks distribute the requirement shown in (4.16) across multiple neurons. In a wider network, each W_1^i does not individually need to be of $\mathcal{O}(1/\epsilon)$, but collectively their contributions must sum up to this scale.

The W_2^i coefficients modulate how strongly each neuron's contribution is weighted within the network's combined output. Specifically, neurons with large $|W_2^i|$ exert greater influence on the weighted average, allowing the network to collectively satisfy (4.16) without relying solely on extreme values of any single W_1^i. However, W_2^i alone cannot induce steep gradients—it can only amplify or attenuate the effect introduced by W_1^i. Therefore, the critical role of W_1^i remains as the primary source of the steepness in $\tanh(xW_1^i + b_1^i)$.

This analysis helps explain why wider networks—with more neurons—are able to distribute



(a) Histogram of $|W_1^i|$

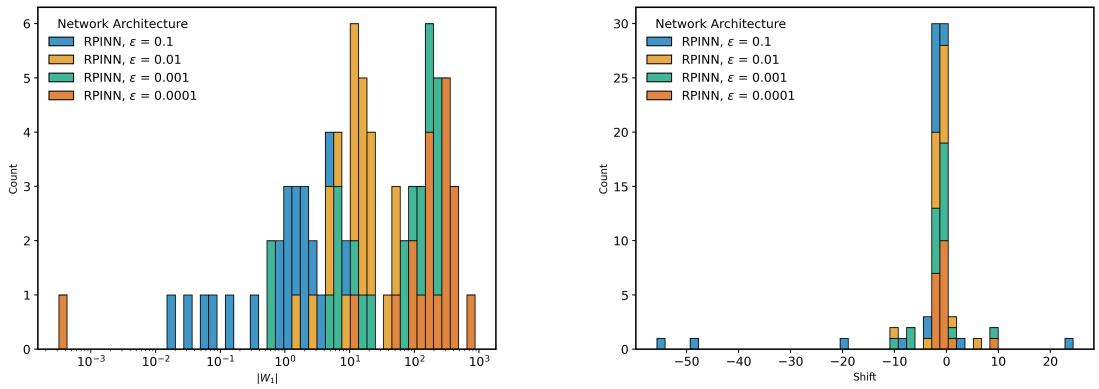
(b) Histogram of effective shift

Figure 4.8: Histograms of $|W_1^i|$ and shift values for networks with one hidden layer of width 2, 5, and 20.

the approximation task more effectively, improving their capacity to resolve steep gradients inherent to singularly perturbed problems.

We can verify our hypodissertation by observing the distributions of $|W_1^i|$ and the shift values empirically. We trained *Transformed* networks with one hidden layer of width 2, 5, and 20 following the same procedures described earlier. The resulting histograms are shown in Figure 4.8. As the network widens, the peaks of $|W_1^i|$ incrementally decrease, partially confirming that wider networks require smaller individual $|W_1^i|$ to achieve the same effect. However, for the network with width 20, the peak is still around 10^2, smaller than $\mathcal{O}(1/\epsilon)$ but still some distance from initialization. Meanwhile, the shift values consistently center around -1, as expected from our earlier activation analysis. Notably, in the network of width 2, one of the $|W_1^i|$ values indeed lies around 10^3, consistent with the requirement imposed by (4.17). This empirical evidence reinforces our analytical findings and highlights how network width impacts the distribution of weights needed to resolve steep boundary layers.

We would also like to verify that $|W_1^i|$ should distribute around $\mathcal{O}(1/\epsilon)$ for networks with one



(a) Histogram of $|W_1^i|$

(b) Histogram of effective shift

Figure 4.9: Histograms of $|W_1^i|$ and shift values for networks with one hidden layer of width 20 for $\epsilon = 10^{-j}$, $j = 1, 2, 3, 4$.

hidden layer. We trained networks with one hidden layer of width 20 for $\epsilon = 10^{-j}$, $j = 1, 2, 3, 4$. The distributions of $|W_1^i|$ and the shift are shown in Figure 4.9. The peaks of $|W_1^i|$ increases as ϵ decreases, and the spread of the shift values still centers around -1, generally aligning with our analysis. Although distributions of $|W_1^i|$ do not peak exactly at $\mathcal{O}(1/\epsilon)$, but the trend is clear. We also take imperfect residual fitting and network width into consideration.

The analysis becomes complicated for more hidden layers, but we observe the same trend. For example, suppose $c_{\boldsymbol{\theta}}$ is a network with two hidden layers, i.e., for $x \in [0, 1]$,

$$c_{\boldsymbol{\theta}} = \sum_{j}^{d_2} W_3^j \tanh\left(\sum_{i}^{d_1} W_2^{i,j} \tanh(xW_1^i + b_1^i) + b_2^j\right) + b_3.$$

Taking derivatives gives

$$c'_{\boldsymbol{\theta}}(x) = \sum_{j=1}^{d_2} W_3^j \operatorname{sech}^2\left(z_j(x)\right) \cdot \sum_{i=1}^{d_1} W_2^{i,j} W_1^i \operatorname{sech}^2\left(xW_1^i + b_1^i\right),$$

$$c''_{\boldsymbol{\theta}}(x) = \sum_{j=1}^{d_2} W_3^j \Bigg[-2\operatorname{sech}^2(z_j(x))\tanh(z_j(x)) \left(\sum_{i=1}^{d_1} W_2^{i,j} W_1^i \operatorname{sech}^2(xW_1^i + b_1^i) \right)^2 + \operatorname{sech}^2(z_j(x)) \sum_{i=1}^{d_1} W_2^{i,j} (W_1^i)^2 \operatorname{sech}^2(xW_1^i + b_1^i) \tanh(xW_1^i + b_1^i) \Bigg].$$

where we define $z_j(x) = \sum_{i=1}^{d_1} W_2^{i,j} \tanh(xW_1^i + b_1^i) + b_2^j$.

Thus, the ratio $\frac{c''_{\boldsymbol{\theta}}(x)}{c'_{\boldsymbol{\theta}}(x)}$ can be expressed as:

$$\frac{c''_{\boldsymbol{\theta}}(x)}{c'_{\boldsymbol{\theta}}(x)} = \frac{\sum_{j=1}^{d_2} W_3^j \operatorname{sech}^2\left(z_j(x)\right)\left[-2\tanh(z_j(x))A_j(x)^2 + B_j(x)\right]}{\sum_{j=1}^{d_2} W_3^j \operatorname{sech}^2\left(z_j(x)\right) A_j(x)}, \tag{4.20}$$

where:

$$A_j(x) = \sum_{i=1}^{d_1} W_2^{i,j} W_1^i \operatorname{sech}^2\left(xW_1^i + b_1^i\right),$$

$$B_j(x) = \sum_{i=1}^{d_1} W_2^{i,j} (W_1^i)^2 \operatorname{sech}^2\left(xW_1^i + b_1^i\right) \tanh\left(xW_1^i + b_1^i\right).$$

Relating (4.20) to (4.16), we analyze the roles of W_1, W_2, and W_3 in shaping the boundary layer. W_1 appears in both $A_j(x)$ and $B_j(x)$ through terms such as W_1^i and $(W_1^i)^2$. The magnitude of W_1 scales the ratio. Thus, large $|W_1^i|$ are directly responsible for generating sharp gradient layers, similar to the one-hidden-layer case. In deeper networks, W_1 still acts as the primary source for steepness, as $A_j(x)$ inherits W_1-dependent scaling.

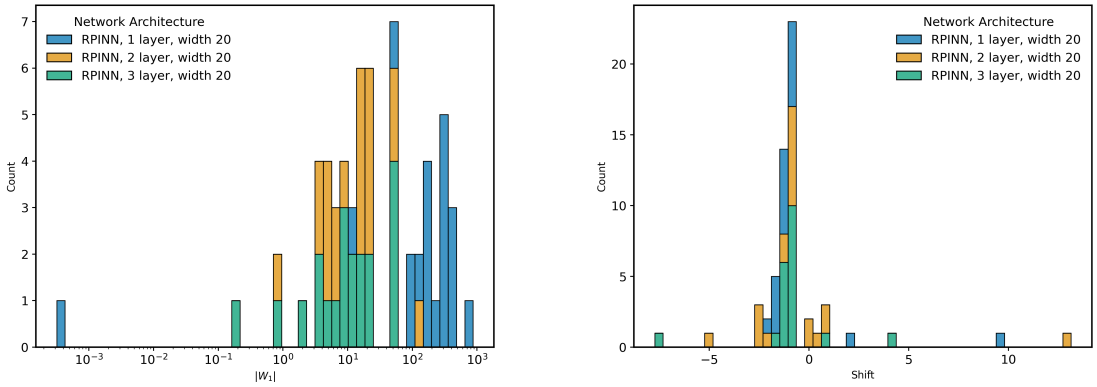
The W_2^i coefficients control how much each neuron's steepness, induced by W_1^i, contributes

to the overall network output. In both $A_j(x)$ and $B_j(x)$, $W_2^{i,j}$ scales the effective contribution of each neuron in the first hidden layer. Larger $|W_2^i|$ amplify the steep gradients produced by corresponding W_1^i, whereas smaller $|W_2^i|$ diminish their effect. Networks with wider layers (d_1 large) can distribute this burden across multiple neurons, allowing $|W_1^i|$ to be smaller while still achieving the desired boundary layer behavior. Thus, W_2 acts as a facilitator, controlling how W_1's influence is passed forward.

W_3 serves as an aggregator across the second hidden layer, weighting how each $z_j(x)$ contributes to the overall ratio. Unlike W_1 and W_2, W_3 does not generate steep gradients directly but modulates how the contributions from earlier layers are combined. Specifically, W_3 impacts whether certain neurons in the second hidden layer dominate the final output or whether the effect is distributed across multiple j indices.

To summarize, the steepness of the boundary layer is predominantly controlled by W_1, while W_2 and W_3 work collaboratively to distribute and amplify W_1's effect throughout the network. As the depth increases, the network has additional flexibility to balance these contributions, although this comes at the cost of increased complexity and potential training difficulties. The analysis of weights and biases in networks with more than two hidden layers follows a similar pattern. The weights in the first layer play a crucial role in generating steep gradients, while the weights in the subsequent hidden layers help distribute the task of approximating these gradients. The output layer then combines the contributions from the hidden layers. Consequently, increasing the number of hidden layers is not necessarily needed improve the network's ability to capture steep gradient layers. The key to accurately approximating steep gradient layers remains primarily in the first layer.

To further validate our hypodissertation, we examined the distributions of $|W_1^i|$ and the shift



(a) Histogram of $|W_1^i|$

(b) Histogram of first-layer effective shift

Figure 4.10: Histograms of $|W_1^i|$ and shift values for networks with one, two, and three hidden layers of width 20.

for networks with two and three hidden layers under the *Transformed* setup and $\epsilon = 0.0001$. The results are shown in Figure 4.10. We observe that, while the shift values consistently center around -1 across all networks, the $|W_1^i|$ distributions differ notably between the one-layer network and the deeper two- and three-layer networks.

Specifically, the two- and three-layer networks exhibit very similar $|W_1^i|$ distributions, suggesting that the additional third layer does not significantly change the behavior compared to the two-layer setup. This similarity confirms our earlier analytical insight: deeper networks share the steep gradient approximation task across multiple layers, thereby reducing the need for extreme magnitudes in W_1^i. The W_2^i coefficients determine how strongly each neuron's steepness, induced by W_1^i, contributes to the overall network output. Larger $|W_2^i|$ amplify steep gradients from W_1^i, while smaller $|W_2^i|$ reduce their effect.

The histogram in Figure 4.11 shows the distribution of $|W_2^i|$ values for networks with two and three hidden layers. We observe that $|W_2^i|$ values are broadly spread but similarly distributed between the two architectures, reinforcing the notion that W_2's role is to aggregate and balance

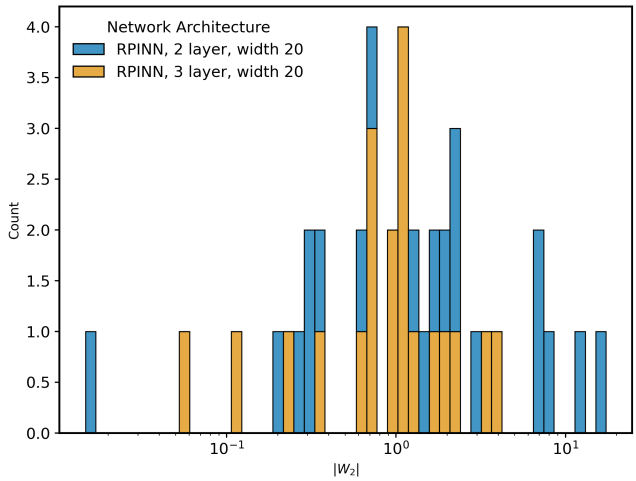


Figure 4.11: Histogram of $|W_2^{i,j}|$ for networks with two and three hidden layers of width 20.

the steepness contributions from W_1. The similarity of these distributions further supports our conclusion that networks deeper than two layers do not substantially alter how steep gradients are aggregated, consistent with the earlier insights drawn from W_1 distributions.

Additionally, this observation indicates diminishing returns from adding layers beyond two in this context. The empirical distributions imply that while increasing depth from one to two layers substantially alters how the network handles steep gradients, adding a third layer provides little additional benefit for resolving the boundary layer in the convection-diffusion equation. Thus, networks with two hidden layers already achieve an effective distribution of steepness-inducing weights across their depth.

Building on the previous insights regarding input shifting and weight distributions, we now explore an additional strategy to address the challenge of large-magnitude weights observed in the first layer. While shifting the input by -1 (i.e., transforming x to $x-1$) helped align the network's sensitivity with the boundary layer location at $x = 1$, we also observed that the network still requires large $|W_1^i|$ values to satisfy the residual condition $\frac{c''_{\boldsymbol{\theta}}}{c'_{\boldsymbol{\theta}}} = \frac{1}{\epsilon}$. To alleviate this burden, we

investigate the role of input stretching—applying a multiplicative scaling factor a to the input—on the performance of RPINNs.

Specifically, we introduce an additional transformation of the form $T(x) = a(x - b)$, combining both shifting and stretching. This modification aims to reduce the required magnitude of W_1^i by sharing the responsibility with the scaling factor a.

To illustrate the concept more clearly, we revisit the simple case of a network with one hidden layer and width one. Suppose the input x is transformed as $T(x) = a(x - b)$, where $x \in [0, 1]$ and $W_1, W_2, b_1, b_2 \in \mathbb{R}$. The network output is:

$$c_{\boldsymbol{\theta}}(T(x)) = \tanh\left(a(x-b)W_1 + b_1\right) W_2 + b_2. \tag{4.21}$$

The effective shift of (4.21) is $-b + \frac{b_1}{aW_1}$, which is predominantly controlled by $-b$ as aW_1 is typically large.

Taking derivatives yields:

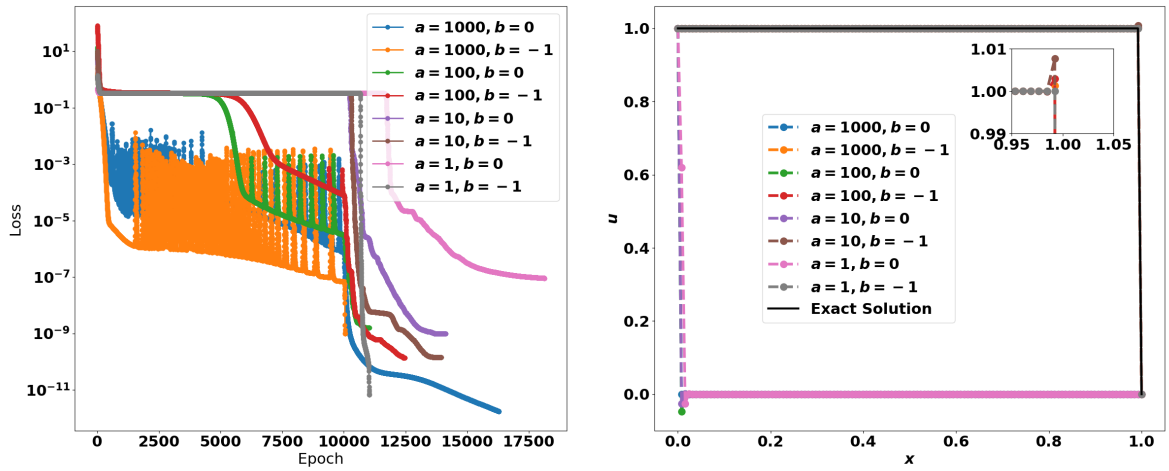
$$c'_{\boldsymbol{\theta}}(T(x)) = aW_1W_2 \operatorname{sech}^2(a(x-b)W_1 + b_1),$$

$$c''_{\boldsymbol{\theta}}(T(x)) = -2(aW_1)^2 W_2 \tanh(a(x+b)W_1 + b_1) \operatorname{sech}^2(a(x-b)W_1 + b_1).$$

Thus, for $c'_{\boldsymbol{\theta}}(x) \neq 0$ and $W_1, W_2 > 0$, we obtain

$$\frac{c''_{\boldsymbol{\theta}}(T(x))}{c'_{\boldsymbol{\theta}}(T(x))} \leq \frac{4}{3\sqrt{3}} aW_1. \tag{4.22}$$

This inequality highlights that, by choosing $a = \mathcal{O}(1/\epsilon)$, the network can satisfy $\frac{c''_{\boldsymbol{\theta}}(T(x))}{c'_{\boldsymbol{\theta}}(T(x))} = \frac{1}{\epsilon}$



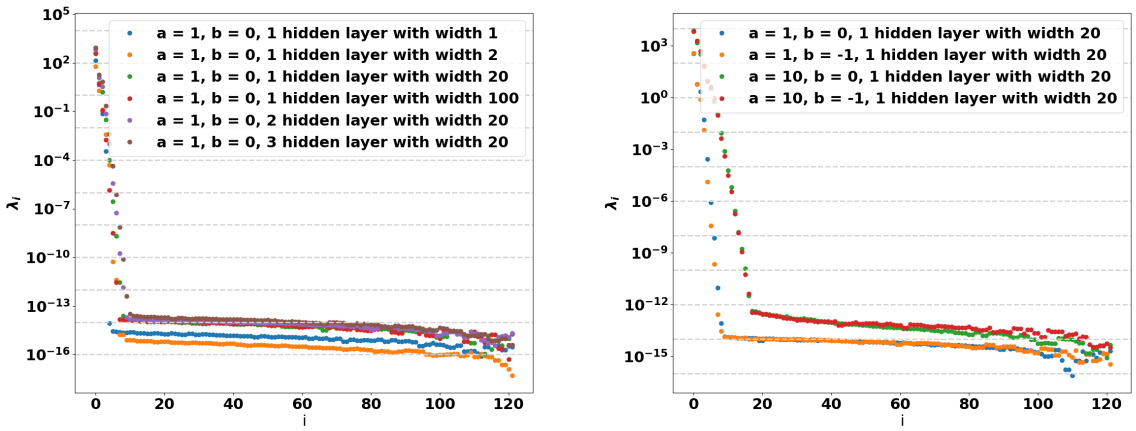
(a) (b)

Figure 4.12: Loss value during training for a network of one hidden layers of width 20 in Figure 4.12a. Corresponding approximations in Figure 4.12b. First 10000 epochs used Adam optimizer, followed by LBFGS optimizer. Here a and b are the linear transformation parameters in (4.21).

without requiring W_1 to be excessively large. Effectively, a acts as a scaling lever, reducing the load placed on W_1 to achieve sharp gradients near the boundary layer.

Figure 4.12a illustrates the empirical benefits of this approach. As a increases, networks converge to smaller loss values more quickly during the Adam optimization phase. However, it is important to note that regardless of the value of a, correct approximations were only consistently achieved when $b = -1$, ensuring that the network's sensitivity is aligned with the outflow boundary at $x = 1$.

In summary, input stretching via the scaling factor a complements input shifting and provides a practical mechanism to mitigate the large-weight issue in RPINNs, promoting faster and more reliable training in singularly perturbed problems.



(a) (b)

Figure 4.13: Eigenvalues of $\boldsymbol{K}$ for different network width and depth (4.13a), and for different transformation parameters a, b (4.13b).

4.4.3 Neural Tangent Kernel (NTK) Analysis

To deepen our understanding of RPINNs, we apply the recently developed Neural Tangent Kernel (NTK) theory [115], a framework that elucidates the training dynamics of neural networks in the infinite-width limit. We follow the work developed in [116] in the following discussion.

In [116], minimizing the loss function by gradient descent with an infinitesimally small learning rate yields the continuous time gradient flow system

$$\frac{d\theta}{dt} = -\nabla \mathcal{L}(\boldsymbol{\theta}). \tag{4.23}$$

Following the derivation of [115], the neural tangent kernel can be generally defined at any time t as the neural network parameters $\theta(t)$ vary during training by gradient descent. The kernel is defined by

$$\boldsymbol{K}_{ert}(x, x') = \left\langle \frac{\partial f(x, \boldsymbol{\theta}(t))}{\partial \boldsymbol{\theta}}, \frac{\partial f(x', \boldsymbol{\theta}(t))}{\partial \boldsymbol{\theta}} \right\rangle. \tag{4.24}$$

As a consequence of [115], a properly randomly initialized and sufficiently wide deep neural network trained by gradient descent is equivalent to a kernel regression with a deterministic kernel.

Let $u_s(t) = u_s(x_u, \boldsymbol{\theta}(t)) = \{u_s(x_u^i, \boldsymbol{\theta}(t))\}_{i=1}^{N_b}$ and $\mathcal{N}[u_s(t)] = \mathcal{N}[u_s(x_r, \boldsymbol{\theta}(t))] = \{\mathcal{N}[u_s(x_r^i, \boldsymbol{\theta}(t))]\}_{i=1}^{N_r}$. By [116], with gradient flow in (4.23), $u_s(t)$ and $\mathcal{N}[u_s(t)]$ obey the following evolution

$$\begin{bmatrix} \frac{du_s(x_u, \boldsymbol{\theta}(t))}{dt} \\ \frac{d\mathcal{N}u_s(x_r, \boldsymbol{\theta}(t))}{dt} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{K}_{uu}(t) & \boldsymbol{K}_{ur}(t) \\ \boldsymbol{K}_{ru}(t) & \boldsymbol{K}_{rr}(t) \end{bmatrix} \begin{bmatrix} u_s(x_u, \boldsymbol{\theta}(t)) - u_{bc} \\ \mathcal{N}[u_s(x_r, \boldsymbol{\theta}(t))] - f(x_r) \end{bmatrix}, \tag{4.25}$$

where $\boldsymbol{K}_{ru}(t) = \boldsymbol{K}_{ur}^T(t)$, $\boldsymbol{K}_{uu}(t) \in \mathbb{R}^{N_b \times N_b}$, $\boldsymbol{K}_{ur}(t) \in \mathbb{R}^{N_b \times N_r}$, and $\boldsymbol{K}_{rr}(t) \in \mathbb{R}^{N_r \times N_r}$ are given by

$$\begin{aligned} (\boldsymbol{K}_{uu})_{ij}(t) &= \Big\langle \frac{du(x_u^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{du(x_u^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \Big\rangle \\ (\boldsymbol{K}_{ur})_{ij}(t) &= \Big\langle \frac{du(x_u^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{N}[u(x_r^j, \boldsymbol{\theta}(t))]}{d\boldsymbol{\theta}} \Big\rangle \\ (\boldsymbol{K}_{rr})_{ij}(t) &= \Big\langle \frac{d\mathcal{N}[u(x_r^i, \boldsymbol{\theta}(t))]}{d\boldsymbol{\theta}}, \frac{d\mathcal{N}[u(x_r^j, \boldsymbol{\theta}(t))]}{d\boldsymbol{\theta}} \Big\rangle, \end{aligned} \tag{4.26}$$

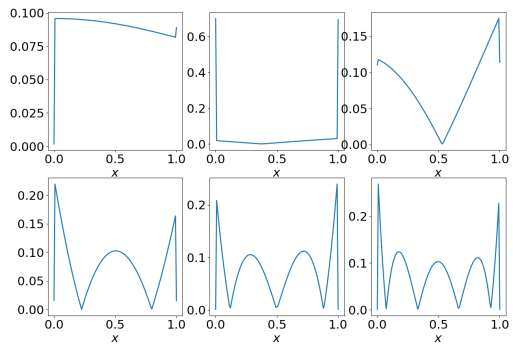
where $\langle \cdot, \cdot \rangle$ denotes the inner product over all trainable parameters in $\boldsymbol{\theta}$. For example,

$$(\boldsymbol{K}_{uu})_{ij}(t) = \sum_{\theta \in \boldsymbol{\theta}} \frac{du(x_u^i, \boldsymbol{\theta}(t))}{d\theta} \cdot \frac{du(x_u^j, \boldsymbol{\theta}(t))}{d\theta}.$$

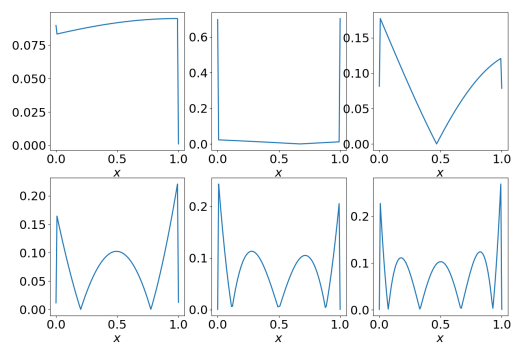
These kernel matrices form a kernel matrix

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{uu}(t) & \boldsymbol{K}_{ur}(t) \\ \boldsymbol{K}_{ru}(t) & \boldsymbol{K}_{rr}(t) \end{bmatrix}. \tag{4.27}$$

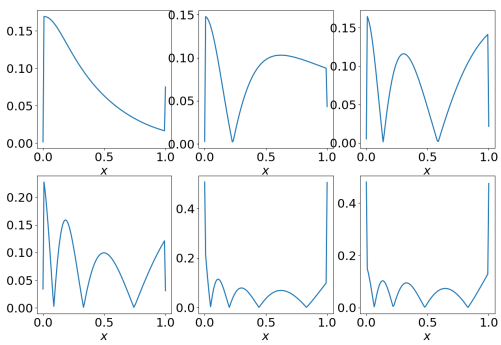
Let $N = N_b + N_r$. By construction, $\boldsymbol{K}$ is a symmetric positive semi-definite matrix of size $N \times N$.



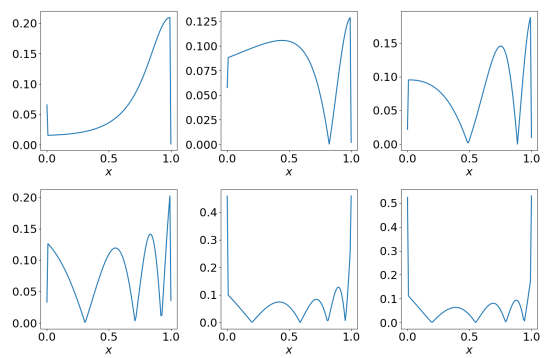
(a) $a = 1, b = 0$



(b) $a = 1, b = -1$



(c) $a = 10, b = 0$



(d) $a = 10, b = -1$

Figure 4.14: Eigenvectors corresponding to the 6 largest eigenvalues of $\boldsymbol{K}$, for eigenvalues decreasing from left to right, top to bottom. Fig 4.14a: the eigenvectors for $a = 1, b = 0$. Fig 4.14b: the eigenvectors for $a = 1, b = -1$. Fig 4.14c: the eigenvectors for $a = 10, b = 0$. Fig 4.14d: the eigenvectors for $a = 10, b = -1$

In addition, the NTK of PINNs converge in probability to a deterministic limiting kernel when the network width $d \to \infty$ [116]. Thus, it is enough to study the behavior of $\boldsymbol{K}$ at initialization. Consider the eigen-decomposition of $\boldsymbol{K}$ at initialization, $\boldsymbol{K} = \boldsymbol{Q}^T \Lambda \boldsymbol{Q}$, where $\boldsymbol{Q}$ is an orthogonal matrix whose columns are eigenvectors of $\boldsymbol{K}$ and Λ is a diagonal matrix whose entries are the corresponding eigenvalues $\lambda_i \geq 0$ of $\boldsymbol{K}$. As shown in [116],

$$\boldsymbol{Q}\left(\begin{bmatrix} \frac{du(x_u, \boldsymbol{\theta}(t))}{dt} \\ \frac{d\mathcal{N}[u(x_r, \boldsymbol{\theta}(t))]}{dt} \end{bmatrix} - \begin{bmatrix} u_{bc} \\ f(x_r) \end{bmatrix}\right) \approx -e^{-\Lambda t}\boldsymbol{Q} \cdot \begin{bmatrix} u_{bc} \\ f(x_r) \end{bmatrix} \tag{4.28}$$

Thus, the i-th component of the left-hand side of (4.28) will decay approximately at the rate of $e^{-\lambda_i t}$. Notice that the left hand side is the evolution of the absolute training error mapped by Q. The components of the target function that correspond to kernel eigenvectors with larger eigenvalues will be learned faster. One can gain insight into the decay process of the training errors by analyzing $\boldsymbol{K}$.

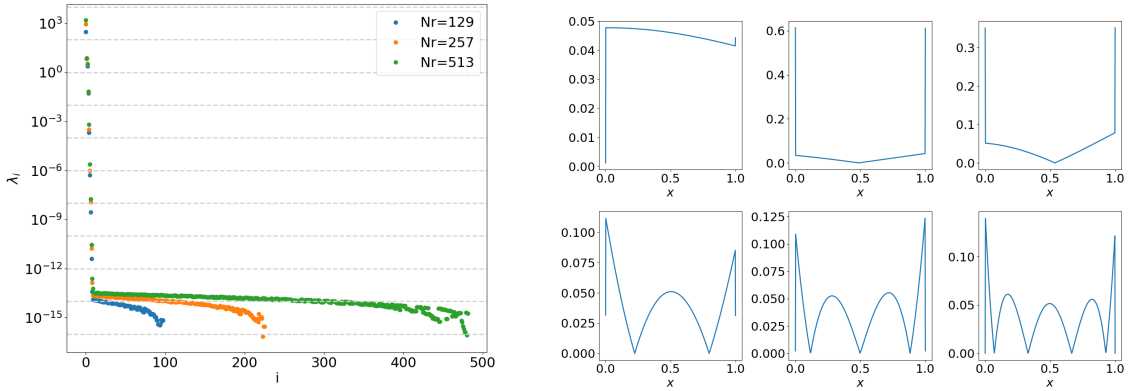
From (4.28), the training errors decay exponentially at a rate determined by the eigenvalues in Λ. Let the average convergence rate c be defined as the mean of all the eigenvalues λ_i, i.e.

$$c = \frac{\sum_{i=1}^{N} \lambda_i}{N} = \frac{Tr(\boldsymbol{K})}{N}. \tag{4.29}$$

The convergence rate measures the how fast the training errors converge to 0. In particular, [116] shows that $Tr(\boldsymbol{K}) = Tr(\boldsymbol{K}_{uu}) + Tr(\boldsymbol{K}_{rr})$, and it follows that $c = \frac{Tr(\boldsymbol{K}_{uu})+Tr(\boldsymbol{K}_{rr})}{N}$. If $Tr(\boldsymbol{K}_{rr}) \gg Tr(\boldsymbol{K}_{uu})$, the surrogate neural network converges to the PDE governing equation faster than to the boundary conditions. In practice, without input transformation, we found experimentally that $Tr(\boldsymbol{K}_{rr}) \gg Tr(\boldsymbol{K}_{uu})$ at initialization as $Tr(\boldsymbol{K}_{rr})/Tr(\boldsymbol{K}_{uu}) \approx 50$, regardless of the network depth or width.

When observing the eigenvalues of $\boldsymbol{K}$, we notice that there are only a few eigenvalues of large magnitude. The situation does not change much when network width or depth changes. Figure 4.13a shows that even for deeper and wider networks, the majority of the eigenvalues are close to zero.

When the linear transformation is applied to the inputs, as shown in Figure 4.13b the eigenvalues of $\boldsymbol{K}$ are similar in magnitude for different values of b. As a increases, the magnitude of eigenvalues increases, which indicates a higher convergence rate. This observation is consistent



(a) (b)

Figure 4.15: 4.15a Eigenvalues of $\boldsymbol{K}$ for different residual sample sizes . 4.15b Eigenvectors of $\boldsymbol{K}$ corresponding to the six largest eigenvalues for sample size 513.

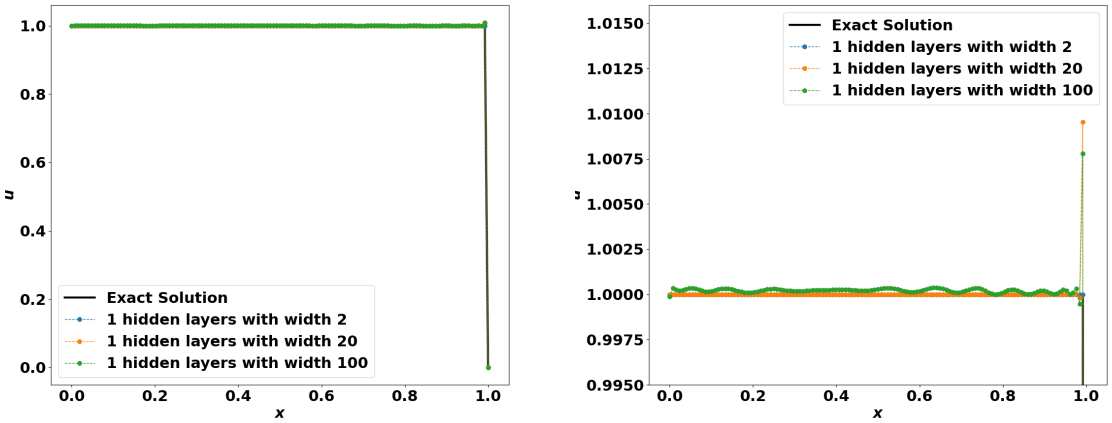
with the faster convergence of loss values for larger a shown in Figure 4.12a. Note that for larger values of a, 1000 for example, the loss values oscillate. This is due to a fixed learning rate applied in Adam optimizations. The oscillatory behavior could be reduced with a decaying learning rate.

The eigenvalues only indicate the convergence rate. But it is still unclear why different values of b lead networks to different approximations. To answer this question, we need to examine the eigenvectors of $\boldsymbol{K}$. Since a majority of the eigenvalues are close to zero, we consider 6 eigenvectors corresponding to the largest 6 eigenvalues. These are plotted in Figure 4.14, which shows a clear distinction between the case of $a = 1, b = 0$ and the case of $a = 1, b = -1$. This distinction becomes more dramatic when $a = 10$. Figure 4.14c suggests that for $b = 0$, the eigenvectors corresponding to the six largest eigenvalues themselves tend to have boundary layers near $x = 0$, whereas for $b = -1$, these eigenvectors tend to form boundary layers near $x = 1$. Since these eigenvectors represent the components in the residual that will decay the fastest, the boundary layers in the top eigenvectors will appear in the approximations obtained by the network. As a result, RPINN approximations will attain boundary layers near $x = 0$ for $b = 0$ and near

$x = 1$ for $b = -1$.

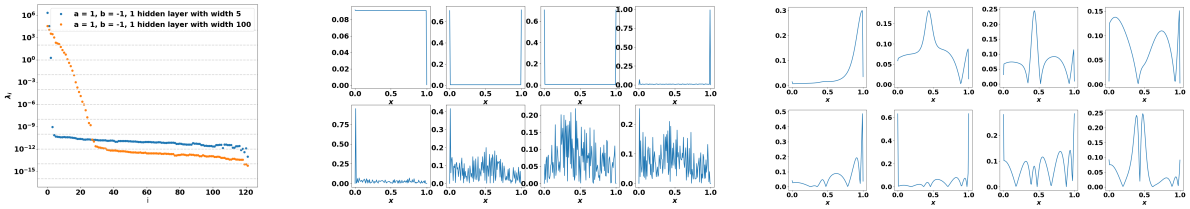
To validate the effect of input transformations, we need to be certain that other factors, such as the network size and residual sample size, do not play a vital role. We examine the eigenvalues of $\boldsymbol{K}$ with different residual sample size $N_r = 129, 257, 513$. As seen in Figure 4.15, the eigenvalues of $\boldsymbol{K}$ do not vary much for different sample sizes and the eigenvectors exhibit similar structures, which translates to the observation that providing more samples does not benefit the training or the accuracy of approximations.

The width of networks is a tricky subject. Using transformation $x \to x - 1$, we fixed the residual sample size to 129 and trained networks of three different widths. Note that the analytic form yielded that networks with larger width could attain the steep gradient more effectively. But the story took a turn. As shown in Figure 4.16, the approximations made with network structure of width 2 actually produced the most accurate surrogate exact solution. The network of width 20 started to produce jumps near the steep gradient layer, and the network of width 100 starts to exhibit oscillations throughout the domain. This behavior is caused by the fact that networks of larger width contain higher frequency modes in their eigenvectors of $\boldsymbol{K}$, as shown in Figure 4.17. We examine the eigenvectors of $\boldsymbol{K}$ after training to illustrate the different modes the networks converged to. It is clear that the eigenvectors of the network with width 100 display oscillations in the eigenvectors corresponding to the eight largest eigenvalues. The oscillations in the eigenvectors explain the oscillations in the approximations shown in Figure 4.16. From Figure 4.17a, one can see that only the largest 4 eigenvalues of $\boldsymbol{K}$ for the network of one hidden layer with width 5 are larger than 10^{-9}, whereas all of the largest 7 eigenvalues of $\boldsymbol{K}$ for the network of width 100 are larger than 10^{2}. Thus, the eigenvectors with noisy oscillations corresponding to the fourth to eighth largest eigenvalues in Figure 4.17b are not significant. The oscillations in the eigenvectors



(a) (b)

Figure 4.16: 4.16a Approximations produced by trained networks of different size. 4.16b Zoom in view near $u = 1$.

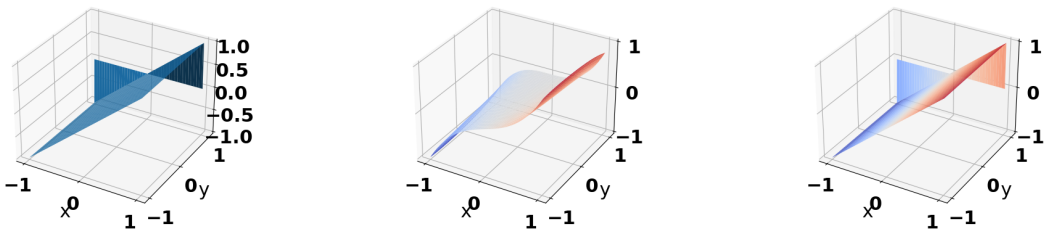


(a) (b) (c)

Figure 4.17: Eigenvalues and eigenvectors corresponding to the 8 largest eigenvalues of $\boldsymbol{K}$ for $a = 1, b = -1$ after training. Eigenvalues decrease from left to right, top to bottom. Fig 4.17a: the eigenvalues of NTK for networks of 1 hidden layer with width 5 and 100, respectively; Fig 4.17b: the eigenvectors for a network of 1 hidden layer with width 5, Fig 4.17c: the eigenvectors for a network of 1 hidden layer with width 100.

in Figure 4.17c are meaningful.

Having analyzed the effectiveness of RPINNs and input transformations in one-dimensional settings, we now extend our investigation to a two-dimensional convection-diffusion equation with a known boundary layer.



(a) (b) (c)

Figure 4.18: Solution of (4.30) for $\epsilon = 1e-4$ in 4.18a. Approximations by PINN in 4.18b and 4.18c. The input transformations are for 4.18b: no transformation; and for 4.18c: $(x, y) \to (x, 10(y-1))$.

4.5 The Two-Dimensional Convection-Diffusion Equation

We can extend our understanding of the performance of RPINN to a simple two-dimensional case, where there is only one steep gradient boundary layer in the domain. Consider the following equation:

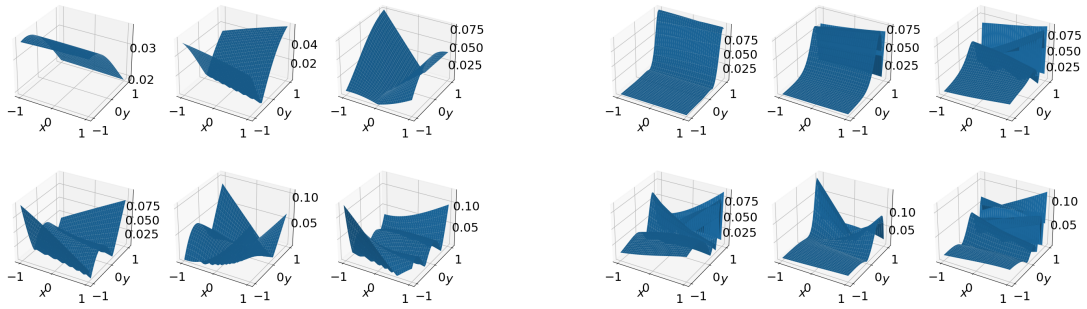
$$
\begin{aligned}
-\epsilon \Delta u + \vec{\omega} \cdot \nabla u = 0, \quad & \text{for } (x, y) \in (-1, 1) \times (-1, 1) \\
u(x, -1) = x, \quad & u(x, 1) = 0, \\
u(-1, y) \approx -1, \quad & u(1, y) \approx 1,
\end{aligned}
\tag{4.30}
$$

where $\vec{\omega} = (0, 1)$. The exact solution of (4.30) is

$$
u(x, y) = x \left(\frac{1 - e^{(y-1)/\epsilon}}{1 - e^{-2/\epsilon}} \right)
$$

The reduced solution is

$$
u(x, y) = x
$$



(a) (b)

Figure 4.19: Eigenvectors corresponding to the 6 largest eigenvalues of $\boldsymbol{K}_{rr}$. Eigenvalues decrease from left to right, top to bottom. Fig 4.19a: the eigenvectors for no transformation. Fig 4.19b: the eigenvectors for transformation $(x, y) \to (x, 10(y-1))$.

The solution has a single boundary layer at the outflow boundary $y = 1$, as seen in Figure 4.18a.

We fix the network structure to one hidden layer of width 5. We select residual samples as an equispaced mesh of size 129 by 129 over $[-1, 1] \times [-1, 1]$. Then we train the networks with different input transformation settings. It is evident that with the input transformations, the networks were able to capture the steep boundary layer at $y = 1$ more effectively.

We examine the behavior of eigenvectors of NTK of the two-dimensional problem to confirm the effect of input transformation. In terms of one-dimensional cases, when we perform input transformation $(x, y) \to (x, 10(y-1))$, the first eigenvector forms a steep boundary layer near $y = 1$. As a result, approximations by RPINNs are more accurate with such input transformation.

4.6 Discussion

In this chapter, we studied the performance of standard PINNs on convection-diffusion problems featuring steep boundary layers, a hallmark of singularly perturbed systems. Our first attempt to use PINNs to correct numerical solutions was unsuccessful, so we set up the

framework of RPINN for PINNs to approximate corrections to reduced solutions. We demonstrated that standard DNN architectures exhibit poor robustness in approximating solutions with sharp gradients when left unmodified. Specifically, RPINNs often misplace boundary layers or fail to develop them altogether, leading to physically incorrect surrogates.

To address this, we introduced simple yet effective input transformations—combinations of input shifting and stretching—that align the neural network's most sensitive regions with the physical location of boundary layers. By centering the activation functions' steep regions near the boundary layer and reducing the need for extreme weight magnitudes, these transformations significantly improved both training robustness and approximation accuracy. Through empirical studies, NTK analysis, and weight distribution investigations, we found that this input transformation serves as a lightweight alternative to more complex multi-scale or Fourier-based architectures.

Our approach consistently resolved steep boundary layers in one-dimensional convection-diffusion problems and extended successfully to a two-dimensional test case. Furthermore, this work highlights how NTK eigenstructures can predict the regions where RPINNs are likely to develop sharp gradients and underscores that proper alignment of these regions with the physics of the problem is essential.

However, the studies done in this chapter build upon the assumption that the location of the steep gradient layer is known apriori. We admit that this assumption is rich for most real-world applications. Thus, we carry our knowledge about necessary conditions DNNs need to satisfy to the next chapter and study situations when the location is not known or too hard to explicitly describe.

Chapter 5: PINNs for Convection-Diffusion Equation: Unknown Steep Gradient Layer Location

5.1 Introduction

Singularly perturbed problems, such as convection-diffusion equations, often exhibit steep boundary or internal layers whose locations are not known a priori. These layers are highly dependent on physical parameters, such as spatially varying velocity fields or complex geometrical features, making it difficult to pre-determine where steep gradients will arise.

In the previous chapter, we considered a scenario where the boundary layer location was known in advance by applying static input transformations. However, in real-world problems, such prior information is often unavailable, and fixed input transformations become ineffective.

To address this challenge, we introduce a new PINN architecture that incorporates an auxiliary input variable. Instead of feeding only the transformed coordinate $T(x)$ into the network, we input the pair $[T(x), a]$, where a is the factor used in the transformation. This framework is conceptually inspired by the Two-Scale PINN (TS-PINN) proposed by [97], where an auxiliary input is included in the network configuration. It was not explained in details how the auxiliary input improves the PINN architecture. We will address its benefits and demonstrates its performance. We will also compare and contrast our framework with TS-PINN in this chapter.

A key benefit of this structure is that it reduces the dependence on network biases and input transformation shift to capture the precise position of the boundary layer. The weights associated with the auxiliary input a directly contribute to learning the location of steep gradients. Furthermore, this design enables the network to handle multiple boundary layer locations within the same domain, as it can flexibly associate different values of a with distinct regions.

This auxiliary input framework is paired with a successive training procedure to enhance robustness and convergence when solving convection-diffusion equations with unknown boundary layer locations. In this chapter, we introduce the architecture, analyze its behavior, and demonstrate its effectiveness through numerical experiments on test cases with nontrivial layer locations.

This chapter is organized as follows. Section 5.2 introduces the Auxiliary-Input PINN (AI-PINN) architecture and compares it to the TS-PINN [97], analyzing how each network structure supports implicit localization of steep gradient layers. Section 5.3 presents the successive training strategy, demonstrating how staged training across varying ϵ values enables more effective approximation in singular perturbation regimes. Section 5.3.3 proposes fixing the input transformation across training stages to stabilize the effective shift, while Section 5.3.4 explores the impact of removing bias terms to further enhance robustness. The performance of AIR-PINN variants is shown in Sections 5.4 and 5.5, which cover 1D reaction-diffusion problems and a series of 2D convection-diffusion cases with varying complexity. Finally, the chapter concludes with a summary of key findings and insights.

5.2 Auxiliary-Input PINNs (AI-PINNs)

The core modification in this framework is the introduction of an auxiliary input alongside the transformed spatial coordinate. In the previous chapter, PINNs or RPINNs took a transformed coordinate $T(x)$ as input. Here, we expand the input space to include both $T(x)$ and the transformation factor a, i.e., the network input is $\mathbf{z} = [T(x), a]$. This idea is inspired by the Two-Scale PINN (TS-PINN) proposed by [97], which employs a similar auxiliary input to address singular perturbations. We provide a detailed comparison between TS-PINN and our Auxiliary-Input Reduced PINN (AIR-PINN) in this section.

Formally, the baseline RPINN architecture with input transformation $T(x)$ is defined as:

$$c_{\boldsymbol{\theta}}(T(x)) = \mathcal{N}(T(x)), \tag{5.1}$$

where $\mathcal{N}$ is a neural network with weights $\boldsymbol{\theta}$ and $T(x) = a(x - b)$ is the transformation using stretching factor a and shift b.

For a single-hidden-layer network with width d, the formulation becomes:

$$c_{\boldsymbol{\theta}}(T(x)) = W_2 \tanh(T(x)W_1 + b_1) + b_2 \tag{5.2}$$

$$= W_2 \tanh(a(x - b)W_1 + b_1) + b_2 \tag{5.3}$$

$$= \sum_{i=1}^{d} W_2^i \tanh(aW_1^i(x - b + \frac{b_1^i}{aW_1^i})) + b_2, \tag{5.4}$$

where $W_1 \in \mathbb{R}^{1\times d}$, $b_1 \in \mathbb{R}^d$, $W_2 \in \mathbb{R}^{d\times 1}$, and $b_2 \in \mathbb{R}$.

As previously identified, the effective shift is $-b + \frac{b_1^i}{aW_1^i}$, which determines the steep layer lo-

cation. Since $aW_1^i = \mathcal{O}(1/\epsilon)$, the $\frac{b_1^i}{aW_1^i}$ term is small when $\epsilon \to 0$, making b the dominant parameter for layer positioning. Therefore, prior knowledge of b is essential for accurate approximation.

To address the limitation of requiring b a priori, we propose the AIR-PINN. The AIR-PINN architecture augments the input with the auxiliary parameter a:

$$c_{\boldsymbol{\theta}}(T(x), a) = \mathcal{N}(T(x), a). \tag{5.5}$$

For a single-hidden-layer AIR-PINN with width d:

$$c_{\boldsymbol{\theta}}(T(x), a) = \sum_{i=1}^{d} W_2^i \tanh(a(x-b)W_1^{1,i} + aW_1^{2,i} + b_1^i) + b_2 \tag{5.6}$$

$$= \sum_{i=1}^{d} W_2^i \tanh(aW_1^i(x - b + \frac{W_1^{2,i}}{W_1^{1,i}} + \frac{b_1^i}{aW_1^{1,i}})) + b_2, \tag{5.7}$$

where $W_1 \in \mathbb{R}^{2\times d}$ includes separate weight rows for $T(x)$ and a.

The additional term $\frac{W_1^{2,i}}{W_1^{1,i}}$ contributes to the effective shift, enabling the network to localize steep gradient regions through the weights linked to a. This decouples the localization task from biases and facilitates the network's capacity to detect boundary layers implicitly.

While this chapter focuses on applying the auxiliary input to RPINNs, the proposed architecture is not inherently restricted to such settings. The introduction of an auxiliary input, such as the transformation factor a, provides a general mechanism to enable neural networks to infer problem-specific scaling, localization, or structural information during training. This design can be extended to other classes of PINNs where solution features—such as steep gradients, localized phenomena, or multiscale effects—emerge from unknown or parameter-dependent transformations. Therefore, the auxiliary-input framework has potential applications beyond Reduced PINNs,

including broader singular perturbation problems and multi-physics PDEs with hidden spatial or temporal scales.

While the above discussion focuses on a single-hidden-layer AIR-PINN for clarity, the architecture generalizes naturally to deeper networks. In such settings, the auxiliary input a can be concatenated with $T(x)$ at the input layer and propagated through subsequent layers without architectural modification. The key property—the contribution of a to the effective shift—persists regardless of depth. However, deeper networks may offer improved expressiveness and capacity, as Chapter 4 demonstrated deeper networks only share the burden fo capturing the steep gradient layer. With the help of the streching factor a, network depth does not necessarily improve the performance of the network.

5.2.0.1 Comparison with Two-Scale PINN (TS-PINN)

Both AIR-PINN and TS-PINN utilize auxiliary inputs to resolve steep gradients associated with singular perturbations. However, there are architectural and interpretational differences.

The TS-PINN introduces a two-scale representation by augmenting the input as:

$$c_{\boldsymbol{\theta}}^{\text{TS-PINN}}(x, T(x), a) = \mathcal{N}(x, T(x), a), \tag{5.8}$$

where $T(x) = a(x - b)$. In [97], a is default at $\epsilon^{-1/2}$ and b is default at x_c, the middle point of the domain. The TS-PINN jointly processes x (the original coordinate) and $T(x)$ (the stretched coordinate), enabling it to capture both the slow and fast scales present in singular perturbation problems.

Again, we consider a single-hidden-layer TS-PINN with width d:

$$c_{\boldsymbol{\theta}}^{\text{TS-PINN}}(x, T(x), a) = \sum_{i=1}^{d} W_2^i \tanh\left(xW_1^{1,i} + a(x-b)W_1^{2,i} + aW_1^{3,i} + b_1^i\right) + b_2, \tag{5.9}$$

$$= \sum_{i=1}^{d} W_2^i \tanh\left((aW_1^{2,i} + W_1^{1,i})\left(x + \frac{-abW_1^{2,i} + aW_1^{3,i} + b_1^i}{aW_1^{2,i} + W_1^{1,i}}\right)\right) + b_2, \tag{5.10}$$

where $W_1 \in \mathbb{R}^{3\times d}$ includes separate weight rows for x, $T(x)$, and a.

Comparing (5.10) to (5.7), we observe two things: the effective streching factor changed from $aW_1^{1,i}$ to $aW_1^{2,i}+W_1^{1,i}$, and the effective shift changed from $-b+\frac{W_1^{2,i}}{W_1^{1,i}}+\frac{b_1^i}{aW_1^i}$ to $\frac{-abW_1^{2,i}+aW_1^{3,i}+b_1^i}{aW_1^{2,i}+W_1^{1,i}}$. We first analyze $aW_1^{2,i} + W_1^{1,i}$. Does the additional $W_1^{1,i}$ provide any additional benefit for approximation solutions to convection-diffusion equation? We know that the streching factor should be of $O(\frac{1}{\epsilon})$, which could be handled through $aW_1^{2,i}$ alone. The additional $W_1^{1,i}$ could potentially introduce additional complexity to the network, which may not be necessary.

Comparing (5.10) to the AIR-PINN formulation in (5.7), we observe that TS-PINN modifies both the effective stretching factor and shift:

- **Stretching factor:** TS-PINN introduces $aW_1^{2,i} + W_1^{1,i}$ instead of $aW_1^{1,i}$.
- **Effective shift:** TS-PINN's effective shift is $\frac{-abW_1^{2,i}+aW_1^{3,i}+b_1^i}{aW_1^{2,i}+W_1^{1,i}}$, compared to AIR-PINN's $-b + \frac{W_1^{2,i}}{W_1^{1,i}} + \frac{b_1^i}{aW_1^i}$.

For convection-diffusion problems where the stretching factor needs to be $\mathcal{O}(1/\epsilon)$, $aW_1^{2,i}$ alone suffices to generate this scaling. Therefore, TS-PINN's additional $W_1^{1,i}$ might introduce unnecessary degrees of freedom, which could increase network complexity and affect generalization. AIR-PINN provides a simpler and more efficient parametrization while retaining the ability to

localize the boundary layer.

The effective shift governs the location of the steep gradient layer. TS-PINN's effective shift can be rewritten as:

$$\frac{-abW_1^{2,i} + aW_1^{3,i} + b_1^i}{aW_1^{2,i} + W_1^{1,i}} = \frac{-b}{1 + \frac{W_1^{1,i}}{W_1^{2,i}}} + \frac{W_1^{3,i}}{\frac{W_1^{1,i}}{a} + W_1^{2,i}} + \frac{b_1^i}{aW_1^{2,i} + W_1^{1,i}}.$$

The first term, $\frac{-b}{1+\frac{W_1^{1,i}}{W_1^{2,i}}}$, corresponds to $-b$ in AIR-PINN, while the second and third terms resemble the $\frac{W_1^{2,i}}{W_1^{1,i}}$ and $\frac{b_1^i}{aW_1^{1,i}}$ terms in AIR-PINN, respectively.

However, deviations emerge when $W_1^{1,i} \neq 0$, causing $\frac{-b}{1+\frac{W_1^{1,i}}{W_1^{2,i}}}$ to differ from $-b$. When a is large, $\frac{W_1^{3,i}}{\frac{W_1^{1,i}}{a}+W_1^{2,i}} \approx \frac{W_1^{3,i}}{W_1^{2,i}}$, diminishing the influence of $W_1^{1,i}$. Additionally, both $\frac{b_1^i}{aW_1^{2,i}+W_1^{1,i}}$ and $\frac{b_1^i}{aW_1^{1,i}}$ decay as a increases.

Thus, while TS-PINN introduces more parameters via $W_1^{1,i}$, their influence on the effective shift is negligible in the regime of large a. This suggests that the added complexity from $W_1^{1,i}$ may not be necessary for accurately resolving steep gradient layers. In contrast, AIR-PINN's simpler parametrization can achieve similar performance with fewer parameters, making it a more efficient choice for singular perturbation problems.

One of the main advantages of AIR-PINN over TS-PINN lies in its simpler and more interpretable structure. In AIR-PINN, the auxiliary input a has a clear and direct role: it participates explicitly in the stretching transformation $T(x) = a(x - b)$ and contributes to the effective shift $\frac{W_1^{2,i}}{W_1^{1,i}}$. This transparency enables a more intuitive understanding of how the network adapts to unknown boundary layer locations. Specifically, a modulates both the steepness and the spatial localization of the gradient layer through well-separated, interpretable terms in the network's

effective representation.

In contrast, TS-PINN's use of $W_1^{1,i}$, $W_1^{2,i}$, and $W_1^{3,i}$, all influencing both x and $T(x)$ scales simultaneously, results in a more entangled structure. While this offers flexibility for multi-scale behavior, it makes it harder to isolate the contribution of each term to the solution features. The coupling of x and $T(x)$ terms complicates the analysis of gradient localization and network dynamics.

In summary, AIR-PINN retains the benefits of auxiliary-input architectures but provides clearer insights into how the network internalizes boundary layer positioning, facilitating easier diagnostics and interpretability during training and analysis.

5.3 Successive Training

Training PINNs to approximate solutions to singularly perturbed problems with unknown boundary layer locations is a longstanding and complex challenge. Conventional training methods often fail to capture steep gradient layers when their positions are unknown, leading to inaccurate solutions and slow convergence. This difficulty stems from the inherent multiscale nature of singularly perturbed problems, where small parameters ϵ induce localized sharp features—so-called boundary layers—whose locations may not be explicitly provided.

To address this challenge, we introduce a successive training strategy inspired by curriculum learning and transfer learning principles. The core idea is to leverage knowledge acquired from solving simpler problems (with larger ϵ values and smoother solutions) to inform the solution process for more challenging problems (with smaller ϵ values and sharper boundary layers). Specifically, we first train the network on a sequence of problems with successively decreasing ϵ,

gradually shifting the focus from capturing global solution structures to resolving highly localized steep gradients.

The key insight originates from the preceding chapter: resolving boundary layers requires scaling factors $a = \mathcal{O}(1/\epsilon)$ within input transformations to focus network capacity on steep regions. However, directly adopting large a values at small ϵ often results in poor approximations if the boundary layer location (parametrized by b) is not known beforehand. This is particularly problematic because steep boundary layers demand highly localized network attention, which, if misaligned, severely limits model performance. Successive training mitigates this issue by employing a multi-phase scheme. In early phases, the model is trained on larger ϵ problems where boundary layers are broad and easier to capture, enabling the network to implicitly infer an effective shift for localizing the steep gradient. In later phases, this learned information is retained and fine-tuned as ϵ is reduced, allowing the network to resolve boundary layers more effectively.

A crucial observation justifying this approach is that, in many singular perturbation problems—such as convection-diffusion equations—the boundary layer location remains invariant as $\epsilon \to 0$, while only the steepness of the gradient increases. This structural property allows the network to extract boundary layer localization at larger ϵ, then progressively sharpen the approximation at the same location as ϵ decreases. The interplay between learned effective shift and successive refinement of steepness is the cornerstone of this strategy.

This concept is conceptually aligned with transfer learning paradigms, where knowledge acquired from simpler tasks is transferred to facilitate the learning of more complex tasks. The use of transfer learning in PINNs has gained traction recently. For instance, [117] employs multi-task learning with uncertainty weighting to improve PINN performance on inverse problems. Likewise, [118] demonstrates that pretraining PINNs on related tasks reduces convergence time and improves

accuracy for reactor kinetics simulations. These studies reinforce the notion that hierarchical training methodologies, including our proposed successive training, are particularly effective in complex PDE-driven applications.

In this chapter, we formalize and extend this idea by systematically applying successive training to different PINN architectures: RPINN, AIR-PINN, and TS-PINN. Our analysis will show how each architecture leverages successive training differently, revealing distinct advantages and limitations in resolving singular perturbation problems with unknown boundary layer locations.

5.3.1 Successive Training Procedure

In this section, we investigate the application of successive training AIR-PINNs. AIR-PINNs, as established in the previous chapter, utilize an input transformation $T(x) = a(x - b)$ to effectively capture steep gradient regions in singularly perturbed problems, while employing the auxiliary input a and its corresponding weight entries to alleviate the burden of correctly identify shift b based on the location of steep gradient layers.

The successive training approach helps mitigate this issue by leveraging the property that the boundary layer location remains nearly invariant across different ϵ values, while only the steepness of the boundary layer varies. Specifically, training on larger ϵ values first allows the network to approximate the global structure and learn an effective shift b implicitly. In subsequent phases, as ϵ decreases, the stretching factor a can be gradually increased, and the model's prior knowledge about the boundary layer location is transferred to facilitate learning the more localized steep gradients. Moreover, we are interested in if AIR-PINN is capable of accurately approximate the solution without the aid of shift in input transformation, i.e. $b = 0$.

To implement successive training for RPINNs, we employ a two-stage procedure as formalized below:

Algorithm 1 Successive Training for Convection-Diffusion Problems

1: **Stage 1:** Initialize RPINN with random weights and biases following the Xavier initialization.
2: Train the RPINN on a problem with large $\epsilon = O(0.1)$ until convergence.
3: **Stage 2:** Load the trained weights and biases.
4: Retrain the RPINN on the target problem with small $\epsilon \ll 0.1$ using the loaded parameters.

In Stage 1, training on a large ϵ allows the network to learn global solution structures and implicitly localize the boundary layer. In Stage 2, this prior knowledge enables the network to resolve the steep gradients in the small ϵ regime without requiring explicit information about the boundary layer location.

This two-stage strategy forms the foundation for successive training, which we will next apply to AIR-PINNs and TS-PINNs.

5.3.2 Network Comparison on 1D Convection-Diffusion Problem

We start with the following 1D convection-diffusion equation:

$$-\epsilon u'' + 2u' = 3, \quad 0 < x < 1, \quad u(0) = 0, \quad u(1) = 0. \tag{5.11}$$

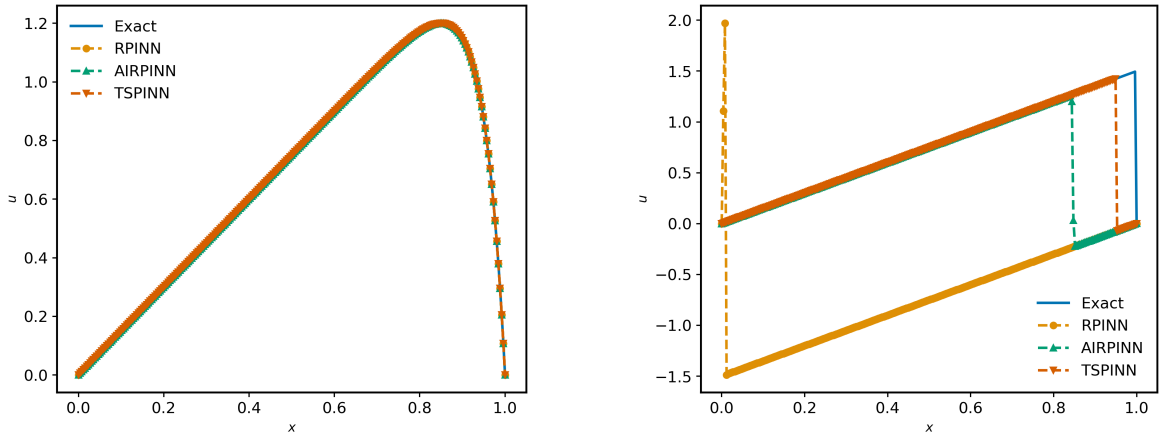
The exact solution of (5.11) is given by

$$u(x) = \frac{3}{2}\left(x - \frac{\exp(-2(1-x)/\epsilon) - \exp(-2/\epsilon)}{1 - \exp(-2/\epsilon)}\right). \tag{5.12}$$

The problem features a boundary layer at $x = 1$ due to the small parameter ϵ, which leads to sharp gradients.

We compare the performance of RPINN, AIR-PINN, and TS-PINN under the successive training strategy. The network architecture is fixed to a two-hidden-layer fully connected neural network with width 20. The training follows the standard PINN loss function defined in (2.19), using $N_b = 1$ boundary sample and $N_f = 1$ residual sample. All networks are trained using the Adam optimizer with a learning rate of 10^{-4} for 20,000 iterations, followed by 5,000 iterations of LBFGS. We use Xavier initialization for all weights and biases, and training data consists of 450 uniformly distributed sample points over the domain $[0, 1]$.

As proposed in [97], the stretching factor a is initialized as $\epsilon^{-1/2}$, and we adopt the same initialization in our experiments. The results of applying successive training from $\epsilon = 10^{-1}$ to $\epsilon = 10^{-5}$ are visualized in Figure 5.1. For each method, we show the network's approximation at both the initial stage ($\epsilon = 10^{-1}$) and the final stage ($\epsilon = 10^{-5}$). The corresponding absolute and relative errors at $\epsilon = 10^{-5}$ are shown in Figure 5.2, and a quantitative summary of average relative errors in the subdomains $[0, 0.9]$ and $[0.9, 1]$ is provided in Table 5.1.



(a) Approximation for $\epsilon = 0.1$.

(b) Approximation for $\epsilon = 1e - 5$.

Figure 5.1: Comparison of approximations for $\epsilon = 0.1$ and $\epsilon = 1e - 5$ using RPINN, AI-PINN, and TS-PINN following 1.

From Figure 5.1a, all methods successfully approximate the solution when $\epsilon = 10^{-1}$,

capturing the relatively mild boundary layer. However, as ϵ decreases to 10^{-5}, notable differences in performance emerge.

RPINN performs the worst under successive training. Although it approximates the solution well for larger ϵ, it fails to resolve the steep boundary layer at $\epsilon = 10^{-5}$, producing a surrogate that captures the global behavior but misses the sharp transition near $x = 1$. In fact, its relative error in the boundary layer region $[0.9, 1]$ exceeds 1.0, indicating a completely inaccurate boundary layer profile. This suggests that RPINN fails to carry forward useful information for boundary layer localization across stages.

AIR-PINN, by contrast, performs significantly better. It maintains low error in the interior domain $[0, 0.9]$ and attempts to form a boundary layer, but places it incorrectly around $x = 0.8$ instead of $x = 1$. This misalignment is evident in both the approximation plot and the relative error, which shows elevated values in the layer region.

TS-PINN exhibits slightly better performance than AIR-PINN. Its boundary layer is formed closer to the correct location, around $x = 0.9$, and its relative error in the steep region is an order of magnitude smaller than AIR-PINN. However, it still fails to accurately capture the boundary layer at $x = 1$, and errors in the region $[0.9, 1]$ remain significant.

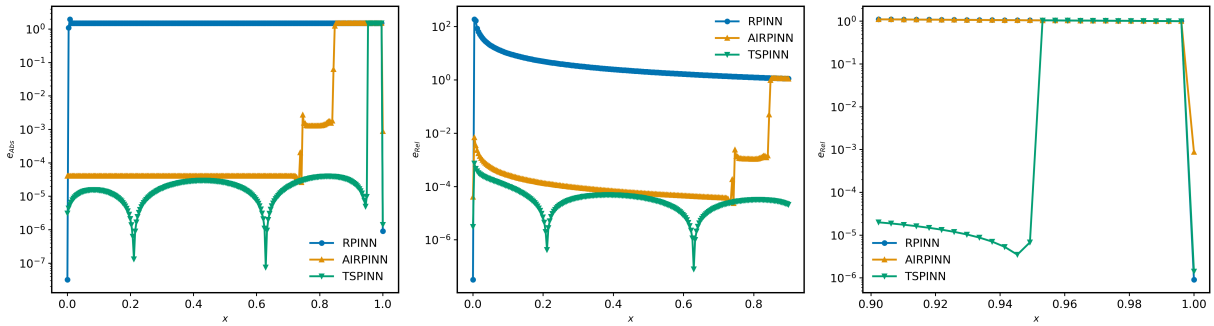


Figure 5.2: Absolute and relative errors of RPINN, AI-PINN, and TS-PINN $\epsilon = 1e - 5$ following 1.

Table 5.1: Relative errors of RPINN, AIR-PINN, and TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ under successive training.

ϵ	Method	$[0, 0.9]$	$[0.9, 1]$
10^{-1}	RPINN	2.92×10^{-8}	6.47×10^{-7}
	AIR-PINN	4.24×10^{-8}	1.17×10^{-7}
	TS-PINN	5.86×10^{-8}	1.62×10^{-7}
10^{-5}	RPINN	1.41×10^{-5}	1.49
	AIR-PINN	6.35×10^{-7}	2.27×10^{-1}
	TS-PINN	1.63×10^{-6}	7.01×10^{-2}

These results indicate that while successive training improves performance for AIR-PINN and TS-PINN compared to RPINN, both architectures still fall short of achieving fully accurate approximations when the boundary layer location is unknown. This limitation motivates the need for an enhanced successive training framework that can better identify and adapt to boundary layer dynamics—particularly under severe singular perturbations. We explore this next.

The approximation errors shown in Figure 5.1b arise primarily from incorrect localization of the steep gradient layer. This localization is controlled by the effective shift in the first layer of the network. To further understand the performance of different architectures under successive training, we analyze how the effective shift evolves as the network transitions from training on large $\epsilon = 0.1$ to small $\epsilon = 10^{-5}$.

The effective shift determines where the network concentrates its capacity to resolve boundary layers. Preserving the learned effective shift is crucial to the success of successive training. If the shift value degrades significantly between stages, the network fails to retain knowledge of where to resolve steep gradients. This section examines whether a neuron with an effective shift of -1 at $\epsilon_0 = 0.1$ maintains a similar shift when retrained on $\epsilon_1 = 10^{-5}$, and how this behavior

differs across RPINN, AIR-PINN, and TS-PINN.

For a network of width d, the effective shift for neuron $i = 1, \ldots, d$ is:

$$\text{RPINN:} \quad -b + \frac{b_1^i}{aW_1^i}, \tag{5.13}$$

$$\text{AIR-PINN:} \quad -b + \frac{W_1^{2,i}}{W_1^{1,i}} + \frac{b_1^i}{aW_1^{1,i}}, \tag{5.14}$$

$$\text{TS-PINN:} \quad \frac{-abW_1^{2,i} + aW_1^{3,i} + b_1^i}{aW_1^{2,i} + W_1^{1,i}}. \tag{5.15}$$

where $i = 1, \ldots, d$. Suppose that a network is trained on a problem with $\epsilon_0 = 0.1$ and the input transformation is $\epsilon_0^{-1/2}x$. There exist one effective shift with the value of -1 for the neurons in the first layer based on results in Chapter 4. Without loss of generality, suppose there exists a neuron i in the first layer with effective shift -1. We study the effect on the same neuron when the network is trained on a problem with $\epsilon_1 = 1e-5$ and the input transformation is $\epsilon_1^{-1/2}x$.

For RPINN, given that $\frac{b_1^i}{\epsilon_0^{-1/2}W_1^i} = -1$, it follows that $b_1^i = -\epsilon_0^{-1/2}W_1^i$. At ϵ_1, the new effective shift becomes:

$$\frac{b_1^i}{\epsilon_1^{-1/2}W_1^i} = \frac{-\epsilon_0^{-1/2}W_1^i}{\epsilon_1^{-1/2}W_1^i} = -\left(\frac{\epsilon_0}{\epsilon_1}\right)^{1/2} \approx 0.$$

This shift approaches zero as $\epsilon_1 \ll \epsilon_0$, indicating that RPINN fails to retain localization, which explains the steep gradient layer forming at $x = 0$ in Figure 5.1b.

For AIR-PINN, at ϵ_0, we assume:

$$\frac{W_1^{2,i}}{W_1^{1,i}} + \frac{b_1^i}{\epsilon_0^{-1/2}W_1^{1,i}} = -1 \quad \Rightarrow \quad b_1^i = W_1^{1,i} \cdot \epsilon_0^{-1/2}\left(-1 - \frac{W_1^{2,i}}{W_1^{1,i}}\right).$$

At ϵ_1, the effective shift becomes:

$$\frac{W_1^{2,i}}{W_1^{1,i}} + \frac{b_1^i}{\epsilon_1^{-1/2} W_1^{1,i}} = \frac{W_1^{2,i}}{W_1^{1,i}} + \left(\frac{\epsilon_0}{\epsilon_1}\right)^{1/2} \left(-1 - \frac{W_1^{2,i}}{W_1^{1,i}}\right).$$

Since $\left(\frac{\epsilon_0}{\epsilon_1}\right)^{1/2} \gg 1$, the effective shift will deviate from -1 unless $b_1^i \to 0$, in which case the shift tends to $\frac{W_1^{2,i}}{W_1^{1,i}}$. This explains AIR-PINN's approximation in Figure 5.1b, where the gradient layer forms around $x = 0.8$, indicating partial but imperfect localization.

For TS-PINN, at ϵ_0, the effective shift is:

$$\frac{W_1^{3,i}}{W_1^{2,i} + \frac{W_1^{1,i}}{\epsilon_0^{-1/2}}} + \frac{b_1^i}{\epsilon_0^{-1/2} W_1^{2,i} + W_1^{1,i}}.$$

At ϵ_1, this becomes:

$$\frac{W_1^{3,i}}{W_1^{2,i} + \frac{W_1^{1,i}}{\epsilon_1^{-1/2}}} + \frac{b_1^i}{\epsilon_1^{-1/2} W_1^{2,i} + W_1^{1,i}}.$$

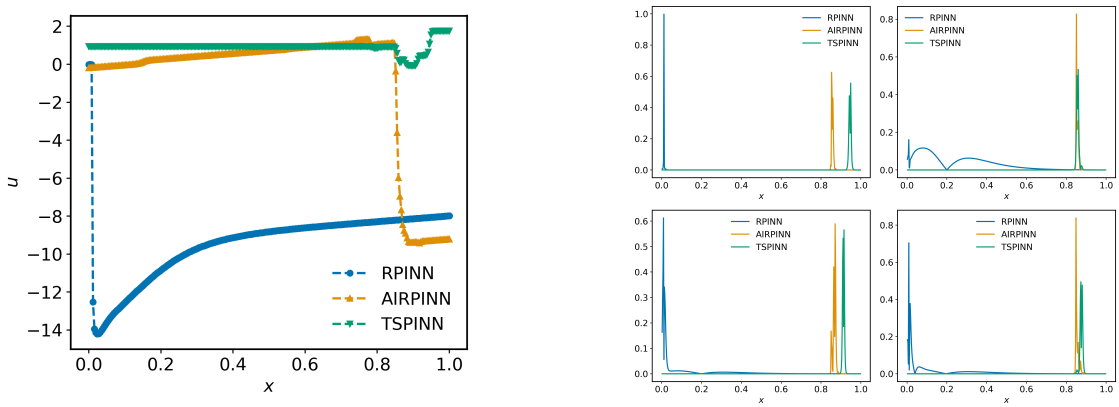
As $\epsilon_1^{-1/2} \to \infty$, we have:

$$\frac{W_1^{1,i}}{\epsilon_1^{-1/2}} \to 0, \quad \frac{b_1^i}{\epsilon_1^{-1/2} W_1^{2,i} + W_1^{1,i}} \to 0,$$

which implies:

$$\text{effective shift} \to \frac{W_1^{3,i}}{W_1^{2,i}}.$$

Therefore, if $W_1^{3,i}/W_1^{2,i} \approx -1$, the shift is preserved. This explains why TS-PINN maintains the gradient layer closer to the correct location ($x = 0.9$), as shown in Figure 5.1b. This also shows that AIR-PINN performs similarly to TS-PINN.



(a) Epoch-zero predicted $u(x)$ for different methods.

(b) Top four NTK eigenvectors for different methods.

Figure 5.3: Comparison of initial predictions after loading trained weights and biases from $\epsilon = 0.1$ and top four NTK eigenvectors for RPINN, AI-PINN, and TS-PINN at $\epsilon = 1e - 5$.

To gain further insight into the behavior of successive training, we examine the network outputs and corresponding NTK eigenstructures immediately after loading weights and biases trained on $\epsilon = 0.1$, before any retraining occurs on $\epsilon = 1e - 5$. As shown in Figure 5.3a, even at epoch zero, the initialized predictions of RPINN, AIR-PINN, and TS-PINN already exhibit signatures of steep gradient layers, located near where these features ultimately emerge after full training. This suggests that the learned representations at large ϵ when translated for smaller ϵ encode useful priors about gradient localization.

To better understand this initialization behavior, we investigate the eigenvectors of the NTK matrix at $\epsilon = 1e - 5$ using the transferred weights. The NTK captures the local linearized training dynamics, and the eigenvectors corresponding to the largest eigenvalues determine the dominant modes that the network will fit first. Figure 5.3b shows the top four NTK eigenvectors for RPINN, AIR-PINN, and TS-PINN. Remarkably, the steep gradients observed in the initial predictions align with the first eigenvectors, each exhibit sharp transitions located at or near the eventual boundary layer regions observed in the final trained solutions.

This observation reinforces the idea that successive training initializes the network in a favorable region of parameter space, where dominant NTK modes are already aligned with physically meaningful features of the solution. However, the ability of each architecture to evolve these features accurately still depends on how well the effective shift is preserved, as previously discussed.

5.3.3 Fixed Input Transformation

The key issue identified in the previous section is that effective shifts, which determine the location of the steep gradient layer, are sensitive to changes in the input transformation—particularly the stretching factor a—across successive training stages. While initializing with larger ϵ can produce an effective shift near the correct boundary layer, retraining on smaller ϵ with a different transformation $T(x) = a(x - b)$ can drastically alter the effective shift, leading to misalignment between learned features and the true boundary layer.

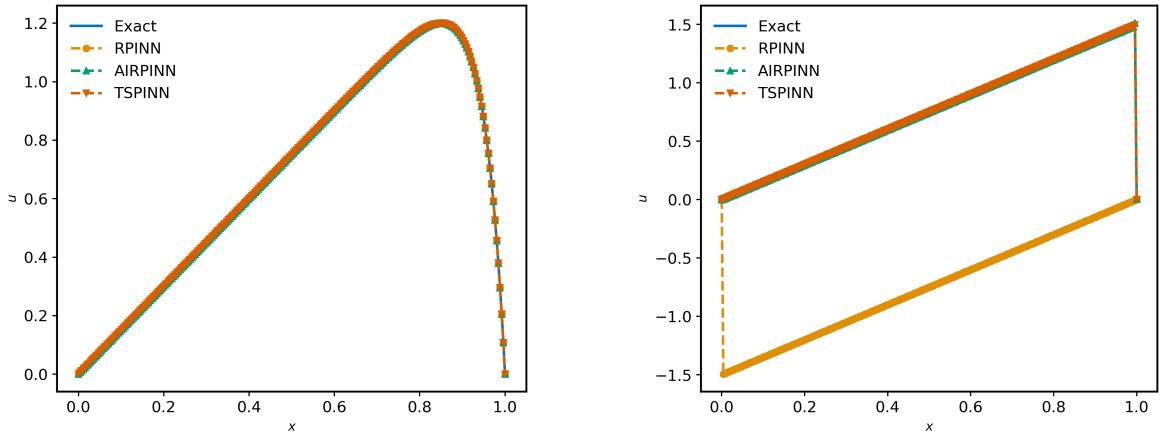
To mitigate this problem, we propose a simple yet effective modification: fix the input transformation $T(x)$ throughout all stages of successive training. Instead of adapting to match each target ϵ, we fix a. This ensures that the spatial scaling used by the network remains consistent, allowing the weights and effective shifts learned from the initial training to remain meaningful during later retraining phases. The fixed transformation allows the learned localization properties to persist, preventing the network from having to rediscover the boundary layer structure after each change in ϵ.

In this section, we evaluate this approach and demonstrate that fixing the input transformation significantly improves the consistency of effective shift behavior and the accuracy of the resulting

approximations at smaller ϵ values.

The choice of a fixed stretching factor a aims to preserve the effective shift across stages of successive training. Rather than adapting a according to the target ϵ, we fix the input transformation as $T(x) = a(x - b)$ with $a = \epsilon_0^{-1} = 10$, based on the initial training value $\epsilon_0 = 0.1$. This selection ensures that the input transformation is sufficiently strong to capture the boundary layer in the later stage (with $\epsilon = 10^{-5}$) while maintaining continuity in the effective shift learned during the earlier stage. Crucially, fixing a eliminates the rescaling inconsistency that previously disrupted the localization of steep gradients.

We apply this fixed transformation to RPINN, AIR-PINN, and TS-PINN under the same successive training protocol and compare their performances against those using dynamic transformation strategies. Figures 5.4 and 5.5 show the approximated solutions and their corresponding errors at both stages of training.



(a) Approximation for $\epsilon = 0.1$.

(b) Approximation for $\epsilon = 1e - 5$.

Figure 5.4: Comparison of approximations for $\epsilon = 0.1$ and $\epsilon = 1e - 5$ using RPINN, AI-PINN, and TS-PINN following 1. The transformation stretching factor is fixed to $a = 10$.

The results demonstrate a substantial improvement in the accuracy and stability of AIR-PINN and TS-PINN. AIR-PINN achieves a relative error of $\sim 10^{-2}$ in the boundary layer and

interior, significantly outperforming its earlier counterpart under adaptive transformations. TS-PINN, in particular, shows strong robustness under this strategy, with relative errors as low as $\sim 10^{-4}$ near the boundary and $\sim 10^{-6}$ in the steep gradient region. These improvements confirm that the fixed transformation enables the network to retain useful inductive bias learned from the first training stage.

RPINN, however, remains unstable even under fixed transformation. The relative error in the interior region rises to ~ 6.67, suggesting that its lack of structural support for implicit localization (as in AIR-PINN or TS-PINN) leaves it incapable of adapting successfully to the more challenging steep boundary layer problem. This aligns with our earlier analysis that RPINN's effective shift is overly reliant on b_1, which can vary unpredictably when a is large.

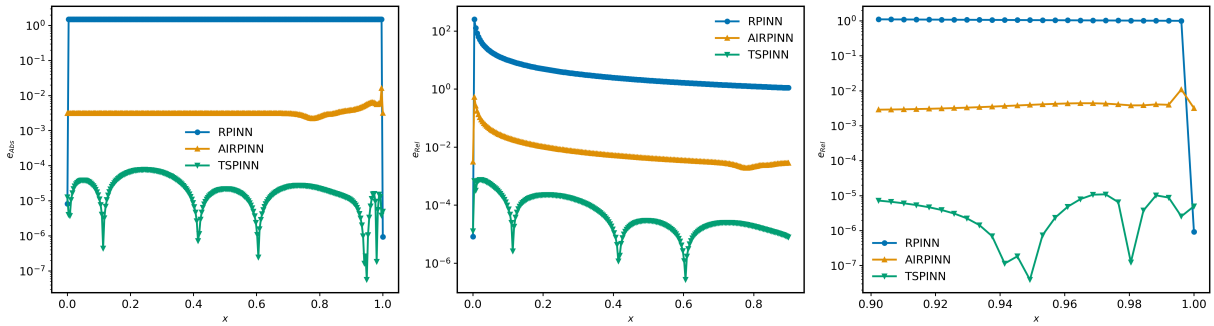


Figure 5.5: Absolute and relative errors of RPINN, AIR-PINN, and TS-PINN $\epsilon = 1e-5$ following 1. The transformation stretching factor is fixed to $a = 10$.

Table 5.2 provides a detailed breakdown of the relative mean errors in the regions $[0, 0.9]$ and $[0.9, 1]$ for both training stages. These results further reinforce the advantage of combining auxiliary inputs and fixed transformations in facilitating effective and transferable learning across scales in singularly perturbed PDEs.

The fixed input transformation strategy provides a straightforward remedy to the problem of misaligned effective shifts by maintaining a consistent spatial scaling throughout successive

Table 5.2: Relative mean errors of RPINN, AIR-PINN, and TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ under successive training. The transformation stretching factor is fixed to $a = 10$.

ϵ	Method	$[0, 0.9]$	$[0.9, 1]$
10^{-1}	RPINN	2.71×10^{-6}	4.07×10^{-7}
	AIR-PINN	1.07×10^{-6}	1.12×10^{-7}
	TS-PINN	3.74×10^{-7}	1.21×10^{-7}
10^{-5}	RPINN	6.67	1.01
	AIR-PINN	1.39×10^{-2}	3.94×10^{-3}
	TS-PINN	1.04×10^{-4}	4.47×10^{-6}

training stages. However, while this method ensures that the learned localization patterns remain meaningful between stages, it does not fully eliminate the sensitivity of the effective shift to other architectural components, such as biases. In particular, the presence of biases in the network introduces additional terms into the effective shift expressions, which can shift its value unpredictably when the stretching factor a changes or becomes large.

To address this residual instability, we next consider an alternative architectural strategy: removing all bias terms from the neural network. This design enforces that the effective shift depends only on the ratio of weights associated with the transformed input and the auxiliary input. As a result, the shift becomes structurally invariant across training stages and decoupled from the influence of a. In the following section, we examine this bias-free AIR-PINN architecture and demonstrate how it improves the stability and reliability of successive training across different problem settings.

5.3.4 Networks without Bias

In addition to fixing the input transformation, we propose another structural modification aimed at improving the robustness of successive training: removing bias terms from the network architecture. This change is motivated by our analysis of effective shifts, where we observed that even small contributions from the bias vector b_1 can distort the effective shift between training stages. These distortions become particularly problematic when transitioning from a larger to a smaller ϵ, as the scaling by $a = \epsilon^{-1/2}$ can amplify otherwise negligible deviations introduced by b_1.

By eliminating biases from all layers of the network, we force the network to rely solely on the interaction between weights corresponding to the transformed spatial coordinates and the auxiliary input a to learn the correct effective shift. This encourages the ratio $\frac{W_1^{2,i}}{W_1^{1,i}}$ in AIR-PINN (and analogous terms in other architectures) to play the dominant role in localizing the steep gradient layer. Crucially, this ratio remains invariant when a changes, making it ideal for preserving learned features across successive stages.

In this section, we test the performance of bias-free AIR-PINNs under successive training and compare their stability and approximation accuracy against standard architectures. We find that removing bias significantly improves the transferability of effective shift information and leads to more accurate solutions at small ϵ without needing to retrain the network to rediscover boundary layers.

While removing biases from the network architecture improves the consistency of effective shift preservation for AIR-PINN and TS-PINN, the same strategy proves ineffective for RPINN. The core reason lies in how each architecture localizes the steep gradient layer. In RPINN, the

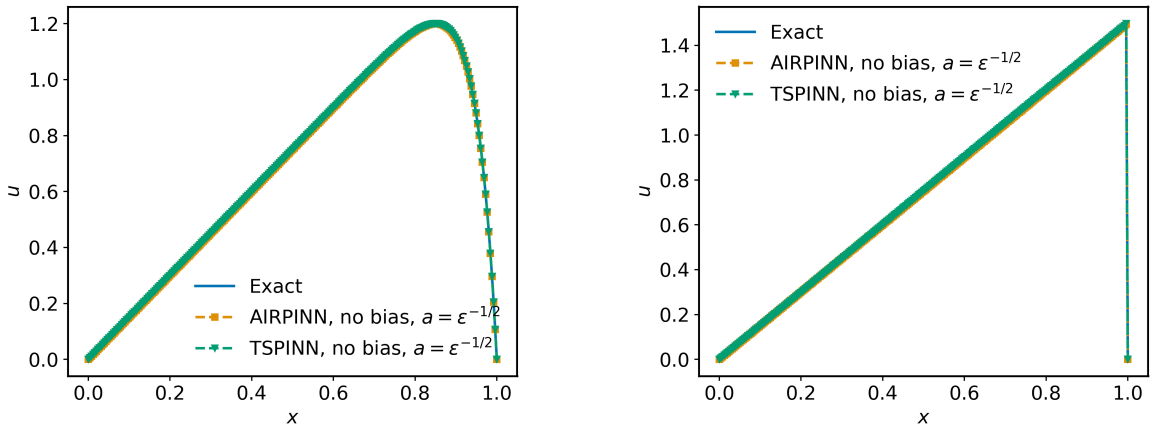
effective shift is determined entirely by the shift parameter b in the input transformation and the bias term b_1 in the first layer:

$$\text{Effective Shift (RPINN)} = -b + \frac{b_1^i}{aW_1^i}.$$

When the bias term b_1^i is removed from the network, the network loses its ability to adjust the effective shift dynamically. This is especially problematic when $b = 0$ is used to minimize prior knowledge, as the effective shift becomes fixed at zero for all neurons, regardless of the steep boundary layer's actual location.

Unlike AIR-PINN and TS-PINN, which incorporate auxiliary input weights to adaptively infer the effective shift, RPINN relies solely on the explicit shift parameter b and bias term b_1 to localize steep gradients. In a bias-free setting, this dependence becomes a liability: without any internal mechanism to learn or adjust the effective shift, RPINN loses its ability to capture boundary layer structures, particularly during successive training where such localization is critical. Our preliminary experiments confirm this limitation—bias-free RPINNs consistently fail to produce meaningful approximations in the presence of sharp boundary layers. Consequently, we exclude RPINN from further bias-free evaluations and instead focus on AIR-PINN and TS-PINN, which are structurally equipped to handle this architectural constraint.

Figure 5.6 shows the approximations of bias-free AIR-PINN and TS-PINN for $\epsilon = 0.1$ and $1e-5$ using a successive training strategy with transformation stretching factor $a = \epsilon^{-1/2}$. Despite the absence of bias, AIR-PINN maintains strong performance, accurately capturing the steep boundary layer even as ϵ becomes small. In contrast, bias-free TS-PINN suffers from performance degradation at smaller ϵ, likely due to its more complex input structure and dependence on weight



(a) Approximation for $\epsilon = 0.1$.

(b) Approximation for $\epsilon = 1e - 5$.

Figure 5.6: Comparison of approximations for $\epsilon = 0.1$ and $\epsilon = 1e - 5$ using bias-free AI-PINN and bias-free TS-PINN following 1. The transformation stretching factor is $a = \epsilon^{-1/2}$.

interactions across multiple input components.

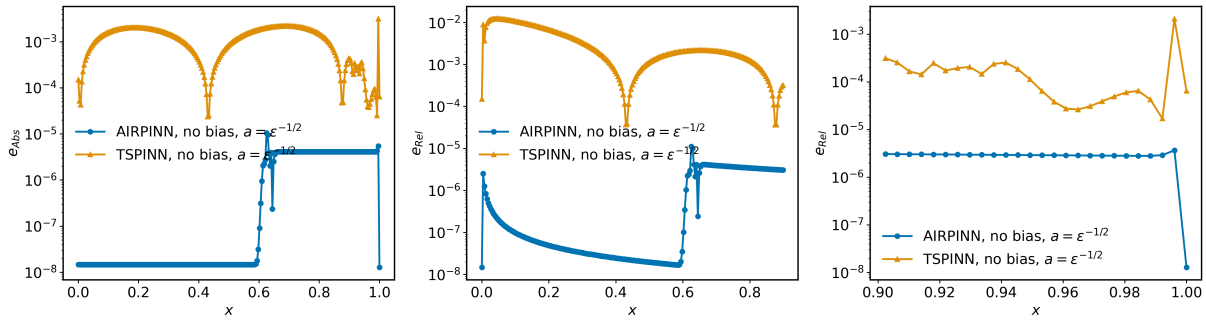


Figure 5.7: Absolute and relative errors of bias-free AIR-PINN and bias-free TS-PINN $\epsilon = 1e - 5$ following 1. The transformation stretching factor is $a = \epsilon^{-1/2}$.

This is further substantiated by the absolute and relative error profiles shown in Figure 5.7 and the quantitative error metrics in Table 5.3. While both methods perform comparably for $\epsilon = 10^{-1}$, bias-free AIR-PINN significantly outperforms TS-PINN for $\epsilon = 10^{-5}$, especially in the boundary region $[0.9, 1]$ where accurate resolution of the gradient is most critical.

To better understand the early-stage behavior under this architectural constraint, we also analyze the NTK eigenvectors at epoch zero in Figure 5.8. For bias-free AIR-PINN, the leading

Table 5.3: Relative mean errors of bias-free AIR-PINN and bias-free TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ with transformation stretching factor $a = \epsilon^{-1/2}$.

ϵ	Method	$[0, 0.9]$	$[0.9, 1]$
10^{-1}	bias-free AIR-PINN	7.20×10^{-7}	1.30×10^{-7}
	bias-free TS-PINN	1.30×10^{-6}	1.35×10^{-7}
10^{-5}	bias-free AIR-PINN	1.24×10^{-6}	2.84×10^{-6}
	bias-free TS-PINN	3.64×10^{-3}	2.03×10^{-4}

eigenvectors of the NTK exhibit clear spatial localization near the steep gradient layer, indicating that the network's initial training direction aligns well with the structure of the solution. In contrast, bias-free TS-PINN displays less coherent localization in its top eigenvectors, suggesting a misalignment between initialization and target features.

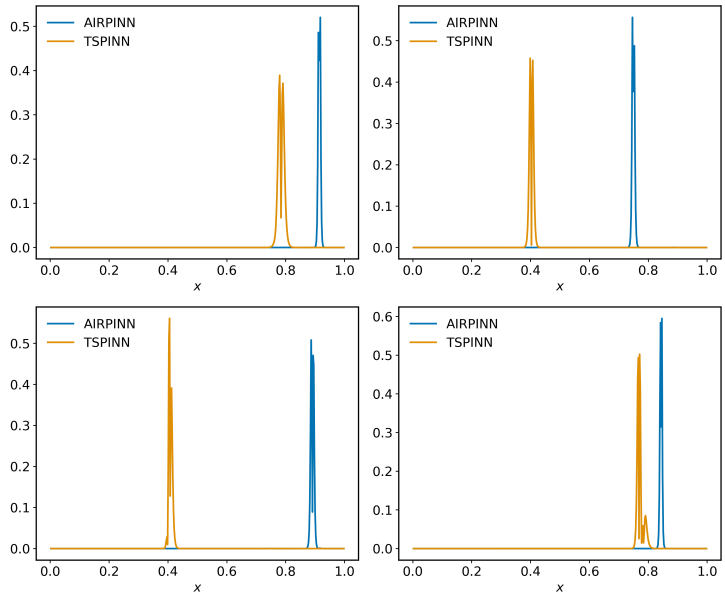


Figure 5.8: Top four NTK eigenvectors for bias-free AIR-PINN and bias-free TS-PINN at $\epsilon = 1e - 5$ at zero epoch.

These results highlight the robustness of AIR-PINN's auxiliary-input structure: even in the absence of biases, the network can leverage the ratio between transformed and auxiliary weights to effectively learn invariant effective shifts. As such, bias-free AIR-PINN offers a

structurally elegant and practically powerful architecture for successive training in singularly perturbed problems.

To explore the full potential of architectural and training strategies discussed so far, we consider combining two effective techniques: fixing the input transformation and using bias-free networks. Specifically, we apply a fixed stretching factor $a = 10$, derived from the initial stage of training, and remove all biases in the network to enforce stability in the learned effective shift. This combination is tested on AIR-PINN and TS-PINN architectures using the successive training strategy.

Additionally, we include a baseline comparison against RPINN trained directly on the target problem with $\epsilon = 1e\text{-}5$ using the transformation $T(x) = \epsilon^{-1/2}(x - 1)$, i.e., without successive training. This baseline represents a scenario where prior knowledge of the boundary layer location is assumed and used to optimize the shift term directly.

The results are summarized in Table 5.4. For $\epsilon = 10^{-1}$, all models—including AIR-PINN and TS-PINN—achieve very low errors, as expected due to the relatively mild boundary layer behavior. At $\epsilon = 10^{-5}$, RPINN trained with known input shift outperforms the bias-free networks, with relative errors on the order of 10^{-6}. However, bias-free AIR-PINN still achieves strong accuracy in both the interior and boundary regions, demonstrating that the architecture can learn effective shifts purely from weight ratios and maintain performance through successive stages. TS-PINN, though benefiting from a fixed transformation, continues to exhibit elevated error in the interior region, echoing prior observations about its sensitivity to parameter initialization and architectural complexity.

These findings suggest that although RPINN benefits from precise initialization with known boundary information, bias-free AIR-PINN offers a practical and generalizable alternative capable

Table 5.4: Relative mean errors of RPINN (non-successive) and bias-free AIR-PINN and TS-PINN in the regions $[0, 0.9]$ and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$. Bias-free networks use fixed transformation $a = 10$.

ϵ	Method	$[0, 0.9]$	$[0.9, 1]$
10^{-1}	bias-free AIR-PINN	4.38×10^{-7}	3.27×10^{-8}
	bias-free TS-PINN	5.09×10^{-8}	7.59×10^{-9}
10^{-5}	RPINN (not successive)	4.89×10^{-7}	9.38×10^{-6}
	bias-free AIR-PINN	4.27×10^{-5}	6.19×10^{-5}
	bias-free TS-PINN	3.60×10^{-3}	3.37×10^{-4}

of resolving steep layers with no prior shift knowledge. The effectiveness of this configuration reinforces AIR-PINN's robustness under both architectural and optimization constraints.

5.4 1D Reaction-Diffusion Problem

Singularly perturbed reaction-diffusion equations represent a fundamental class of partial differential equations characterized by the interaction between a small diffusion term and a reactive term. Unlike convection-diffusion equations, where steep boundary layers arise asymmetrically at outflow boundaries, reaction-diffusion problems typically exhibit symmetric boundary layers at both ends of the domain due to balanced reaction terms and symmetric boundary conditions.

The standard form of the one-dimensional reaction-diffusion equation is:

$$-\epsilon u''(x) + r(x)u(x) = f(x), \quad x \in (0, 1), \qquad u(0) = u(1) = 0, \tag{5.16}$$

where $\epsilon > 0$ is a small perturbation parameter, $r(x) \geq 0$ is a reaction coefficient, and $f(x)$ is a known source term. As $\epsilon \to 0$, the solution develops steep boundary layers near both $x = 0$ and $x = 1$, while remaining relatively flat in the interior.

A key difference between reaction-diffusion and convection-diffusion problems lies in the boundary layer width. For the convection-diffusion equation, the boundary layer has width $\mathcal{O}(\epsilon)$, whereas for reaction-diffusion problems, the boundary layer is typically broader, with width $\mathcal{O}(\epsilon^{1/2})$ [119]. This makes the reaction-diffusion problem more tractable from a training perspective, as the steep gradients are less severe for the same value of ϵ.

This setup provides a useful intermediate test case for evaluating the effectiveness of AIR-PINNs. In particular, the reaction-diffusion equation offers an opportunity to assess the network's ability to detect and resolve multiple boundary layers—without prior knowledge of their locations—using a relatively milder singular perturbation. It also allows us to further test the generalizability of techniques like successive training, fixed input transformations, and bias-free networks before applying them to more challenging multi-dimensional problems.

In this section, we explore the application of AIR-PINNs to the reaction-diffusion equation, comparing its performance under different training strategies and architectural variations.

We start with the following 1D reaction-diffusion equation:

$$-\epsilon u'' + u = 0, \quad 0 < x < 1, \quad u(0) = 0, \quad u(1) = 0. \tag{5.17}$$

The exact solution of (5.17) is given by

$$u(x) = 1 - \frac{e^{-x/\sqrt{\epsilon}} + e^{-(1-x)/\sqrt{\epsilon}}}{1 + e^{-1/\sqrt{\epsilon}}}. \tag{5.18}$$

The problem features two boundary layers at $x = 0, 1$ due to the small parameter ϵ, which leads to sharp gradients.

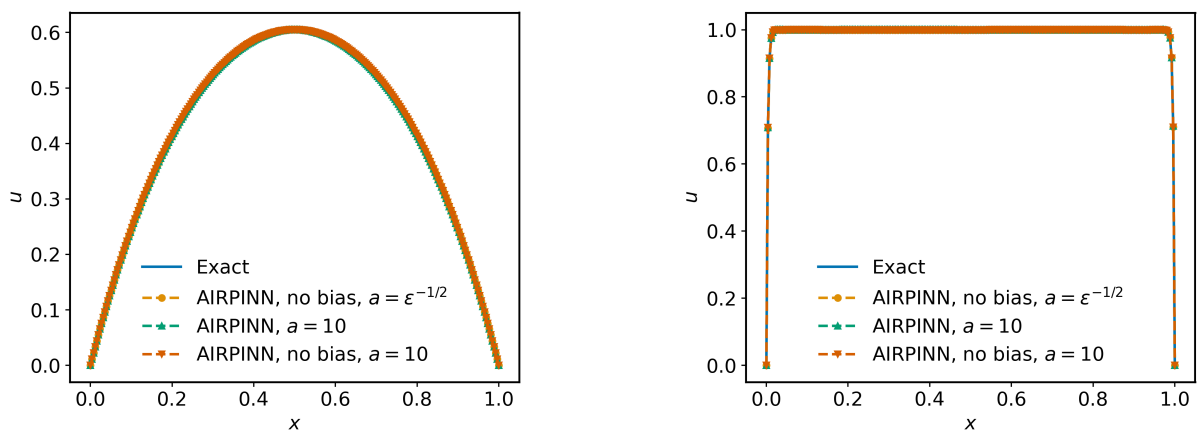
We test three variations of AIR-PINN on this problem:

1. a bias-free AIR-PINN with input transformation $a = \epsilon^{-1/2}$,

2. a standard AIR-PINN with fixed transformation $a = 10$, and

3. a bias-free AIR-PINN with fixed transformation $a = 10$.

All models use a two-layer neural network with width 20 and are trained using a successive training strategy, where the model is first trained on a problem with $\epsilon_0 = 0.1$ and then retrained on $\epsilon_1 = 10^{-5}$. The training protocol, optimizer, and data sampling remain consistent across all settings to ensure a fair comparison.



(a) Approximation for $\epsilon = 0.1$.

(b) Approximation for $\epsilon = 1e - 5$.

Figure 5.9: Comparison of approximations produced for $\epsilon = 0.1$ and $\epsilon = 1e - 5$ of (5.17).

The approximations for both stages of training are shown in Figure 5.9, while the corresponding absolute and relative errors for $\epsilon = 10^{-5}$ are visualized in Figure 5.10. The quantitative errors across subdomains are reported in Table 5.5.

All three AIR-PINN variants achieve high accuracy in resolving the solution, with relative errors in the interior region $[0.1, 0.9]$ remaining consistently on the order of 10^{-5} or lower, and

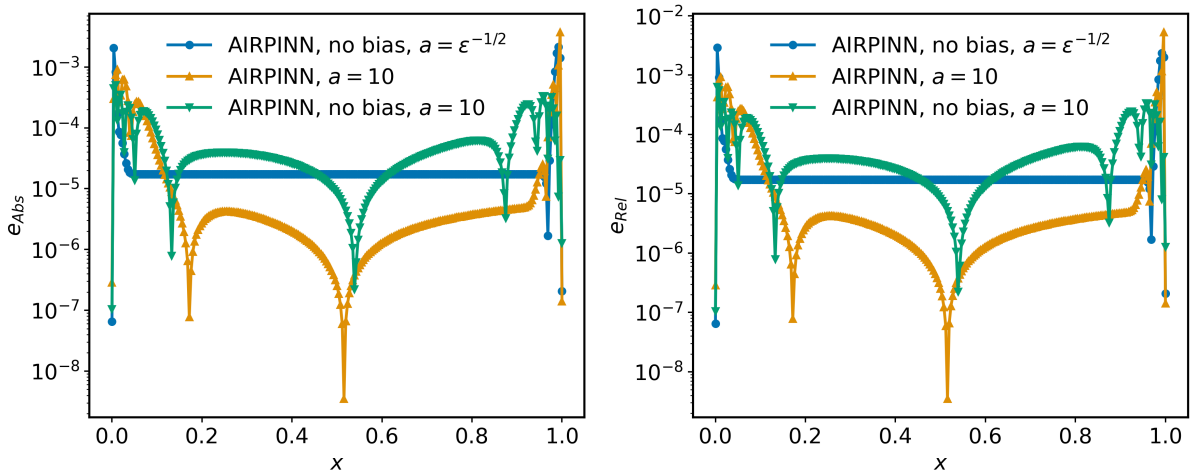


Figure 5.10: Absolute and relative errors of approximations for $\epsilon = 1e - 5$ using AI-PINN.

boundary layer errors in $[0, 0.1]$ and $[0.9, 1]$ remaining below 10^{-3}. Notably, the standard AIR-PINN with fixed $a = 10$ achieves the lowest overall error in the interior, demonstrating that a moderate, fixed stretching factor can offer robust performance even without exploiting a highly localized transformation.

Table 5.5: Relative mean errors of AIR-PINN variants in the regions $[0, 0.1]$, $[0.1, 0.9]$, and $[0.9, 1]$ for $\epsilon = 10^{-1}$ and $\epsilon = 10^{-5}$ under successive training with fixed input transformation.

ϵ	Method	$[0, 0.1]$	$[0.1, 0.9]$	$[0.9, 1]$
10^{-1}	bias-free AIR-PINN, $a = \epsilon^{-1/2}$	4.52×10^{-7}	5.25×10^{-7}	8.39×10^{-7}
	AIR-PINN, $a = 10$	2.22×10^{-7}	2.77×10^{-7}	2.58×10^{-7}
	bias-free AIR-PINN, $a = 10$	2.18×10^{-7}	2.12×10^{-7}	3.23×10^{-7}
10^{-5}	bias-free AIR-PINN, $a = \epsilon^{-1/2}$	1.82×10^{-4}	1.73×10^{-5}	2.97×10^{-4}
	AIR-PINN, $a = 10$	2.87×10^{-4}	3.49×10^{-6}	3.22×10^{-4}
	bias-free AIR-PINN, $a = 10$	1.87×10^{-4}	3.13×10^{-5}	1.71×10^{-4}

The bias-free AIR-PINNs also perform remarkably well. The variant with $a = 10$ slightly outperforms the one with $a = \epsilon^{-1/2}$, suggesting that removing biases from the network does not hinder its ability to detect both boundary layers. Instead, it potentially stabilizes the effective shift

mechanism across training stages, aligning with observations from previous sections.

These results highlight that AIR-PINN, particularly in a bias-free configuration, can successfully handle problems with multiple boundary layers. The architectural flexibility, combined with judicious choice of input transformation, makes it a powerful tool for singular perturbation problems beyond the convection-diffusion setting.

5.5 2D Convection-Diffusion Problems

In this section, we extend our investigation of AIR-PINNs and successive training to the two-dimensional setting, focusing on convection-diffusion problems with boundary layers. Two-dimensional (2D) convection-diffusion equations arise in a wide range of physical applications, including heat transfer, fluid flow, and transport phenomena. Similar to their one-dimensional counterparts, these equations become singularly perturbed when the diffusion coefficient ϵ is small, resulting in steep boundary layers near outflow boundaries.

Compared to 1D problems, the 2D setting introduces new challenges: boundary layers may form along edges, corners, or within the domain, and their locations may vary depending on the direction and strength of the convection field. This makes 2D problems an ideal testbed for evaluating the ability of AIR-PINNs to learn and resolve multiple or unknown boundary layers.

We consider three representative examples of 2D convection-diffusion equations with known analytic solutions. Each example introduces different configurations of convection fields and boundary conditions, resulting in distinct boundary layer structures. These problems are featured in [89]:

- **Example 1:** A problem with constant vertical wind and zero source term. The solution

exhibits an exponential boundary layer of width $\mathcal{O}(\epsilon)$ along the outflow boundary $y = 1$.

- **Example 2:** A problem with variable vertical wind that generates symmetric characteristic boundary layers on the left and right sides of the domain. These so-called *shear layers* have width $\mathcal{O}(\sqrt{\epsilon})$, making them milder but still challenging to capture.

- **Example 3:** A problem with oblique wind direction and discontinuous boundary conditions that produce both an exponential boundary layer near $y = 1$ and an internal shear layer within the domain. The bottom and right boundary conditions are modified to provide a more localized and steep signal, making this a stringent test for network-based solvers.

We apply the same three settings in Section 5.4 and evaluate the effectiveness of the proposed methods in resolving sharp boundary layers. All models are trained using the successive training strategy described in previous sections. Quantitative and qualitative results are presented for each case, including error distributions and comparison against exact solutions. When exact solutions are not avaiable, we compare AIR-PINN approximations to FEM approximations generated over a fine mesh using IFISS [120, 121, 122]. We compare the performance of AIR-PINN variants with TS-PINN counterparts.

Recall that the 2D convection-diffusion equation is given by,

$$-\epsilon\nabla^2 u + \vec{w} \cdot \nabla u = f. \tag{5.19}$$

The three examples used in this chapter follow (5.19) with different wind $\vec{w}$ and boundary conditions.

5.5.1 Example 1: Vertical Wind, One Boundary Layer

The first example is also referenced in Chapter 4. Recan that the 2D convection-diffusion problem has $\vec{w} = \langle 0, 1 \rangle$ and $f = 0$. The exact solution is given by

$$u(x, y) = x \left(\frac{1 - e^{(y-1)/\epsilon}}{1 - e^{-2/\epsilon}} \right), \tag{5.20}$$

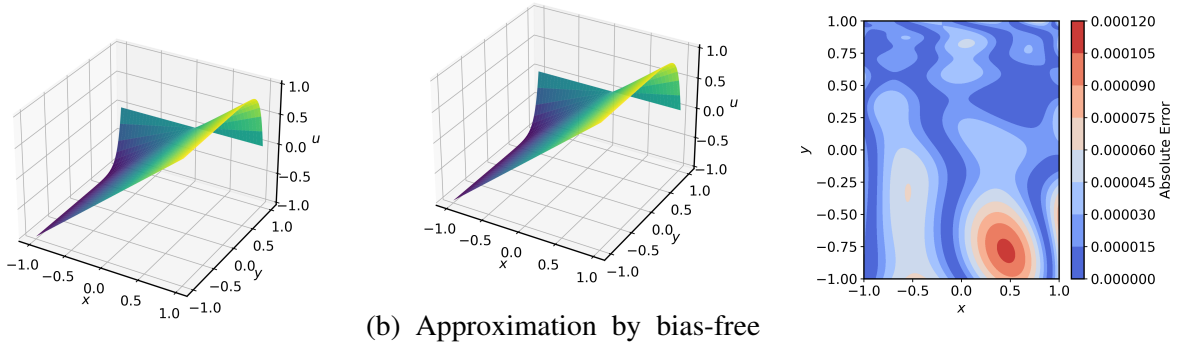
This solution satisfies the Dirichlet boundary conditions

$$u(x, -1) = x, \quad u(x, 1) = 0, \quad u(-1, y) \approx -1, \quad u(1, y) \approx 1, \tag{5.21}$$

where the latter two conditions are approximations that hold except near $y = 1$.

The problem features a vertical flow characterized by streamlines of the form $c(s) = (\alpha, s)$ for $\alpha \in (-1, 1)$, transporting solution values upward from $y = -1$ to $y = 1$. As $\epsilon \to 0$, the solution becomes approximately $u(x, y) \approx x$ throughout the domain except near the outflow boundary $y = 1$, where a steep exponential boundary layer emerges. This layer has width $O(\epsilon)$, and the solution undergoes a sharp transition to zero, as required by the Dirichlet condition at $y = 1$.

We evaluate the performance of AIR-PINN and TS-PINN variants on the first 2D convection-diffusion example, which features a vertical wind field $\vec{w} = \langle 0, 1 \rangle$ and a boundary layer forming along the outflow boundary $y = 1$. This setting serves as a baseline test for validating whether AIR-PINNs—particularly under bias-free architectures—can capture steep boundary transitions in 2D problems.

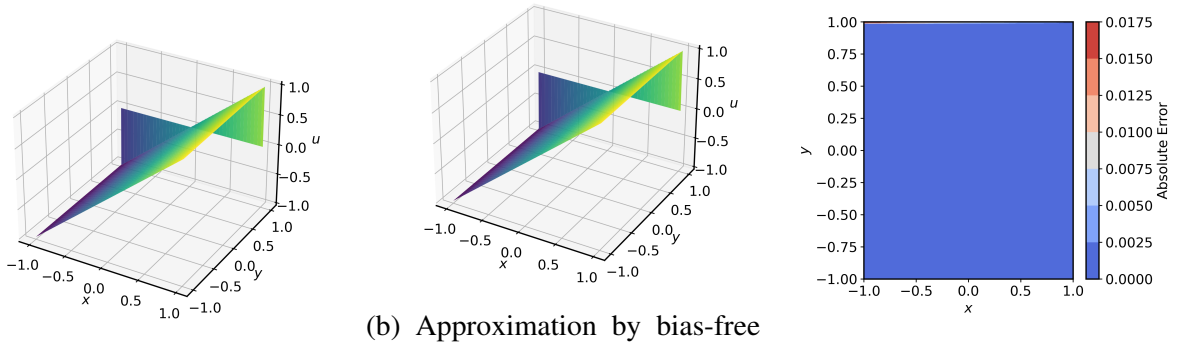


(a) Exact Solution for $\epsilon = 0.1$.

(b) Approximation by bias-free AIR-PINN.

(c) Absolute Error.

Figure 5.11: Comparison of exact solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.1$ in Example 1.



(a) Exact Solution for $\epsilon = 0.0001$.

(b) Approximation by bias-free AIR-PINN.

(c) Absolute Error.

Figure 5.12: Comparison of exact solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.0001$ in Example 1.

Figure 5.11 illustrates the results at $\epsilon = 0.1$. All configurations, including the bias-free AIR-PINN with either fixed transformation $a = 10$ or dynamic scaling $a = \epsilon^{-1/2}$, produce accurate approximations, with small pointwise errors concentrated near the outflow boundary. As seen in Table 5.6, the mean absolute errors remain below 10^{-4}, and the maximum error does not exceed 3.12×10^{-4} in any case. These results affirm the capability of all three AIR-PINN variants to resolve moderately sharp layers in relatively simple regimes.

As ϵ decreases to 10^{-4}, the challenge of resolving the boundary layer becomes more severe, as shown in Figure 5.12. Here, differences between the three configurations become

more pronounced. The bias-free AIR-PINN with transformation $a = \epsilon^{-1/2}$ delivers the best approximation among all variants, reducing both maximum and mean absolute errors to 1.70×10^{-2} and 3.53×10^{-5}, respectively. In contrast, the bias-free and standard AIR-PINNs using fixed $a = 10$ exhibit significantly larger errors, particularly near the outflow boundary. This behavior highlights that an inadequately scaled transformation can prevent the network from concentrating resolution capacity where the layer forms.

Table 5.6: Absolute errors (max and mean) of bias-free AIR-PINN and TS-PINN with fixed transformation, bias-free AIR-PINN and TS-PINN with $a = \epsilon^{-1/2}$ and standard AIR-PINN and TS-PINN with fixed transformation for various ϵ values and transformation settings.

ϵ	Configuration	Max Abs. Error	Mean Abs. Error
10^{-1}	bias-free AIR-PINN, $a = 10$	3.12×10^{-4}	7.85×10^{-5}
10^{-1}	bias-free AIR-PINN, $a = \epsilon^{-1/2}$	1.09×10^{-4}	3.00×10^{-5}
10^{-1}	AIR-PINN, $a = 10$	1.09×10^{-4}	7.78×10^{-6}
10^{-1}	bias-free TS-PINN, $a = 10$	2.78×10^{-4}	4.39×10^{-5}
10^{-1}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	6.22×10^{-5}	9.83×10^{-6}
10^{-1}	TS-PINN, $a = 10$	7.53×10^{-5}	1.09×10^{-5}
10^{-4}	bias-free AIR-PINN, $a = 10$	8.96×10^{-2}	4.62×10^{-4}
10^{-4}	bias-free AIR-PINN, $a = \epsilon^{-1/2}$	1.70×10^{-2}	3.53×10^{-5}
10^{-4}	AIR-PINN, $a = 10$	8.41×10^{-2}	1.51×10^{-4}
10^{-4}	bias-free TS-PINN, $a = 10$	2.28×10^{-1}	1.90×10^{-3}
10^{-4}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	2.31×10^{-2}	2.77×10^{-4}
10^{-4}	TS-PINN, $a = 10$	1.07×10^{-1}	8.27×10^{-4}

These observations support two key conclusions. First, the AIR-PINN architecture—especially when bias-free—can successfully learn steep boundary layers, provided the input transformation is suitably matched to the problem's scale. Second, the dynamic transformation $a = \epsilon^{-1/2}$ provides a more consistent strategy across varying ϵ, as it maintains the relative spatial scaling needed to target the steep boundary accurately. This validates the importance of either careful transformation

tuning or adaptive mechanisms (like auxiliary inputs) when applying AIR-PINNs to singularly perturbed 2D problems.

5.5.2 Example 2: Vertical Wind, Characteristic Layers

In this example, we consider the two-dimensional convection-diffusion equation with zero source term and a spatially varying vertical wind,

$$\vec{w} = \left(0,\ 1 + \frac{(x+1)^2}{4}\right), \quad f = 0. \tag{5.22}$$

The flow is strictly vertical, but its magnitude increases from left to right. Dirichlet boundary conditions are specified on the inflow and characteristic boundaries, while a homogeneous Neumann condition is imposed on the top boundary.

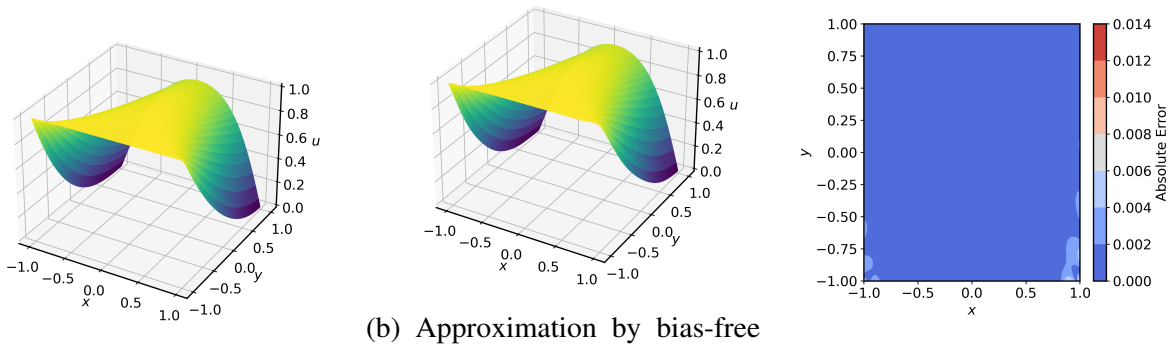
The boundary conditions are given as follows:

- $u = 1$ on the inflow boundary ($y = -1$),
- $\partial_y u = 0$ on the top boundary ($y = 1$),
- $u = 1 - (x+1)^2$ on the right boundary ($x = 1$),
- $u = 1 - (x+1)^3$ on the left boundary ($x = -1$).

Unlike Example 1, this setup does not produce exponential boundary layers. Instead, the mismatch between the reduced solution $\hat{u} \equiv 1$ and the boundary conditions on the characteristic sides leads to the formation of *characteristic layers*, also referred to as *shear layers*. These layers typically emerge in fluid flow problems along characteristic boundaries, and their width scales

with $\sqrt{\epsilon}$ instead of ϵ. Therefore, they are less steep than exponential boundary layers and easier to resolve numerically.

As the wind speed increases with x, the characteristic layer on the right boundary is sharper than that on the left. This setup allows us to evaluate the ability of AIR-PINN to handle spatially varying shear layers, complementing the boundary layer behavior studied in Example 1.

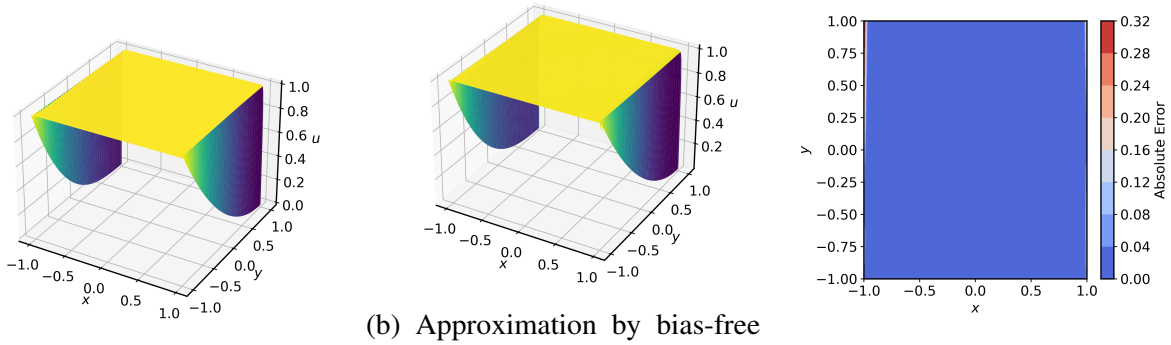


(a) IFISS Solution for $\epsilon = 0.1$.

(b) Approximation by bias-free AIR-PINN.

(c) Absolute Error.

Figure 5.13: Comparison of IFISS solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.1$ in Example 2.



(a) IFISS Solution for $\epsilon = 0.0001$.

(b) Approximation by bias-free AIR-PINN.

(c) Absolute Error.

Figure 5.14: Comparison of IFISS solution, AIR-PINN approximation, and absolute error for $\epsilon = 0.0001$ in Example 2.

In the second example, we consider a more complex convection-diffusion scenario where the vertical wind varies across the domain, leading to the formation of characteristic boundary layers. Unlike Example 1, this problem does not admit a closed-form analytical solution. Therefore, we

Table 5.7: Absolute difference (max and mean) between IFISS solution and bias-free AIR-PINN and TS-PINN with fixed transformation, bias-free AIR-PINN and TS-PINN with $a = \epsilon^{-1/2}$, and standard AIR-PINN and TS-PINN with fixed transformation for various ϵ values.

ϵ	Configuration	Max Abs. Error	Mean Abs. Error
10^{-1}	bias-free AIR-PINN, $a = 10$	6.04×10^{-3}	1.12×10^{-4}
10^{-1}	bias-free AIR-PINN, $a = \epsilon^{-1/2}$	8.98×10^{-3}	1.43×10^{-4}
10^{-1}	AIR-PINN, $a = 10$	9.60×10^{-3}	1.91×10^{-4}
10^{-1}	bias-free TS-PINN, $a = 10$	7.82×10^{-3}	1.79×10^{-4}
10^{-1}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	3.98×10^{-3}	7.49×10^{-5}
10^{-1}	TS-PINN, $a = 10$	7.61×10^{-3}	1.33×10^{-4}
10^{-4}	bias-free AIR-PINN, $a = 10$	1.14×10^{-1}	2.39×10^{-3}
10^{-4}	bias-free AIR-PINN, $a = \epsilon^{-1/2}$	4.25×10^{-2}	2.96×10^{-4}
10^{-4}	AIR-PINN, $a = 10$	6.52×10^{-2}	3.27×10^{-3}
10^{-4}	bias-free TS-PINN, $a = 10$	1.50×10^{-1}	3.25×10^{-3}
10^{-4}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	1.58×10^{-1}	3.12×10^{-3}
10^{-4}	TS-PINN, $a = 10$	3.67×10^{-1}	3.18×10^{-3}

compare AIR-PINN predictions to high-fidelity finite element approximations generated using IFISS. It is important to note that these IFISS approximations themselves carry discretization and numerical errors. As a result, the computed ”absolute errors” presented here reflect discrepancies relative to the IFISS reference, rather than true analytical deviations. In particular, large error values do not necessarily imply poor performance of the AIR-PINN model—indeed, the network may outperform the IFISS reference in some regions.

Figure 5.13 presents the comparison for $\epsilon = 0.1$. All three AIR-PINN configurations—bias-free with fixed $a = 10$, bias-free with dynamic $a = \epsilon^{-1/2}$, and standard AIR-PINN—produce good approximations. The absolute errors are low and mainly concentrated near the boundary layers, particularly where the IFISS solution exhibits rapid transitions due to varying inflow strength.

As ϵ decreases to 10^{-4}, the solution exhibits sharper layers, particularly near the left and

right characteristic boundaries. This poses a greater challenge for approximation. Figure 5.14 shows that the AIR-PINN variants remain capable of approximating the solution well, though the error patterns vary. The bias-free AIR-PINN with dynamical scaling achieves the lowest mean and max errors (Table 5.7). Although, it is important to keep in mind that the larger errors observed could partly reflect improved fidelity to the underlying PDE solution, particularly in regions where the IFISS solution may suffer from mesh-related inaccuracies.

Taken together, these results highlight the practical utility of AIR-PINNs for solving challenging 2D convection-diffusion settings. Even in the absence of ground truth, AIR-PINNs—especially in bias-free configurations—are able to learn solutions that track the known physical features of the problem. The fixed transformation and auxiliary input allow the network to consistently identify and resolve characteristic layers, with performance that is competitive with or potentially superior to standard numerical solvers.

5.5.3 Example 3: Constant 30° Wind, Multiple Boundary Layers

The third example is adapted from the standard benchmark in [89], originally featuring a constant wind field at a 30° angle to the left of vertical. In our soften the sharp inflow signals for larger ϵ. The constant wind direction is

$$\vec{w} = \left(-\frac{1}{2}, \frac{\sqrt{3}}{2}\right), \tag{5.23}$$

corresponding to a flow direction at 30° to the left of vertical.

The boundary conditions are as follows:

$$
\begin{aligned}
u(x,-1) &= \sigma\left(-\frac{2x}{\epsilon}\right), && \text{(bottom boundary)} \\
u(1,y) &= 1-\exp\left(\frac{y-1}{\epsilon}\right), && \text{(right boundary)} \\
u(x,y) &= 0 && \text{on the remaining boundaries.}
\end{aligned}
$$

Here, $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function, which introduces a steep transition in the boundary data along the bottom inflow edge.

Due to the oblique wind and the imposed inflow discontinuities, the solution features both:

- an **exponential boundary layer** along the top boundary $y = 1$, where the solution transitions from near 1 to 0 over a width $\mathcal{O}(\epsilon)$, and
- an **interior shear layer** aligned with the characteristic line $y + \sqrt{3}x =$ constant, along which the discontinuity in the inflow data is propagated.

This setup poses a significant challenge due to the simultaneous presence of steep boundary and interior layers and is used to test the robustness of AIR-PINN under complex multiscale dynamics.

In Example 3, we shift focus to a more intricate setting where the wind field is constant but tilted at a 30-degree angle toward the upper-left, inducing both downstream boundary layers and an interior layer. Unlike the previous cases, this problem introduces a discontinuous boundary condition, which results in a discontinuity in the reduced solution along a specific characteristic curve. Because of this discontinuity and the nontrivial directional structure of the convection, we do not employ the reduced PINN approach; instead, we directly approximate the solution using

the standard AI-PINN architecture with full inputs (x, y, a).

Successive training is conducted across three stages: from $\epsilon = 10^{-1}$ to 10^{-2}, then to 10^{-3}, and finally to 10^{-4}. This gradual refinement enables the network to progressively adapt to the evolving sharp features in the solution, particularly the interior layer that becomes increasingly localized as ϵ decreases.

Unlike Example 1, there is no closed-form solution for this problem. Moreover, the IFISS finite element package, used as a baseline in the previous example, cannot reliably approximate the solution at $\epsilon = 10^{-4}$. At such small diffusivity, the associated Péclet number grows large, and accurate resolution of the boundary and internal layers requires an extremely fine mesh. For instance, achieving acceptable resolution at $\epsilon = 10^{-4}$ would necessitate a grid of size $2^{13} \times 2^{13}$, resulting in a stiffness matrix with roughly $(2^{13})^2 \approx 6.7 \times 10^7$ degrees of freedom, and a corresponding solver scale of over 10^{14} entries in the worst-case dense matrix representation—beyond the capabilities of typical finite element solvers.

As a result, IFISS-generated approximations are only used as a reference up to $\epsilon = 10^{-3}$. At $\epsilon = 10^{-4}$, we rely entirely on the consistency of the successive AI-PINN predictions across stages and their ability to identify the correct locations and structures of boundary and interior layers without overfitting to noisy or incomplete reference data. This example serves as a stress test for the network's ability to resolve sharp gradients, discontinuities, and layered features without explicit reduction or finely-tuned numerical benchmarks.

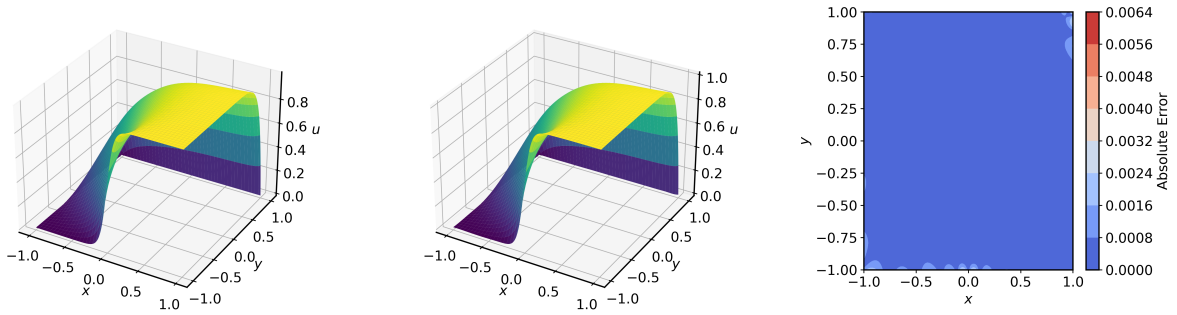
To visualize the performance of AI-PINN in this more complex 2D convection-diffusion example, we show results across four stages of successive training: $\epsilon = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$. At $\epsilon = 10^{-4}$, we only show the AI-PINN approximations.

Table 5.8: Absolute errors (max and mean) of bias-free AI-PINN with fixed transformation, bias-free AI-PINN with $a = \epsilon^{-1/2}$, and standard AI-PINN with fixed transformation for various ϵ values.

ϵ	Configuration	Max Abs. Error	Mean Abs. Error
10^{-1}	bias-free AI-PINN, $a = 10$	7.25×10^{-3}	1.39×10^{-4}
10^{-1}	bias-free AI-PINN, $a = \epsilon^{-1/2}$	6.52×10^{-3}	7.63×10^{-5}
10^{-1}	AI-PINN, $a = 10$	5.84×10^{-3}	1.40×10^{-4}
10^{-1}	bias-free TS-PINN, $a = 10$	4.51×10^{-3}	1.92×10^{-4}
10^{-1}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	5.12×10^{-3}	9.79×10^{-5}
10^{-1}	TS-PINN, $a = 10$	5.45×10^{-3}	1.29×10^{-4}
10^{-2}	bias-free AI-PINN, $a = 10$	3.34×10^{-2}	1.37×10^{-3}
10^{-2}	bias-free AI-PINN, $a = \epsilon^{-1/2}$	8.74×10^{-3}	3.14×10^{-4}
10^{-2}	AI-PINN, $a = 10$	9.20×10^{-3}	5.23×10^{-4}
10^{-2}	bias-free TS-PINN, $a = 10$	4.01×10^{-2}	6.71×10^{-4}
10^{-2}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	4.01×10^{-2}	6.71×10^{-4}
10^{-2}	TS-PINN, $a = 10$	4.74×10^{-2}	1.96×10^{-3}
10^{-3}	bias-free AI-PINN, $a = 10$	2.16×10^{-1}	2.42×10^{-3}
10^{-3}	bias-free AI-PINN, $a = \epsilon^{-1/2}$	4.56×10^{-1}	5.19×10^{-3}
10^{-3}	AI-PINN, $a = 10$	9.64×10^{-2}	4.38×10^{-4}
10^{-3}	bias-free TS-PINN, $a = 10$	1.41×10^{-1}	1.64×10^{-3}
10^{-3}	bias-free TS-PINN, $a = \epsilon^{-1/2}$	1.39×10^{-1}	1.59×10^{-3}
10^{-3}	TS-PINN, $a = 10$	5.62×10^{-2}	1.73×10^{-3}

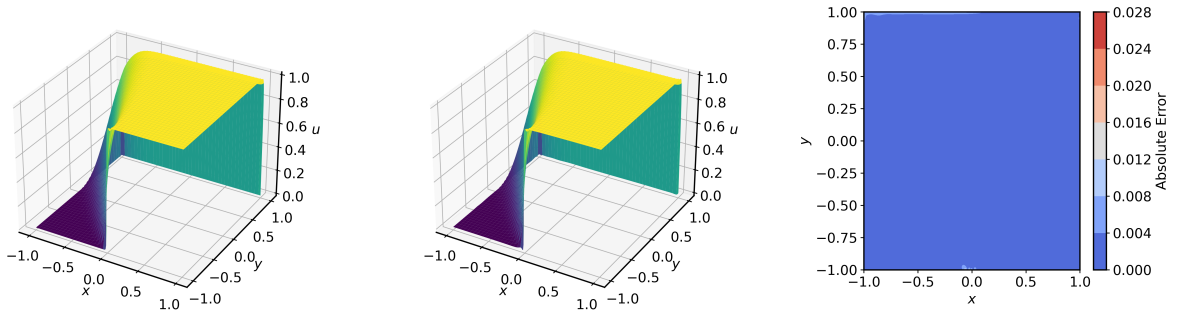
Table 5.8 summarizes the absolute maximum and mean errors for AI-PINN variants under successive training on Example 3, where no reduced solution is used. The configurations include a standard AI-PINN with fixed transformation, and two bias-free AI-PINNs with different transformation strategies: fixed $a = 10$ and adaptive $a = \epsilon^{-1/2}$.

Across all ϵ values, we observe that while all methods perform well at $\epsilon = 10^{-1}$, the relative performance begins to diverge as ϵ decreases. In particular, the standard AI-PINN with fixed $a = 10$ consistently achieves the lowest mean and max errors at smaller ϵ, especially at $\epsilon = 10^{-3}$,



(a) IFISS ($\epsilon = 10^{-1}$) (b) AI-PINN Prediction (c) Absolute Error

Figure 5.15: Comparison at $\epsilon = 10^{-1}$ using AI-PINN with constant a.

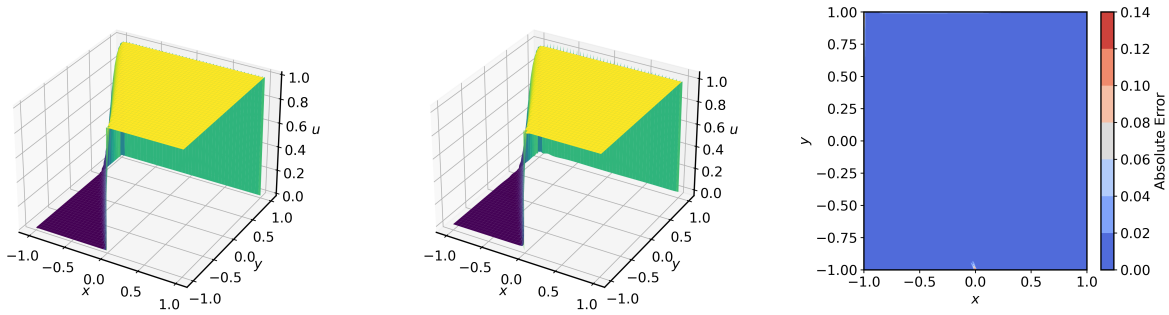


(a) IFISS ($\epsilon = 10^{-2}$) (b) AI-PINN Prediction (c) Absolute Error

Figure 5.16: Comparison at $\epsilon = 10^{-2}$ using AI-PINN with constant a.

where it outperforms both bias-free models by a significant margin. The bias-free AI-PINN with adaptive transformation performs well at moderate ϵ but exhibits larger errors at $\epsilon = 10^{-3}$, suggesting sensitivity to rapidly steepening gradients and potential issues with implicit shift localization. When examining the absolute error, the largest error occurs at the bottom boundary. Therefore, a tailored weighting scheme where heavier weights used for boundary samples could improve the performance.

Interestingly, the bias-free AI-PINN with fixed transformation $a = 10$ remains competitive throughout, highlighting the utility of combining fixed spatial scaling with a simplified architecture for stability. However, its performance also degrades at smaller ϵ, albeit more gracefully than the adaptive counterpart. These results reinforce the importance of balancing architectural constraints



(a) IFISS ($\epsilon = 10^{-3}$) (b) AI-PINN Prediction (c) Absolute Error

Figure 5.17: Comparison at $\epsilon = 10^{-3}$ using AI-PINN with constant a.

with transformation strategy, especially in multi-phase successive training.

Figure 5.18 shows the AI-PINN approximation generated for $\epsilon = 0.0001$. The approximation sharply transitions along the vertical interface, capturing the internal boundary layer characteristic of this regime. Despite the high Péclet number, the AI-PINN manages to adequately resolve internal and boundary layers. This highlights the model's ability to extend beyond traditional solvers. The smooth behavior away from the layer and the lack of spurious oscillations further illustrate the effectiveness of combining fixed input transformation and architectural simplicity in the presence of steep gradient layers.

5.6 Discussion

This chapter presented a comprehensive study of PINNs for solving singularly perturbed PDEs with unknown boundary layer locations, with a focus on designing architectures and training strategies that allow for effective gradient localization without prior knowledge. Our central contribution—the Auxiliary-Input PINN (AI-PINN)—demonstrated significant improvements from RPINN or standard PINN in adaptability, particularly when combined with carefully chosen training strategies.

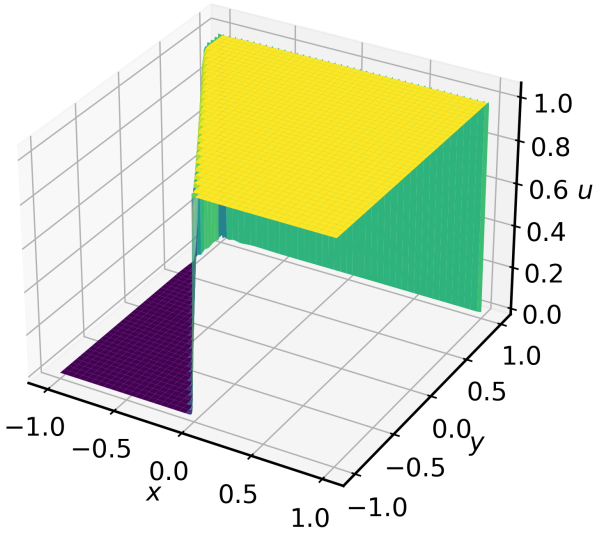


Figure 5.18: AI-PINN with constant a prediction at $\epsilon = 10^{-4}$; no IFISS reference due to mesh limitations.

The introduction of the auxiliary input a expanded the expressiveness of the network by allowing it to implicitly learn the shift required to localize steep gradients. This innovation was validated through theoretical analysis of the effective shift, and extensive testing on convection-diffusion and reaction-diffusion problems in both one and two dimensions. In 1D settings, Auxiliary-Input Reduced PINN (AIR-PINN) was shown to resolve boundary layers more effectively than Reduced PINN (RPINN) or Two-Scale PINN (TS-PINN), particularly when aided by successive training. In 2D, AIR-PINN generalized well, resolving multiple boundary layers even under steep gradients and complex flows.

However, naive application of successive training was shown to introduce inconsistencies due to varying input transformations between training stages. Our analysis of effective shift dynamics revealed that RPINN suffers significantly from this variation, while AIR-PINN and TS-PINN offer greater resilience due to their ability to learn shift information via auxiliary weights. To address this, we proposed two remedies: (1) fixing the input transformation across stages,

and (2) using bias-free networks to eliminate spurious contributions to the effective shift. Both strategies were found to stabilize the learning process, improve performance, and preserve the learned boundary layer locations across training phases.

We also extended the analysis with a Neural Tangent Kernel (NTK) perspective, which supported the idea that the early dynamics of gradient descent are driven by implicit structure learned during previous stages. NTK eigenvector analysis further revealed that the learned modes in the early stage already align with the eventual steep gradient location in later stages, confirming the importance of initializing with informative prior solutions.

Finally, we evaluated AIR-PINN across multiple 2D convection-diffusion examples, each designed to challenge different aspects of gradient localization, such as multiple boundary layers or directional transport. Even in the absence of a reduced formulation or exact solutions, AIR-PINN produced highly accurate results, outperforming baselines and offering a robust pathway to tackling high-dimensional singularly perturbed PDEs.

These results illustrate the power and flexibility of AIR-PINN combined with principled training strategies. Looking forward, these methods offer a promising foundation for extending PINNs to even more complex systems, such as time-dependent problems, anisotropic diffusions, or high-Péclet-number flows in irregular geometries.

Chapter 6: Conclusion and Future Work

In this dissertation, we have shown that the most effective way for a Physics-Informed Neural Network (PINN) to resolve narrow boundary or internal layers is to embed *layer-aware information* directly into its inputs. By incorporating such information, the network's internal dynamics—such as weights, biases, activation sensitivities, and ultimately the neural tangent kernel (NTK) spectrum—are reorganized. This reorganization ensures that steep-gradient modes are captured and prioritized early in training, enabling the network to effectively approximate complex features in the solution.

We demonstrated this principle in two distinct contexts: Richards' equation and convection–diffusion problems. In both cases, we showed that network performance improved significantly when the correct input features were provided. This strategy not only enhanced learning but also reduced the need for extensive hyperparameter tuning or architectural changes. For Richards' equation, incorporating the surface flux $q(t)$ proved essential. Without this information, the network fails to detect abrupt infiltration or evaporation events that generate wetting fronts. Similarly, for convection–diffusion problems, the key input was a stretched and shifted spatial coordinate $T(x) = a(x - b)$. This transformation magnifies the boundary layer and centers the activation's sensitive region precisely where the gradient is sharp, allowing the network to resolve steep solution variations.

We also introduced problem-specific modifications to tailor the networks. For Richards' equation, the network suffered from causality issues that resulted in physically inconsistent approximations. To address this, we proposed a *Discrete-Residual* (DR) loss function that enforces temporal causality at each time step Δt in the residual sample data. This causality-aware formulation penalizes deviations from the expected temporal behavior, yielding more accurate predictions. We validated this approach under various conditions, including cases with noise-contaminated samples, and demonstrated strong performance in Chapter 3.

For convection–diffusion problems, we initially assumed that the location of the gradient layer was known. We introduced the transformation $T(x) = a(x - b)$, where a controls the width of the layer and b determines its location. Our initial attempt used PINNs to correct oscillatory approximations from traditional methods like finite difference and finite element methods. However, we observed that the network struggled to eliminate oscillations, and iterative corrections failed to converge. We then shifted to using PINNs as correctors for reduced solutions of the PDE, leading to the *Reduced PINN* (RPINN) framework. This approach enabled us to concentrate the network's capacity on resolving steep gradients and to generalize our analysis.

We found that standard PINNs often fail to learn the correct gradient layer, producing solutions with incorrect flow direction. Through analysis, we identified necessary conditions for the network to correctly learn the steep gradient: the weights in the first layer must be large enough for the activation functions to respond to sharp features. Moreover, we found that the location of the gradient layer corresponds to the *effective shift* in the first layer. NTK analysis confirmed that this shift determines where steep gradients appear in the dominant eigenvectors—i.e., the modes learned first. This insight motivated the development of the transformation $T(x) = a(x - b)$, which grants control over the effective shift and improves network convergence. We successfully

demonstrated this approach in Chapter 4 for both 1D and 2D convection–diffusion problems.

Recognizing that the gradient layer location is not always known in practice, we proposed the *Auxiliary-Input Reduced PINN* (AIR-PINN) framework. This method allows the network to learn the optimal shift internally via *successive training* when b is unknown. By transferring weights and biases learned from training at higher viscosity values, the network is guided toward the correct gradient layer location at lower viscosity. This eliminates the need for prior knowledge of the gradient location. However, we observed that dynamically adjusting the stretch factor a as $\epsilon^{-1/2}$ causes previously learned information to be lost. To overcome this, we introduced two fixes: (1) fixing the stretching factor a across successive training cycles, and (2) using a network without bias terms. These modifications significantly improved the network's ability to resolve steep gradients by leveraging the effective shift learned in earlier cycles. Chapter 5 illustrates the success of this approach on 1D and 2D problems, showing that it can provide accurate approximations when traditional solvers become computationally expensive.

Future work will advance the "input-guided" PINN philosophy along three complementary directions. First, we will transition from hand-crafted transformations to *adaptive input learning*, enabling the network to optimize its own stretch-and-shift map—or learn a low-dimensional latent map—so that steep layers can be automatically identified in cases with unknown geometry. Second, we will extend this methodology to three-dimensional transport and flow problems, which involves learning coordinates aligned with complex boundary or internal layers, and integrating them into tensor-product neural architectures that scale effectively with dimensionality. Third, we will evaluate the framework on more complex PDEs, such as the Navier–Stokes equations, where the solution involves multiple interacting layers. This will necessitate more advanced input transformations and architectural choices to ensure accurate gradient resolution. We will also

explore the use of this framework for time-dependent PDEs in which the steep gradient layers evolve over time.

Resolving steep gradients in PDEs is a persistent challenge for both classical numerical solvers and neural network-based approaches like PINNs. This dissertation introduced a promising framework—input-guided PINNs—that embeds layer-aware information directly into the inputs, thereby significantly enhancing the network's ability to learn and approximate solutions with steep features. We are excited about the potential of this approach and look forward to extending its reach in future research.

Bibliography

[1] CivilDigital. Infiltration process diagram, n.d. Accessed: 2025-03-30.

[2] Qiu-Yu Yan, Pei Leng, Zhao-Liang Li, Qian-Yu Liao, Fang-Cheng Zhou, Xiao-Jing Han, Jianwei Ma, Ya-Yong Sun, Xia Zhang, and Guo-Fei Shang. A method for estimating spatially continuous soil moisture from the synergistic use of geostationary and polar-orbit satellite data. *Journal of Hydrology*, 608:127590, 2022.

[3] Y. Wang and Q. Ren. *Deep Learning-Based Forward Modeling and Inversion Techniques for Computational Physics Problems*. Springer, 2023.

[4] W. Chen, J. Song, and P. Ren. Data-efficient operator learning via unsupervised pretraining and in-context learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[5] A. J. Said and A. M. Ismail. Trends in natural language processing for text classification: A comprehensive survey. *ResearchGate*, 2025.

[6] H. Koushal, R. Kaur, and C. Kaur. The relative review of machine learning in natural language processing (nlp). *ResearchGate*, 2025.

[7] M. A. Awan, K. M. Abbasi, and T. Ismail. A deep learning approach to semantic clarity in urdu translations of the holy quran. *ResearchGate*, 2025.

[8] J. Mashhura. Advancing machine translation: Challenges, innovations, and the future of nlp in low-resource languages. *Journal of International Scientific Research*, 2025.

[9] I. S. B. Azhar and W. K. Sari. A comparative study of deep learning's performance methods for news article using word representations. *Sistemasi: Jurnal Sistem Informasi*, 2025.

[10] B. Jalali, Y. Zhou, and A. Kadambi. Physics-ai symbiosis. *Machine Learning*, 2022.

[11] F. A. Gonzalez. *On Deep Learning for Computational Fluid Dynamics*. PhD thesis, HAL Thesis, 2023.

[12] P. Rajendra and V. Brahmajirao. Modeling of dynamical systems through deep learning. *Biophysical Reviews*, 2020.

[13] G. Akhtar and W. Shah. Ai in mathematics and steam learning: A design-based research approach to enhancing digital education. *ResearchGate*, 2025.

[14] R. Ajax, S. Joseph, and A. Luz. Machine learning for cost estimation in new product development. *ResearchGate*, 2025.

[15] Y. Xu, X. Gan, J. Pei, W. Wang, and J. Chen. Applications of artificial intelligence and computational intelligence in hydraulic optimization of centrifugal pumps: A comprehensive review. *Engineering Applications of Artificial Intelligence*, 2025.

[16] A. N. Mir, D. R. Rizvi, and M. R. Ahmad. Enhancing histopathological image analysis: An explainable vision transformer approach with comprehensive interpretation methods. *Engineering Applications of Artificial Intelligence*, 2025.

[17] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. Available at: `https://arxiv.org/pdf/1703.09039`.

[18] A. Shrestha and A. Mahmood. Review of deep learning algorithms and architectures. *IEEE Access*, 7:53040–53065, 2019. Available at: `https://ieeexplore.ieee.org/iel7/6287639/8600701/08694781.pdf`.

[19] I. Ahmed. Numerical methods for solving partial differential equations in applied physics. *Frontiers in Applied Physics and Mathematics*, 2024. Available at: `https://sprcopen.org/FAPM/article/download/26/13`.

[20] M. Frehner, S. M. Schmalholz, and E. H. Saenger. Comparison of finite difference and finite element methods for simulating two-dimensional scattering of elastic waves. *Physics of the Earth and Planetary Interiors*, 171(1-4):101–108, 2008. Available at: `https://www.academia.edu/download/111739269/Frehner_et_al_PEPI_2008.pdf`.

[21] L. E. Jalil, N. H. Malik, and M. K. Hussein. A comparative study of partial differential equation solving methods and their applications. *Bilangan: Jurnal Ilmiah Matematika dan Pendidikan Matematika*, 2025. Available at: `https://journal.arimsi.or.id/index.php/Bilangan/article/download/375/551`.

[22] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[24] L. Mandl, S. Goswami, L. Lambers, and T. Ricken. Separable deeponet: Breaking the curse of dimensionality in physics-informed machine learning. *arXiv Preprint*, 2024. Available at: `https://arxiv.org/pdf/2407.15887`.

[25] L. Mandl, S. Goswami, L. Lambers, and T. Ricken. Separable physics-informed deeponet: Breaking the curse of dimensionality in physics-informed machine learning. *Computer Methods in Applied Mechanics and Engineering*, 2025.

[26] A. R. Brink, D. A. Najera-Flores, and C. Martinez. The neural network collocation method for solving partial differential equations. *Neural Computing and Applications*, 2021.

[27] Z. Li, H. Song, D. Xiao, Z. Lai, and W. Wang. Harnessing scale and physics: A multi-graph neural operator framework for pdes on arbitrary geometries. *arXiv preprint*, 2024.

[28] X. Morales Ferez, J. Mill, K. A. Juhl, and C. Acebes. Deep learning framework for real-time estimation of in-silico thrombotic risk indices in the left atrial appendage. *Frontiers in Physiology*, 2021.

[29] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[30] Weilong Guan, Kaihan Yang, Yinsheng Chen, Shaolin Liao, and Zhong Guan. A dimension-augmented physics-informed neural network (dapinn) with high level accuracy and efficiency. *J. Comput. Phys.*, 491:Paper No. 112360, 2023.

[31] D. Bigoni. *Uncertainty quantification with applications to engineering problems*. PhD thesis, Technical University of Denmark, 2015.

[32] S. Arridge, P. Maass, O. Öktem, and C. B. Schönlieb. Solving inverse problems using data-driven models. *Acta Numerica*, 2019.

[33] H. Kersting. *Uncertainty-Aware Numerical Solutions of ODEs by Bayesian Filtering*. PhD thesis, University of Tübingen, 2021.

[34] W. Cho, M. Jo, H. Lim, K. Lee, and S. Hong. Parameterized physics-informed neural networks for parameterized pdes. *arXiv preprint arXiv:2408.09446*, 2024.

[35] M. Penwarden, S. Zhe, and A. Narayan. Physics-informed neural networks (pinns) for parameterized pdes: a metalearning approach. *arXiv preprint arXiv:2110.13361*, 2021.

[36] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[37] Luning Sun, Han Gao, Shaowu Pan, and Jian-Xun Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

[38] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

[39] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

[40] Zhiwei Fang and Justin Zhan. Deep physical informed neural networks for metamaterial design. *IEEE Access*, 8:24506–24513, 2019.

[41] Dehao Liu and Yan Wang. Multi-fidelity physics-constrained neural network and its application in materials modeling. *Journal of Mechanical Design*, 141(12):121403, 2019.

[42] Sifan Wang and Paris Perdikaris. Deep learning of free boundary and stefan problems. *Journal of Computational Physics*, 428:109914, 2021.

[43] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[44] Jeremy Yu, Lu Lu, Xuhui Meng, and George Em Karniadakis. Gradient-enhanced physics-informed neural networks for forward and inverse pde problems. *Comput. Methods Appl. Mech. Engrg.*, 393:Paper No. 114823, 22, 2022.

[45] H. Gao, M. J. Zahr, and J. X. Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 2022.

[46] A. Vadeboncoeur, Ö. D. Akyildiz, and I. Kazlauskaite. Fully probabilistic deep models for forward and inverse problems in parametric pdes. *Journal of Computational Physics*, 2023.

[47] A. Lininger. *Design Optimization and Fabrication of Photonic Metamaterials*. PhD thesis, ProQuest Dissertations Publishing, 2024.

[48] M. Jokar. *Machine Learning Models for Computational Structural Mechanics*. PhD thesis, ProQuest Dissertations Publishing, 2023.

[49] Z. Wu, H. Wang, C. He, B. Zhang, and T. Xu. The application of physics-informed machine learning in multiphysics modeling in chemical engineering. *Industrial & Engineering Chemistry Research*, 2023.

[50] S. Huang, W. Feng, C. Tang, and J. Lv. Partial differential equations meet deep neural networks: A survey. *arXiv preprint*, 2022.

[51] D. He, S. Li, W. Shi, X. Gao, and J. Zhang. Learning physics-informed neural networks without stacked back-propagation. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2023.

[52] Sifan Wang, Xuhui Yu, and Paris Perdikaris. On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

[53] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks, 2022.

[54] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310, 2019.

[55] Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Commun. Comput. Phys.*, 28(5):1746–1767, 2020.

[56] Guangtao Zhang, Huiyu Yang, Fang Zhu, and et al. Yang Chen. Dasa-pinns: Differentiable adversarial self-adaptive pointwise weighting scheme for physics-informed neural networks. *SSRN*, 2023.

[57] Xi-An Li, Zhi-Qin John Xu, and Lei Zhang. A multi-scale dnn algorithm for nonlinear elliptic equations with multiple scales. *Commun. Comput. Phys.*, 28(5):1886–1906, 2020.

[58] Xi-An Li, Zhi-Qin John Xu, and Lei Zhang. Subspace decomposition based dnn algorithm for elliptic type multi-scale pdes. *J. Comput. Phys.*, 488:Paper No. 112242, 17, 2023.

[59] Bo Wang, Wenzhong Zhang, and Wei Cai. Multi-scale deep neural network (mscalednn) methods for oscillatory stokes flows in complex domains. *Commun. Comput. Phys.*, 28(5):2139–2157, 2020.

[60] Wei Cai, Xiaoguang Li, and Lizuo Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM J. Sci. Comput.*, 42(5):A3285–A3312, 2020.

[61] Zhiwei Fang, Sifan Wang, and Paris Perdikaris. Ensemble learning for physics informed neural networks: A gradient boosting approach. *arXiv preprint*, 2023.

[62] Zhiwei Gao, Liang Yan, and Tao Zhou. Failure-informed adaptive sampling for pinns. *SIAM J. Sci. Comput.*, 45(4):A1971–A1994, 2023.

[63] Zhiwei Gao, Tao Tang, Liang Yan, and Tao Zhou. Failure-informed adaptive sampling for pinns, part ii: Combining with re-sampling and subset simulation. *arXiv preprint*, 2023.

[64] Giulia Bertaglia, Chuan Lu, Lorenzo Pareschi, and Xueyu Zhu. Asymptotic-preserving neural networks for multiscale hyperbolic models of epidemic spread. *Math. Models Methods Appl. Sci.*, 32(10):1949–1985, 2022.

[65] Yiqi Gu, Haizhao Yang, and Chao Zhou. Selectnet: Self-paced learning for high-dimensional partial differential equations. *J. Comput. Phys.*, 441:Paper No. 110444, 18, 2021.

[66] Sokratis J. Anagnostopoulos, Juan Diego Toscano, Nikolaos Stergiopulos, and George Em Karniadakis. Residual-based attention and connection to information bottleneck theory in pinns. *Comput. Methods Appl. Mech. Engrg.*, 421:116805, 2024.

[67] Wojciech M. Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30. Curran Associates, Inc., 2017.

[68] Hwijae Son, Jin Woo Jang, Woo Jin Han, and Hyung Ju Hwang. Sobolev training for physics-informed neural networks. *Commun. Math. Sci.*, 21(6):1679–1705, 2023.

[69] Zixue Xiang, Wei Peng, Xu Liu, and Wen Yao. Self-adaptive loss balanced physics-informed neural networks. *Neurocomputing*, 496:11–34, 2022.

[70] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *Comput. Methods Appl. Mech. Engrg.*, 421:116813, 2024.

[71] Jihun Han and Yoonsang Lee. Hierarchical learning to solve pdes using physics-informed neural networks. In *Computational Science – ICCS 2023: 23rd International Conference, Prague, Czech Republic, July 3–5, 2023, Proceedings, Part III*, page 548–562, Berlin, Heidelberg, 2023. Springer-Verlag.

[72] Sven Lämmle, Can Bogoclu, Kevin Cremanns, and Dirk Roos. Gradient and uncertainty enhanced sequential sampling for global fit. *Comput. Methods Appl. Mech. Engrg.*, 415:Paper No. 116226, 23, 2023.

[73] Jay L Clausen, Jason R Dorvee, Anna M Wagner, Susan Frankenstein, Blaine F Morriss, Keran J Claffey, Terrance M Sobecki, Christopher R Williams, Stephen D Newman, Brandon K Booker, et al. Spatial and temporal variance in the thermal response of buried objects. Technical Report ERDC/CRREL TR-20-10, US Army Engineer Research and Development Center, Cold Regions Research and Engineering Laboratory, 2020.

[74] Jay L Clausen, Susan Frankenstein, Jason R Dorvee, Austin K Workman, Blaine F Morriss, Keran J Claffey, Terrance M Sobecki, Christopher R Williams, Stephen D Newman, Brandon K Booker, et al. Spatial and temporal variance of soil and meteorological properties affecting sensor performance—phase 2. Technical Report ERDC/CRREL TR-21-10, U.S. Army Engineer Research and Development Center, Cold Regions Research and Engineering Laboratory, 2021.

[75] Jay L Clausen, Christopher Felt, Michael Musty, Vuong Truong, Susan Frankenstein, Anna Wagner, Rosa Affleck, Steven Peckham, and Christopher Williams. Modernizing environmental signature physics for target detection - phase 3. Technical Report ERDC/CRREL TR-22-4, U.S. Army Engineer Research and Development Center, Cold Regions Research and Engineering Laboratory, 2022.

[76] Michael Seidel, Michael Vohland, Isabel Greenberg, Bernard Ludwig, Malte Ortner, Sören Thiele-Bruhn, and Christopher Hutengs. Soil moisture effects on predictive vnir and mir modeling of soil organic carbon and clay content. *Geoderma*, 427:116103, 2022.

[77] Lawrence Carin, Norbert Geng, Mark McClure, Yanting Dong, Zhijun Liu, Jiangqi He, Jeffrey Sichina, Marc Ressler, Lam Nguyen, and Anders Sullivan. Wide-area detection of land mines and unexploded ordnance. *Inverse Problems*, 18(3):575, apr 2002.

[78] Volker Dworak, Sven Augustin, and Robin Gebbers. Application of terahertz radiation to soil measurements: Initial results. *Sensors*, 11(10):9973–9988, 2011.

[79] Jay L Clausen, Vuong Truong, Sophia Bragdon, Susan Frankenstein, Ann Wagner, Rosa Affleck, and Christopher Williams. Buried-object detection improvements incorporating

environmental phenomenology into signature physics. Technical Report ERDC/CRREL TR-22-19, U.S. Army Engineer Research and Development Center, Cold Regions Research and Engineering Laboratory, 2022.

[80] Matthew W Farthing and Fred L Ogden. Numerical solution of richards' equation: A review of advances and challenges. *Soil Science Society of America Journal*, 81(6):1257–1269, 2017.

[81] DA Barry, J-Y Parlange, GC Sander, and M Sivaplan. A class of exact solutions for richards' equation. *Journal of Hydrology*, 142(1-4):29–46, 1993.

[82] Po Jo Ross. Efficient numerical methods for infiltration using richards' equation. *Water Resources Research*, 26(2):279–290, 1990.

[83] Florian List and Florin A Radu. A study on iterative methods for solving richards' equation. *Computational Geosciences*, 20:341–353, 2016.

[84] Zhanpeng Pan, Yu Tong, Jingming Hou, Jian Zheng, Yongde Kang, Yan Wang, and Changchong Cao. Hole irrigation process simulation using a soil water dynamical model with parameter inversion method. *Agricultural Water Management*, 245:106542, 2021.

[85] G.L. Mahon. Assessment of groundwater pathways and contaminant transport in florida and georgia using multiple chemical and microbiological indicators. `https://pubs.usgs.gov/fs/2011/3070/`, 2011. U.S. Geological Survey Fact Sheet 2011–3070. Available online only.

[86] Yulong Zhu, Tatsuya Ishikawa, Srikrishnan Siva Subramanian, and Bin Luo. Simultaneous analysis of slope instabilities on a small catchment-scale using coupled surface and subsurface flows. *Engineering Geology*, 275:105750, 2020.

[87] Michael A Celia, Efthimios T Bouloutas, and Rebecca L Zarba. A general mass-conservative numerical solution for the unsaturated flow equation. *Water resources research*, 26(7):1483–1496, 1990.

[88] M Th Van Genuchten. A closed-form equation for predicting the hydraulic conductivity of unsaturated soils. *Soil science society of America journal*, 44(5):892–898, 1980.

[89] Howard Elman, David Silvester, and Andy Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics*. Oxford University Press, 06 2014.

[90] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[91] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[92] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[93] Bart Van Merriënboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ml: Where we are and where we should be going. *Advances in Neural Information Processing Systems*, 31, 2018.

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

[95] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(1–3):503–528, August 1989.

[96] Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Nuñez. A multiscale neural network based on hierarchical matrices. *Multiscale Modeling & Simulation*, 17(4):1189–1213, 2019.

[97] Qiao Zhuang, Chris Ziyi Yao, Zhongqiang Zhang, and George Em Karniadakis. Two-scale neural networks for partial differential equations with small parameters. *arXiv preprint arXiv:2402.17232*, 2024.

[98] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Neel Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.

[99] Jiajing Guan, Sophia P. Bragdon, Jay L. Clausen, United States Army Corps of Engineers, Engineer Research, Development Center (U.S.), Cold Regions Research, Engineering Laboratory (U.S.), and Military Engineering Technology Program (U.S.). *Predicting soil moisture content using Physics-Informed Neural Networks (PINNs)*. U.S. Army Engineer Research and Development Center, Cold Regions Research and Engineering Laboratory, Hanover, NH, 2024. ERDC/CRREL TR; 24-8; TR-24-8. 1 online resource (52 pages) : pdf.

[100] Hennie Daniels and Marina Velikova. Monotone and partially monotone neural networks. *IEEE Transactions on Neural Networks*, 21(6):906–917, 2010.

[101] Toshiyuki Bandai and Teamrat A Ghezzehei. Physics-informed neural networks with monotonicity constraints for richardson-richards equation: Estimation of constitutive relationships and soil water flux density from volumetric water content measurements. *Water Resources Research*, 57(2):e2020WR027642, 2021.

[102] T. Bandai and T. A. Ghezzehei. Forward and inverse modeling of water flow in unsaturated soils with discontinuous hydraulic conductivities using physics-informed neural networks with domain decomposition. *Hydrology and Earth System Sciences*, 26(16):4469–4495, 2022. [doi:10.5194/hess-26-4469-2022].

[103] Jirka Simunek, M Th Van Genuchten, and M Sejna. The hydrus-1d software package for simulating the one-dimensional movement of water, heat, and multiple solutes in variably-saturated media. *University of California-Riverside Research Reports*, 3:1–240, 2005.

[104] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[105] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pages 1–1, 2016.

[106] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

[107] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468, 1997.

[108] Ravi G. Patel, Indu Manickam, Nathaniel A. Trask, Mitchell A. Wood, Myoungkyu Lee, Ignacio Tomas, and Eric C. Cyr. Thermodynamically consistent physics-informed neural networks for hyperbolic systems. *Journal of Computational Physics*, 449:110754, 2022.

[109] *Physics informed deep learning for flow and transport in porous media*, volume Day 1 Tue, October 26, 2021 of *SPE Reservoir Simulation Conference*, 10 2021.

[110] Olga Fuks and Hamdi A Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020.

[111] Bernardo Llanas, Sagrario Lantarón, and Francisco Sáinz. Constructive approximation of discontinuous functions by neural networks. *Neural Processing Letters*, 27:209–226, 06 2008.

[112] W. He, J. Li, X. Kong, and L. Deng. Multi-level physics informed deep learning for solving partial differential equations in computational structural mechanics. *Communications Engineering*, 3(1):1–12, 2024.

[113] Ziad Aldirany, R. Cottereau, and S. Prudhomme. Error reduction in neural network approximations of boundary-value problems by a multi-level approach. Technical report, HAL Open Science, 2024.

[114] Ziad Aldirany. *Accurate Approximations of the Wave Equation: From Spectral Element Methods to Deep Learning Approaches*. PhD thesis, Polytechnique Montréal, 2024.

[115] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[116] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[117] Chen Xu, Ba Trung Cao, Yong Yuan, and Günther Meschke. Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. *Computer Methods in Applied Mechanics and Engineering*, 405:115852, 2023.

[118] Raphaël Pellegrin, Blake Bullwinkel, Marios Mattheakis, and Pavlos Protopapas. Transfer learning with physics-informed neural networks for efficient simulation of branched flows. *arXiv preprint arXiv:2211.00214*, 2022.

[119] Hans-Görg Roos, Martin Stynes, and Lutz Tobiska. *Robust Numerical Methods for Singularly Perturbed Differential Equations*. Springer, 2008.

[120] Howard Elman, Alison Ramage, and David Silvester. Algorithm 866: IFISS, a Matlab toolbox for modelling incompressible flow. *ACM Trans. Math. Softw.*, 33:2–14, 2007.

[121] Howard Elman, Alison Ramage, and David Silvester. IFISS: A computational laboratory for investigating incompressible flow problems. *SIAM Review*, 56:261–273, 2014.

[122] David Silvester, Howard Elman, and Alison Ramage. Incompressible Flow and Iterative Solver Software (IFISS) version 3.5, September 2016. `http://www.manchester.ac.uk/ifiss/`.