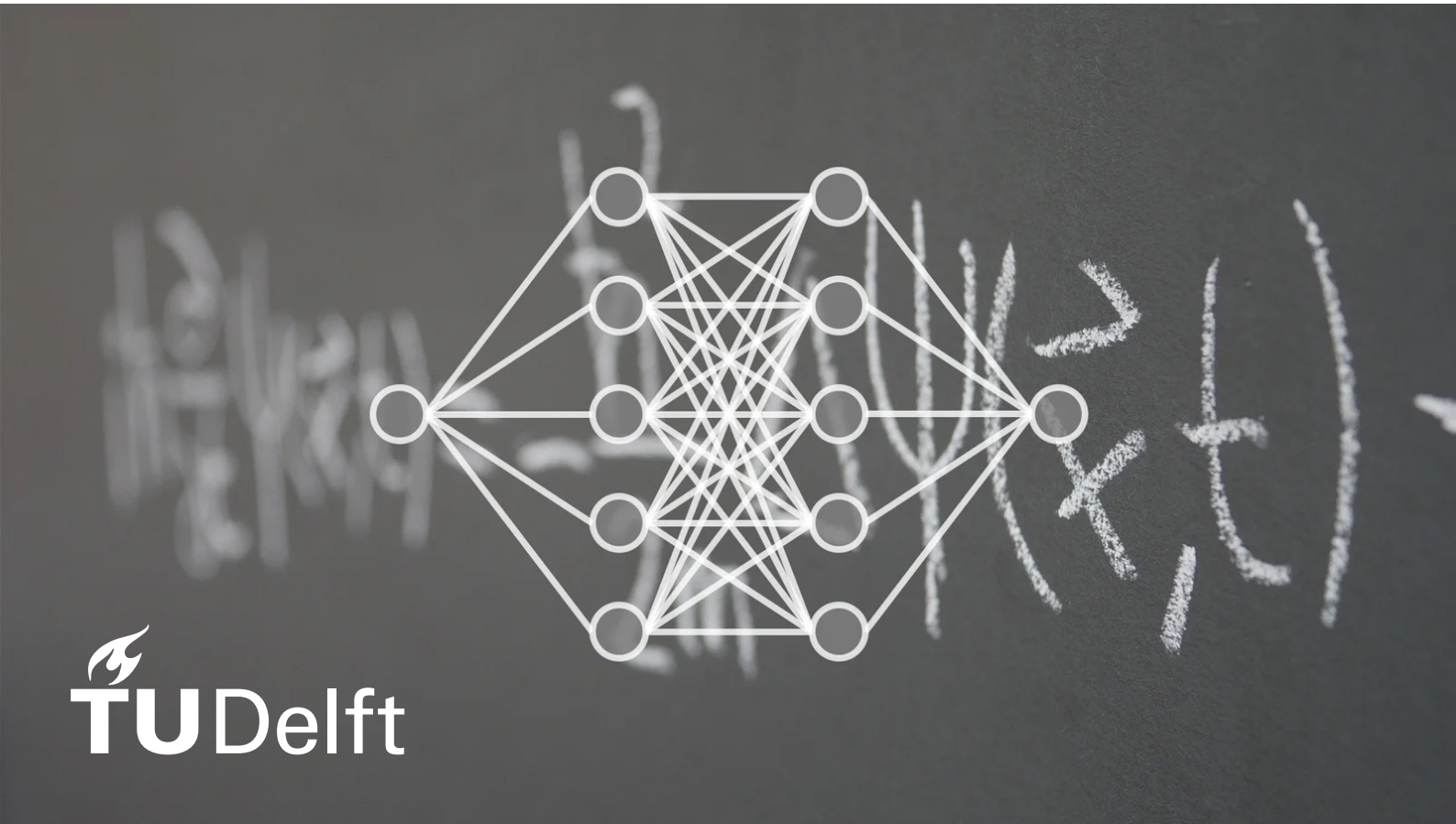
Domain decomposition-based neural networks for complex shaped domains

Wanxin Chen



TU Delft

Domain decomposition-based neural networks for complex shaped domains

by

Wanxin Chen,

to obtain the degree of Master of Science at the Delft University of Technology,
to be defended publicly on Thursday August 28, 2025.

Student numbers:	5931401
Thesis committee:	Dr. A.Heinlein, TU Delft, supervisor
	Dr. A.Howard, Pacific Northwest National Laboratory, supervisor
	Prof. Dr. A.Papapantoleon, TU Delft
Project Duration:	11, 2024 - 8, 2025
Faculty:	Faculty of Electrical Engineering, Mathematics & Computer Science (EEMCS), Delft

TU Delft

Acknowledgements

First of all, I am sincerely grateful to my supervisors, Dr. Heinlein and Dr. Howard, for their guidance, encouragement, and enthusiasm at every stage. Their support made this thesis possible.

I would also like to extend my gratitude to my friends. This journey, my first extended time away from home city, would not have been possible without their support. To my friends in China, thank you for the shared memories and enduring moral support. To my friends in the Netherlands, especially my companions for our Friday evening card games, thank you for all the laughter and moments of respite. To my friends in applied mathematics, I am grateful for your help and support with my studies.

Finally, I wish to express my deepest gratitude to my family. To my parents, Wu Xianhong and Chen Dingquan, and my brother, Chen Zian, your encouragement and support were my driving force.

Wanxin Chen
Delft, August 2025

Abstract

Physics-informed neural networks (PINNs) provide a powerful framework for solving differential equations but often encounter difficulties when addressing high-frequency solutions. Finite basis physics-informed neural networks (FBPINNs) improve PINN performance through uniform overlapping domain decomposition, yet they may still struggle with problems involving non-uniform frequency solutions. In this work, we introduce a novel framework called adaptive domain decomposition-based FBPINNs (Adaptive DD FBPINNs), which incorporates partition of unity networks (POUnets) to learn domain partitions that adaptively decompose the domain in a data-driven manner. This dynamic decomposition significantly enhances the accuracy and efficiency of PDE solvers, particularly for problems with high-frequency components and complex geometries. Furthermore, the framework integrates a residual-based adaptive distribution (RAD) resampling strategy that concentrates training on regions with high residuals, further boosting performance. Experimental results demonstrate that the Adaptive DD FBPINN outperforms standard FBPINN in terms of accuracy, providing a flexible and robust solution for both regular and complex-shaped domains, while efficiently enforcing Dirichlet boundary conditions as hard constraints. Overall, this work provides an exploratory contribution, presenting a promising approach for adaptively learning partitions by combining data-driven POUnets and FBPINNs, which can be further generalized to complex-shaped domains.

The code for this work is available at `https://github.com/orangebowl/thesis_project`

Key words: Physics-Informed Neural Networks, Partition of Unity Networks, Adaptive Domain Decomposition, Finite Basis Physics-Informed Neural Networks

Contents

1

Introduction

Prior to advancements in data-driven machine learning, physics-based modeling dominated fields such as engineering, relying heavily on partial differential equations (PDEs). Classical numerical methods, including finite-difference, finite-element, and spectral methods, have long provided reliable solutions to both forward and inverse PDE problems. However, these methods still face inherent challenges when dealing with complex problems [2].

Recently, with the exponential growth of data and computational resources, machine learning techniques, particularly deep learning, have achieved breakthroughs across different domains such as image recognition [32], natural language processing [37], cognitive science [35], and genomics [1]. Combining Scientific machine learning (SciML) has emerged as a promising field that integrates machine learning with physics-based modeling to address complex scientific problems [3, 65]. Among SciML approaches, Physics-informed neural networks (PINNs) have gained significant attention for their ability to incorporate physical laws directly into the learning process by embedding PDE residuals into the loss function [50, 49, 48]. This formulation enables data-efficient training without requiring large labeled datasets.

Despite their success, PINNs also suffer from several limitations, including spectral bias, slow convergence, and poor scalability to large or complex domains. Recent advances such as domain decomposition methods have sought to alleviate these issues. Notably, finite basis PINNs (FBPINNs) uses local window functions to form a global function expressed as the sum of on the entire domain to enhance training efficiency and accuracy [43]. However, these methods typically rely on manually predefined window functions for domain partitioning.

To overcome this limitation, recent work has explored self-adaptive domain decomposition strategies using data-driven partitioning, such as partition of unity networks (POUnets), which provide valuable insights, suggesting that POUnets can be used to learn features of the target function in a data-driven way [36].

Furthermore, for problems defined on complex-shaped domains, the inexact imposition of boundary conditions can adversely affect both the training of neural networks and the accuracy of PINNs [63]. This difficulty is further amplified on intricate geometries, where Sukumar et al. have proposed methods to impose boundary conditions exactly [59].

Building upon these ideas, this work proposes a novel framework of adaptive domain decomposition-based neural networks that integrates FBPINNs with the domain paritions learning via POUnets. Our method aims to enable data-driven adaptive domain decomposition, thereby improving both solution accuracy and computational efficiency on complex shaped domains.

The remainder of this thesis is structured as follows. Chapter 2 critically reviews prior work on PINNs, domain decomposition techniques, and adaptive partitioning strategies, and distills the key research gaps that motivate this study. Chapter 3 consolidates the requisite background, covering sampling strategies, neural network preliminaries, and the advancement of hard boundary constraints and their

application to complex-shaped domains. Chapter 4 details our proposed *Adaptive Domain Decomposition FBPINN* framework, including its mathematical description, partition of unity construction, and adaptive subdomain learning procedure. Chapter 5 demonstrates the effectiveness of the framework on a hexagon-shaped domain. Finally, Chapter 6 summarizes the main findings, discusses their broader implications, and outlines promising directions for future research.

2

Related Works

This section first reviews the state-of-the-art framework of PINNs and its main extensions, paying special attention to domain-decomposition techniques and recent advances in adaptive partitioning. We then highlight the remaining research gaps and articulate the questions that motivate the present study.

2.1. Physics-Informed Neural Networks

The use of artificial neural networks to solve differential equations dates back over two decades [33, 34, 46], though early efforts were limited by computational resources. With the rise of deep learning, the broader field of SciML has advanced significantly, demonstrating potential to complement traditional solvers by embedding scientific knowledge [65].

Within this context, PINNs were first proposed by Raissi et al [50, 49, 48]. Unlike conventional neural networks that rely solely on data for training, PINNs incorporate physical laws directly into their training process by embedding the squared PDE residuals into the loss function. By minimizing these residuals at collocation points within the domain, PINNs effectively combine data and the governing physics of the system. This inherent alignment with the physical principles not only enhances the explanatory power of the model but also expands its applicability to a wide range of scientific and engineering domains.

In the field of fluid mechanics, PINNs have been widely adopted to reconstruct velocity and pressure fields from sparse or noisy observations [8], to simulate high-speed flows [40], and even to perform flow control [22]. A comprehensive review categorizes these contributions into methodological innovations, scalable software platforms, and theoretical analyses, highlighting the effectiveness of PINNs in solving Navier–Stokes equations and related transport phenomena [8].

In power systems, PINNs have demonstrated strong potential in modeling complex dynamics and solving optimization problems that involve physical constraints. Their applications include state estimation, power flow analysis, transient stability simulation, and data–model synthesis. By encoding grid physics directly into the learning objective, PINNs can overcome challenges related to sparse measurements, model uncertainty, and dynamic complexity [24].

In solid earth geophysics, PINNs are actively explored for wave propagation and seismic inversion. A PINN-based framework has been proposed for solving the acoustic wave equation and performing full waveform inversions, demonstrating that complex subsurface velocity structures can be reconstructed from surface seismograms and a few early-time wavefield snapshots. The mesh-free nature of PINNs enables flexible treatment of free-surface and absorbing boundary conditions and supports the inversions of complex geometries using only partial observations [51].

Despite their growing popularity and effectiveness, PINNs suffer from several limitations that hinder their scalability and applicability in certain scenarios.

2.1.1. Challenges in PINNs

The first challenge lies in the generation of collocation points. As previously discussed, PINNs rely on calculating PDE residuals at collocation points, making their selection crucial. A comprehensive study [66] indicates that uniform sampling works well for simple PDEs but is inefficient for complex problems. Adaptive sampling methods like Residual-based adaptive distribution (RAD) and residual-based adaptive refinement with distribution (RAR-D) significantly improve accuracy with fewer points for both forward and inverse problems [66]. However, these methods produce non-uniform point distributions that introduce pathological loss landscapes. To resolve this, VW-PINNs [57] integrate volume-weighted residuals, explicitly embedding spatial distribution characteristics to balance gradient updates.

Empirical and theoretical studies confirm that fully connected networks exhibit spectral bias: they converge to low-frequency solution components orders of magnitude faster than high-frequency ones [47]. Within PINNs, this bias manifests as overly smooth predictions when solving PDEs with sharp gradients or multiscale features. Neural Tangent Kernel (NTK) analysis reveals that the eigenvalue spectrum of the NTK matrix in PINNs often exhibits rapid decay (e.g., polynomial or faster). This means the eigenvalues associated with high-frequency components are significantly smaller than those associated with low-frequency ones [31]. This disparity leads to an ill-conditioned optimization problem, where different components in the solution converge at vastly different rates. This failure mode, rooted in the spectral bias of neural networks, is particularly pronounced and detrimental in solving multiscale problems where accurately capturing high-frequency phenomena is crucial [64].

This issue is further compounded by gradient pathologies, which manifest as vanishing gradients—specifically, the gradients of the loss function with respect to model parameters during backpropagation—that impede the learning of fine-scale structures [63]. This phenomenon is especially pronounced near discontinuities or within boundary layers, where the physical solution exhibits large spatial gradients (e.g., $\frac{\partial u}{\partial x}$). As a result, the gradients from the PDE loss term become extremely small compared to those from boundary or initial conditions, leading to an imbalance that stalls training and prevents the accurate resolution of high-frequency features.

These accuracy limitations directly impact computational efficiency. Overcoming spectral bias necessitates larger networks with increased parameters to adequately represent high-frequency content, while multi-scale problems require domain upscaling that further escalates model complexity. Consequently, the combined effect substantially elevates training costs, creating significant barriers to real-world deployment as problem domains expand.

Recent adaptive strategies face significant implementation constraints. Fourier feature embeddings [60] explicitly encode high frequencies but require manual tuning of frequency scales, leading to suboptimal performance when scales mismatch the problem's spectral characteristics. Similarly, MscaleDNN [67] employs frequency-domain scaling to transform high-frequency components to lower frequencies, yet its effectiveness depends critically on careful parameter selection and deteriorates when the frequency range of interest remains poorly captured. Both approaches consequently demand prior knowledge of solution characteristics and lack generalizability to unknown frequency regimes.

2.1.2. Other Extension of PINNs

There are also other extensions of PINNs. For instance, Variational PINNs (vPINNs) [28] use the variational form of PDEs to optimize the loss function, improving numerical stability and making them particularly effective for systems with complex geometries or discontinuities. Normalized PINNs (nPINNs) [45] introduce adaptive normalization techniques to mitigate numerical instabilities in multi-scale PDEs, enhancing convergence rates and accuracy in large-scale simulations. Δ-PINNs [11] encode geometric features using Laplace-Beltrami eigenfunctions, enabling better enforcement of boundary conditions and improved performance on irregular domains.

To further enhance the efficiency and stability of PINNs for complex problems, domain decomposition methods will be introduced as an alternative strategy.

2.2. Domain Decomposition in Neural Networks

A significant advancement in PINNs is the integration of domain decomposition methods (DDMs). DDMs are widely used in numerical analysis to address PDEs over large, complex domains or in multi-

physics and multi-material scenarios. DDMs are scalable and robust for both linear and nonlinear systems arising from PDE discretizations. The fundamental principle is to ”divide and conquer”: the computational domain is split into multiple subdomains, each of which is solved separately, with solutions coordinated by exchanging information—typically boundary conditions—at the interfaces between subdomains to achieve a consistent global solution.

DDMs are primarily categorized into two main types. The first type is overlapping domain decomposition, exemplified by the Schwarz method [68] and its variants, such as the Additive Schwarz (AS), and the Restricted Additive Schwarz (RAS) methods [9]. The second type is non-overlapping domain decomposition, including the Schur complement method [10], Finite Element Tearing and Interconnecting (FETI) [16], and its significant improvements FETI-DP (Dual-Primal) [17] and Balancing Domain Decomposition by Constraints (BDDC) [13, 39].

Driven by the rapid advancement of SciML, integrating DDMs with machine learning has emerged as a promising strategy to enhance the efficiency of large-scale scientific computations. Recent reviews [20, 30] indicate that current research primarily follows two related paths: one involves incorporating learning algorithms into traditional DDM frameworks to accelerate convergence and reduce computational costs, such as in [19]. The other replaces conventional finite element or finite difference discretizations with deep neural networks that serve as subdomain solvers when combined with DDMs such as [38], which combines Schwarz iteration with PINNs, where each subdomain is solved by an independent PINN exchanging boundary conditions.

The remainder of this section explores how these ML-assisted DDM concepts can be used within the PINN framework. With the advent of PINNs, researchers have revisited domain decomposition to enhance neural network-based PDE solvers. Several PINN extensions utilize domain decomposition to improve convergence, enable parallelization, and increase accuracy in complex scenarios.

One notable approach as conservative PINNs (cPINNs) [26], which explicitly enforce conservation laws across subdomains. By imposing continuity constraints at subdomain interfaces, cPINNs improve physical consistency and numerical stability while using domain decomposition for efficient training. Nevertheless, cPINNs rely on strong enforcement of flux-matching losses at interfaces, making them less flexible for non-conservative PDEs or domains with complex boundary conditions. The requirement for exact conservation can also slow convergence and increase computational overhead.

The idea of cPINNs is further extended to generic PDEs and arbitrary spacetime domains in Extended Physics-Informed Neural Networks (XPINNs) [25], which partition the computational domain into subdomains and assign independent neural networks to each region. This framework allows for parallel training, increasing scalability for high-dimensional PDEs and complex geometries. By solving subdomain problems independently and enforcing continuity constraints at interfaces, XPINNs achieve faster convergence and lower computational costs compared to standard PINNs. However, XPINNs require carefully balanced interface loss terms and explicit residual or flux matching, which can complicate hyperparameter tuning and introduce instabilities when dealing with highly heterogeneous or multi-scale problems.

FBPINNs [43] localize the approximation of the solution via overlapping subdomains and window functions. Each local network operates on an overlapping subdomain, and their outputs are summed to represent the global solution. This design mitigates spectral bias, guarantees continuity without additional interface losses, and simplifies training by maintaining a unified global loss. FBPINNs are particularly well suited for solving problems on large domains or with multi-scale features, and they avoid the manual tuning of interface penalties required by XPINN and cPINN.

Expanding on the FBPINN framework, [14] introduced a multilevel domain decomposition structure that further improves the resolution of multi-scale and high-frequency PDEs. Numerical experiments have demonstrated that multilevel FBPINNs outperform both standard PINNs and single-level FBPINNs in terms of accuracy, efficiency, and scalability, making them well-suited for large-scale problems. A notable extension is Finite Basis Kolmogorov-Arnold Networks (FBKANs) [21], which replace the multilayer perceptrons in FBPINNs with Kolmogorov-Arnold Networks (KANs) to enhance noise robustness and function expressivity.

These advancements illustrate how domain decomposition—originally developed for finite element and

finite difference methods—has been successfully integrated into PINNs to improve parallelization, conservation enforcement, convergence rates, and scalability. However, challenges remain, particularly regarding adaptive domain partitioning and handling complex shaped domains, which continue to be active areas of research.

2.3. Adapted Domain Decomposition

Domain decomposition techniques have played a crucial role in solving PDEs efficiently, particularly within the PINN framework. The FBPINN method [43], as mentioned earlier, relies on predefined window functions to partition the domain. While this approach is effective for structured geometries, it cannot be easily adapted to complex or irregular problems. The absence of adaptive domain partitioning can result in inefficient training and suboptimal convergence.

Recent gated network methods introduce trainable weighting networks that dynamically adjust domain partitions. APINNs [23] initialize its gate using XPINN decompositions while enabling flexible parameter sharing and continuity through unit decomposition. Similarly, GatedPINNs [58] optimize computational efficiency via conditional expert activation, while MoE-PINN [5] automates expert selection via sparsity regularization. These approaches collectively eliminate manual interface conditions and enhance adaptability for complex domains, yet fundamental challenges persist: the black-box nature of gating networks obscures geometric interpretability.

In the study of POUnets introduced by Lee et al. [36], researchers have explored data-driven and adaptive partitioning techniques that utilize deep neural networks to construct space partitions without relying on explicit mesh generation. This approach enables localized polynomial approximation and exhibits hp-convergence, making it a promising tool for domain decomposition in scientific computing. Building on POUnets, Hierarchical Partition of Unity Networks (HPOUnets) [62] employ a multilevel, multigrid-inspired training strategy to accelerate adaptive partitioning. Similarly, Probabilistic Partition of Unity Networks (PPOUnets) [61] use Gaussian mixture models to estimate partitioning uncertainty, enhancing robustness in spatially varying problems.

Although POUnets are data-driven function approximation methods and were not originally developed as self-adaptive domain decomposition techniques, they provide valuable insights for flexible, data-driven space partitioning. Building on this concept, Partition of Unity PINNs (POU-PINNs) [53] introduce an unsupervised, residual-driven domain decomposition approach within the PINN framework. Instead of relying on manually predefined subdomains, POU-PINNs learn the partitioning through unsupervised training, enabling a more flexible representation of heterogeneous physics while reducing the complexity of traditional domain decomposition. However, the authors still prescribe a piecewise conductivity field before training. This known conductivity is first used as a reference target, allowing the network to align its composite prediction with the ground truth. Consequently, the reported results validate the POU-PINNs under ideal, fully specified conditions rather than demonstrating a truly unsupervised method. Future work is therefore needed to test whether POU-PINNs can identify spatial partitions and local properties when no analytic conductivity is available.

Building on these advancements, we can draw inspiration from POU-based methodologies while addressing their inherent limitations, such as ensuring adequate overlap between subdomains and defining governing partitioning functions. This represents a promising direction for enhancing the adaptability and efficiency of PINNs in solving complex real-world PDE problems.

2.4. Research Gap and Questions

PINNs [48] have gained significant attention as a promising approach for solving PDEs by integrating physical laws into neural network training. Despite their success, PINNs face several challenges, including spectral bias, gradient pathologies, and high computational costs. Various improvements, such as Fourier feature embeddings, variational formulations, and domain decomposition techniques, have been proposed to address these issues. Among these, domain decomposition-based approaches such as FBPINNs [43] enhance parallelization and scalability, especially for high-frequency problems. However, FBPINNs rely on manually predefined window functions, which are not adaptive to different problems or complex-shaped domains.

This limitation highlights a key research gap: existing domain decomposition methods lack adaptive partitioning mechanisms that can dynamically adjust subdomains based on problem complexity. Recent works on POUnets [36] suggest a promising pathway for achieving data-driven adaptive domain partitioning; however, these approaches are primarily based on labeled data and may not fully incorporate underlying physical principles.

To bridge this gap, we would like to answer the following research questions:

How can data-driven adaptive partitions be learned so that they implicitly refine in regions exhibiting rich spectral content, thereby enabling adaptive partitioning of complex shaped domains and improving the convergence and accuracy of domain decomposition-based neural networks?

Specifically, we aim to address the following key subquestions:

1. How does the baseline PINN perform, and what are its limitations?
2. How does FBPINN improve the performance of PINNs?
   (a) To what extent do FBPINNs with uniform domain decomposition improve accuracy?
   (b) Do manually designed domain decompositions outperform uniform domain decompositions?
3. Instead of manually designing the window functions as partitions in FBPINN, can we adaptively design partitions that adjust according to different problems to enhance model flexibility?
   (a) Can we learn partitions in a data-driven way that adjust dynamically for different problems?
   (b) How can such learned partitions be integrated with the FBPINN framework?
4. How well does the integrated method generalize to complex-shaped domains?

By addressing these questions, we aim to integrate the ideas from FBPINNs with adaptive paritions inspired by POUnets, thereby enabling an adaptive domain decomposition framework. This approach seeks to enhance the capability of neural PDE solvers in handling high frequencies problem and complex shaped domain, ultimately improving both training efficiency and solution accuracy.

3

Knowledge Background

This chapter introduced some knowledge used throughout the thesis. We begin by reviewing the neural networks. We then formulate PINN for general boundary-value problems and discuss the challenge of balancing residual and boundary losses. To address the weakness of soft boundary enforcement, we adopt a general scheme of ansatz-based constructions that satisfy different boundary conditions exactly. And for complex geometries, these ansatzes are built using approximate distance functions (ADFs) composed via R-equivalence operations. We further describe the sampling methods, and specify the evaluation metrics and experimental setup used later in the chapter.

3.1. Deep Neural Networks

A fully connected neural network, also known as a Multi-Layer Perceptron (MLP), is a fundamental class of artificial neural network models in machine learning. It is a powerful function approximator used for a wide range of supervised learning tasks, including regression and classification. In this thesis, the MLP forms the basis of the domain decomposition-based neural networks.

3.1.1. Network Architecture

An MLP is organized as a sequence of layers: an input layer, one or more hidden layers, and an output layer. We define a network with L hidden layers.

The network comprises an input layer, L hidden layers with widths $n_1, \dots, n_L$ respectively, and an output layer. We consider an input vector $\mathbf{x} \in \mathbb{R}^d$ and an output $\mathbf{y} \in \mathbb{R}^m$. Each hidden layer performs a weighted sum plus bias, followed by a non-linear activation function. Denoting the pre-activation and activation of the k-th hidden layer by $\mathbf{z}^{(k)}$ and $\mathbf{a}^{(k)}$, the neural network can be formulated as:

$$
\begin{aligned}
\mathbf{z}_j^{(1)} &= \sum_{i=1}^{n_0} W_{ji}^{(1)}\, \mathbf{a}_i^{(0)} + b_j^{(1)}, \qquad \mathbf{a}_j^{(1)} = \sigma\big(\mathbf{z}_j^{(1)}\big), \\
\mathbf{z}_j^{(2)} &= \sum_{i=1}^{n_1} W_{ji}^{(2)}\, \mathbf{a}_i^{(1)} + b_j^{(2)}, \qquad \mathbf{a}_j^{(2)} = \sigma\big(\mathbf{z}_j^{(2)}\big), \\
&\vdots \\
\mathbf{z}_j^{(L)} &= \sum_{i=1}^{n_{L-1}} W_{ji}^{(L)}\, \mathbf{a}_i^{(L-1)} + b_j^{(L)}, \qquad \mathbf{a}_j^{(L)} = \sigma\big(\mathbf{z}_j^{(L)}\big), \\
y_j &= \sum_{i=1}^{n_L} W_{ji}^{(L+1)}\, \mathbf{a}_i^{(L)} + b_j^{(L+1)}, \qquad \mathbf{y} = (y_1, \dots, y_m)^\top.
\end{aligned}
$$

The common choice for activation functions in MLPs is the Rectified Linear Unit (ReLU), defined as $\mathrm{ReLU}(x) = \max(0, x)$, or the hyperbolic tangent (tanh) function, given by $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. In this work, we use the tanh activation function, illustrated in Figure 3.1.

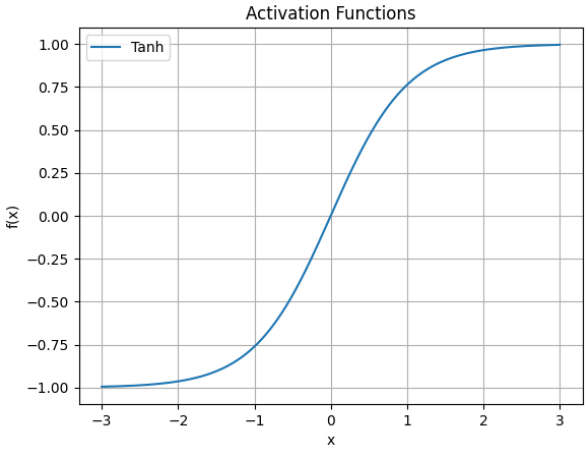


Figure 3.1: Graphs of the tanh activation functions.

In summary, the neural network constitutes a composite function formed by these sequential layers. This function is parameterized by the weights $\mathbf{W}^{(i)}$ and biases $\mathbf{b}^{(i)}$, and can be expressed more compactly as:

$$\begin{aligned}\mathcal{NN}(\mathbf{x};\theta) &= \mathbf{W}^{(L+1)}\sigma\left(\mathbf{b}^{(L)} + \mathbf{W}^{(L)}\sigma\left(\dots\sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right)\dots\right)\right) + \mathbf{b}^{(L+1)} \\ &= \mathbf{W}^{(L+1)} \circ \mathbf{a}^{(L)} \circ \cdots \circ \mathbf{a}^{(1)}(\mathbf{x}) + \mathbf{b}^{(L+1)}.\end{aligned}$$

where $\theta = \{\mathbf{W}^{(i)}, \mathbf{b}^{(i)}\}_{i=1}^{L+1}$ is the set of all trainable parameters.

A suitable weight initialization is also critical to avoid vanishing or exploding signals across layers during both the forward and the backward passes. Since we use tanh, we adopt the Glorot (Xavier) initialization [18] to preserve activation and gradient variance across layers, initial biases are set to zero.

3.1.2. Network Training

To enable the network to make accurate predictions, its parameters θ must be learned from data. This is accomplished through a training process where θ is optimized to minimize a loss function, which measures the discrepancy between the network's predictions and the true target values.

For a standard regression task, the Mean Squared Error (MSE) is a commonly used loss function. Given a training dataset of N samples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, the MSE is defined as:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \mathcal{NN}(\mathbf{x}_i;\theta)\|^2, \tag{3.1}$$

where $\mathbf{y}_i$ is the true target for the i-th sample and $\mathcal{NN}(\mathbf{x}_i;\theta)$ is the prediction generated by the network for the input $\mathbf{x}_i$.

Given a loss function, the training data are propagated forward through the neural network to compute the loss; subsequently, the parameters θ are updated using gradient descent or to obtain optimal values—this process involves backpropagation.

Backpropagation is a critical technique for training neural networks, enabling them to adjust parameters to minimize the loss function. This is done by propagating the error backward from the output to the input layers and computing gradients to indicate how much each parameter contributes to the error. The gradients are then used to update the parameters, typically through gradient descent.

The gradient of the loss function $\mathcal{L}$ with respect to the parameters $\theta^{(\ell)}$ of a given layer ℓ is computed using the chain rule. For instance, the gradient with respect to the weights $W^{(\ell)}$ is:

$$\frac{\partial \mathcal{L}}{\partial W^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(\ell)}} \frac{\partial \mathbf{a}^{(\ell)}}{\partial \mathbf{z}^{(\ell)}} \frac{\partial \mathbf{z}^{(\ell)}}{\partial W^{(\ell)}},$$

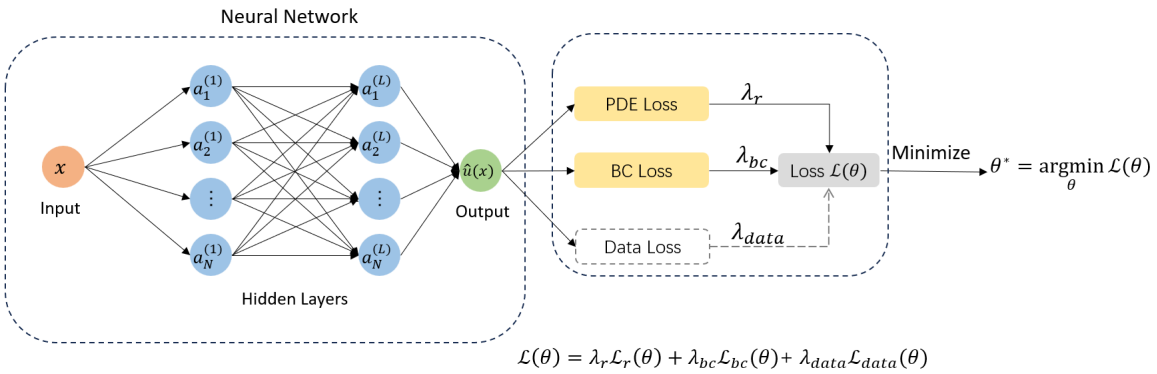


Figure 3.2: Illustration of the PINN architecture for $L = 2$ layers, each with $N = 4$ nodes per layer.

where $\mathbf{a}^{(\ell)}$ is the activation and $\mathbf{z}^{(\ell)}$ is the pre-activation of layer ℓ. This process is applied iteratively backward from the output layer to the input layer.

During the forward pass, the network computes predictions by propagating the input through its layers. The loss function then measures the discrepancy between these predictions and the true values. In the backward pass, backpropagation efficiently computes the gradient of the loss with respect to each network parameter. These gradients are then used by an optimization algorithm to update the parameters. We train in a full-batch regime: at each optimization step, the loss is evaluated over the entire set of training points.

Gradient descent updates parameters by moving opposite to the gradient:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}).$$

A popular and effective variant of gradient descent is the Adam optimizer [29]. Adam computes adaptive learning rates for each parameter from estimates of the first and second moments of the gradients, which typically leads to faster and more reliable convergence.

3.2. Physics-informed Neural Networks

As introduced in [48], PINNs embed the differential and boundary operators of a partial differential equation (PDE) into the neural network's loss function as a form of regularization. When the total loss is minimized during the training, PINN will not only try to match its prediction to the training data, but also enabling the network to learn the solution while adhering to the underlying physical laws.

To formally define the PINN, we consider a general form of a Boundary Value Problem (BVP) as follows:

$$\begin{aligned} \mathcal{D}[\mathbf{u}(\mathbf{x})] &= \mathbf{f}(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \\ \mathcal{B}_k[\mathbf{u}(\mathbf{x})] &= \mathbf{g}_k(\mathbf{x}), \qquad \mathbf{x} \in \Gamma_k \subset \partial\Omega,\ k = 1, 2, \ldots, K. \end{aligned} \tag{3.2}$$

Here, $\mathbf{x}$ is an input d-dimensional vector in the domain $\Omega \subset \mathbb{R}^d$, where $\partial\Omega$ denotes the boundary of Ω, $\mathcal{D}$ represents the differential operator, and $\mathcal{B}_k$ denotes the operators enforcing the boundary or initial conditions. The solution to the PDE is represented by $\mathbf{u} \in \mathbb{R}^m$, with $\mathbf{f}(\mathbf{x})$ as the forcing term, and $\mathbf{g}_k(\mathbf{x})$ defining the boundary conditions, $\mathbf{f}, \mathbf{g}_k : \mathbb{R}^d \to \mathbb{R}^m$. The total number of boundary conditions is denoted by K, and Γ_k refers to the k-th boundary.

Given the problem defined in Equation 3.2, PINNs train the neural network: $\mathcal{NN}(\mathbf{x}; \boldsymbol{\theta}) \approx \mathbf{u}(\mathbf{x})$, where $\mathbf{x}$ denotes the input, and $\boldsymbol{\theta}$ represents all the parameters of the neural network.

A general PINN framework is illustrated in Figure 3.2. The total objective $\mathcal{L}(\boldsymbol{\theta})$ comprises three components: the residual (physics) loss $\mathcal{L}_r(\boldsymbol{\theta})$, which penalizes violations of the governing PDE; the boundary loss $\mathcal{L}_{bc}(\boldsymbol{\theta})$, which enforces the prescribed boundary conditions; and, when labeled observations are

available, the data loss $\mathcal{L}_{\text{data}}(\boldsymbol{\theta})$. These terms are combined with nonnegative weights λ_r, λ_{bc}, and λ_{data} to balance their relative scales:

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_r \, \mathcal{L}_r(\boldsymbol{\theta}) + \lambda_{bc} \, \mathcal{L}_{bc}(\boldsymbol{\theta}) + \lambda_{\text{data}} \, \mathcal{L}_{\text{data}}(\boldsymbol{\theta}).$$

Since our study focuses on PINNs without labeled data rather than data-driven PINNs, we adopt the physics-only formulation and omit $\mathcal{L}_{\text{data}}(\boldsymbol{\theta})$ and its weight λ_{data}.

Therefore, the loss function is now defined as a weighted sum of the residual loss and the boundary loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_r \mathcal{L}_r(\boldsymbol{\theta}) + \lambda_{bc} \mathcal{L}_{bc}(\boldsymbol{\theta}),$$

The parameters $\boldsymbol{\theta}$ are optimized by minimizing a predefined total loss function $\mathcal{L}(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}),$$

The residual loss is defined as:

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_r} \sum_{i=1}^{N_r} \| \mathcal{D}[\mathcal{NN}(\mathbf{x}_i; \boldsymbol{\theta})] - \mathbf{f}(\mathbf{x}_i) \|^2, \tag{3.3}$$

where $\{\mathbf{x}_r^i\}_{i=1}^{N_r}$ are training points sampled from the domain Ω.

The boundary loss is defined as:

$$\mathcal{L}_{bc}(\boldsymbol{\theta}) = \sum_{k=1}^{K} \frac{1}{N_{b_k}} \sum_{j=1}^{N_{b_k}} \| \mathcal{B}_k[\mathcal{NN}(\mathbf{x}_{k_j}; \boldsymbol{\theta})] - \mathbf{g}_k(\mathbf{x}_{k_j}) \|^2, \tag{3.4}$$

where $\{\mathbf{x}_{k_j}\}_{j=1}^{N_{b_k}}$ are points sampled from the k-th boundary Γ_k associated with each condition.

And λ_r and λ_{bc} are the weights that balance the residual and boundary losses.

However, in practice, balancing the residual and boundary condition terms in the loss function can be challenging, often leading to poor convergence and reduced solution accuracy. The values of λ_r and λ_{bc} are not necessarily optimal, and many studies have focused on optimizing these parameters. For example, a study by Wang et al. [63] analyzed a fundamental failure mode of PINNs in solving PDEs, specifically the gradient imbalance during training caused by numerical stiffness. In response, they proposed a learning rate annealing algorithm that dynamically adjusts the learning rate to balance the gradients of different terms in the loss function, though it introduces additional hyperparameters such as the annealing factor. Jin et al. [27] introduced gradient normalization with an auxiliary objective function to adaptively weight the loss terms, though at increased computational cost. More recently, McClemy and Braga-Neto [41] developed a fundamentally different approach using trainable soft-attention weights updated via min-max optimization, which dynamically amplifies penalties for high-error regions without manual tuning or additional hyperparameters.

Given the aforementioned drawback that boundary conditions are enforced only weakly in the loss function, we next introduce a strongly constrained approach that not only applies to standard rectangular domains but also generalizes to complex-shaped domains.

3.3. Strong Boundary Constraints

An alternative method, as introduced by [34], strictly enforces boundary conditions by building them into the solution ansatz. Let $\Omega \subset \mathbb{R}^d$ be the computational domain with boundary $\partial\Omega$, and let $\mathbf{u} : \Omega \to \mathbb{R}^m$ denote the target solution. For $\mathbf{x} \in \partial\Omega$, let $\mathbf{n}(\mathbf{x})$ be the outward unit normal and define the normal derivative of a smooth scalar field v by

$$\partial_n v(\mathbf{x}) := \nabla v(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}).$$

Instead of approximating the solution $\mathbf{u}(\mathbf{x})$ directly with a neural network, we construct an approximation $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta})$ that satisfies the boundary conditions for any network parameters $\boldsymbol{\theta}$. We now outline the general approach for building such an ansatz for different types of boundary conditions.

Dirichlet BCs:

Given:

$$\mathbf{u}(\mathbf{x}) = \mathbf{g}_D(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega, \tag{3.5}$$

We choose a scalar function $\phi_D : \overline{\Omega} \to \mathbb{R}$ with $\phi_D > 0$ in Ω and $\phi_D|_{\partial\Omega} = 0$, and set

$$\hat{\mathbf{u}}_D(\mathbf{x};\boldsymbol{\theta}) = \mathbf{g}_D(\mathbf{x}) + \phi_D(\mathbf{x})\,\mathcal{NN}(\mathbf{x};\boldsymbol{\theta}).$$

Then $\hat{\mathbf{u}}_D|_{\partial\Omega} = \mathbf{g}_D$ holds by construction, independently of $\boldsymbol{\theta}$.

Neumann BCs:

Given:

$$\partial_n \mathbf{u}(\mathbf{x}) = \mathbf{h}_N(\mathbf{x}), \qquad \mathbf{x} \in \partial\Omega, \tag{3.6}$$

We then build a function $\mathbf{g}_N$ such that $\partial_n \mathbf{g}_N|_{\partial\Omega} = \mathbf{h}_N$.

Next we need to select ϕ_N that vanishes to first order in the normal direction,

$$\phi_N|_{\partial\Omega} = 0, \qquad \partial_n \phi_N|_{\partial\Omega} = 0, \tag{3.7}$$

and define

$$\hat{\mathbf{u}}_N(\mathbf{x};\boldsymbol{\theta}) = \mathbf{g}_N(\mathbf{x}) + \phi_N(\mathbf{x})\,\mathcal{NN}(\mathbf{x};\boldsymbol{\theta}). \tag{3.8}$$

Differentiating (3.8) in the normal direction gives

$$\partial_n \hat{\mathbf{u}}_N = \partial_n \mathbf{g}_N + \phi_N\,\partial_n \mathcal{NN}(\mathbf{x};\boldsymbol{\theta}) + \mathcal{NN}(\mathbf{x};\boldsymbol{\theta})\,\partial_n \phi_N,$$

and the last two terms vanish on $\partial\Omega$ by (3.7). Hence $\partial_n \hat{\mathbf{u}}_N|_{\partial\Omega} = \mathbf{h}_N$ which satisfies the Neumann boundary condition (3.6).

Mixed BCs:

Let the boundary be partitioned into disjoint parts Γ_D and Γ_N with

$$\begin{cases} \mathbf{u}(\mathbf{x}) = \mathbf{g}_D(\mathbf{x}), & \mathbf{x} \in \Gamma_D, \\ \partial_n \mathbf{u}(\mathbf{x}) = \mathbf{h}_N(\mathbf{x}), & \mathbf{x} \in \Gamma_N, \end{cases} \tag{3.9}$$

where $\Gamma_D \cup \Gamma_N = \partial\Omega$ and $\Gamma_D \cap \Gamma_N = \varnothing$.

To build the solution ansatz for mixed BCs, we need to build a function $\mathbf{B}(\mathbf{x})$ such that $\mathbf{B}(\mathbf{x})|_{\Gamma_D} = \mathbf{g}_D(\mathbf{x})$ and $\partial_n \mathbf{B}(\mathbf{x})|_{\Gamma_N} = \mathbf{h}_N(\mathbf{x})$. Let $\phi_D(\mathbf{x})$ satisfy $\phi_D(\mathbf{x})|_{\Gamma_D} = 0$ and let $\phi_N(\mathbf{x})$ satisfy (3.7) on Γ_N.

Then the mixed ansatz is defined as

$$\hat{\mathbf{u}}_M(\mathbf{x};\boldsymbol{\theta}) = \mathbf{B}(\mathbf{x}) + \phi_D(\mathbf{x})\,\phi_N(\mathbf{x})\,\mathcal{NN}(\mathbf{x};\boldsymbol{\theta}).$$

On Γ_D, we have $\phi_D(\mathbf{x}) = 0$, hence

$$\hat{\mathbf{u}}_M(\mathbf{x};\boldsymbol{\theta}) = \mathbf{B}(\mathbf{x}) = \mathbf{g}_D(\mathbf{x}).$$

On Γ_N, the normal derivative of the second term is

$$\begin{aligned} \partial_n\big(\phi_D(\mathbf{x})\phi_N(\mathbf{x})\,\mathcal{NN}(\mathbf{x};\boldsymbol{\theta})\big) = {} & (\partial_n \phi_D(\mathbf{x}))\,\phi_N(\mathbf{x})\,\mathcal{NN}(\mathbf{x};\boldsymbol{\theta}) + \phi_D(\mathbf{x})\,(\partial_n \phi_N(\mathbf{x}))\,\mathcal{NN}(\mathbf{x};\boldsymbol{\theta}) \\ & + \phi_D(\mathbf{x})\,\phi_N(\mathbf{x})\,\partial_n \mathcal{NN}(\mathbf{x};\boldsymbol{\theta}). \end{aligned}$$

Since $\phi_N|_{\Gamma_N} = 0$ and $\partial_n \phi_N|_{\Gamma_N} = 0$, the expression above vanishes on Γ_N. Therefore,

$$\partial_n \hat{\mathbf{u}}_M|_{\Gamma_N} = \partial_n \mathbf{B}|_{\Gamma_N} = \mathbf{h}_N.$$

With these ansatzes, the boundary conditions are satisfied exactly for any $\boldsymbol{\theta}$, allowing training to focus on minimizing the interior residual of the governing PDE.

However, while ansatz-based constructions are straightforward on simple domains (e.g., rectangles or disks), designing the functions ϕ_D and ϕ_N that satisfy $\phi_D|_{\partial\Omega} = 0$, $\phi_N|_{\partial\Omega} = 0$, and $\partial_n \phi_N|_{\partial\Omega} = 0$ becomes challenging for complex shaped domains. To address this, we introduced an approach to exactly impose boundary conditions in PINNs that is based on approximate distance functions (ADFs).

3.3.1. Approximate Distance Function

The signed distance function provides an implicit representation for curves and surfaces, enabling efficient evaluation of geometric predicates. And the exact distance function $d(\mathbf{x})$ is defined as the minimum distance from any point $\mathbf{x}$ in the domain Ω to the boundary $\partial\Omega$. By construction, $d(\mathbf{x})$ is zero on the boundary and positive within the domain. However, computing the exact distance function can be challenging, particularly for domains with complex geometries.

In the context of the ansatz construction in Section 3.3, it is desirable to construct an approximate distance function (ADF) $\phi(\mathbf{x})$ that approximates the distance to the boundary ∂S of a domain $S \subset \mathbb{R}^d$. It is essential that $\phi(\mathbf{x})$ vanishes on the boundary ∂S and is positive inside the domain S.

Based on the work by Shapiro and Tsukanov [56], we describes the construction of the ADF $\phi(\mathbf{x})$ of a given point to a line segment and the ADF $\phi(\mathbf{x})$ to a quarter-circular arc.

Approximate distance to a Line Segment:

Let a line segment be defined by its endpoints $\mathbf{x}_1 := (x_1, y_1)$ and $\mathbf{x}_2 := (x_2, y_2)$. The center of this segment is denoted by $\mathbf{x}_c$:

$$\mathbf{x}_c := \frac{\mathbf{x}_1 + \mathbf{x}_2}{2}.$$

The length of the segment is L:

$$L := \|\mathbf{x}_2 - \mathbf{x}_1\|.$$

The signed distance function $f(\mathbf{x})$ from a point $\mathbf{x} := (x, y)$ to the infinite line passing through $\mathbf{x}_1$ and $\mathbf{x}_2$ is defined as [54]:

$$f(\mathbf{x}) = \frac{(x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)}{L} \tag{3.10}$$

Since the representation of the segment can be viewed as the intersection of an infinite line with a disk of radius L/2, we consider the following trimming function. To account for the finite length of the segment, a trimming function $t(\mathbf{x})$ is introduced. The segment can be viewed as the intersection of the infinite line with a disk of radius $L/2$ centered at $\mathbf{x}_c$ [54].

$$t(\mathbf{x}) = \frac{1}{L}\left(\left(\frac{L}{2}\right)^2 - \|\mathbf{x} - \mathbf{x}_c\|^2\right) \tag{3.11}$$

Note that $t(\mathbf{x}) \geq 0$ defines a disk with center at $\mathbf{x}_c$ and radius $L/2$.

First, an intermediate function $\varphi(\mathbf{x})$ is defined to help smooth the transition near $t(\mathbf{x}) = 0$:

$$\varphi(\mathbf{x}) = \sqrt{t(\mathbf{x})^2 + f(\mathbf{x})^4} \tag{3.12}$$

Then, the ADF $\phi(\mathbf{x})$ to the segment with endpoints $\mathbf{x_1}$ and $\mathbf{x_2}$ is given by [54]:

$$\phi(\mathbf{x}) = \sqrt{f(\mathbf{x})^2 + \left(\frac{\varphi(\mathbf{x}) - t(\mathbf{x})}{2}\right)^2} \tag{3.13}$$

This function $\phi(\mathbf{x})$ in (3.13) is an approximation of the distance function to the segment with endpoints $\mathbf{x}_1$ and $\mathbf{x}_2$.

Based on the given function, we can draw the ADF to a segment as shown in Figure 3.3.

Approximate distance to a quarter-circular arc: And for a quarter-circular arc with endpoints $\mathbf{x}_1$ and $\mathbf{x}_2$, the ADF is constructed similarly. The representation of the quarter-circular arc can be viewed as the intersection of a circle of radius R and the segments defined by the two endpoints of the arc. The distance function $f(\mathbf{x})$ is defined as [54]:

$$f(\mathbf{x}) = \frac{R^2 - \|\mathbf{x} - \mathbf{x}_c\|^2}{2R}$$

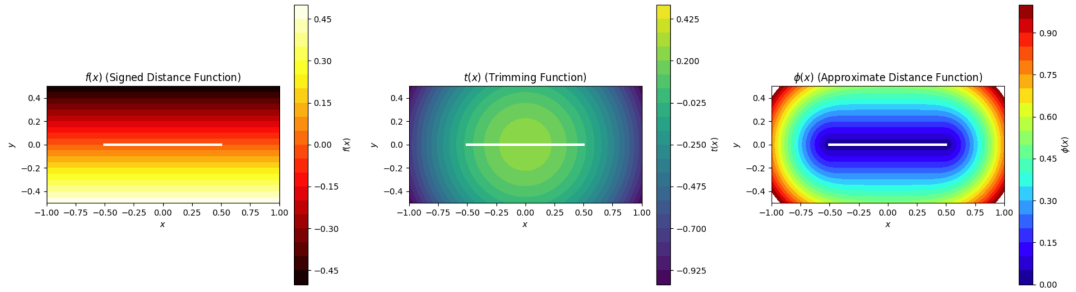


Figure 3.3: Construction of the ADF to a line segment:(left) the function f;(middle) $t(x)$; (right) the ADF to a line segment.

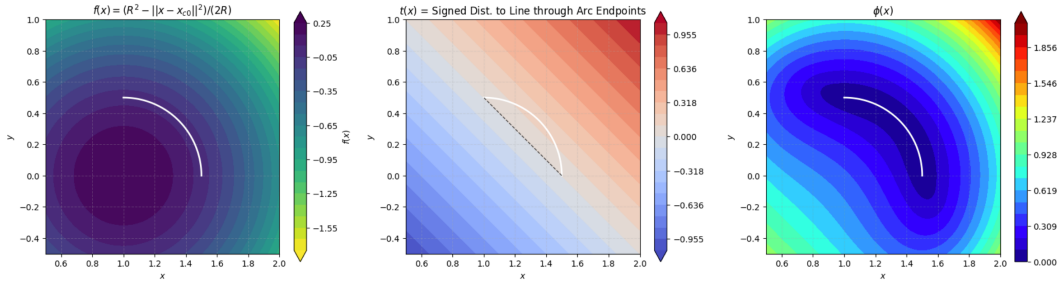


Figure 3.4: Construction of the ADF to a curve: f;(middle) $t(x)$; (right) the ADF to the quarter-circular arc.

where R is the radius of the arc and $\mathbf{x}_c$ is the center of the circular. The trimming function $t(\mathbf{x})$ is defined as [54]:

$$t(\mathbf{x}) = \frac{(x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1)}{L} \tag{3.14}$$

The intermediate function $\varphi(\mathbf{x})$ and the final ADF $\phi(\mathbf{x})$ are constructed using Equations 3.12 and 3.13. Figure 3.4 illustrates the functions f, t, and ϕ for a quarter-circular arc.

R-equivalence operation

For a boundary $\partial\Omega \subset \mathbb{R}^2$ composed of piecewise curves and line segments, we denote the ADF for each piece by $\phi_i(\mathbf{x})$. A key property of any ADF ϕ is that it must vanish on the boundary, i.e., $\phi(\mathbf{x}) = 0$ for all $\mathbf{x} \in \partial\Omega$.

An ADF is defined as being of m-th order if its normal derivatives from the second to the m-th order are zero [55]:

$$\frac{\partial \phi}{\partial \nu} = 1, \quad \frac{\partial^k \phi}{\partial \nu^k} = 0 \quad (k = 2, 3, \ldots, m),$$

Here, ν is the unit inward normal vector on the boundary. Such a function is called m-th order normalized. For the exact distance function, the normal derivative on the boundary should be 1, i.e., $\frac{\partial d}{\partial \nu} = \nabla d \cdot \nu = 1$, and all higher-order normal derivatives vanish. For finite m, the normalized function only matches the exact distance function near the boundary [55]. To more closely emulate the exact distance function, an ADF should also have a unit normal derivative ($\partial\phi/\partial\nu = 1$) and vanishing higher-order normal derivatives at regular points on the boundary.

A R-function is a real-valued function constructed using Boolean operations on other functions, where the sign of the resulting function depends solely on the signs of its arguments [55].

Mathematically, an R-function is a real-valued function $F(\omega_1, \omega_2, \ldots, \omega_q)$, where each $\omega_i(x)$ is a real-valued function. The key property of an R-function is that its sign is determined entirely by the signs of the functions $\omega_1(x), \omega_2(x), \ldots, \omega_q(x)$. The R-equivalence operation is a specific type of R-function:

$$R_{\sim}(\omega_1, \omega_2) = \frac{\omega_1 \omega_2}{\sqrt{\omega_1^2 + \omega_2^2}}$$

This operation combines two functions so that the resulting function smoothly blends the characteristics of both inputs.

To combine the distance functions ϕ_1 and ϕ_2 of two curves, c_1 and c_2, into a single distance function ϕ for their union $c_1 \cup c_2$, the resulting function must be zero on the boundary and positive elsewhere. A simple product $\phi_1 \phi_2$ satisfies the zero condition but fails to preserve normalization at regular points of the segments [59]. To address this, an R-equivalence operation can be used, which maintains normalization up to a specified order m at all regular points (i.e., non-vertex points on polygonal curves) [6]:

$$\phi(\phi_1, \phi_2) = \frac{\phi_1 \phi_2}{\sqrt[m]{\phi_1^m + \phi_2^m}} = \frac{1}{\sqrt[m]{\frac{1}{\phi_1^m} + \frac{1}{\phi_2^m}}}, \tag{3.15}$$

When $\partial\Omega$ is composed of n pieces, then a ϕ that is normalized up to order m is given by:

$$\phi(\phi_1, \ldots, \phi_n) = \frac{1}{\sqrt[m]{\frac{1}{\phi_1^m} + \frac{1}{\phi_2^m} + \cdots + \frac{1}{\phi_n^m}}}. \tag{3.16}$$

We are not going to prove any property of the normalization here, the rigorous proofs are in [42] which is out of the scope of our work. And in the following, we will use the R-equivalence operation to construct the ADF for two L-shaped line segments and a hexagon.

ADF for 2 line segments

Consider the two axis-aligned unit segments emanating from the origin,

$$c_1 = \{(0, y) \mid 0 \le y \le 1\}, \qquad c_2 = \{(x, 0) \mid 0 \le x \le 1\}.$$

$$\phi(x, y) = \frac{1}{\left(x^{-m} + y^{-m}\right)^{1/m}} = \frac{x\, y}{\left(x^m + y^m\right)^{1/m}}, \qquad m \ge 1. \tag{3.17}$$

The ADFs with different m are shown in Figure 3.5. With an increasing m, the ADF approaches the exact distance function to the segments.

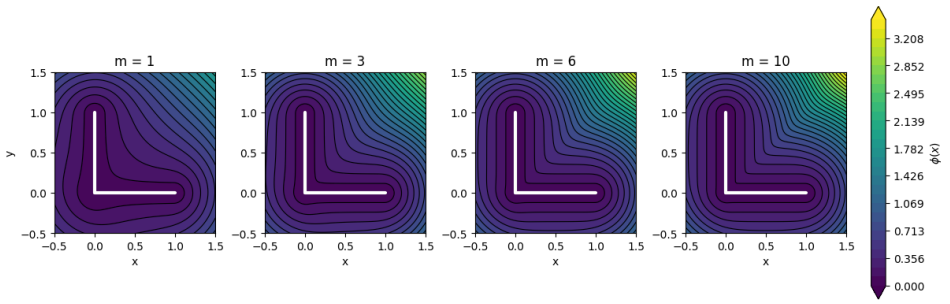


Figure 3.5: Plots of ADF to two line segments with different orders of the normalizing parameter $m = 1, 3, 6, 10$.

ADF for a Hexagon

We next construct a ADF for $\phi_{\text{hex}}(\mathbf{x})$. Let the hexagon be specified by its vertices $\mathbf{v}_i := (x_i, y_i)$, $i = 1, \ldots, 6$, ordered counterclockwise, with $\mathbf{v}_7 \equiv \mathbf{v}_1$. Each edge is the line segment $e_i = (\mathbf{v}_i, \mathbf{v}_{i+1})$. For a query point $\mathbf{x} := (x, y) \in \mathbb{R}^2$, we associate to each edge e_i the segment center and length

$$\mathbf{x}_{c,i} := \frac{\mathbf{v}_i + \mathbf{v}_{i+1}}{2}, \qquad L_i := \|\mathbf{v}_{i+1} - \mathbf{v}_i\|.$$

Following the equation (3.10), the signed distance from $\mathbf{x}$ to the infinite line through e_i is (3.10)

$$f_i(\mathbf{x}) = \frac{(x - x_i)(y_{i+1} - y_i) - (y - y_i)(x_{i+1} - x_i)}{L_i},$$

and following the equation (3.11), the first-order normalized trimming function that encodes the finite segment extent is:

$$t_i(\mathbf{x}) = \frac{1}{L_i}\left(\left(\frac{L_i}{2}\right)^2 - \|\mathbf{x} - \mathbf{x}_{c,i}\|^2\right).$$

Following (3.12)–(3.13), define

$$\varphi_i(\mathbf{x}) = \sqrt{t_i(\mathbf{x})^2 + f_i(\mathbf{x})^4}, \qquad \phi_i(\mathbf{x}) = \sqrt{f_i(\mathbf{x})^2 + \left(\frac{\varphi_i(\mathbf{x}) - t_i(\mathbf{x})}{2}\right)^2},$$

so that each ϕ_i is an unsigned, first-order normalized approximation to the distance from $\mathbf{x}$ to the edge segment e_i. The hexagon-level function is then obtained by an R-equivalence operation (3.16):

$$\phi_{\text{hex}}(\mathbf{x}; m) = \left(\sum_{i=1}^{6} \phi_i(\mathbf{x})^{-m}\right)^{-1/m}. \tag{3.18}$$

By inspecting the contours, we can see that as m increases, the ADF approaches the exact distance function to the segments.

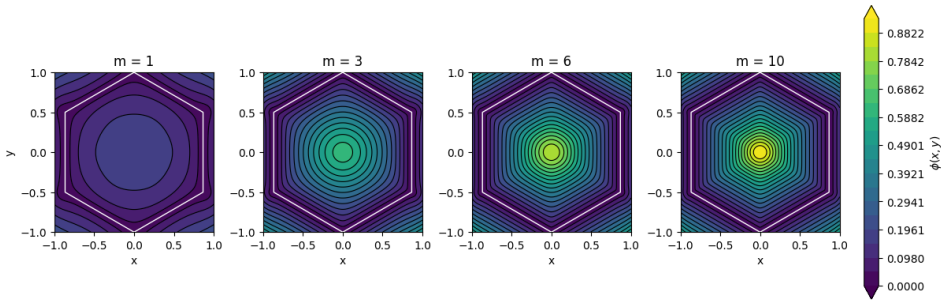


Figure 3.6: Plots of ADF to Hexagon with different orders of the normalizing parameter $m = 1, 3, 6, 10$.

As m increases, the ADF becomes a closer approximation to the exact distance function. However, higher-order normalization can lead to pronounced oscillations in the Laplacian [59]. Figure 3.7 shows that the Laplacian is largest near the corners and along the boundary. Therefore, in our experiments, we use $m = 1$ for constructing the ADF. In line with the guidance from [59], we also avoid placing collocation points directly on the boundary and near corners.

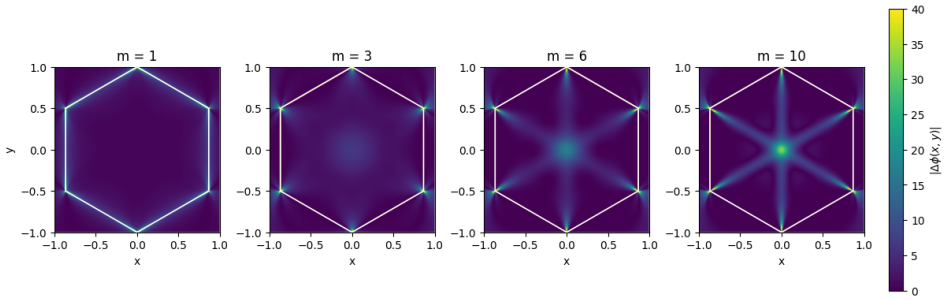


Figure 3.7: Plots of the absolute of Laplacian of Hexagon with different orders of the normalizing parameter $m = 1, 3, 6, 10$.

3.4. Sampling Methods

The placement and spatial coverage of collocation points in the computational domain Ω are critical for training PINNs, as they strongly affect convergence and accuracy [66]. Since we enforce boundary conditions strongly via the solution ansatz, we only need to sample collocation points in the interior of the domain. The most straightforward approach, which we adopt as our baseline, is to sample these points from a uniform distribution over Ω.

To improve robustness, Wu et al. proposed the Residual-based Adaptive Distribution (RAD) method [66], outlined in Algorithm 1.

At each resampling step, a new set of points is drawn from a probability density function (PDF) $p(\mathbf{x})$ that is proportional to the current residual error. For any point $\mathbf{x} \in \Omega$, let the residual be $\varepsilon(\mathbf{x}) = \|\mathcal{D}[\hat{u}(\mathbf{x};\theta)] - \mathbf{f}(\mathbf{x})\|$. The sampling PDF is defined as

$$p(\mathbf{x}) \ \propto \ \frac{\varepsilon(\mathbf{x})^k}{\mathbb{E}[\varepsilon(\mathbf{x})^k]} + c, \qquad k \geq 0, \ c \geq 0, \tag{3.19}$$

where the expectation is taken over the uniform measure on Ω and is estimated numerically. The hyperparameters k and c are tunable. When $k = 0$, the expression simplifies, and sampling becomes uniform. As k increases, the sampling distribution becomes more concentrated in regions with larger residuals, thereby focusing on the areas that is hard to learn. The parameter c acts as a regularizer; as c increases, the sampling distribution tends toward uniformity. As suggested by [66], the combination of $k = 1$ and $c = 1$ is a good default choice for many problems.

Algorithm 1 Residual-based Adaptive Distribution (RAD) [66]

1: Initialize a set $\mathcal{T}$ of collocation points using uniform sampling over Ω;
2: Train the PINN for a prescribed number of iterations;
3: **repeat**
4: $\quad \mathcal{T} \leftarrow$ points resampled according to the PDF in Equation (3.19);
5: $\quad$ Train the PINN for a prescribed number of iterations;
6: **until** reach the total training steps

In our work, We caculate the PDF in Equation 3.19 as follows:

1. Draw a dense points set $\mathcal{S}_0$ (poolsize M) uniformly over Ω;
2. Compute the $p(\mathbf{x})$ for all the points in $\mathcal{S}_0$;
3. Form the probability mass function $\tilde{p}(\mathbf{x}) = p(\mathbf{x}) / \sum_{\mathbf{x}\in\mathcal{S}_0} p(\mathbf{x})$;
4. Sample N_r points from $\mathcal{S}_0$ according to $\tilde{p}$.

We resample every N_{resamp} iterations during training; $(N_r, N_{\text{resamp}}, k, c)$ are the sampling hyperparameters reported in our experiments.

3.5. Numerical Experiments

Software and hardware environment. All PINN and FBPINN experiments were implemented in a unified training framework built with JAX [52]. All the training was conducted with a single NVIDIA GeForce RTX 4060 GPU.

For evaluation in the following examples, since we are using strong boundary constraints, we only need to sample test points that are equally spaced in the domain for evaluation. Let $\{\mathbf{x}_j\}_{j=1}^N \subset \Omega$ be testing points with exact or reference values $u(\mathbf{x}_j)$ and predictions $\hat{u}(\mathbf{x}_j)$. We choose the metrics relative L_2 error and root-mean-squared error (RMSE).

Then

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{j=1}^{N} \|\hat{u}(\mathbf{x}_j) - u(\mathbf{x}_j)\|^2}$$

and

$$\text{Relative } L_2 \text{ error} = \frac{\left(\sum_{j=1}^{N} \|\hat{u}(\mathbf{x}_j) - u(\mathbf{x}_j)\|^2\right)^{1/2}}{\left(\sum_{j=1}^{N} \|u(\mathbf{x}_j)\|^2\right)^{1/2}}$$

For visualization in 2D examples, the testing points $\{(\mathbf{x}_i, \mathbf{y}_j)\}_{i,j}$ are equally spaced in the domain, and we plot the pointwise difference:

$$\text{Difference}[\mathbf{x}_i, \mathbf{y}_j] = \hat{u}(\mathbf{x}_i, \mathbf{y}_j) - u(\mathbf{x}_i, \mathbf{y}_j). \tag{3.20}$$

The first example is taken from the study in [43], which considers the following ODE:

$$\frac{du}{dx} = \cos(\omega x), \quad u(0) = 0 \tag{3.21}$$

whose exact solution is

$$u(x) = \frac{1}{\omega} \sin(\omega x). \tag{3.22}$$

the researchers incorporate a solution ansatz in the PINN as:

$$\hat{u}(x; \theta) = \tanh(\omega x)\, \mathcal{NN}(x; \theta), \tag{3.23}$$

which automatically satisfies the initial condition.

When the frequency is low as $\omega = 1$, the domain is $x \in [-2\pi, 2\pi]$, and a fully connected neural network with 2 hidden layers of width 16 is used. A total of 3000 training points are uniformly sampled across the domain. The input variable x is normalized to the range $[-1, 1]$. The model is trained 50000 steps with the Adam optimizer at a learning rate of 0.001. We observe as shown in Figure 3.8a that the PINN quickly and accurately converges to the exact solution.

However, as the frequency increases when $\omega = 15$,the PINN with the same parameter settings fails to make accurate predictions, as shown in Figure 3.8b. Even when the network is enlarged to 5 hidden layers with 128 neurons each (Figure 3.8c), the PINN still struggles to converge. As discussed in Section 2.1.1, PINNs become increasingly difficult to train at higher frequencies.

To further evaluate the capability of PINNs in handling high-frequency problems, we consider the following first-order PDE on 2D domain defined as follows:

$$\begin{aligned} u_x + u_y &= f(x, y), \quad \Omega = [0, 1]^2, \\ u(x, 0) &= 0, \quad u(0, y) = 0 \end{aligned} \tag{3.24}$$

with exact solution:

$$u(x, y) = \sin(10\pi x^2) \sin(10\pi y^2).$$

Therefore the source term is given by

$$f(x, y) = -\left(20\pi^2 x \cos(10\pi x^2) + 20\pi^2 y \cos(10\pi y^2)\right) \sin(10\pi x^2) \sin(10\pi y^2).$$

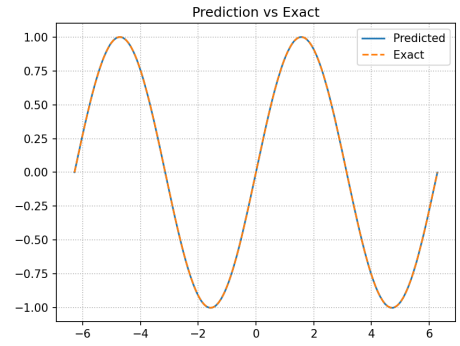
As shown in Figure 3.9, the solution exhibits higher-frequency oscillations toward the corner $(1, 1)$.

To enforce homogeneous Dirichlet boundary conditions on the coordinate axes, we represent the solution as

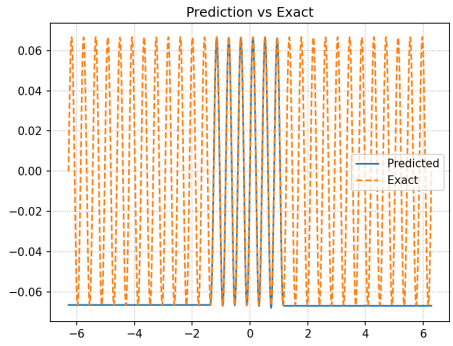
$$\hat{u}(\mathbf{x}; \theta) = \phi(\mathbf{x})\, \text{NN}(\mathbf{x}; \theta), \tag{3.25}$$

where $\phi(\mathbf{x})$ is a function that vanishes on the boundary. Following the ADF construction in Equation 3.3.1, and choosing $m = 1$, we set

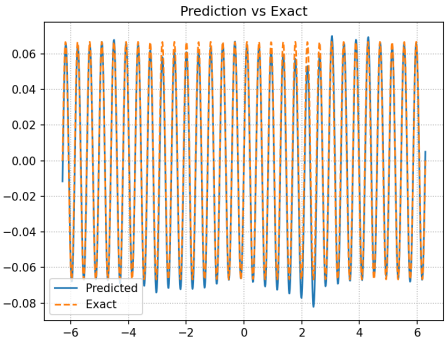
$$\phi(x, y) = \left(x^{-1} + y^{-1}\right)^{-1} = \frac{xy}{x + y}. \tag{3.26}$$



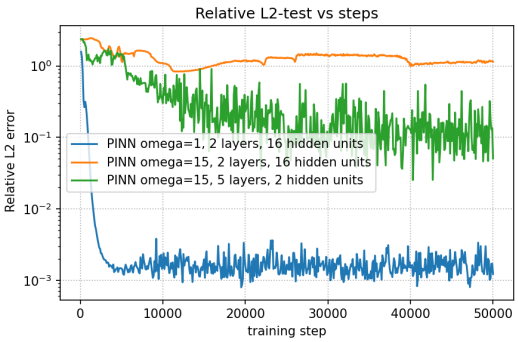
(a) PINN, $\omega = 1$, 2 hidden layers with 16 neurons each



(b) PINN, $\omega = 15$, 2 hidden layers with 16 neurons each



(c) PINN, $\omega = 15$, 5 hidden layers with 128 neurons each



(d) Relative L_2 error comparison

Figure 3.8: Using PINNs to solve the cosine ODE example. (a) and (b) show the PINN predictions for frequencies $\omega = 1$ and $\omega = 15$ respectively with a 2 layers, each layers contain 16 neurons. (c) shows the PINN prediction for $\omega = 15$ with a larger 5 layers, 128 neurons for each layer. (d) compares the relative L_2 error convergence over training steps.

In practice, we add a small constant $\varepsilon = 10^{-8}$ to the denominator to ensure numerical stability.

For the PINN training, we use a network with 5 hidden layers of 128 neurons each. The model is trained for 30,000 steps using the Adam optimizer with a learning rate of 10^{-3}, on collocation points sampled uniformly from the domain $\Omega = [0, 1]^2$. The components of the input vector $\mathbf{x} = (x, y)$ are normalized to the range $[-1, 1]$. The training points are sampled uniformly from the domain $\Omega = [0, 1]^2$.

Table 3.1: Relative L_2 and RMSE under different numbers of collocation points.

Number of collocation points	Relative L_2 error	RMSE
80×80	$2.18 \times 10^{-1} \pm 1.02 \times 10^{-1}$	$9.92 \times 10^{-2} \pm 4.65 \times 10^{-2}$
100×100	$\mathbf{1.41 \times 10^{-1} \pm 1.06 \times 10^{-1}}$	$\mathbf{6.44 \times 10^{-2} \pm 4.81 \times 10^{-2}}$
150×150	$1.46 \times 10^{-1} \pm 6.92 \times 10^{-2}$	$6.66 \times 10^{-2} \pm 3.16 \times 10^{-2}$
200×200	$1.67 \times 10^{-1} \pm 5.98 \times 10^{-2}$	$7.60 \times 10^{-2} \pm 2.73 \times 10^{-2}$

As shown in Table 3.1, when the number of collocation points is increased from 6400 to 10000, the relative L_2 error and RMSE decrease, indicating better performance. However, when the number of collocation points is increased from 10000 to 22500 and 40000, the relative L_2 error and RMSE increase slightly, indicating that increasing the number of collocation points does not guarantee better performance. Therefore, we choose the best average performance with 100×100 collocation points.

We also investigate the effect of the RAD resampling strategies based on the 100x100 number of collocation points with a poolsize = 20000. And with the same random key, we compare the performance of the RAD with different combinations of parameters.

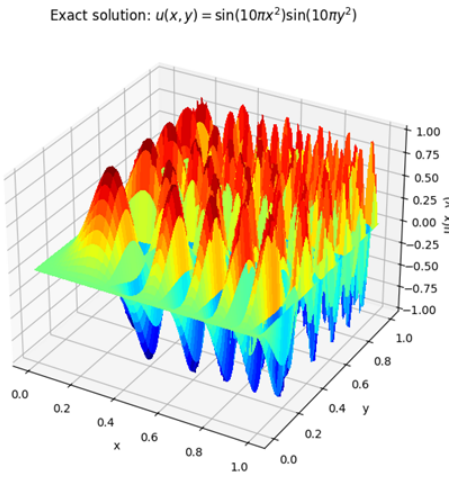


Figure 3.9: 3D plot of exact solution.

As suggested in [66] and follow the RAD algorithms 1, we first set the parameters $k = 1$ and $c = 1$ for the RAD resampling strategy as a starting point. We resample every 2000, 5000, and 10000 steps, respectively.

Table 3.2: Performance metrics under RAD resampling schedules with 100×100 collocation points, $k = 1$, $c = 1$.

Resampling Schedule	Relative L_2 error	RMSE
No RAD	2.46×10^{-1}	1.12×10^{-1}
Resample every 2000 steps	2.23×10^{-1}	1.02×10^{-1}
Resample every 5000 steps	1.97×10^{-1}	8.98×10^{-2}
Resample every 10000 steps	$\mathbf{1.94 \times 10^{-1}}$	$\mathbf{8.82 \times 10^{-2}}$
Resample every 15000 steps	2.05×10^{-1}	9.36×10^{-2}

As shown in Table 3.2, a resampling interval of 10000 iterations slightly outperforms the others. We therefore fix this setting and next examine different combinations of the hyperparameters k and c.

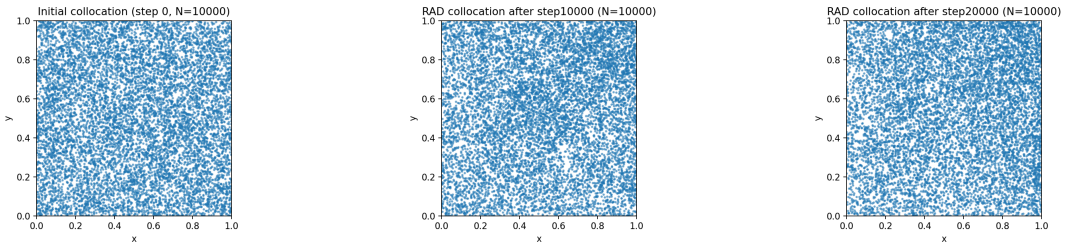
Table 3.3: Performance metrics for different k and c combinations.

(k, c)	Relative L_2 error	RMSE
$k = 0.5, c = 1$	2.15×10^{-1}	9.82×10^{-2}
$k = 1, c = 1$	$\mathbf{1.94 \times 10^{-1}}$	$\mathbf{8.82 \times 10^{-2}}$
$k = 2, c = 1$	2.02×10^{-1}	9.20×10^{-2}
$k = 3, c = 1$	2.15×10^{-1}	9.80×10^{-2}
$k = 1, c = 0.5$	2.04×10^{-1}	9.29×10^{-2}
$k = 1, c = 0.1$	2.16×10^{-1}	9.86×10^{-2}

As shown in Table 3.3, the setting $k = 1, c = 1$ achieves the lowest relative L_2 error, and RMSE. Overall, $k = 1, c = 1$ offers the best balance and has the best overall performance.

As shown in Figure 3.10, the collocation points at steps 0, 10000, and 20000 indicate that resampling at 10000 and 20000 steps concentrates samples along the main diagonal and in the upper-right region, aligning with the model's high-frequency areas. Consistently, the difference maps in Figure 3.12 also exhibit their largest magnitudes along the diagonal and in the upper-right corner.

As shown in Figure 3.11, the PDE loss (blue) drops sharply at the start and then continues to decrease with pronounced oscillations, while the relative L_2 error (orange) decays smoothly but plateaus at a



(a) step 0 **(b)** step 10000 **(c)** step 20000

Figure 3.10: The resampling collocation points using RAD at training steps 0, 10000, and 20000.

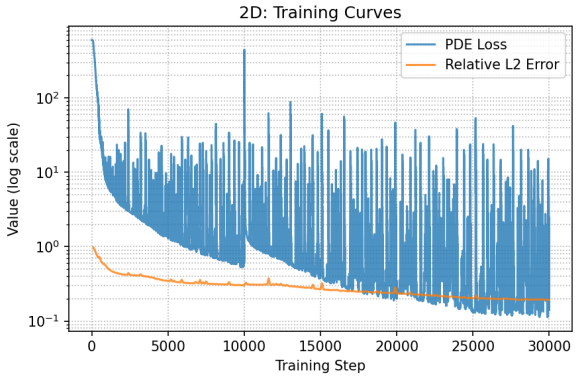


Figure 3.11: The loss of using PINNs to solve $u_x + u_y = f(x, y)$.

value that is not sufficiently small. Together with the prediction and difference maps in Figure 3.12, this indicates that the PINN fails to recover the exact solution of $u_x + u_y = f(x, y)$, particularly along the diagonal and near the upper-right corner.

This pattern matches the problem's structure. The PDE is a first-order linear advection equation whose characteristics satisfy $x - y = C$ and transport information from the inflow boundaries toward $(1, 1)$, where errors tend to accumulate. Moreover, the exact solution $u = \sin(10\pi x^2)\sin(10\pi y^2)$ exhibits high-frequency oscillations, concentrating complexity along the diagonal and near $(1, 1)$, which leads to steep gradients and makes the PINN notably harder to train in those regions.

The RAD method does not seem to be very effective in this test. This might be due to the difficulty of training the PINN; as shown in Figure 3.11, the training loss oscillates heavily during training. The effectiveness of RAD will be further examined in later sections.

In the following sections, we introduce domain decomposition-based PINNs as a potential solution to address these high frequency challenges and how to adaptively chose the domain decomposition according to the frequency of the problem.

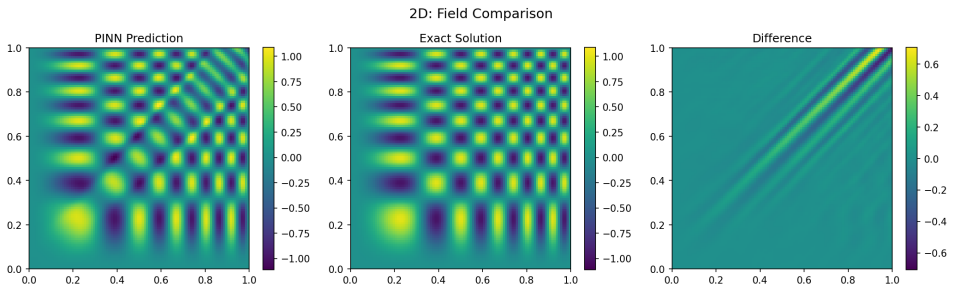


Figure 3.12: Using PINNs to solve $u_x + u_y = f(x, y)$: The PINN is trained with 100x100 uniformly sampled collocation points with RAD ($k = 1$,$c = 1$) using a fully connected neural network with 5 hidden layers and 128 neurons each.

4

Adaptive Domain Decomposition-based FBPINNs

In this chapter, we begin by introducing the domain decomposition based FBPINNs framework and the method of POUnets. Building upon these foundations, we then proposed a method that take advantages of both components. The method starts with the sampling of collocation points and is subsequently refined through the integration of an adaptive domain decomposition strategy.

4.1. Finite Basis Physics-informed Neural Networks

The mathematical formulation of FBPINNs [43] divides the domain $\Omega \subset \mathbb{R}^d$ into n overlapping subdomains Ω_i as shown in Figure

Each subdomain is assigned a separate neural network $\mathcal{NN}_i(\mathbf{x}; \boldsymbol{\theta}_i)$, and smooth, differentiable window functions $w_i(\mathbf{x})$ are used to combine their outputs into a global solution. The approximate solution $\hat{u}(\mathbf{x}; \boldsymbol{\theta})$ is given by:

$$\hat{u}(\mathbf{x}; \boldsymbol{\theta}) = \mathcal{C}\left[\overline{\mathcal{NN}}(\mathbf{x}; \boldsymbol{\theta})\right], \tag{4.1}$$

where

$$\overline{\mathcal{NN}}(\mathbf{x}; \boldsymbol{\theta}) = \sum_{i=1}^{n} w_i(\mathbf{x}) \cdot \mathcal{NN}_i\left(\text{norm}_i(\mathbf{x}); \boldsymbol{\theta}_i\right). \tag{4.2}$$

Here:

- $w_i(\mathbf{x})$ is a smooth window function that confines each network to its subdomain. As described in the following section 4.1.1, ensures each neural network contributes only within its respective subdomain, with its influence rapidly decaying to zero outside that region. Its partition of unity (POU) is constructed using the expression:

$$\omega_i = \frac{\hat{\omega}_i}{\sum_{i=1}^{L} \hat{\omega}_i}, \tag{4.3}$$

  where $\hat{\omega}_i$ represents the unnormalized weight function.L is the total number of partitions.
- $\mathcal{NN}_i(\mathbf{x}; \boldsymbol{\theta}_i)$ is the neural network specific to subdomain Ω_i.
- $\text{norm}_i(\mathbf{x})$ denotes normalization applied to inputs of the subdomain.
- $\mathcal{C}$ is a constraining operator that enforces boundary conditions on the solution ansatz, as described in Section 3.3.

- $\boldsymbol{\theta} = \{\boldsymbol{\theta}_i\}$ represents all trainable parameters.

4.1.1. Window Functions

We use a set of compactly supported window functions $\{w_i\}$ to construct a partition of unity (POU). This means that at every point $\mathbf{x}$ in the domain, the sum of all window function weights equals one. Specifically, a collection $\{w_i(\mathbf{x})\}_{i=1}^n$ defined on Ω forms a partition of unity if it meets the following criteria:

1. Non-negativity: $w_i(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \Omega$ and for each i.
2. Unity sum: $\sum_{i=1}^n w_i(\mathbf{x}) = 1$ for all $\mathbf{x} \in \Omega$.

We first define the window function $\tilde{w}_i(\mathbf{x})$ by multiplying the function ψ across each dimension:

$$\tilde{w}_i(\mathbf{x}) \;=\; \prod_{j=1}^{d} \psi\left(\frac{x^j - c_i^j}{h_i^j}\right), \tag{4.4}$$

where the function $\psi(t)$ is defined as:

$$\psi(t) = \begin{cases} \big(1+\cos(\pi t)\big)^2, & |t| \leq 1, \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

In Equation (4.4), $\mathbf{x} = (x^1, \ldots, x^d)$ is a point in the d-dimensional space, $\mathbf{c}_i = (c_i^1, \ldots, c_i^d)$ is the center of the i-th window, and $\mathbf{h}_i = (h_i^1, \ldots, h_i^d)$ defines its radius, or half-width of the support, in each dimension.

This particular form of $\psi(t)$, which can be expressed as $4\cos^4(\pi t/2)$ for $|t| \leq 1$, is locally supported.

The final window functions $w_i(\mathbf{x})$ with an overlap of 0.1 that form the POU are obtained by normalization in Equation 4.3 as illustrated in Figure 4.1.

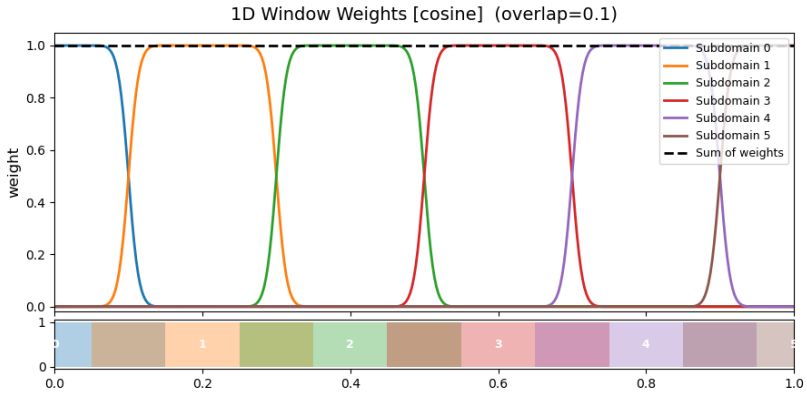


Figure 4.1: 1D window functions with 0.1 overlap

Other window functions can be used as well, such as the sigmoid window function, Gaussian window function and bump window function, but these are outside the scope of this work.

Resulting loss for FBPINNs:

By enforcing the boundary conditions through the operator $\mathcal{C}$, the FBPINN is trained by minimizing a hard-constrained loss function. This loss is defined as the mean squared residual of the governing PDE over N collocation points $\{\mathbf{x}_i\}_{i=1}^N$:

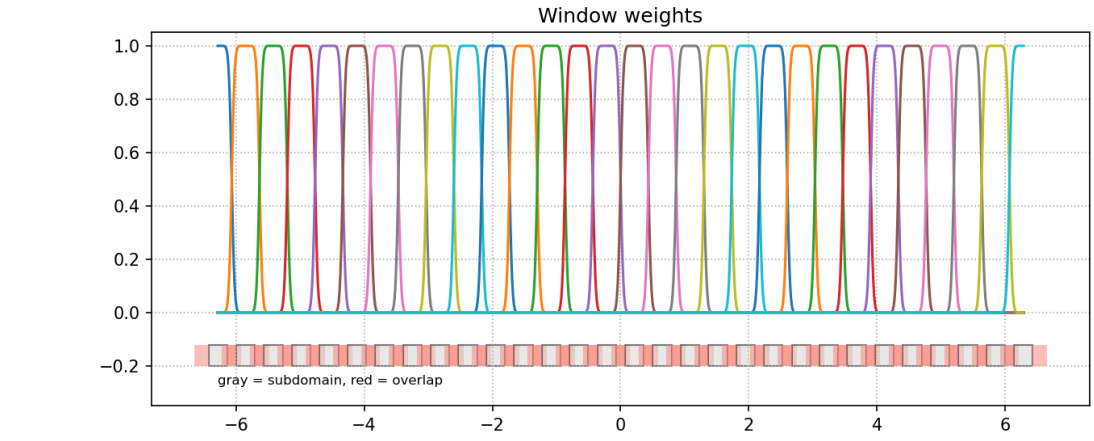
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \left(\mathcal{D}\hat{u}(\mathbf{x}_i;\boldsymbol{\theta}) - f(\mathbf{x}_i)\right)^2$$

4.1.2. Numerical Experiments

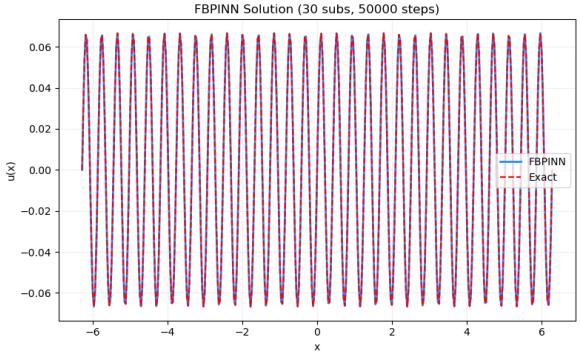
To demonstrate the effectiveness of FBPINNs, we revisit the 1D Cosine ODE problem with $\omega = 15$ (defined in a previous chapter by Equation 3.21), a case where standard PINNs struggle, as discussed in Section 3.5. We apply the FBPINN method by dividing the domain into $n = 30$ subdomains with an overlap of $\delta = 0.3$. The window functions, described by Equation 4.4, are illustrated in Figure 4.2a.

With 3000 collocation points uniformly sampled over the domain $[-2\pi, 2\pi]$, we use, in each subdomain, a network with two hidden layers of 16 neurons each. For testing, we uniformly sample 1000 test points on $[-2\pi, 2\pi]$ and compare the FBPINN against a baseline PINN with five hidden layers and 128 neurons per layer. Figure 4.2c presents the results.

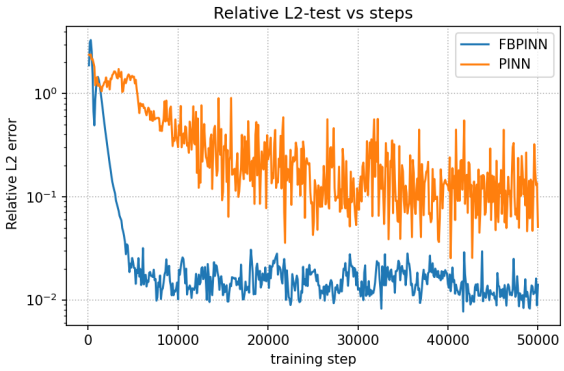
As shown in Figure 4.1.1, the FBPINN attains substantially lower relative L_2 error than the PINN and converges to the solution in fewer training steps.



(a) Cosine window function with an overlap of 0.3



(b) FBPINN solution



(c) Relative L_2 error of FBPINN vs PINN

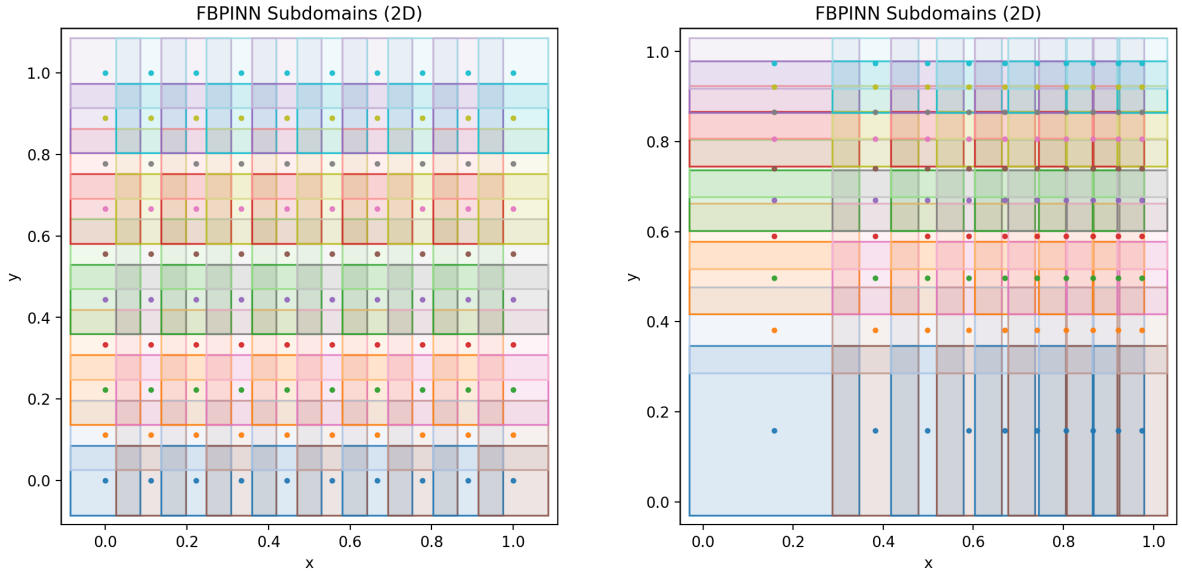
Figure 4.2: FBPINN vs PINN on the 1D Cosine ODE ($\omega = 15$): (a) cosine window function with overlap 0.3; (b) FBPINN solution; (c) relative L_2 error of FBPINN vs PINN.

Although FBPINNs are designed to address problems with strong high-frequency components, real-world PDEs are typically more complex and vary across the domain. This raises a natural question: how does the method perform on domains where the frequency is high yet non-uniform?

We use FBPINN to solve the problem $u_x + u_y = f(x, y)$ defined in Equation 3.24, sampling 100×100 collocation points uniformly on $[0, 1]^2$. Each subdomain network has 2 layers with 16 neurons, and an overlap size of 0.06.

We compare two decomposition strategies as shown in Figure 4.3:

1. **Uniform Domain Decomposition.** As shown in Figure 4.3a, we generate equally spaced centers



(a) Uniform domain decomposition

(b) Feature-aligned domain decomposition

Figure 4.3: Comparison of uniform and feature-aligned domain decomposition strategies. Dots indicate subdomain centers; colors distinguish neighboring subdomains.

on $[0,1]$,

$$x_i = \frac{i}{n_x - 1}, \quad i = 0, 1, \ldots, n_x - 1, \qquad y_j = \frac{j}{n_y - 1}, \quad j = 0, 1, \ldots, n_y - 1.$$

$\Delta x = 1/(n_x - 1)$ and $\Delta y = 1/(n_y - 1)$. The subdomain without overlap centered at (x_i, y_j) is

$$\Omega_{ij}^0 = \left[x_i - \tfrac{\Delta x}{2}, x_i + \tfrac{\Delta x}{2}\right] \times \left[y_j - \tfrac{\Delta y}{2}, y_j + \tfrac{\Delta y}{2}\right].$$

With an overlap $\delta \geq 0$, we expand each side by $\delta/2$:

$$I_i^x = \left[x_i - \tfrac{\Delta x + \delta}{2}, x_i + \tfrac{\Delta x + \delta}{2}\right], \qquad I_j^y = \left[y_j - \tfrac{\Delta y + \delta}{2}, y_j + \tfrac{\Delta y + \delta}{2}\right],$$

and set $\Omega_{ij} = I_i^x \times I_j^y$.

2. **Feature-aligned Domain Decomposition.** As shown in Figure 4.3b, we generate nodes on $[0,1]$,

$$z_k^x = \sqrt{\frac{k}{n_x - 1}}, \quad k = 0, \ldots, n_x - 1, \qquad z_\ell^y = \sqrt{\frac{\ell}{n_y - 1}}, \quad \ell = 0, \ldots, n_y - 1,$$

and define non-uniform centers as midpoints

$$x_i = \tfrac{z_i^x + z_{i+1}^x}{2}, \quad i = 0, \ldots, n_x - 2, \qquad y_j = \tfrac{z_j^y + z_{j+1}^y}{2}, \quad j = 0, \ldots, n_y - 2.$$

$$\Delta x_i = z_{i+1}^x - z_i^x, \qquad \Delta y_j = z_{j+1}^y - z_j^y.$$

The subdomain without overlap centered at (x_i, y_j) is

$$\Omega_{ij}^0 = \left[x_i - \tfrac{\Delta x_i}{2}, x_i + \tfrac{\Delta x_i}{2}\right] \times \left[y_j - \tfrac{\Delta y_j}{2}, y_j + \tfrac{\Delta y_j}{2}\right].$$

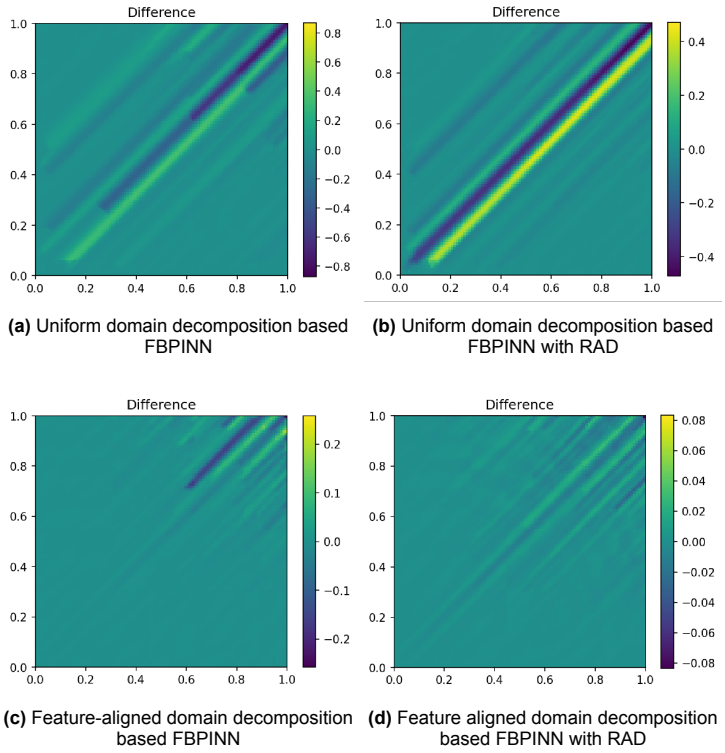
With an overlap $\delta \geq 0$, we expand each side by $\delta/2$:

$$I_i^x = \left[x_i - \tfrac{\Delta x_i + \delta}{2}, x_i + \tfrac{\Delta x_i + \delta}{2}\right], \qquad I_j^y = \left[y_j - \tfrac{\Delta y_j + \delta}{2}, y_j + \tfrac{\Delta y_j + \delta}{2}\right],$$

and set $\Omega_{ij} = I_i^x \times I_j^y$.

Table 4.1: FBPINN performance metrics under uniform vs. feature-aligned domain decomposition, with and without RAD.

Method	**Relative L_2 error**	**RMSE**
Uniform domain decomposition-based FBPINN	2.64×10^{-1}	1.20×10^{-1}
Uniform domain decomposition-based FBPINN(RAD)	2.32×10^{-1}	1.06×10^{-1}
Feature-aligned domain decomposition-based FBPINN	4.37×10^{-2}	1.99×10^{-2}
Feature-aligned domain decomposition-based FBPINN(RAD)	$\mathbf{1.15 \times 10^{-2}}$	$\mathbf{5.26 \times 10^{-3}}$



(a) Uniform domain decomposition based FBPINN

(b) Uniform domain decomposition based FBPINN with RAD

(c) Feature-aligned domain decomposition based FBPINN

(d) Feature aligned domain decomposition based FBPINN with RAD

Figure 4.4: Difference plots comparing domain decomposition strategies (uniform, feature-aligned) and the effect of RAD resampling. Note that the color scales differ between subplots.

We then compare the performance of the FBPINN based on the uniform domain decomposition and feature-aligned domain decomposition and further evaluate how the RAD resampling method mentioned in Section 3.4.

As shown in Figure 4.4 which shows the map of difference. Under the uniform domain decomposition as shown in Figure 4.4a, equally spaced subdomains do not align with solution features, producing large error larger than 0.8 on the diagonal. Applying RAD resampling reduces but does not solve this pattern—the diagonal still shows an error larger than 0.4. In contrast, the feature-aligned decomposition as shown in Figure 4.4c places smaller subdomains along regions of high variation, markedly lowering the difference relative to the uniform domain decomposition, although the upper-right corner still exceeds an error larger than 0.2. With RAD (Figure 4.4d), both the relative L_2 error and RMSE decrease further, and the remaining hotspots along the diagonal and near $(1, 1)$ shrink to approximately 0.08.

Table 4.1 shows that feature-aligned domain decomposition based FBPINN with RAD performs best with the lowest relative L_2 error of 1.15×10^{-2} and the lowest RMSE $= 5.26 \times 10^{-3}$. Adding RAD provides a modest improvement under uniform domain decomposition, but a much larger gain under feature-aligned domain decomposition. Overall, the feature-aligned design of domain decomposition increase the accuracy, and RAD amplifies its benefits.

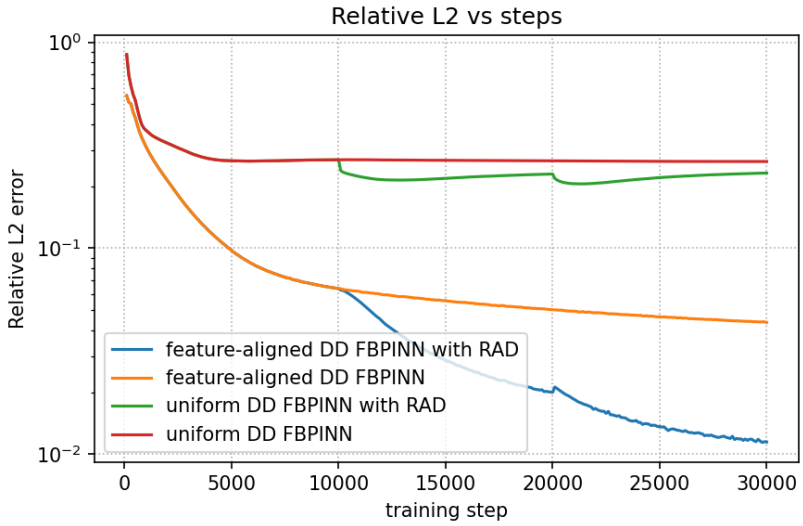


Figure 4.5: Relative L_2 error vs. training steps for four FBPINN settings; feature-aligned domain decomposition with RAD converges fastest, while uniform domain decomposition-based FBPINN plateaus at higher error.

Figure 4.5 plots the relative L_2 error against training steps for the four settings. The feature-aligned FBPINN with RAD resampling shows the fastest convergence; its error decay accelerates at steps 10,000 and 20,000, corresponding to the RAD resampling method. The feature-aligned FBPINN without RAD also improves steadily, albeit more slowly. In contrast, the uniform FBPINN plateaus early at a high error level, and adding RAD provides only a modest improvement. These results confirm that feature-aligned decomposition is the primary driver of accuracy, with RAD further amplifying its benefits.

The experimental results show that different domain decomposition strategies yield varying performance. The feature-aligned domain decomposition, which adapts to the solution's characteristics, significantly outperforms the uniform approach. This highlights the importance of aligning subdomains with solution features for effective training and generalization.

This prompts a key question: how can we determine a suitable domain decomposition, especially when the exact solution is unknown? The following section introduces POUnets, which provide a way to learn the domain decomposition adaptively in a data-driven way.

4.2. Partition of Unity Networks

The goal of POUnets is to approximate functions based on a dataset: Given a dataset

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{data}}},$$

where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i = y(\mathbf{x}_i)$ are the function values at the data points.

POUnets directly incorporate Partition-of-Unity (POU) functions and polynomial approximations into the network architecture. This enables the neural network to focus exclusively on partitioning the space, while localized polynomial spaces handle high-order approximations.

The approximation form used by POUnets is given by:

$$y_{\text{pou}}(\mathbf{x}) = \sum_{\alpha=1}^{N_{\text{part}}} \phi_\alpha(\mathbf{x}) \sum_{\beta=1}^{\dim(V)} c_{\alpha,\beta}\, P_\beta(\mathbf{x}), \tag{4.6}$$

where:

- $V = \pi_m(\mathbb{R}^d)$ denotes the space of polynomials of total degree at most m, with $\dim(V) = \binom{m+d}{d}$ and basis $\{P_\beta(\mathbf{x})\}_{\beta=1}^{\dim(V)}$.

- $\Phi = \{\phi_\alpha(\mathbf{x})\}_{\alpha=1}^{N_{\text{part}}}$ are the partitions functions satisfying POU:

$$\sum_{\alpha=1}^{N_{\text{part}}} \phi_\alpha(\mathbf{x}) = 1, \quad \phi_\alpha(\mathbf{x}) \geq 0.$$

- $c_{\alpha,\beta} \in \mathbb{R}$ are coefficients in the polynomial approximations.

For instance, in a one-dimensional setting ($d = 1$), using linear polynomials ($m = 1$) results in a two-dimensional approximation space, $\dim \pi_1(\mathbb{R}) = \binom{2}{1} = 2$. The standard basis for this space is $\{1, x\}$, and each local approximant $p_\alpha(x)$ within a partition α takes the form of a simple linear function: $p_\alpha(x) = c_{\alpha,0} + c_{\alpha,1}x$. For more intricate functions, a higher polynomial degree can be employed. In the one-dimensional case with quadratic polynomials ($m = 2$), the approximation space $\pi_2(\mathbb{R})$ is of dimension $\binom{3}{1} = 3$. This space is spanned by the basis $\{1, x, x^2\}$, enabling each local approximant $p_\alpha(x) = c_{\alpha,0} + c_{\alpha,1}x + c_{\alpha,2}x^2$ to capture local curvature.

This framework extends naturally to higher dimensions. In a two-dimensional domain ($d = 2$) with quadratic polynomials ($m = 2$), the approximation space $\pi_2(\mathbb{R}^2)$ has a dimension of $\binom{4}{2} = 6$. The basis includes all monomials up to degree two, namely $\{1, x, y, x^2, xy, y^2\}$. This allows the local approximant $p_\alpha(x, y)$ to represent more complex surfaces, as described by $p_\alpha(x, y) = c_{\alpha,0} + c_{\alpha,1}x + c_{\alpha,2}y + c_{\alpha,3}x^2 + c_{\alpha,4}xy + c_{\alpha,5}y^2$.

The POU functions $\phi_\alpha(\mathbf{x}; \boldsymbol{\theta}_{\text{pou}})$ are parameterized by chosen networks in one of the POU architectures described in Section 4.2.1 where $\boldsymbol{\theta}_{\text{pou}}$ represents the parameters of the network.

4.2.1. POU Architectures

Here we introduced two architectures for the POU:

RBF-Net [7, 4]:

$$\phi_\alpha(\mathbf{x}; \boldsymbol{\theta}_{pou}) = \frac{\exp(-|\mathbf{x} - \boldsymbol{\xi}_{1,\alpha}|^2/\xi_{2,\alpha}^2)}{\sum_\beta \exp(-|\mathbf{x} - \boldsymbol{\xi}_{1,\beta}|^2/\xi_{2,\beta}^2)}, \quad 1 \leq \alpha \leq N_{\text{part}}. \tag{4.7}$$

This is a shallow RBF-network, using a single hidden layer without additional nonlinear transformations. And here the output is determined by the centers and width parameters of the RBF functions, without any additional non-linear layers. Here, $\boldsymbol{\theta}_{pou} = (\boldsymbol{\xi}_1, \xi_2)$, where $\boldsymbol{\xi}_1$ denotes the RBF centers and ξ_2 denotes RBF shape parameters, both of which evolve during training.

MLP:

The POU functions can also be generated by the neural network $\mathcal{NN}(\mathbf{x}; \boldsymbol{\theta}_{\text{pou}})$ that we introduced in Section 3.1.

Training Optimization Strategy

The optimization problem for POUnets is formulated as:

$$\min_{\boldsymbol{\theta}_{\text{pou}},\, \mathbf{c}} \sum_{i=1}^{N_{\text{data}}} \left| \sum_{\alpha=1}^{N_{\text{part}}} \phi_\alpha(\mathbf{x}_i; \boldsymbol{\theta}_{\text{pou}}) \sum_{\beta=1}^{\dim(V)} c_{\alpha,\beta} P_\beta(\mathbf{x}_i) - y_i \right|^2. \tag{4.8}$$

However, the parameter optimization is slightly different from traditional methods, since y_{pou} in Equation 4.6 includes two sets of parameters: one is the network parameters $\boldsymbol{\theta}_{\text{pou}}$, and the other is the polynomial approximation parameters $\mathbf{c}$, which are optimized jointly. Therefore, we will introduce the algorithm of Least-squares gradient descent(LSGD).

In Algorithm 2, the inputs are the initial POUnets parameters $\boldsymbol{\theta}_{\text{pou,old}}$, the initial polynomial coefficients $\mathbf{c}_{\text{old}}$, and the number of outer iterations n_{epochs}, which specifies how many LS + GD cycles are executed. And the η is the learning rate used for the GD update of the POUnets parameters $\boldsymbol{\theta}_{\text{pou}}$. The algorithm outputs the updated network parameters $\boldsymbol{\theta}_{\text{pou}}$ together with the updated polynomial coefficients $\mathbf{c}$.

In each LS + GD epoch, we perform the following steps:

Algorithm 2 Least-squares gradient descent (LSGD) [12]

Input: $\boldsymbol{\theta}_{\text{pou, old}},\ \mathbf{c}_{\text{old}},\ n_{\text{epochs}},\ \rho,\ \eta$

Output: $\boldsymbol{\theta}_{\text{pou, new}},\ \mathbf{c}_{\text{new}}$

function LSGD($\boldsymbol{\theta}_{\text{pou, old}}, \mathbf{c}_{\text{old}}, n_{\text{epochs}}, \rho, \eta$)
  $\boldsymbol{\theta}_{\text{pou}} \leftarrow \boldsymbol{\theta}_{\text{pou, old}},\ \ \mathbf{c} \leftarrow \mathbf{c}_{\text{old}}$
  $\mathbf{c} \leftarrow \text{LS}(\boldsymbol{\theta}_{\text{pou}})$ ▷ solve for $\mathbf{c}$ with given $\boldsymbol{\theta}_{\text{pou}}$ using least-squares
  $\boldsymbol{\theta}_{\text{pou}} \leftarrow \text{GD}(\boldsymbol{\theta}_{\text{pou}}, \mathbf{c}, \eta)$ ▷ one gradient step on $\boldsymbol{\theta}_{\text{pou}}$ using learning rate η
end function
$\mathbf{c}_{\text{new}} \leftarrow \text{LS}(\boldsymbol{\theta}_{\text{pou}})$ ▷ final solve for $\mathbf{c}$
$\boldsymbol{\theta}_{\text{pou, new}} \leftarrow \boldsymbol{\theta}_{\text{pou}}$

1. **Least-squares update.** Fix $\boldsymbol{\theta}_{\text{pou}}$, and solve for the optimal coefficients:

$$\mathbf{c}^* = \arg\min_{c_{\alpha,\beta}} \sum_{i=1}^{N_{\text{data}}} \left[\sum_{\alpha=1}^{N_{\text{part}}} \sum_{\beta=1}^{\dim(V)} c_{\alpha,\beta}\, \phi_\alpha(\mathbf{x}_i; \boldsymbol{\theta}_{\text{pou}})\, P_\beta(\mathbf{x}_i) - y_i \right]^2$$

   This is a standard least-squares problem, which can be written in matrix form as

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} \|A\,\mathbf{c} - \mathbf{y}\|_2^2$$

   where $A_{i,(\alpha,\beta)} = \phi_\alpha(\mathbf{x}_i; \boldsymbol{\theta}_{\text{pou}})\, P_\beta(\mathbf{x}_i)$. The solution is $\mathbf{c}^* = (A^\mathsf{T} A)^{-1} A^\mathsf{T} \mathbf{y}$.

2. **Gradient-descent update.** With $\mathbf{c} = \mathbf{c}^*$ fixed, take one gradient step on the POU parameters:

$$\boldsymbol{\theta}_{\text{pou}} \leftarrow \boldsymbol{\theta}_{\text{pou}} - \eta\, \nabla_{\boldsymbol{\theta}_{\text{pou}}} \mathcal{L}(\boldsymbol{\theta}_{\text{pou}}, \mathbf{c}^*)$$

   where

$$\mathcal{L}(\boldsymbol{\theta}_{\text{pou}}, \mathbf{c}^*) = \sum_{i=1}^{N_{\text{data}}} \left| y_{\text{pou}}(\mathbf{x}_i; \boldsymbol{\theta}_{\text{pou}}, \mathbf{c}^*) - y_i \right|^2$$

This alternating LSGD procedure enables continuous adaptation of local polynomials to evolving partitions and has demonstrated accelerated convergence in practice [36].

To assess the performance and extend the applicability of POUnets beyond their original scope, we conduct a series of numerical experiments. We begin by reproducing the 1D experiments from the original paper [36] on both piecewise linear and smooth functions. Subsequently, we extend the methodology to 2D domains, highlighting the challenges of partition learning in higher dimensions. This motivates our primary contribution: a novel Separable POUnets (Sep-POUnets) architecture designed to generate structured partitions suitable for downstream tasks.

4.2.2. 1D Piecewise Linear Function

We first evaluate POUnets on a piecewise linear triangular wave function defined as:

$$f(x, p) = 4\left| px \ - \ \left\lfloor px + \tfrac{1}{2} \right\rfloor \right| - 1$$

where the integer p controls the frequency. For this task, the local approximation space consists of linear polynomials (degree 1) with the basis $\{1, x\}$. The training data consists of 1000 pairs (x, y), where x is equally sampled from $[0, 1]$ and $y = f(x, p)$.

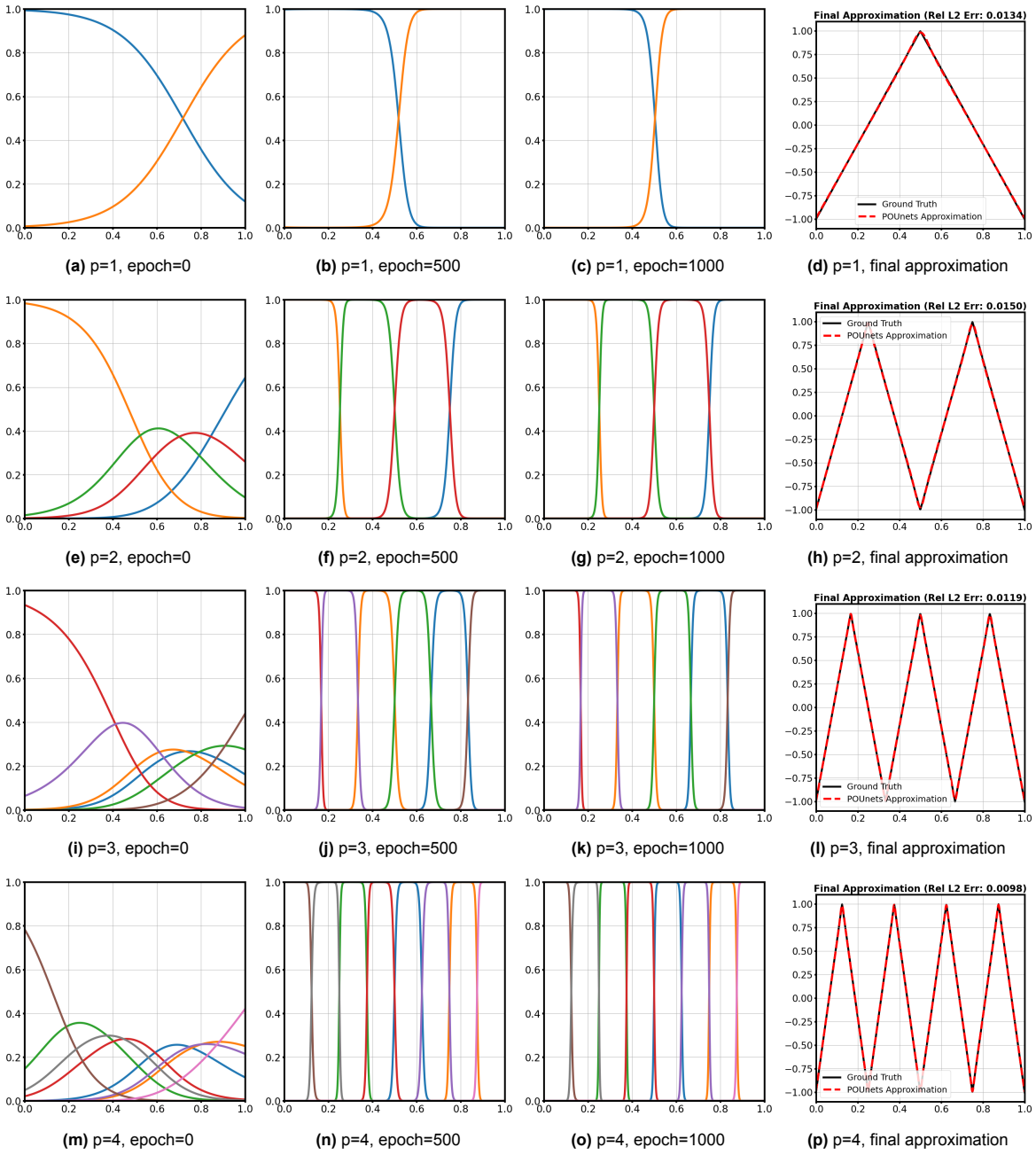
RBF-POUnet:

We employ the Algorithm 2 with an Adam optimizer (learning rate $1\text{e}{-3}$) for 1000 epochs. RBF centers are initialized uniformly in $[0, 1]$ with initial widths equal to 0.25. As shown in Figure 4.6, the RBF-POUnet can successfully learn the underlying partitions and approximates the target function. Owing to the locality of RBF kernels, the network begins with favorable partitioning, which promotes compact, target-consistent partitions during POUnet training.

MLP-POUnet:

The MLP architecture consists of 2 hidden layers with 8 neurons each, trained with Adam (learning rate $1e{-3}$) for 10000 epochs. As illustrated in Figure 4.7, the MLP-POUnet performs well for lower frequencies ($p = 1, 2$). However, for higher frequencies ($p = 3, 4$), it fails to learn effective partitions. In these cases, several partitions collapse or vanish early in training, and this also lead to poor approximation accuracy.

We also experimented with different hyperparameter settings, such as adjusting the learning rate, modifying the network size, and increasing the number of training epochs, but these adjustments did not yield significant improvements. The MLP-POUnet still struggled to learn effective partitions for higher frequencies. Detailed results are provided in Appendix A.1.



(a) p=1, epoch=0 **(b)** p=1, epoch=500 **(c)** p=1, epoch=1000 **(d)** p=1, final approximation

(e) p=2, epoch=0 **(f)** p=2, epoch=500 **(g)** p=2, epoch=1000 **(h)** p=2, final approximation

(i) p=3, epoch=0 **(j)** p=3, epoch=500 **(k)** p=3, epoch=1000 **(l)** p=3, final approximation

(m) p=4, epoch=0 **(n)** p=4, epoch=500 **(o)** p=4, epoch=1000 **(p)** p=4, final approximation

Figure 4.6: Training progression of RBF-POUnet on the triangular wave function for $p \in \{1, 2, 3, 4\}$. The first three columns show the evolution of the learned partitions over 1000 epochs. The final column displays the resulting function approximation.

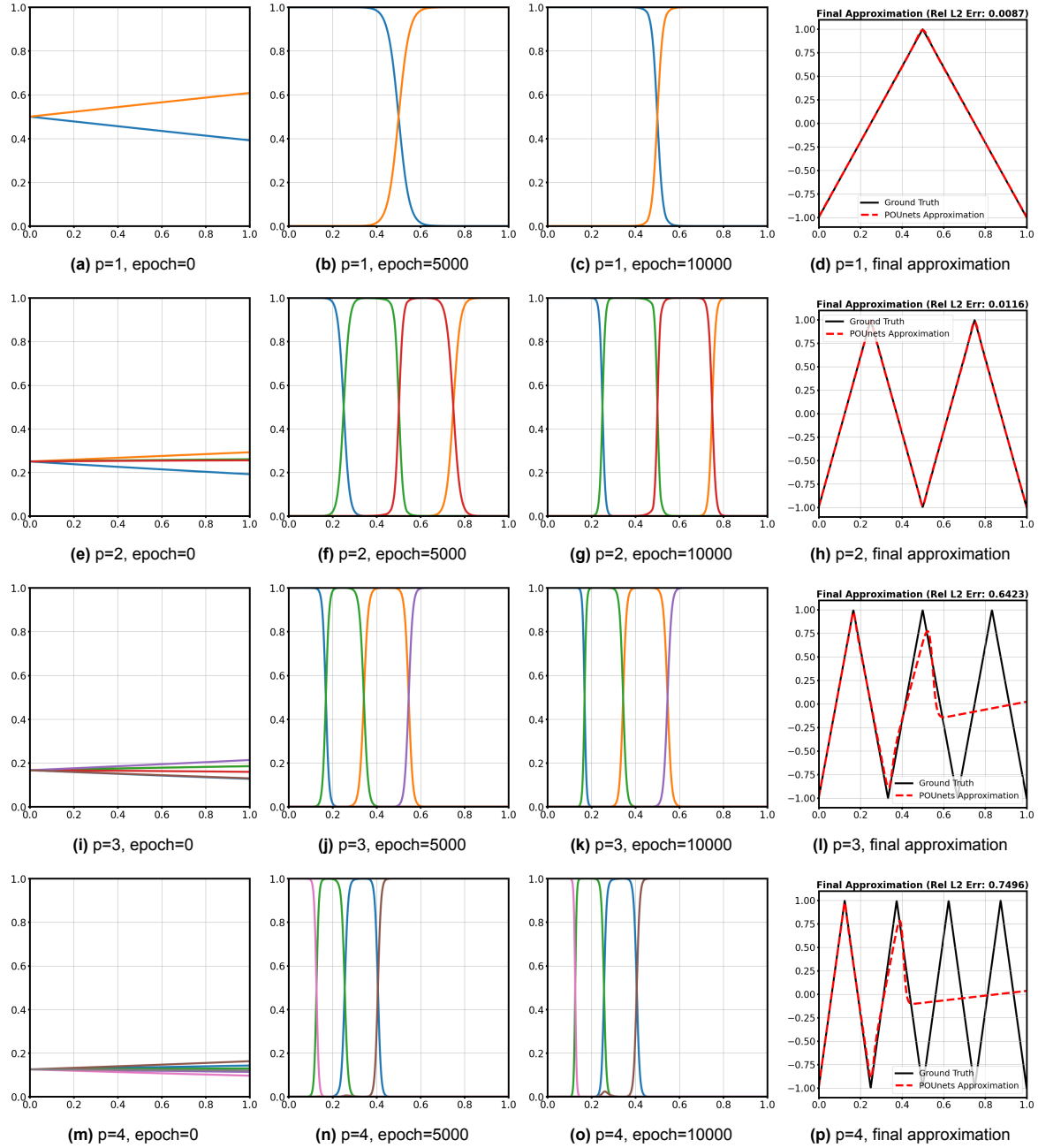


Figure 4.7: Training progression of MLP-POUnet on the triangular wave function for $p \in \{1, 2, 3, 4\}$. The first three columns show the evolution of the learned partitions over 1000 epochs. The final column displays the resulting function approximation. For $p \geq 3$, the network fails to maintain efficient partitions, resulting in a poor final approximation.

4.2.3. 1D Smooth Function

To investigate the model's applicability beyond piecewise linear functions, we test its performance on a smooth sinusoidal function:

$$f(x, k) = \sin(2\pi k x), \quad x \in [0, 1]$$

where k ajust the frequency. For this problem, we use a local quadratic basis (degree 2) $\{1, x, x^2\}$

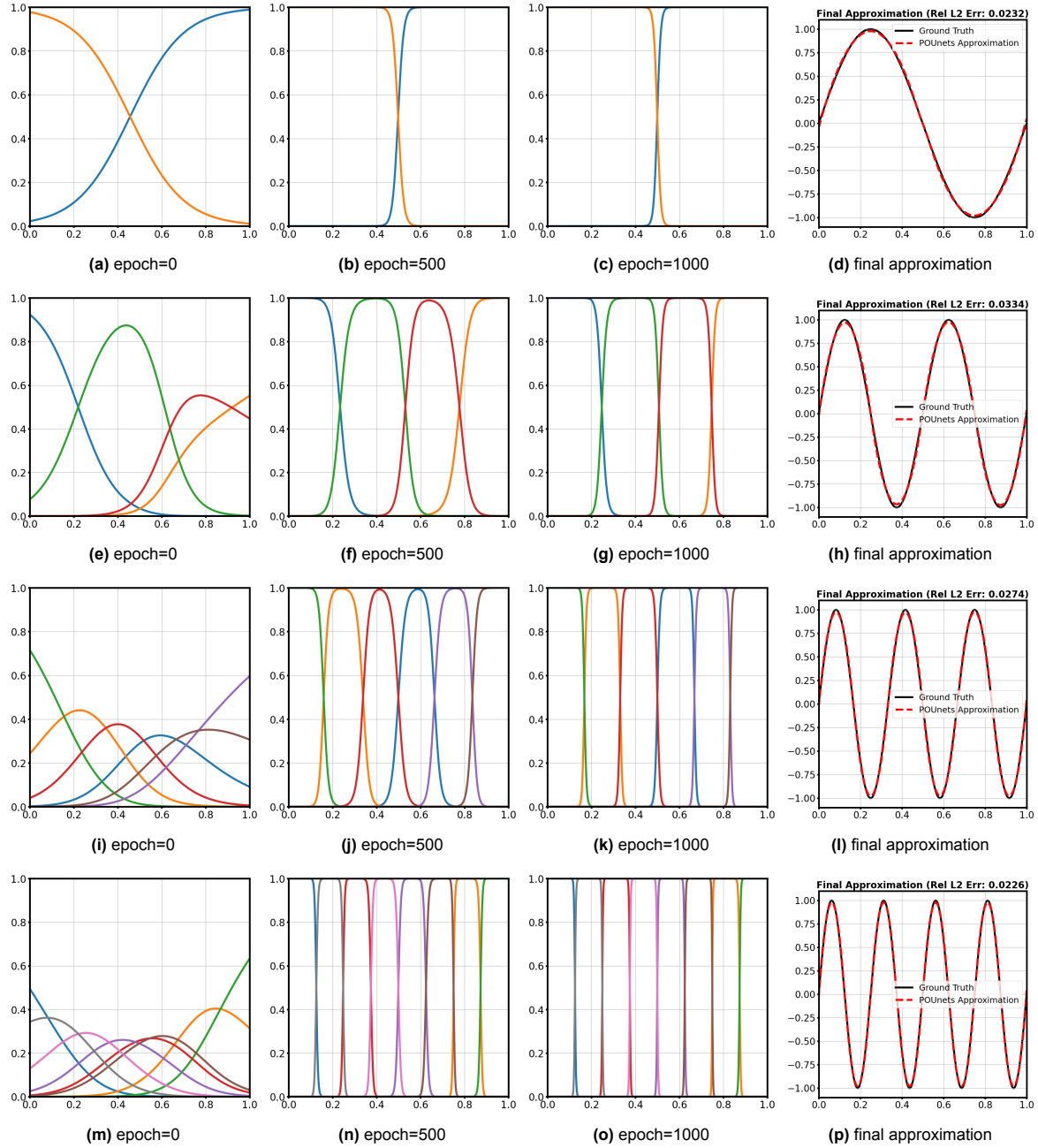


Figure 4.8: Training progression of MLP-POUnet on the smooth function for $k \in \{1, 2, 3, 4\}$. The first three columns show the evolution of the learned partitions over 1000 epochs. The final column displays the resulting function approximation.

For the smooth target function, the RBF-POUnet can also learn good partitions with different value of k as can be seen from Figure 4.8.

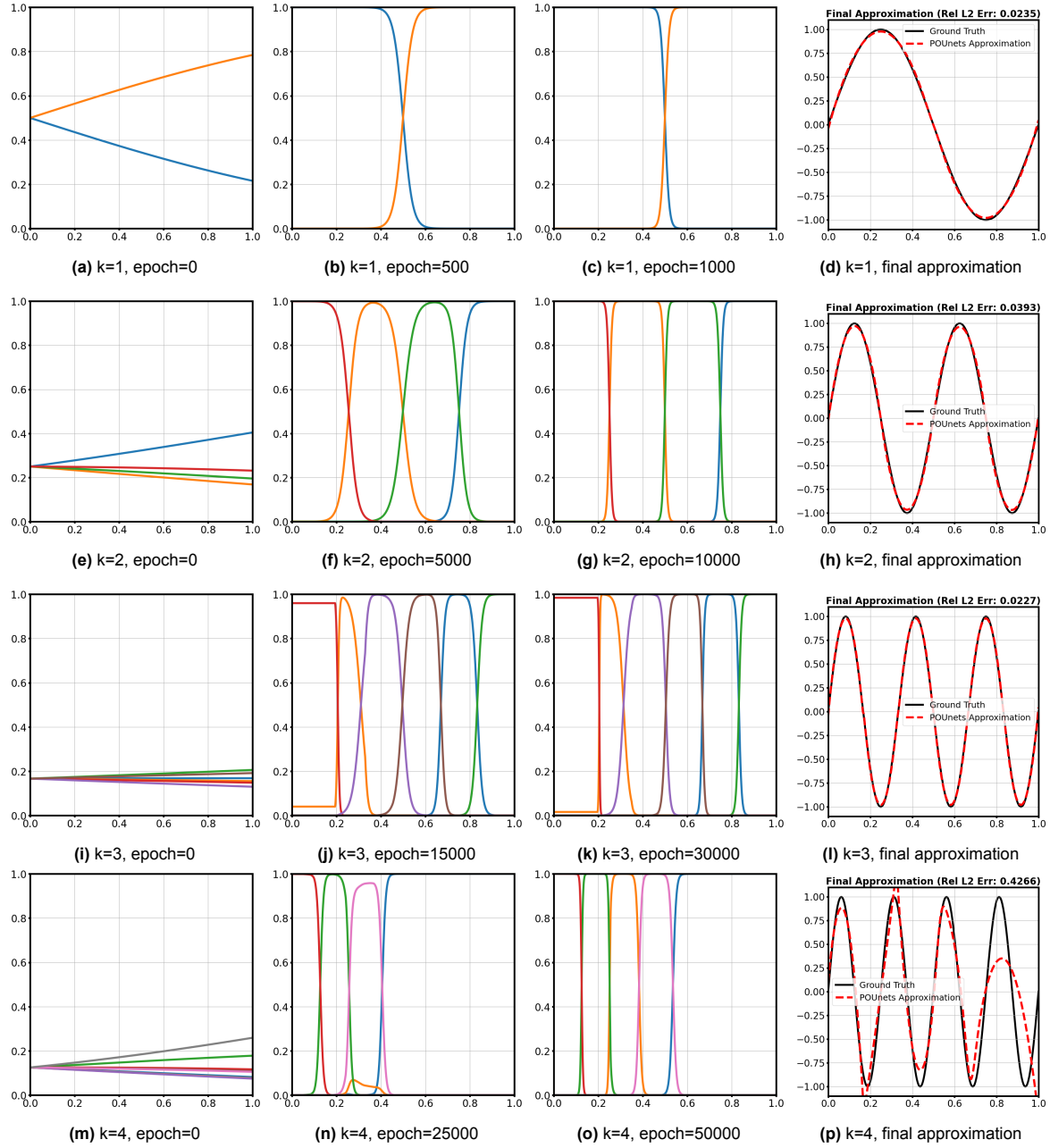


Figure 4.9: Training progression of MLP-POUnet on the smooth function for $k \in \{1, 2, 3, 4\}$. The first three columns show the evolution of the learned partitions over 1000 epochs. The final column displays the resulting function approximation. For k=1, MLP-POUnet is trained at a learning rate of 1e-3 only for 1000 epochs with 2 layers and 16 units. For k = 2, we train the POUnet for 10000 epochs at a learning rate of 1e-3 for 10000 epochs. For k = 3, we train the POUnet with learning rate = 1e-4 for 30000 epochs with 2 layers and 8 units MLP. For k = 4, we train the POUnet with learning rate = 1e-4 for 50000 epochs with 2 layers and 8 units MLP.

We observe that for $k = 1$ and $k = 2$, the MLP-POUnet consistently learns effective partitions and achieves good approximation results. However, for higher frequencies ($k = 3$ and $k = 4$), the MLP-POUnet fails to learn effective partitions, leading to their collapse or disappearance and resulting in poor approximations, even after increasing the number of training epochs and adjusting the learning rate.

Remarks:

In the 1D examples for both piecewise-linear and smooth target functions, successful training of RBF-

POUnets and MLP-POUnets reveals a common pattern: the partitions first identify the dominant regions. As training progresses, these partitions are gradually refined, becoming more compact with reduced overlap.

And for MLP-POUnet, some partitions collapsed to zero at an early epoch. The original POUnet paper [36] suggests applying a regularization term during the LS stage—and decaying it during training when a stagnation of loss for many steps is detected. This technique, which we did not use in the experiments, could be a future direction to improve the performance of POUnet.

4.2.4. 2D Sine Function

The extension of POUnets to 2D domains introduces significant geometric complexity in partition learning. We first test the standard RBF-POUnet and MLP-POUnet on the following 2D sine function:

$$u(\mathbf{x}) = \sin(2\pi x)\sin(2\pi y), \quad \mathbf{x} = (x, y) \in [0, 1]^2 \tag{4.9}$$

Both models are trained for 10,000 epochs with a learning rate of $1\mathrm{e}{-3}$. The local approximation space is composed of second-order polynomials.

The training data are generated by sampling on a uniform 100×100 grid in the domain $[0, 1]^2$.

RBF-POUnet

As shown in Figure 4.10a-4.10e, the RBF-POUnet learns partitions that qualitatively match the target function's features, dividing the domain into four small squares. However, the plot of the difference between the exact solution and approximation reveals high inaccuracies, particularly at the corners of the domain. We will analysis this phenomena with the approximation results of MLP-POUnet together later in this section.

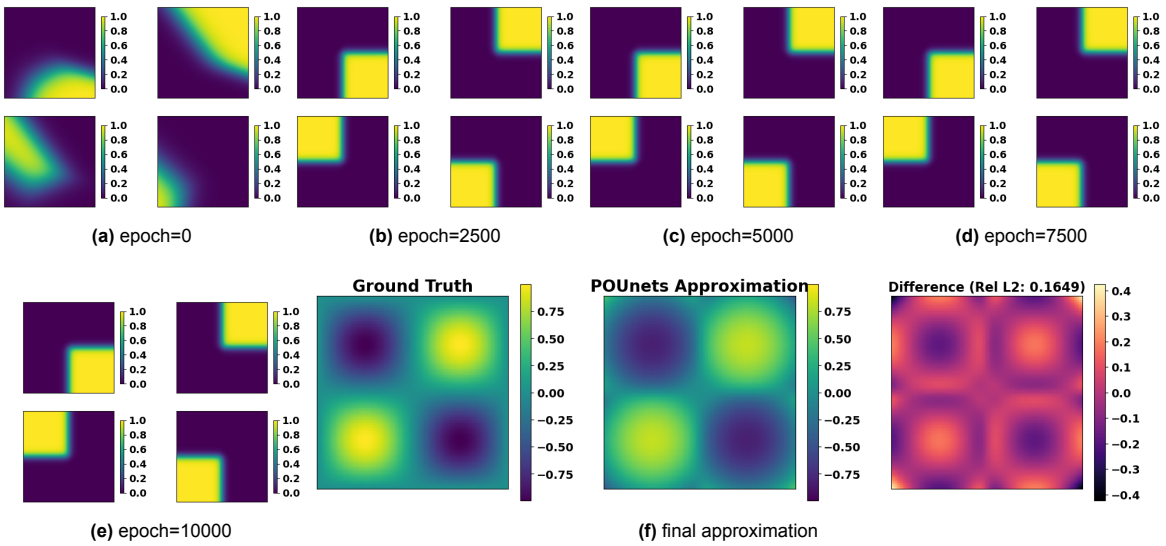


Figure 4.10: Training progression of the standard RBF-POUnet on the 2D sine function. (a-e): Evolution of the irregular domain partitions over 10,000 epochs. (f): Final results showing the ground truth, the POUnet approximation, and their difference.

MLP-POUnet

Compared with the partitions learned with RBF-POUnet as shown in Figure 4.10a-4.10e, the MLP-POUnet produces more irregular partitions as shown in Figure 4.11a-4.11e. Interestingly, we observe an inconsistent relationship between partition regularity and corner accuracy. For instance, subdomains that are more regular as square-liked exhibit high corner errors, whereas on the irregular subdomains, it has lower error in the corners but higher error near the center.

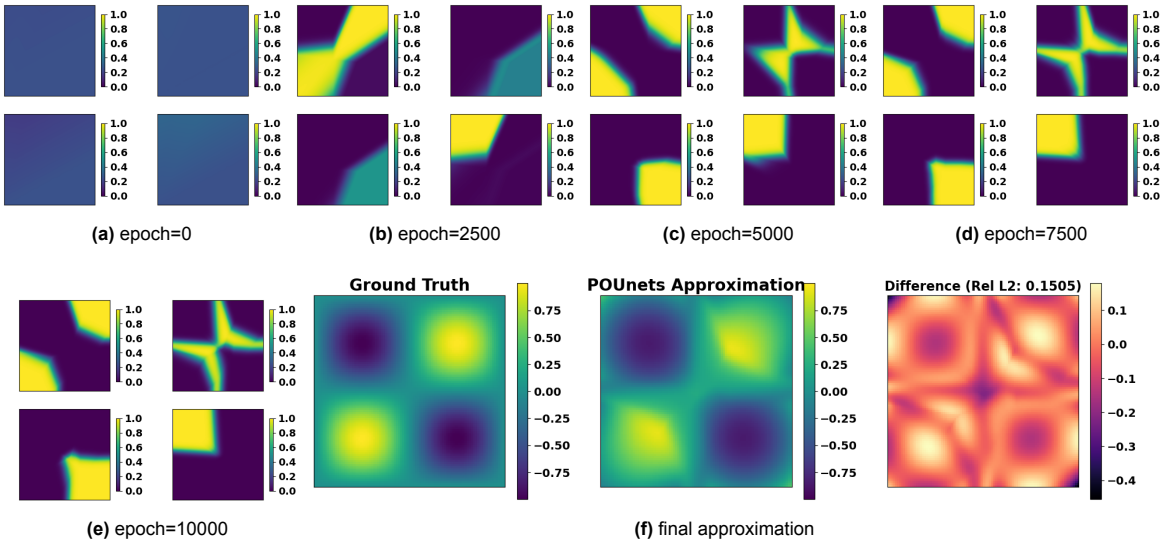


Figure 4.11: Training progression of the MLP-POUnet on the 2D sine function. (a-e): Evolution of the irregular domain partitions over 10,000 epochs. (f): Final results showing the ground truth, the POUnet approximation, and their difference.

This observation is critical: visually ”optimal” or regular partitions do not necessarily yield the best function approximation. While these irregular partitions may be effective for the task of functions approximation, they may be poorly suited for providing a subdomain partitions for the downstream tasks. That said, our goal is not to obtain the best function approximation with POUnets, but to use their learned partition as a guidance for FBPINNs. To this end, we introduce Separable POUnets in the following, which aim to generate more regular partitions.

Separable POUnets (Sep-POUnets)

$$\mathbf{x} = (x, y) \in \mathbb{R}^2.$$

We construct a set of functions, where n_x and n_y are the number of partitions along the x and y dimensions, respectively. The total number of partitions is $N = n_x \times n_y$. Each partition, indexed by (i, j), has an associated partitions function:

$$\{w_{ij}(\mathbf{x})\}_{i=1,j=1}^{n_x,n_y}$$

These partitions satisfy POU condition, ensuring that their sum is always one at any point in the domain:

$$\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} w_{ij}(\mathbf{x}) = 1, \qquad \forall \mathbf{x}.$$

Two independent one-dimensional functions (such as MLPs, RBF-Nets, or other network architectures) produce axis-wise predictions:

$$w_{ij}(\mathbf{x}) = \frac{\exp\big(f_x(x;\boldsymbol{\theta}_x)_i + f_y(y;\boldsymbol{\theta}_y)_j\big)}{\sum_{p=1}^{n_x}\sum_{q=1}^{n_y}\exp\big(f_x(x;\boldsymbol{\theta}_x)_p + f_y(y;\boldsymbol{\theta}_y)_q\big)}, \qquad i = 1,\dots,n_x,\ j = 1,\dots,n_y.$$

Since the denominator factorizes, we have

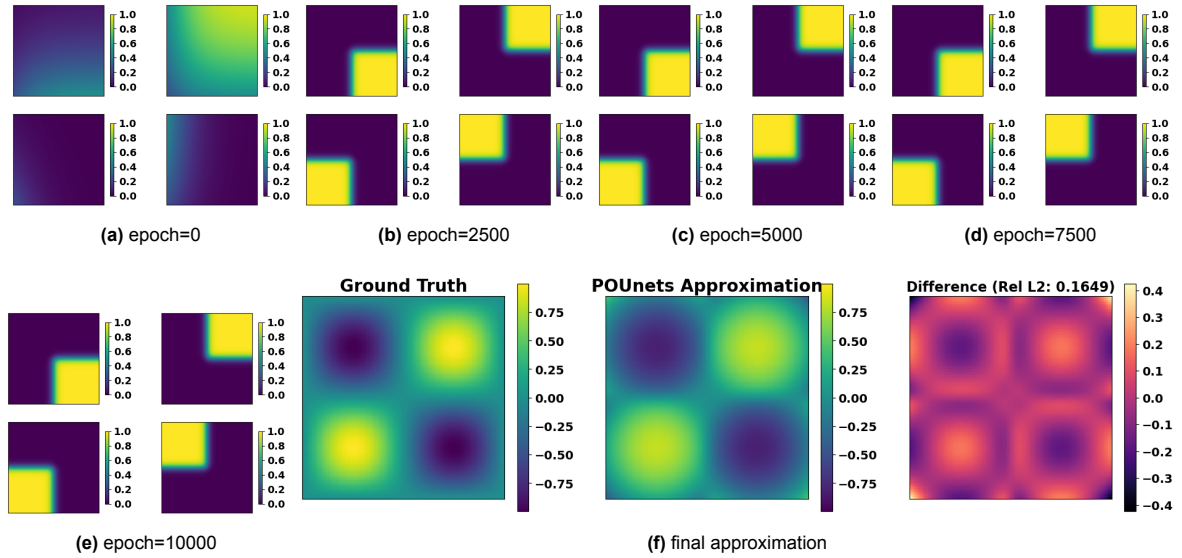
$$\sum_{p=1}^{n_x}\sum_{q=1}^{n_y}\exp\big(f_x(x;\boldsymbol{\theta}_x)_p + f_y(y;\boldsymbol{\theta}_y)_q\big) = \Big(\sum_{p=1}^{n_x}\exp f_x(x;\boldsymbol{\theta}_x)_p\Big)\Big(\sum_{q=1}^{n_y}\exp f_y(y;\boldsymbol{\theta}_y)_q\Big),$$

so we can rewrite w_{ij} in the fully factorized form:

$$w_{ij}(\mathbf{x}) = \frac{\exp f_x(x;\boldsymbol{\theta}_x)_i}{\sum_p \exp f_x(x;\boldsymbol{\theta}_x)_p}\ \frac{\exp f_y(y;\boldsymbol{\theta}_y)_j}{\sum_q \exp f_y(y;\boldsymbol{\theta}_y)_q},$$

which makes it explicit that the single softmax is algebraically identical to first normalizing each axis ($\sum_i w_i = 1,\ \sum_j w_j = 1$) and then taking their outer product. This compact formulation preserves the POU property.

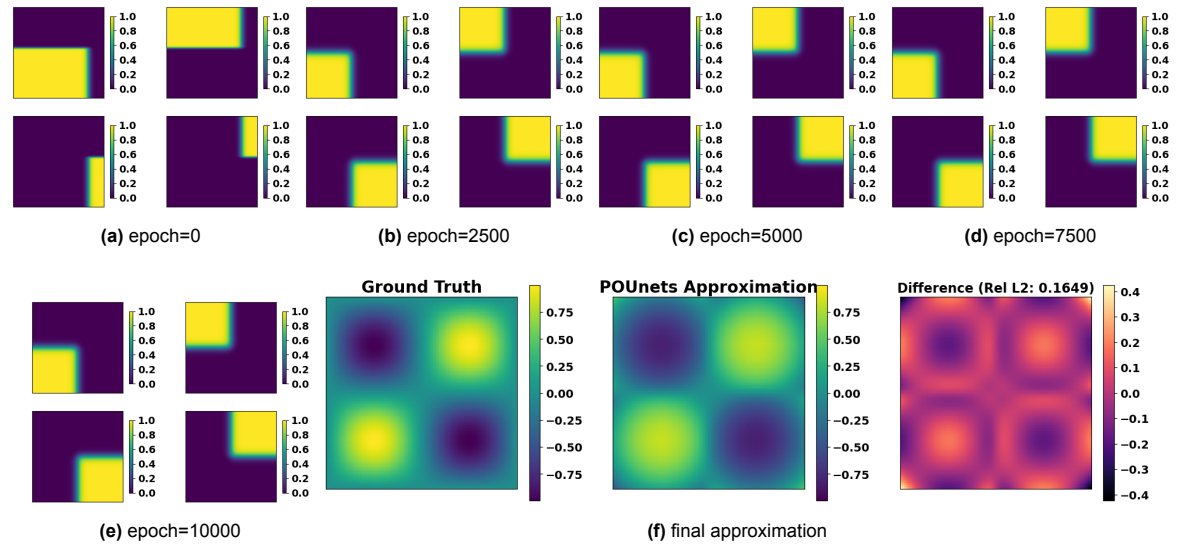
Next, we evaluate Sep-POUnets on the same 2D problem in (4.9). In the separable construction, the partitions learned across dimensions, and each factor is parameterized by a chosen network. We consider MLP and RBF-Net based POUnets on each dimension, denoting as Sep-MLP-POUnet and Sep-RBF-POUnet respectively.



(a) epoch=0 **(b)** epoch=2500 **(c)** epoch=5000 **(d)** epoch=7500

(e) epoch=10000 **(f)** final approximation

Figure 4.12: Training progression of the **Sep-MLP-POUnet** on the 2D sine problem. (a–e): Evolution of learned partitions over 10000 epochs. (f): Final results showing the ground truth, the POUnet approximation, and their difference.



(a) epoch=0 **(b)** epoch=2500 **(c)** epoch=5000 **(d)** epoch=7500

(e) epoch=10000 **(f)** final approximation

Figure 4.13: Training progression of the **Sep-RBF-POUnet** on the 2D sine problem. (a–e) Evolution of learned partitions over 10000 epochs. (f) Final results: ground truth, POUnet approximation, and their difference..

As shown in Figures 4.12f and 4.13e, both Sep-POUnets have the learned partitions closely align with the feature of the target function, producing four near-regular, approximately square partitions. And this indicate that the Sep-POUnets are workable for finding the square like partitions even though it has the same problem as the RBF-POUnet with high error in the four corners.

4.3. Proposed Method

Since POUnets are data-driven and PDE problems lack labeled data, our initial approach is to use a standard PINN to generate a low-fidelity solution. This solution then serves as the target data for training a POUnet, which learns partitions that satisfy the POU property. These learned partitions, in turn, guide the subsequent FBPINN training.

The integration of FBPINNs and POUnets is motivated by the need for flexible, data-driven domain decomposition when solving complex PDEs. As demonstrated in Section 4.1.2, which compares a uniform domain decomposition-based FBPINN with a feature-aligned domain decomposition-based FBPINN, FBPINNs effectively leverage local neural networks on overlapping subdomains; however, their performance depends critically on the chosen partitioning, making the decomposition strategy a key determinant of accuracy and efficiency.

POUnets, on the other hand, can learn partitions directly from data. As shown in Section 4.2, POUnets are able to learn partitions that adapt to the frequency and features of the target function, capturing regions of high variation and aligning with solution features. This enables adaptive, feature-aligned domain decompositions in a data-driven way. By combining these approaches, we aim to improve solution accuracy.

This section introduces the framework of adaptive domain decomposition-based FBPINN, which we will refer to as **Adaptive DD FBPINN**.

Algorithm 3 Adaptive DD FBPINN

Train FBPINN at $N_1{=}1$ and obtain $\hat{u}^{(1)}$; $\mathbf{y}_{\text{target}} \leftarrow \hat{u}^{(1)}$
Schedule partitions $\{N_1, N_2, \ldots, N_S\}$; choose threshold τ
$\Phi^\star \leftarrow \Phi^{(1)}$, $N_{\text{prev}} \leftarrow N_1$
for $N \in \{N_2, \ldots, N_S\}$ **do**
  Train POUnet at N using $\mathbf{y}_{\text{target}}$; obtain partitions $\Phi^{(N)}$
  if $\exists \alpha$ s.t. $\max_{\mathbf{x}} \phi_\alpha^{(N)}(\mathbf{x}) < \tau$ **then** ▷ Collapse detected at N
    Train FBPINN at N_{prev} with $\Phi^\star$ to get $\hat{u}^{(N_{\text{prev}})}$
    $\mathbf{y}_{\text{target}} \leftarrow \hat{u}^{(N_{\text{prev}})}$ ▷ Update target with a better solution
    Train POUnet at N using new $\mathbf{y}_{\text{target}}$ and obtain new $\Phi^{(N)}$ ▷ Retry POUnet training for N
    if $\exists \alpha$ s.t. $\max_{\mathbf{x}} \phi_\alpha^{(N)}(\mathbf{x}) < \tau$ **then** ▷ Collapse again at N
      return last FBPINN (at N_{prev}) and $\Phi^\star$ ▷ Stop
    end if
  end if
  $\Phi^\star \leftarrow \Phi^{(N)}$, $N_{\text{prev}} \leftarrow N$ ▷ Update partitions and proceed to next N
end for
Train FBPINN at N_{prev} with $\Phi^\star$
return last FBPINN and $\Phi^\star$

This method combines FBPINNs and POUnets. The adaptive procedure, detailed in Algorithm 3 and illustrated in Figure 4.14, consists of the following stages:

1. **Initialization:** A baseline FBPINN with a single subdomain ($N_1 = 1$) is trained to generate an initial, low-fidelity solution $\hat{u}^{(1)}$. This solution serves as the first target, $\mathbf{y}_{\text{target}}$, for the POUnet. A schedule of increasing partition numbers, $N = [N_1, N_2, \ldots, N_S]$, is defined.
2. **Iterative Partitioning and Refinement:** The algorithm iterates through the schedule of partition numbers. For each number of partitions N:
   (a) **POUnet Training:** A POUnet is trained to learn a set of N partitions, $\Phi^{(N)}$, using the prediction basedcurrent $\mathbf{y}_{\text{target}}$.
   (b) **Collapse Detection:** The algorithm checks if any learned partition has ”collapsed”.
      - **If a collapse occurs:** The algorithm assumes the current partitions are not good enough we denote it as N_{fail}. It reverts to the last successful partition configuration ($\Phi^\star$ at N_{prev})

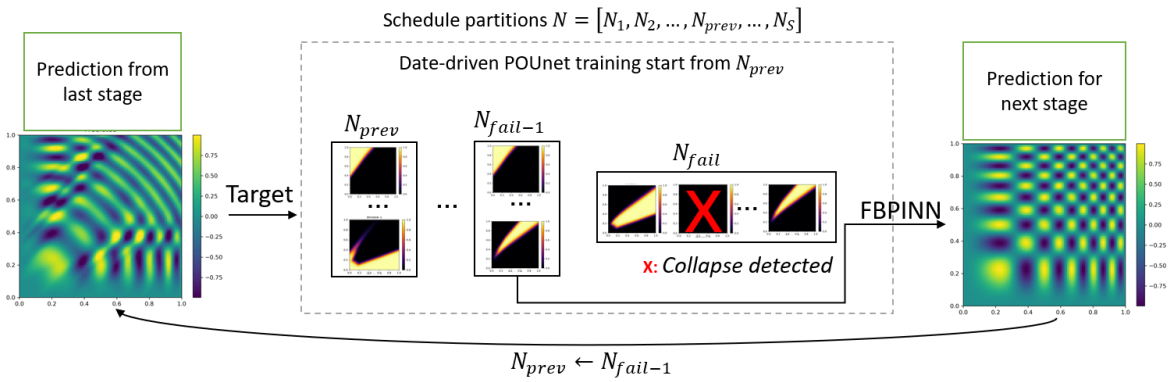


Figure 4.14: Illustration of the Adaptive DD FBPINN procedure: (left) prediction from the previous stage; (middle) POUnet training with scheduled partitions; (right) FBPINN prediction for the next stage.

and trains an FBPINN based on the success partitions to obtain a more accurate solution. This new solution becomes the updated $\mathbf{y}_{\text{target}}$, and the POUnet training is retried for the same N. If it collapses again, the loop terminates, returning the last successful result.

- **If no collapse occurs:** The learned partitions $\Phi^{(N)}$ are accepted as the new best configuration $\Phi^{\star}$, and the algorithm proceeds to the next partition number in the schedule.

3. **Final FBPINN Training:** Once the loop completes (either by finishing the schedule or by stopping due to repeated collapse), a final FBPINN is trained using the last successful set of partitions, $\Phi^{\star}$. This yields the final, high-fidelity solution.

As described in the algorithm, we define a ”collapse” as occurring when the maximum value of any partition function drops below a threshold τ, i.e., $\max_{\mathbf{x}} \phi_{\alpha}^{(N)}(\mathbf{x}) < \tau$ for some partition α. A partition is fully effective when its maximum value is 1; its influence diminishes as this value approaches zero. As the experiments on 1D piecewise-smooth and smooth functions show, the collapse of partitions consistently leads to a poor approximation of the target function. Thus, this threshold serves as a practical criterion for identifying partitions that are no longer contributing meaningfully.

In addition, we employ RAD resampling throughout training of the Adaptive DD FBPINN, continually resampling collocation points according to the evolving PDE residual so that learning concentrates on the most challenging regions.

Here we use the problem defined in Equation (3.24) as a motivating example for the Adaptive DD FBPINN. The build of FBPINN remains the same as in the previous section. The only difference is that we use the learned partitions from the POUnet as the window functions in the FBPINN training.

We then present the results of the Adaptive DD FBPINN using the four POUnet architectures discussed previously: RBF-POUnet, MLP-POUnet, Sep-RBF-POUnet, and Sep-MLP-POUnet. The performance is compared using both quantitative metrics and visual inspection of the approximations.

The shared hyperparameters for the Adaptive DD FBPINN are recorded in Table 4.2. The only component that varies across runs is the choice of POUnet architecture and its specific parameters, which are detailed in each example below.

Table 4.2: Fixed hyperparameters for the Adaptive DD FBPINN.

Adaptive DD FBPINN	
Schedule partitions	$N = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$
Collapse threshold (τ)	0.8
Collocation points	100×100 uniformly sampled
Training steps	30000
Learning rate	10^{-3}
Training optimizer	Adam
Subdomain Network	2 layers, 64 neurons for each layer
RAD method	Poolsize=20000, resample every 10000 steps ($k = 1, c = 1$)

All the Adaptive DD FBPINN share the same initialization, a low-fidelity reference solution is generated by training an FBPINN with a single subdomain (equivalent to a standard PINN), as shown in Figure 4.15. This solution serves as the initial training target for all POUnet variants. Then, we use this target to train the POUnet to learn the partitions.

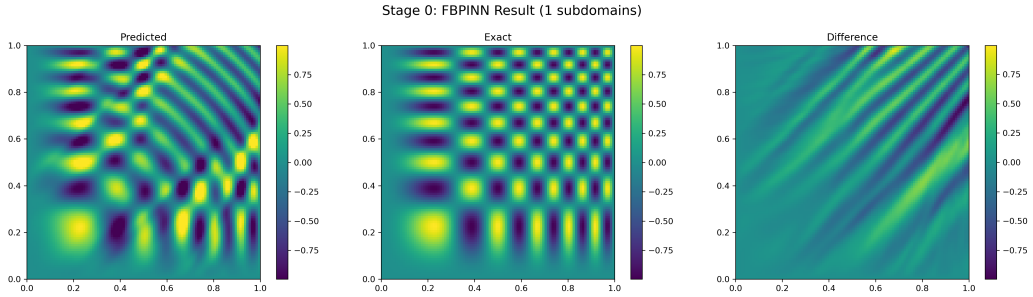
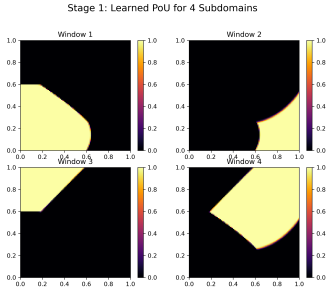


Figure 4.15: Initialization: (left): one subdomain FBPINN prediction.(middle): ground truth; (right): plot of difference. This serves as the initial target for the POUnets training in the Adaptive DD FBPINN.

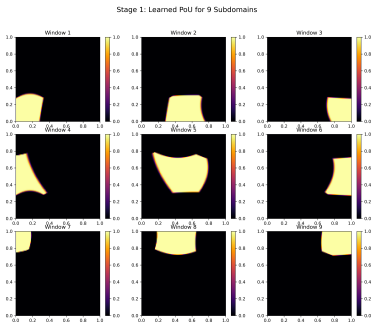
4.3.1. Adaptive DD FBPINN with RBF-POUnet

Based on the prediction in the initialization, we train a RBF-POUnet to learn the POU functions. The initial centers of RBF-Net is generately uniformly on the domain with fixed initial width equals to 0.25. There's no extra needed hyperparameters for the RBF-POUnet.

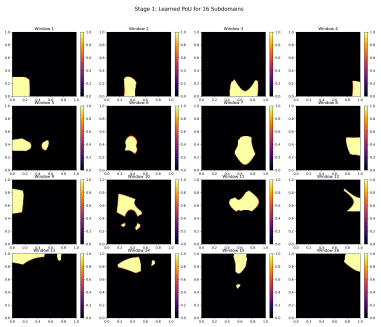
We trained the RBF-POUnet for partition numbers starts at $N = 4$, to $N = 9, 16, 25, 36, 49$. The first collapse occurred at $N = 64$, therefore, we use the partitions learned at $N = 49$ to train a new FBPINN. All the learned partitions are shown in Figure 4.16.



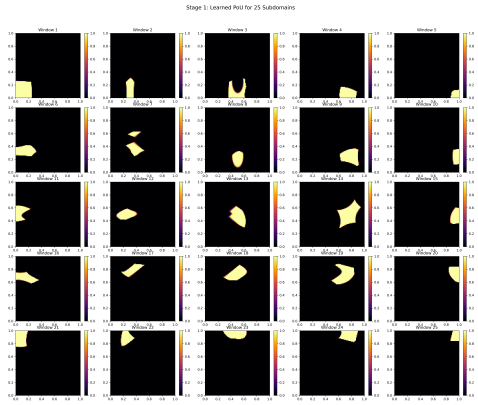
(a) $N = 4$



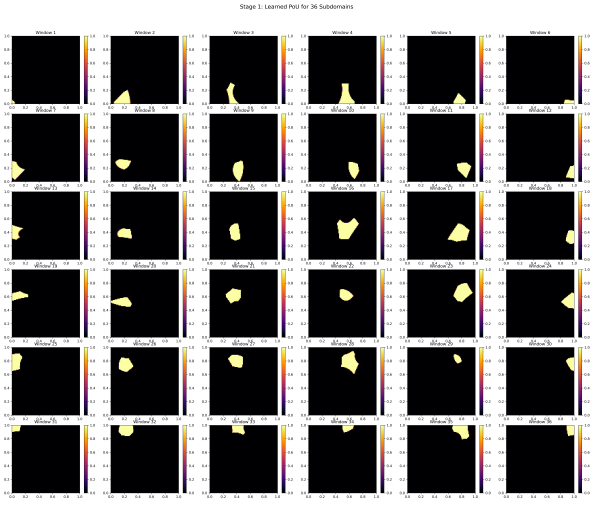
(b) $N = 9$



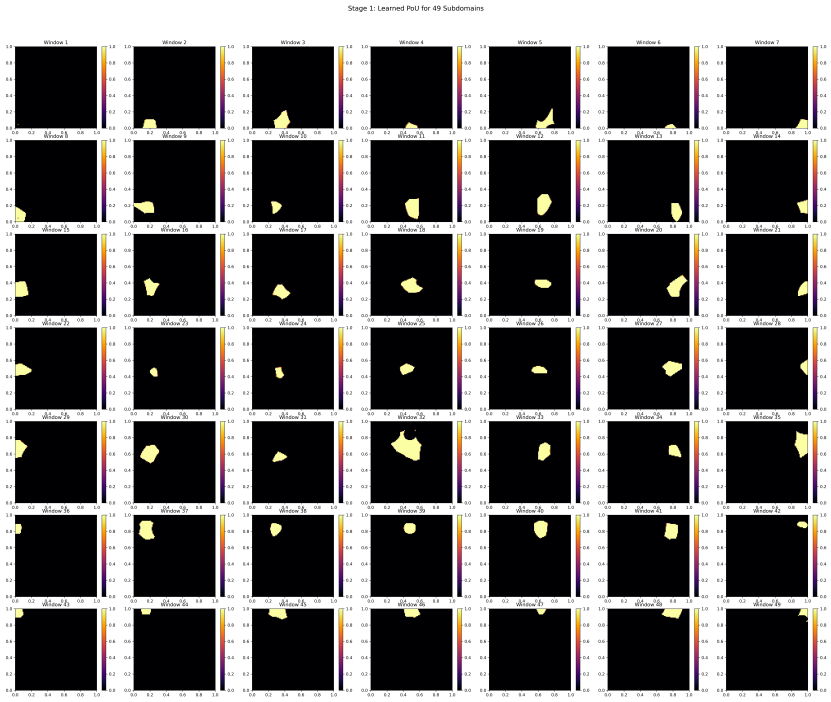
(c) $N = 16$



(d) $N = 25$



(e) $N = 36$



(f) $N = 49$

Figure 4.16: RBF-POUnet: (a)-(f) Learned partitions of the adaptive DD FBPINN with RBF-POUnet, for partitions of $N \in \{4, 9, 16, 25, 36, 49\}$ subdomains. And the collapse is dectected at $N = 64$.

As Figure 4.16f shows, at $N = 49$ the resulting partitioning is irregular, with limited inter-subdomain overlap. Consequently, the FBPINN trained on these learned partitions still exhibits pronounced errors along the diagonal and in the upper-right corner as shown in Figure 4.17.

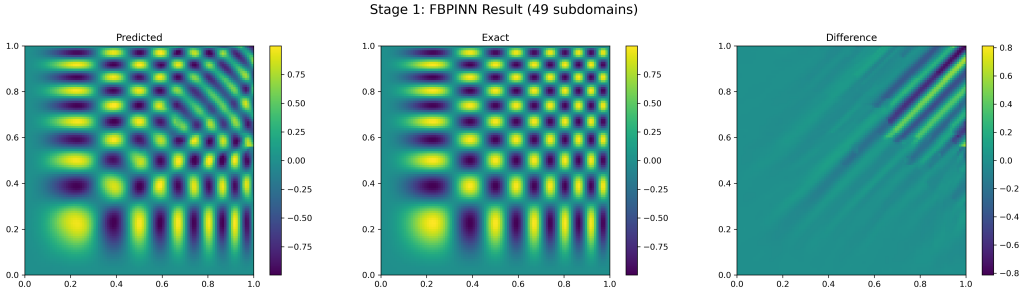
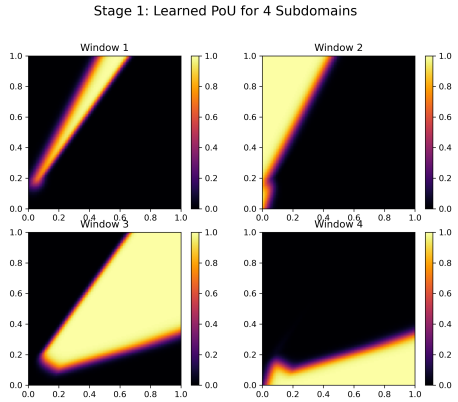


Figure 4.17: RBF-POUnet: (left): FBPINN prediction based on the learned partitions for $N = 49$ from Figure 4.16f; (middle): ground truth; (right): plot of difference.

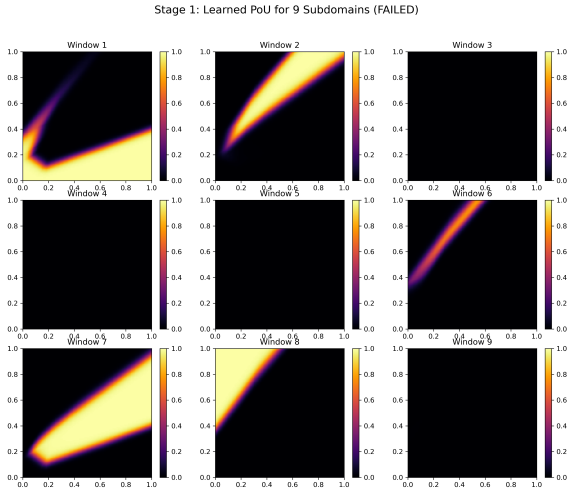
4.3.2. Adaptive DD FBPINN with MLP-POUnet

In this case, we use the MLP-POUnet to learn partitions for the same PDE problem. For each number of parititions, the MLP-POUnet is trained with 10000 epochs and a learning rate of 1e-3, 2 layers with 16 neurons each.

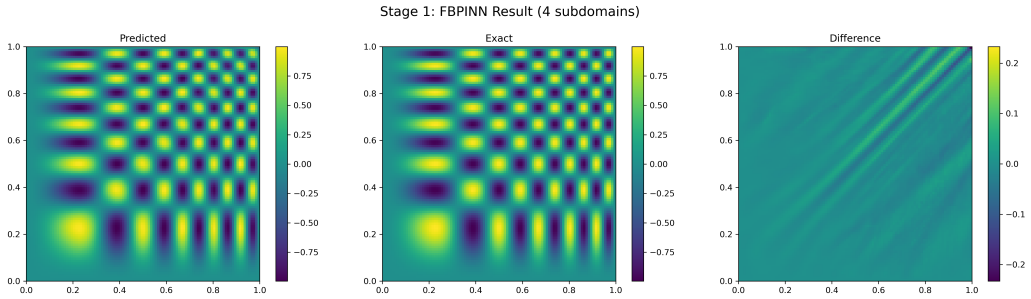
Then, based on the prediction in initialization, we train the MLP-POUnet with 4 partitions. The learned partitions are shown in Figure 4.18a. Although the resulting partition from the MLP-POUnet is not perfectly regular, it is still a meaningful decomposition. The partition boundaries are somewhat irregular, but they effectively cover the entire domain.



(a) Learned partitions for 4 subdomains.



(b) Learned partitions for 9 subdomains (collapsed).



(c) FBPINN prediction using the learned partitions for 4 subdomains.

Figure 4.18: MLP-POUnet: (a) Learned partitions for 4 subdomains; (b) Learned partitions for 9 subdomains (collapsed); (c) FBPINN prediction using the learned partitions for 4 subdomains.

Using the learned partitions, we train FBPINN. The prediction is shown in Figure 4.18c. However, it can be seen that the prediction is not satisfactory, with a significant error of approximately 0.2 along the diagonal. This indicates that the partitioning is not effective for this case.

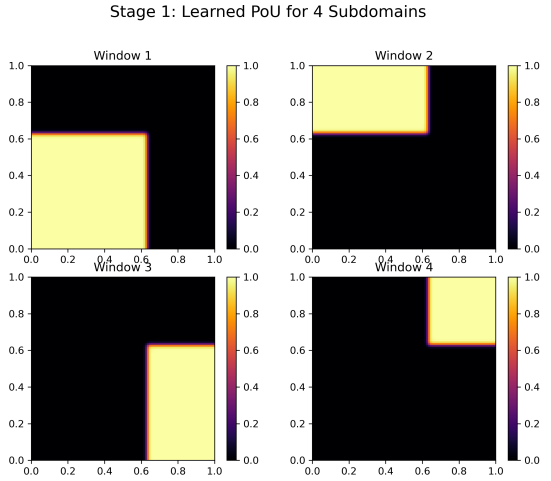
Then we continue to train the MLP-POUnet, which based on the less satisfactory predicition of FBPINN, we train the MLP-POUnet with 9 partitions. However, the partitions collapse again at $N = 9$. This indicates that the MLP-POUnet is not able to learn meaningful partitions for this case, therefore we stop.

For the two non-separable POUnets, the RBF-POUnet learns highly irregular subdomain boundaries with poor overlap. The MLP-POUnet, on the other hand, is easy to collapse when learning a large number of partitions. This issue for MLP-POUnet was also validated in the 1D experiments on smooth and piecewise linear functions: as the problem's frequency increases, so does the required number of partitions, making collapse more likely.

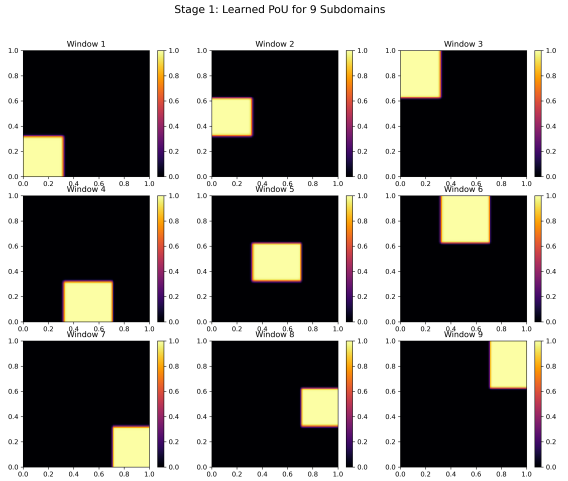
4.3.3. Adaptive DD FBPINN with Sep-RBF-POUnet

For Sep-RBF-POUnet in Adaptive DD FBPINN, we train for 10000 epochs with a learning rate of 0.0001. On each dimension, the RBF centers are initialized uniformly over the domain, with a fixed initial width of 0.25.

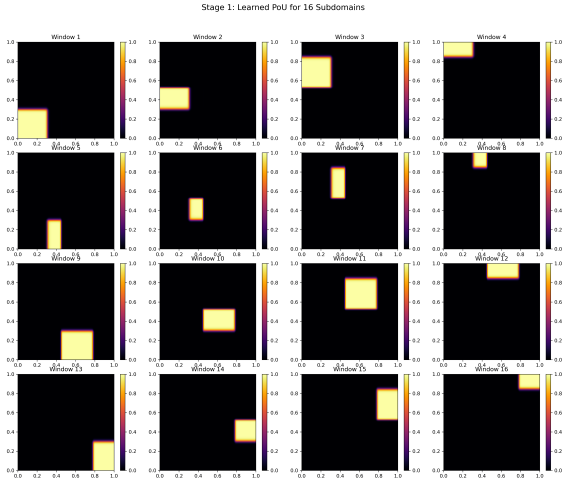
Starting from the prediction in the initialization stage, as the number of partitions increases, the Sep-RBF-POUnet successfully learns partitions for $N = 4$, $N = 9$, and $N = 16$. However, at $N = 25$, the learned partitions collapse (see Figure 4.19d).



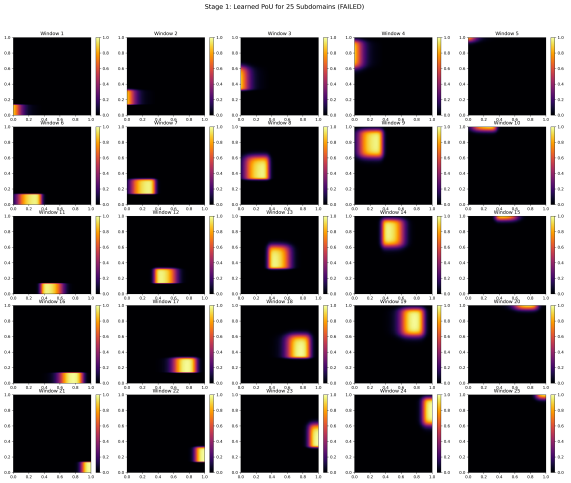
(a) Learned partitions for $N = 4$ subdomains.



(b) Learned partitions for $N = 9$ subdomains.



(c) Learned partitions for $N = 16$ subdomains.



(d) Learned partitions for $N = 25$ subdomains (collapsed).

Figure 4.19: Sep-RBF-POUnet: Learned partitions for (a) $N = 4$, (b) $N = 9$, (c) $N = 16$, and (d) $N = 25$ subdomains. Collapse is detected at $N = 25$.

The partitions learned by the Sep-RBF-POUnet align with the structure of the exact solution: they allocate smaller subdomains near the upper-right corner where the exact solution exhibits higher frequency. Using the $N = 16$ partition, we then train the next FBPINN, yielding the results shown in Figure 4.20.

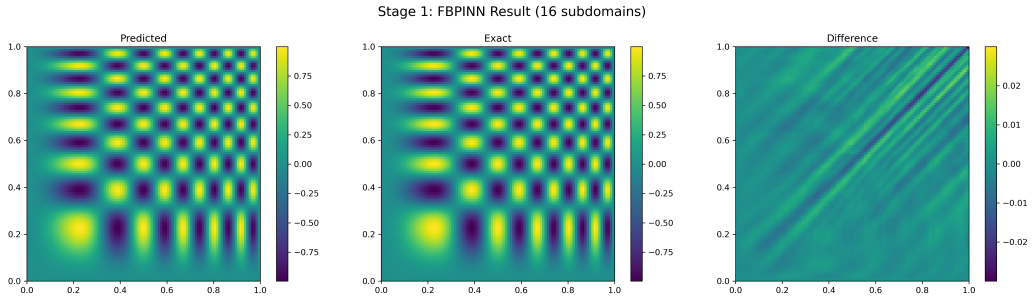


Figure 4.20: Sep-RBF-POUnet: FBPINN prediction based on the learned partitions for $N = 16$.

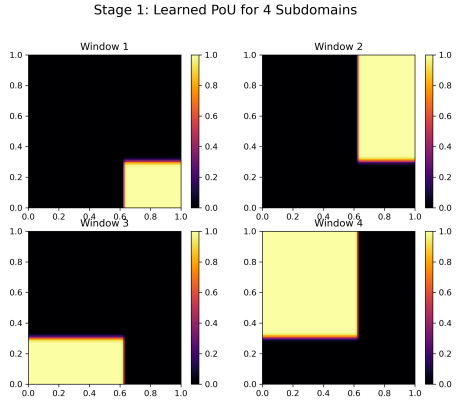
We also observe that the partitions generated by the Sep-RBF-POUnet align with the structure of the exact solution: smaller subdomains are allocated to the upper-right corner, where the exact solution

exhibits higher frequency. This is consistent with the intuition of assigning smaller partitions to more complex regions. Using the $N = 16$ partition, combined with the RAD resampling method that allocates more collocation points to regions with high residuals, these two strategies work in concert to improve the predictive accuracy of the FBPINN.

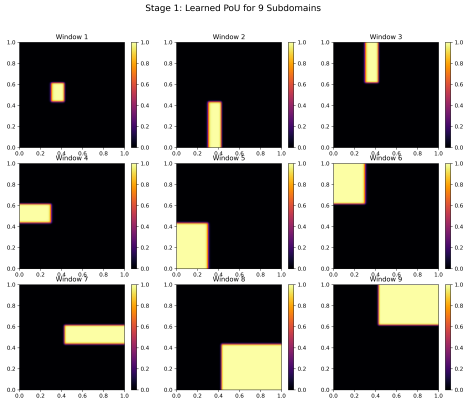
4.3.4. Adaptive DD FBPINN with Sep-MLP-POUnet

The Sep-MLP-POUnet used in the adaptive DD FBPINN is a separable model where each per-dimension branch is a two layers MLP with eight neurons per layer; training uses the Adam optimizer with a fixed learning rate of 1×10^{-4} for $10{,}000$ epochs.

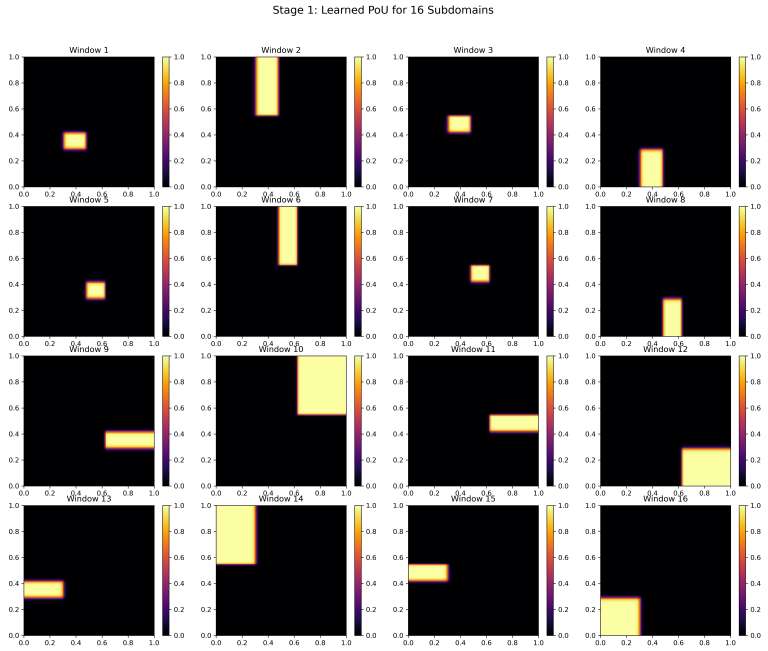
The Sep-MLP-POUnet successfully learns partitions for $N = 4$, $N = 9$ and $N = 16$. A collapse occurs at $N = 25$, so we use the last successful partitions at $N = 16$. The learned partitions are shown in Figure 4.21. The FBPINN prediction based on the learned 16 partitions is shown in Figure 4.22.



(a) $N = 4$



(b) $N = 9$



(c) $N = 16$

Figure 4.21: Sep-MLP-POUnet: Learned partitions for (a) $N = 4$, (b) $N = 9$, and (c) $N = 16$ subdomains. A collapse is detected at $N = 25$.

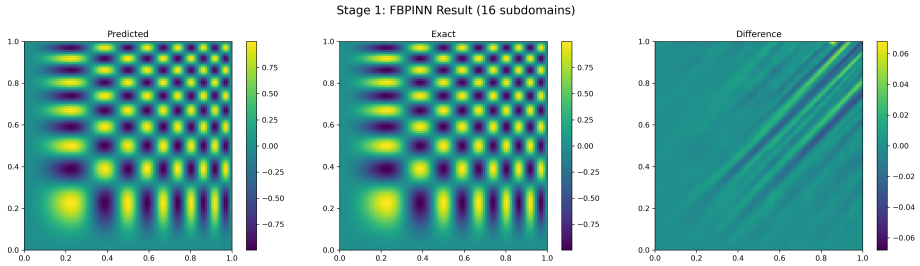


Figure 4.22: Sep-MLP-POUnet: (left) FBPINN prediction based on the learned partitions for $N = 16$.(middle)ground truth; (right) plot of difference.

Then we continue to train to FBPINN based on the learned partitions for $N = 16$. The prediction is shown in Figure 4.22. The prediction is not satisfactory, with a significant error of approximately 0.2 along the diagonal.

4.3.5. Comparison of the Adaptive DD FBPINN with 4 different POUnets

Method	Relative L_2 error	RMSE
Adaptive DD FBPINN (Sep-MLP-POUnet)		
PINN (initial)	5.24×10^{-1}	2.39×10^{-1}
FBPINN (16 partitions)	1.72×10^{-2}	9.88×10^{-3}
Adaptive DD FBPINN (Sep-RBF-POUnet)		
PINN (initial)	5.24×10^{-1}	2.39×10^{-1}
FBPINN (16 partitions)	$\mathbf{8.59 \times 10^{-3}}$	$\mathbf{3.91 \times 10^{-3}}$
Adaptive DD FBPINN (MLP-POUnet)		
PINN (initial)	5.24×10^{-1}	2.39×10^{-1}
FBPINN (4 partitions)	4.20×10^{-2}	1.91×10^{-2}
Adaptive DD FBPINN (RBF-POUnet)		
PINN (initial)	5.24×10^{-1}	2.39×10^{-1}
FBPINN (49 partitions)	2.60×10^{-1}	1.19×10^{-1}
PINN (5 layers, 128 neurons)	2.01×10^{-1}	1.12×10^{-1}
FBPINN (uniform domain decomposition 10×10, overlap=0.06)	2.64×10^{-1}	9.15×10^{-2}
FBPINN (uniform domain decomposition 4×4, overlap=0.2)	3.46×10^{-1}	1.58×10^{-1}

Table 4.3: Comparison of the relative L_2 error and RMSE for PINN and FBPINN with different numbers of partitions. The first three rows correspond to different stages of Adaptive DD FBPINN. All models are trained for 30000 steps. Each FBPINN subnetwork uses a 2-layer, 64-neuron fully connected architecture. The PINN uses a 5-layer, 128-neuron fully connected architecture.

The performance of different Adaptive DD FBPINN strategies and baseline models is summarized in Table 4.3. All adaptive methods start from an initial single-domain FBPINN (equivalent to a PINN) with a relative L_2 error of 5.24×10^{-1}.

The Adaptive DD FBPINN with Sep-RBF-POUnet achieves the best performance, reducing the relative L_2 error to 8.59×10^{-3} with 16 learned partitions. The separable Sep-MLP-POUnet also shows strong performance, with the error decreasing progressively as the number of partitions increases from 9 to 16.

In contrast, the non-separable POUnet architectures are less effective. The standard RBF-POUnet only

offers a marginal improvement over the baseline, while the MLP-POUnet, despite showing promise with 4 partitions, fails to scale to a higher number of subdomains.

Notably, the top-performing Adaptive DD FBPINN with Sep-RBF-POUnet significantly outperforms both the baseline PINN and FBPINN with uniform domain decomposition. This highlights the effectiveness of using learned, separable partitions to guide the domain decomposition, leading to substantial gains in accuracy.

Remark:

We evaluated two non-separable and two separable POUnets. While the non-separable POUnets did not perform as well as their separable counterparts in our tests, they generated more irregular partitions that can potentially adapt better to complex function features. The separable POUnets, in contrast, produced more regular partitions, which proved more suitable for the FBPINNs training framework.

The success of the Adaptive DD FBPINN, particularly with the Sep-RBF-POUnet, stems from a two-pronged improvement. First, it assigns smaller subdomains to the most challenging regions of the function, helping the neural networks overcome spectral bias. Second, after the subdomains are defined more precisely, the adaptation of collocation points becomes crucial. Concentrating collocation points in areas with high residuals provides the necessary training collocation pionts for the neural networks on these smaller, more focused subdomains.

5

Application on Complex Geometries

In previous chapters, we defined the loss by constructing an ansatz solution. For regular domains such as rectangles, boundary constraints are easy to satisfy, but for complex-shaped domains, constructing the ansatz solution is more challenging. Based on the ADF generated for the hexagon-shaped domain in Equation 3.18, and using the Adaptive DD FBPINN, we can solve the Poisson problem on hexagon-shaped domains with zero Dirichlet boundary conditions as a motivating example.

We consider the Poisson equation

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned}$$

where $\Omega \subset \mathbb{R}^2$ is the interior of a regular hexagon of circumradius 1.

The corresponding ADF $\phi(x)$ was derived in Equation 3.18 and is used here directly.

With Dirichlet BC, the solution ansatz for this Poisson problem is formed as:

$$\hat{u}(\mathbf{x};\boldsymbol{\theta}) = \phi(\mathbf{x})\mathcal{NN}(\mathbf{x};\boldsymbol{\theta}), \tag{5.1}$$

For mixed Dirichlet and Neumann boundary conditions, the situation is more complicated. As suggested in [59], Neumann conditions can be imposed via the potential energy functional in the variational principle, as in the Deep Ritz method [15]. If Robin boundary conditions are also present, constructing the solution ansatz becomes even more complex. Therefore, in this chapter, we focus exclusively on Dirichlet boundary conditions to demonstrate the application of the Adaptive DD FBPINN on a hexagon-shaped domain.

We attempt to construct a complicated right-hand side $f(x, y)$:

$$f(x,y) = 100\left[\sin\big(\pi x\,(2x^2+4x+4)\big) + \sin\big(\pi y\,(2y^2+4y+4)\big)\right].$$

Reference solution:

The reference solution is computed on a uniform cartesian grid. We discretize $[-1,1]^2$ into a $G \times G$ grid with spacing $h = 2/(G-1)$ and use a mask function to select the interior grid indices

$$\Omega_h \;=\; \{(i,j) \,:\, (x_i, y_j) \in \Omega_{\text{hex}}\},$$

where Ω_{hex} denotes the regular hexagonal domain. Homogeneous Dirichlet boundary conditions are imposed by setting $u \equiv 0$ outside the hexagon. For each interior node $(i,j) \in \Omega_h$, we assemble the five-point finite-difference stencil for $-\Delta u = f$: For each interior node $(i,j) \in \Omega_h$, the five-point finite-difference stencil for $-\Delta u = f$ is

$$\frac{4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}}{h^2} = f_{i,j},$$

where neighbors that lie outside Ω_{hex} are treated as Dirichlet zeros. This yields a sparse linear system $A\mathbf{u} = \mathbf{b}$ over the unknowns $\{u_{i,j}\}_{(i,j)\in\Omega_h}$; solving it gives the reference solution $u^{\text{fdm}}_{i,j}$. In our work, we set $G = 201$.

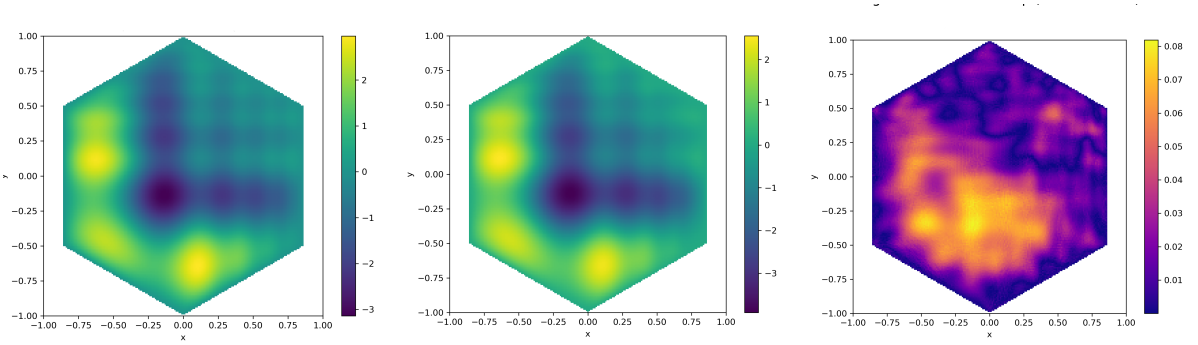
Following Algorithm 3, we first set the number of partitions starting from 1, then increase to 4, 9, 16, 25, 36 and so on. When the partition learned by POU-LSGD collapses, we go back to the last healthy partition and train a new FBPINN as target function. Then, we start from where we failed and training a FBPINN from the previously collapsed number of partitions. If it still collapses, training stops. If there is no collapse, we continue increasing the number of partitions and repeat the algorithm until the maximum value is reached.

Training parameters: For the hexagonal domain, we uniformly sample collocation points within the domain, drawing 6380 collocation points for training. Each subdomain network in the FBPINN is an MLP with two hidden layers of 16 neurons each. We train for 30,000 steps using the Adam optimizer with a learning rate of 10^{-3}. For testing, we use exactly the same grid nodes as those used to compute the FDM reference solution.

For the POUnet, we test two different architectures. The first is a standard MLP-POUnet, which consists of a 2-layer MLP with 32 neurons per layer. The second is a Sep-MLP-POUnet, where each dimension is processed by a separate 2-layer MLP with 16 neurons per layer. Both POUnet architectures are trained for 5,000 epochs with a learning rate of 1×10^{-3} for each partition configuration.

The adaptive training process is stopped if the maximum value of any partition of unity function drops below 0.8, which indicates a partition collapse. The RAD resampling strategy is applied with parameters $k = 1$ and $c = 1$, with resampling performed every 10000 steps.

The prediction of the initial FBPINN ($n = 1$) is shown in Figure 5.1b, and the absolute error with respect to the reference solution is shown in Figure 5.1c. Since we are using hard constraints, the Dirichlet boundary conditions are strictly satisfied on the boundary, so we only need to analyze the error in the interior. As can be seen from the figure, there is a relatively large error in the lower-left region of the hexagon, which matches the reference solution with high frequency in this region.



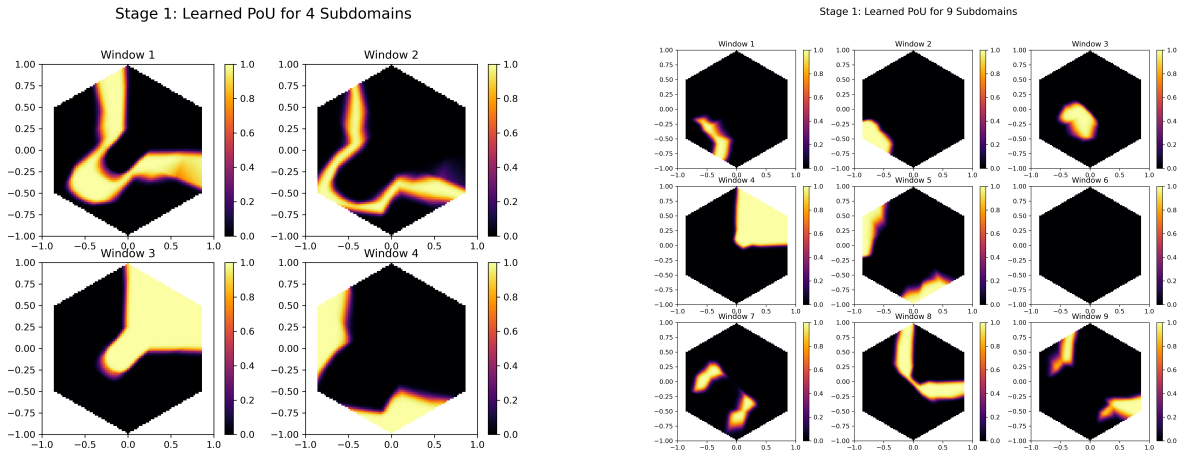
(a) Reference solution **(b)** FBPINN solution **(c)** Plot of absolute error

Figure 5.1: Initial FBPINN: (a) reference solution, (b) FBPINN prediction with 1 subdomain, and (c) absolute error.

Adaptive DD FBPINN with MLP-POUnet:

We first test the MLP-POUnet architecture: for each subdomain of the FBPINN, we use an MLP with 2 layers and 32 neurons per layer. The MLP-POUnet is trained for 5,000 epochs with a learning rate of 10^{-3}. The results are shown in Figure 5.2. When the number of partitions is 4, the POU function is successfully learned, as shown in Figure 5.2a. When the number of partitions is increased to 9, the POU function collapses, as shown in Figure 5.2b. The learned partitions exhibit irregular shapes. We then proceed to train the FBPINN based on these irregular partitions, and the final results are shown in Figure 5.3. As can be seen, the FBPINN solution is inaccurate, with a large absolute error, indicating that these irregularly shaped partitions are not suitable for the FBPINN.

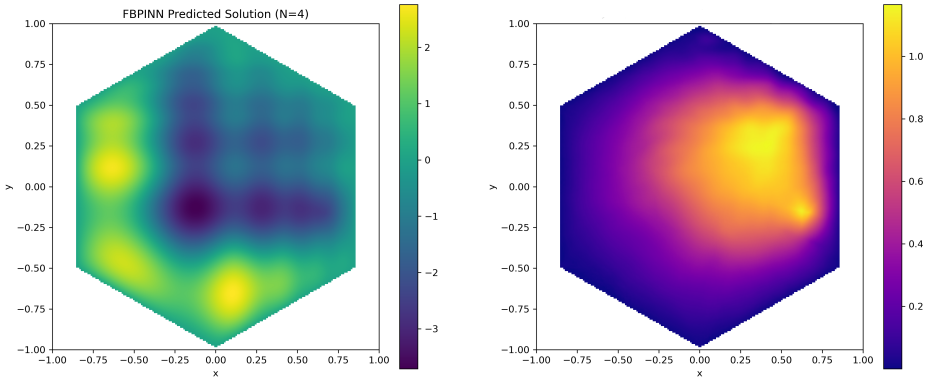
Adaptive DD FBPINN with Sep-MLP-POUnet:



(a) Learned 4 partitions

(b) Learned 9 partitions (collapsed)

Figure 5.2: MLP-POUnet: Learned POU functions with (a) 4 partitions and (b) 9 partitions (collapsed).



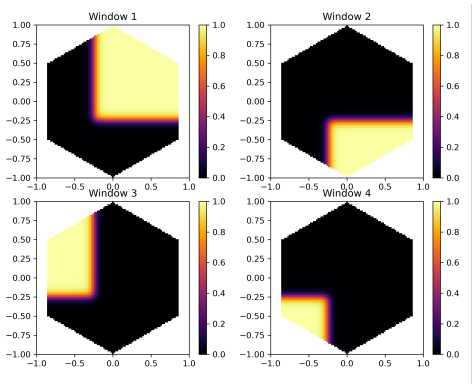
(a) FBPINN solution (4 partitions)

(b) Absolute error

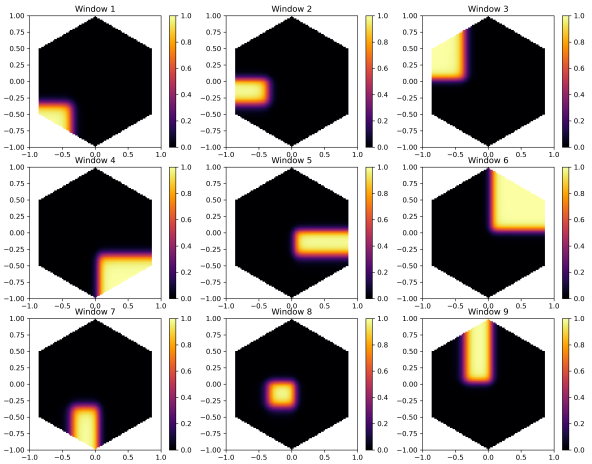
Figure 5.3: Computed FBPINN solution with 4 partitions (left) and absolute error (right).

Based on Figure 5.1b, for Sep-MLP-POUnet, increasing the number of partitions from 4 to 9 to 16. We observe that POUnet collapses when using 16 number of partitions. So we use the POU functions with 9 partitions and train the FBPINN based on that partitions; the results are shown in Figures 5.4a and 5.4b.

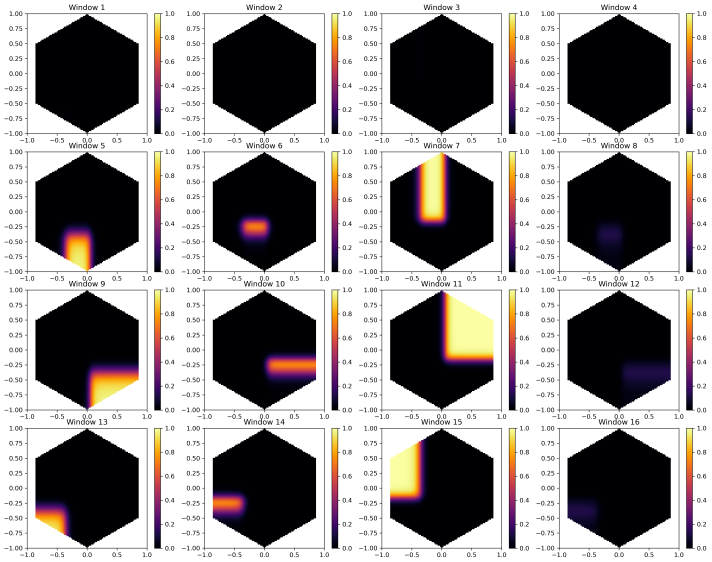
Based on the FBPINN results learned with 9 partitions, we continue POU-LSGD training with 16 partitions, but it still collapses. Training is terminated.



(a) Learned 4 partitions



(b) Learned 9 partitions

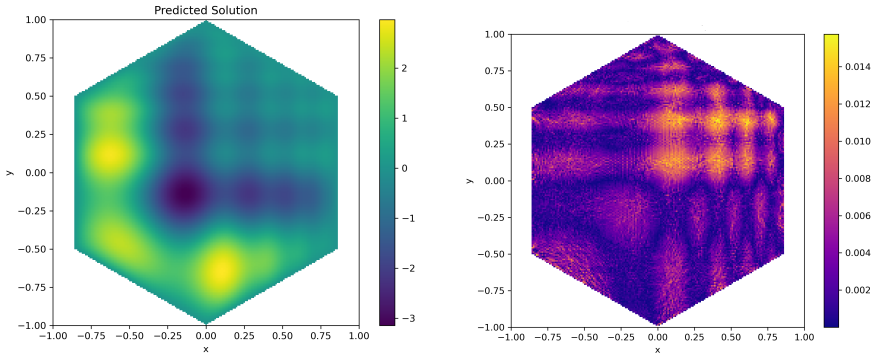


(c) Learned 16 partitions (collapsed)

Figure 5.4: Sep-MLP-POUnet: Learned POU functions with (a) 4 partitions, (b) 9 partitions, and (c) 16 partitions (collapse detected).

Method	Relative L2 error	RMSE
Adaptive DD FBPINN (Sep-MLP-POUnet)		
(Stage 0) PINN (initial)	2.74×10^{-2}	3.45×10^{-2}
(Stage 1) FBPINN (9 partitions)	$\mathbf{3.52 \times 10^{-3}}$	$\mathbf{4.44 \times 10^{-3}}$
Adaptive DD FBPINN (MLP-POUnet)		
(Stage 0) PINN (initial)	2.74×10^{-2}	3.45×10^{-2}
(Stage 1) FBPINN (4 partitions)	4.21×10^{-1}	5.30×10^{-1}
FBPINN (DD 3×3=9, overlap=0.2)	9.09×10^{-2}	1.15×10^{-1}

Table 5.1: Comparison of Adaptive DD FBPINN with Sep-MLP-POUnet, MLP-POUnet, and baseline FBPINN for the Poisson problem on a hexagonal domain.



(a) FBPINN solution (9 partitions)

(b) Absolute error

Figure 5.5: Computed FBPINN solution with 9 partitions (a) and absolute error (b).

As a baseline experiment, we evaluate the performance of the Adaptive DD FBPINN against the Fixed Uniform DD FBPINN for solving the Poisson equation on the hexagonal domain. The uniform DD is first performed on the bounding rectangle of the hexagon, and then only the parts within the hexagonal domain are selected.

Using the relative L_2 error and RMSE as the primary evaluation metrics, as shown in the Table 5.1, the results clearly demonstrate the superiority of the adaptive approach. Based on the results in Table 5.1, several key conclusions can be drawn. The Adaptive DD FBPINN using the MLP-POUnet architecture, when trained with 4 partitions, achieves a relative L_2 error of 4.21×10^{-1}, which is not only significantly worse than the Sep-MLP-POUnet, but also worse than the initial single PINN solution.

Among all the methods, the Sep-MLP-POUnet outperforms the others. With 9 partitions, the adaptive method achieves a relative L_2 error of 3.52×10^{-3}, which is much lower than the 9.09×10^{-2} error from the uniform 3×3 FBPINN. This represents an improvement of nearly an order of magnitude over the initial single PINN error of 2.74×10^{-2}, confirming the effectiveness of the adaptive process.

The choice of the POUnet architecture is critical for this success. While the Sep-MLP-POUnet facilitated a substantial error reduction, the standard MLP-POUnet failed, causing the error to increase dramatically to 9.59×10^{-1}. This underscores the necessity of a suitable POUnet structure for the adaptive decomposition to function effectively. In contrast, uniform partitioning proved to be counterproductive; the uniform FBPINN with 9 partitions performed worse than the initial single PINN, suggesting that an inappropriate or non-optimal decomposition can degrade accuracy rather than improve it.

As shown in Figure 5.5b, the error of the FBPINN prediction is concentrated in the top-right region, whereas after the first training of the PINN, the error was located in the bottom-left region. This indicates that the learned 4 partitions helped improve accuracy in the bottom-left area by using smaller partitions within the domain.

In summary, this comparative study provides strong evidence that for problems on complex shaped domains like hexagon, the adaptive DD FBPINN, by using a well-suited POUnet (like Sep-MLP-POUnet or Sep-RBF-POUnet) to guide domain decomposition, achieves higher accuracy compared to the traditional FBPINN with a fixed uniform decomposition. This highlights the immense potential of adaptive domain decomposition strategies in scientific computing.

Remarks

For the reference solution, we compute a FDM solution on a uniform Cartesian grid for convenience. While FDM is straightforward and efficient for regular domains, its reliance on structured grids makes it less suitable for complex geometries. Therefore, in future work, we plan to explore alternative methods better suited to irregular domains, such as the finite element method (FEM).

In this study, we focus on Dirichlet boundary conditions. Extending the framework to handle Neumann or Robin boundary conditions will require additional development and is left for future research. Since

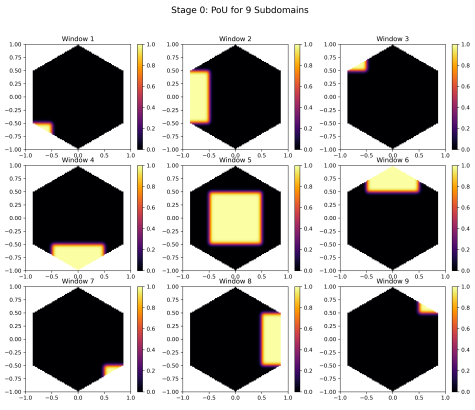


Figure 5.6: 3x3 Uniform Domain Decomposition

our goal is to demonstrate the effectiveness of the adaptive DD FBPINN on complex geometries, Dirichlet conditions serve as a clear and appropriate illustrative case.

As observed in the original FBPINN paper [43], simply increasing the number of partitions does not necessarily lead to improved accuracy. We observe the same phenomenon in our Adaptive DD FBPINN experiments, where the error does not consistently decrease as the number of partitions increases. This suggests that more principled adaptive strategies are required to achieve further improvements.

When considering domain decomposition, non-separable POUnets should not be dismissed. While the Sep-MLP-POUnet, which utilizes a mask function for clipping, performs well on complex-shaped domains like hexagons, it may not be optimal for more intricate or non-convex domains. Non-separable POUnets, despite generating irregular partitions and potentially providing less accurate predictions, are more adaptable to complex-shaped domains. As a result, they contribute to the flexibility and effectiveness of the Adaptive DD FBPINN, enabling better handling of diverse geometries.

6

Discussion and Conclusion

6.1. Discussion and Conclusion

This work investigates the role of adaptive partitions within the FBPINN framework and validates the ability of data-driven POUnets to learn partitions for 1D and 2D problems. We introduce separable POUnets (Sep-POUnets) that, for 2D domains, learn coordinate-wise partitions, yielding more stable and geometrically interpretable data-driven decompositions. By combining data-driven POUnets with FBPINNs, we present an Adaptive Domain Decomposition FBPINN (Adaptive DD FBPINN) framework for solving PDEs. This method applies not only to rectangular domains but also generalizes to complex-shaped domains. Several conclusions can be drawn from the experiments:

1. Compared with FBPINNs using uniform domain decomposition, feature-aligned domain decomposition achieves better performance, especially with the RAD resampling strategy. This underscores the significant impact of partition strategies on accuracy in FBPINNs. Moreover, using smaller partitions together with collocation points resampling method further improves accuracy by focusing learning on challenging regions.
2. In purely data-driven experiments using POUnets, the learned partitions adapt more effectively to problem characteristics (e.g., local frequency). Compared with non-separable POUnets (RBF-POUnets and MLP-POUnets), our new Sep-POUnet architecture better learns geometrically interpretable partitions.
3. By combining FBPINN and POUnets, the Adaptive DD FBPINN framework improves PDE-solution accuracy via partition updates. Partitions learned by Sep-POUnets are particularly effective, adapting better to problem characteristics and thus yielding more accurate solutions.
4. Based on approximate distance functions (ADF), for problems with Dirichlet boundary conditions, the Adaptive DD FBPINN can be readily applied to complex-shaped domains such as hexagons. The method adaptively learns partitions that align with the solution's features, improving accuracy.

Several key findings and observations arose from the experimental results:

- **Irregular but Usable Partitions:** Although POUnet-learned partitions can be irregular, they still provide effective guidance for the FBPINN framework. Strict geometric regularity is therefore not required, provided the POU constraint is met and sufficient overlap is maintained. For complex-shaped domains, non-separable POUnets can be more flexible because they do not rely on geometry-aligned clipping.
- **Importance of Partition Overlap:** A recurring challenge is that the learned partitions sometimes yield narrow or insufficient overlaps. This limits information exchange between subdomains and can cause discontinuities or inconsistencies in the global solution. Enhancing overlap quality through explicit regularization is thus an important direction for future research.
- **Performance of POUnet Architectures:** Among the tested architectures, Sep-RBF-POUnet achieved the best performance, consistently producing partitions aligned with salient solution

features and delivering markedly improved accuracy. In contrast, non-separable POUnets such as MLP-POUnet tended to struggle as problem complexity increased.

- **Generalizability to Complex Geometries:** The approach produced promising results on irregular domains such as hexagons, demonstrating the scalability and flexibility of Adaptive DD FBPINN for problems with complex boundary conditions.

Limitations and Challenges:

- **Retraining and Efficiency:** Retraining the FBPINN each time a new partition is accepted can be costly, especially when the number of partitions grows. This can reduce training efficiency, particularly for large or complex domains.
- **Non-Monotonic Behavior with Increasing Partitions:** Simply increasing the number of partitions does not guarantee improved accuracy. Over-fragmentation or insufficient overlap can raise errors. Balancing partition count and overlap is therefore crucial for both accuracy and efficiency.
- **Tension Between POUnet and FBPINN Training:** In the Adaptive DD FBPINN framework, the objectives of POUnet training and FBPINN optimization can diverge. For example, POUnets may produce partitions that yield good local approximation yet are suboptimal for FBPINN training with insufficient overlap. A careful balance between POUnet approximation quality and FBPINN training efficiency is needed.

6.2. Future Work

Future research may focus on the following directions:

- **End-to-End Co-Optimization:** A fully integrated approach that co-optimizes POUnet and FBPINN in a unified training loop could eliminate repeated retraining after each partition update. Starting from an initial decomposition and adaptively adjusting partitions during training may yield more efficient training and better convergence.
- **Overlap-Aware Regularization:** Introducing regularizers that explicitly promote adequate and well-distributed overlaps in POUnet partitions could mitigate cases where narrow overlaps arise from overly compact partitions.
- **Generalization to Complex Boundary Conditions:** Extending the Adaptive DD FBPINN framework to PDEs with mixed, Neumann, or Robin boundary conditions would broaden its applicability.
- **Residual-Aware Resampling:** Beyond computing residuals over all collocation points, dynamically reallocating points according to the evolving learned partitions may further improve training efficiency and convergence.

In conclusion, the Adaptive DD FBPINN method shows strong potential for solving complex PDEs, particularly on domains with intricate geometry. Data-driven, dynamically adapted partitions enhance both the flexibility and the accuracy of the FBPINN framework, making it a promising avenue for future research and an exploratory contribution to applying deep learning in scientific computing.

Disclaimer

Generative AI tools—specifically OpenAI's ChatGPT [44]—were used for grammar correction and LaTeX formula formatting. No AI was involved in idea development, data analysis, or result generation. All content was produced solely by the author and reviewed to ensure academic integrity.

References

[1] Babak Alipanahi et al. “Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning”. In: *Nature Biotechnology* 33.8 (2015), pp. 831–838. doi: 10.1038/nbt.3300.

[2] William F. Ames. *Numerical Methods for Partial Differential Equations*. 2nd. Computer Science and Applied Mathematics. Amsterdam: Academic Press, 2014, p. 380. isbn: 9781483262420.

[3] Nathan Baker et al. *Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence*. Tech. rep. USDOE Office of Science (SC), Washington, D.C. (United States), Feb. 2019. doi: 10.2172/1478744. url: https://www.osti.gov/biblio/1478744.

[4] Steve A. Billings and Guang L. Zheng. “Radial basis function network configuration using genetic algorithms”. In: *Neural Networks* 8.6 (1995), pp. 877–890. issn: 0893-6080. doi: https://doi.org/10.1016/0893-6080(95)00029-Y. url: https://www.sciencedirect.com/science/article/pii/089360809500029Y.

[5] Rafael Bischof and Michael A. Kraus. “Mixture-of-Experts-Ensemble Meta-Learning for Physics-Informed Neural Networks”. In: *Proceedings of the 33rd Forum Bauinformatik*. Technical University of Munich, Sept. 2022. url: https://mediatum.ub.tum.de/doc/1688403/uic8b0xn1c845e7rac1or092o.Bischof%20et%20Al.%202022.pdf.

[6] Arpan Biswas and Vadim Shapiro. “Approximate distance fields with non-vanishing gradients”. In: *Graphical Models* 66.3 (May 2004), pp. 133–159. doi: 10.1016/j.gmod.2004.01.003.

[7] David Broomhead and David Lowe. “Radial basis functions, multi-variable functional interpolation and adaptive networks”. In: *ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM)* RSRE-MEMO-4148 (Mar. 1988).

[8] S. Cai et al. “Physics-informed neural networks (pinns) for fluid mechanics: a review”. In: *Acta Mechanica Sinica* 37 (12 2021), pp. 1727–1738. doi: 10.1007/s10409-021-01148-1.

[9] Xiao-Chuan Cai and Marcus Sarkis. “A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems”. In: *SIAM Journal on Scientific Computing* 21.2 (1999), pp. 792–797. doi: 10.1137/S106482759732678X. eprint: https://doi.org/10.1137/S106482759732678X. url: https://doi.org/10.1137/S106482759732678X.

[10] Tony F. Chan and Tarek P. Mathew. “Domain decomposition algorithms”. In: *Acta Numerica* 3 (1994), pp. 61–143. doi: 10.1017/s0962492900002427.

[11] Francisco Sahli Costabal, Simone Pezzuto, and Paris Perdikaris. “Δ-PINNs: physics-informed neural networks on complex geometries”. In: (2022). arXiv: 2209.03984 [cs.LG]. url: https://arxiv.org/abs/2209.03984.

[12] Eric C. Cyr et al. “Robust Training and Initialization of Deep Neural Networks: An Adaptive Basis Viewpoint”. In: (2019). arXiv: 1912.04862 [cs.LG]. url: https://arxiv.org/abs/1912.04862.

[13] Clark R. Dohrmann. “A Preconditioner for Substructuring Based on Constrained Energy Minimization”. In: *SIAM Journal on Scientific Computing* 25.1 (2003), pp. 246–258. doi: 10.1137/S1064827502412887. eprint: https://doi.org/10.1137/S1064827502412887. url: https://doi.org/10.1137/S1064827502412887.

[14] Victorita Dolean et al. “Multilevel domain decomposition-based architectures for physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 429 (Sept. 2024), p. 117116. issn: 0045-7825. doi: 10.1016/j.cma.2024.117116. url: http://dx.doi.org/10.1016/j.cma.2024.117116.

[15] Weinan E and Bing Yu. “The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems”. In: (2017). arXiv: 1710.00211 [cs.LG]. url: https://arxiv.org/abs/1710.00211.

[16] Charbel Farhat and Francois-Xavier Roux. “A method of finite element tearing and interconnecting and its parallel solution algorithm”. In: *International Journal for Numerical Methods in Engineering* 32.6 (1991), pp. 1205–1227. doi: 10.1002/nme.1620320604.

[17] Charbel Farhat et al. “FETI-DP: a dual–primal unified FETI method—part i: a faster alternative to the two-level FETI method”. In: *International Journal for Numerical Methods in Engineering* 50 (Mar. 2001), pp. 1523–1544. doi: 10.1002/nme.76.

[18] 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*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. url: https://proceedings.mlr.press/v9/glorot10a.html.

[19] Alexander Heinlein et al. “Combining Machine Learning and Adaptive Coarse Spaces—A Hybrid Approach for Robust FETI-DP Methods in Three Dimensions”. In: *SIAM Journal on Scientific Computing* 43.5 (2021), S816–S838. doi: 10.1137/20M1344913. eprint: https://doi.org/10.1137/20M1344913. url: https://doi.org/10.1137/20M1344913.

[20] Alexander Heinlein et al. “Combining machine learning and domain decomposition methods for the solution of partial differential equations—A review”. In: *GAMM-Mitteilungen* 44.1 (2021). doi: 10.1002/gamm.202100001.

[21] Amanda A. Howard et al. “Finite basis Kolmogorov-Arnold networks: domain decomposition for data-driven and physics-informed problems”. In: (2024). arXiv: 2406.19662 [cs.LG]. url: https://arxiv.org/abs/2406.19662.

[22] Wulong Hu et al. “Efficient deep reinforcement learning strategies for active flow control based on physics-informed neural networks”. In: *Physics of Fluids* 36.7 (2024). doi: 10.1063/5.0213256.

[23] Zheyuan Hu et al. “Augmented Physics-Informed Neural Networks (APINNs): A gating network-based soft domain decomposition methodology”. In: *Engineering Applications of Artificial Intelligence* 126 (2023), p. 107183. issn: 0952-1976. doi: https://doi.org/10.1016/j.engappai.2023.107183. url: https://www.sciencedirect.com/science/article/pii/S0952197623013672.

[24] Bin Huang and Jianhui Wang. “Applications of Physics-Informed Neural Networks in Power Systems - A Review”. In: *IEEE Transactions on Power Systems* 38.1 (2023), pp. 572–588. doi: 10.1109/TPWRS.2022.3162473.

[25] Ameya D Jagtap and George Em Karniadakis. “Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations”. In: *Communications in Computational Physics* 28.5 (2020), pp. 2002–2041.

[26] Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. “Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems”. In: *Computer Methods in Applied Mechanics and Engineering* 365 (2020), p. 113028. issn: 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113028. url: https://www.sciencedirect.com/science/article/pii/S0045782520302127.

[27] Xiaowei Jin et al. “NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations”. In: *Journal of Computational Physics* 426 (Feb. 2021), p. 109951. issn: 0021-9991. doi: 10.1016/j.jcp.2020.109951. url: http://dx.doi.org/10.1016/j.jcp.2020.109951.

[28] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. *Variational Physics-Informed Neural Networks For Solving Partial Differential Equations*. 2019. arXiv: 1912.00873 [cs.NE]. url: https://arxiv.org/abs/1912.00873.

[29] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (2017). arXiv: 1412.6980 [cs.LG]. url: https://arxiv.org/abs/1412.6980.

[30] Axel Klawonn, Martin Lanser, and Janine Weber. “Machine learning and domain decomposition methods – a survey”. In: (2023). arXiv: 2312.14050 [math.NA]. url: https://arxiv.org/abs/2312.14050.

[31] Aditi S. Krishnapriyan et al. “Characterizing possible failure modes in physics-informed neural networks”. In: (2021). arXiv: 2109.01050 [cs.LG]. url: https://arxiv.org/abs/2109.01050.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. doi: 10.1145/3065386.

[33] I.E. Lagaris, A. Likas, and D.I. Fotiadis. “Artificial neural networks for solving ordinary and partial differential equations”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. issn: 1045-9227. doi: 10.1109/72.712178. url: http://dx.doi.org/10.1109/72.712178.

[34] Isaac Lagaris, Aristidis Likas, and Dimitrios Papageorgiou. “Neural-network methods for boundary value problems with irregular boundaries”. In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 11 (Feb. 2000), pp. 1041–9. doi: 10.1109/72.870037.

[35] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (2015), pp. 1332–1338. doi: 10.1126/science.aab3050.

[36] Kookjin Lee et al. “Partition of unity networks: deep hp-approximation”. In: (2021). arXiv: 2101.11256 [cs.LG]. url: https://arxiv.org/abs/2101.11256.

[37] Hang Li. “Deep learning for natural language processing: advantages and challenges”. In: *National Science Review* 5.1 (2017), pp. 24–26. doi: 10.1093/nsr/nwx110.

[38] Wuyang Li, Xueshuang Xiang, and Yingxiang Xu. “Deep Domain Decomposition Method: Elliptic Problems”. In: (2020). arXiv: 2004.04884 [math.NA]. url: https://arxiv.org/abs/2004.04884.

[39] Jan Mandel and C Dohrmann. “Convergence of a balancing domain decomposition by constraints and energy minimization”. In: *Numerical Linear Algebra with Applications* 10 (Dec. 2002), pp. 639–659.

[40] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. “Physics-informed neural networks for high-speed flows”. In: *Computer Methods in Applied Mechanics and Engineering* 360 (2020), p. 112789. doi: 10.1016/j.cma.2019.112789.

[41] Levi D. McClenny and Ulisses M. Braga-Neto. “Self-adaptive physics-informed neural networks”. In: *Journal of Computational Physics* 474 (2023), p. 111722. doi: 10.1016/j.jcp.2022.111722.

[42] Daniel Millán, N. Sukumar, and Marino Arroyo. “Cell-based maximum-entropy approximants”. In: *Computer Methods in Applied Mechanics and Engineering* 284 (2015). Isogeometric Analysis Special Issue, pp. 712–731. issn: 0045-7825. doi: https://doi.org/10.1016/j.cma.2014.10.012. url: https://www.sciencedirect.com/science/article/pii/S004578251400382X.

[43] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. “Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations”. In: *Advances in Computational Mathematics* 49.4 (July 2023). issn: 1572-9044. doi: 10.1007/s10444-023-10065-9. url: http://dx.doi.org/10.1007/s10444-023-10065-9.

[44] OpenAI. *ChatGPT*. https://openai.com/chatgpt/. Accessed: 2025-05-17. 2023.

[45] Guofei Pang et al. “nPINNs: nonlocal Physics-Informed Neural Networks for a parametrized nonlocal universal Laplacian operator. Algorithms and Applications”. In: (2020). arXiv: 2004.04276 [math.AP]. url: https://arxiv.org/abs/2004.04276.

[46] Dimitris C. Psichogios and Lyle H. Ungar. “A hybrid neural network-first principles approach to process modeling”. In: *Aiche Journal* 38 (1992), pp. 1499–1511. url: https://api.semanticscholar.org/CorpusID:15491561.

[47] Nasim Rahaman et al. “On the Spectral Bias of Neural Networks”. In: (2019). arXiv: 1806.08734 [stat.ML]. url: https://arxiv.org/abs/1806.08734.

[48] M. Raissi, P. Perdikaris, and G.E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. issn: 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. url: https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[49] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations”. In: (2017). arXiv: 1711.10561 [cs.AI]. url: https://arxiv.org/abs/1711.10561.

[50] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations”. In: (2017). arXiv: 1711.10566 [cs.AI]. url: https://arxiv.org/abs/1711.10566.

[51] M. Rasht-Behesht et al. “Physics-informed neural networks (pinns) for wave propagation and full waveform inversions”. In: *Journal of Geophysical Research: Solid Earth* 127 (5 2022). doi: 10.1029/2021jb023120.

[52] Google Research. *JAX: Autograd and XLA*. GitHub repository. 2018. url: https://github.com/google/jax.

[53] Arturo Rodriguez et al. “Partition of Unity Physics-Informed Neural Networks (POU-PINNs): An Unsupervised Framework for Physics-Informed Domain Decomposition and Mixtures of Experts”. In: (2024). arXiv: 2412.06842 [cs.LG]. url: https://arxiv.org/abs/2412.06842.

[54] V.L. Rvachev et al. “Transfinite interpolation over implicitly defined sets”. In: *Computer Aided Geometric Design* 18.3 (2001), pp. 195–220. issn: 0167-8396. doi: https://doi.org/10.1016/S0167-8396(01)00015-2. url: https://www.sciencedirect.com/science/article/pii/S0167839601000152.

[55] Vadim Shapiro. *Theory of R-functions and Applications: A Primer*. Technical Report CPA88-3. Tech. Rep. CPA88-3. Ithaca, NY 14853, USA: Cornell Programmable Automation, Sibley School of Mechanical Engineering, 1991.

[56] Vadim Shapiro and Igor Tsukanov. “Implicit functions with guaranteed differential properties”. In: *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*. SMA ’99. Ann Arbor, Michigan, USA: Association for Computing Machinery, 1999, pp. 258–269. isbn: 1581130805. doi: 10.1145/304012.304038. url: https://doi.org/10.1145/304012.304038.

[57] Jiahao Song et al. “VW-PINNs: A volume weighting method for PDE residuals in physics-informed neural networks”. In: (2024). arXiv: 2401.06196 [cs.CE]. url: https://arxiv.org/abs/2401.06196.

[58] Patrick Stiller et al. “Large-scale Neural Solvers for Partial Differential Equations”. In: (2020). arXiv: 2009.03730 [cs.LG]. url: https://arxiv.org/abs/2009.03730.

[59] N. Sukumar and Ankit Srivastava. “Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 389 (Feb. 2022), p. 114333. issn: 0045-7825. doi: 10.1016/j.cma.2021.114333. url: http://dx.doi.org/10.1016/j.cma.2021.114333.

[60] Matthew Tancik et al. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 7537–7547. url: https://proceedings.neurips.cc/paper_files/paper/2020/file/55053683268957697aa39fba6f231c68-Paper.pdf.

[61] Nat Trask et al. “Probabilistic partition of unity networks: clustering based deep approximation”. In: (2021). arXiv: 2107.03066 [cs.LG]. url: https://arxiv.org/abs/2107.03066.

[62] Nathaniel Trask et al. “Hierarchical partition of unity networks: fast multilevel training”. In: *Proceedings of Mathematical and Scientific Machine Learning*. Ed. by Bin Dong et al. Vol. 190. Proceedings of Machine Learning Research. PMLR, Aug. 2022, pp. 271–286. url: https://proceedings.mlr.press/v190/trask22a.html.

[63] Sifan Wang, Yujun Teng, and Paris Perdikaris. “Understanding and mitigating gradient pathologies in physics-informed neural networks”. In: (2020). arXiv: 2001.04536 [cs.LG]. url: https://arxiv.org/abs/2001.04536.

[64] Sifan Wang, Hanwen Wang, and Paris Perdikaris. “On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 384 (Oct. 2021), p. 113938. issn: 0045-7825. doi: 10.1016/j.cma.2021.113938. url: http://dx.doi.org/10.1016/j.cma.2021.113938.

[65] Jared Willard et al. “Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems”. In: (2022). arXiv: 2003.04919 [physics.comp-ph]. url: https://arxiv.org/abs/2003.04919.

[66] Chenxi Wu et al. “A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks”. In: *Computer Methods in Applied Mechanics and Engineering* 403 (2023), p. 115671.

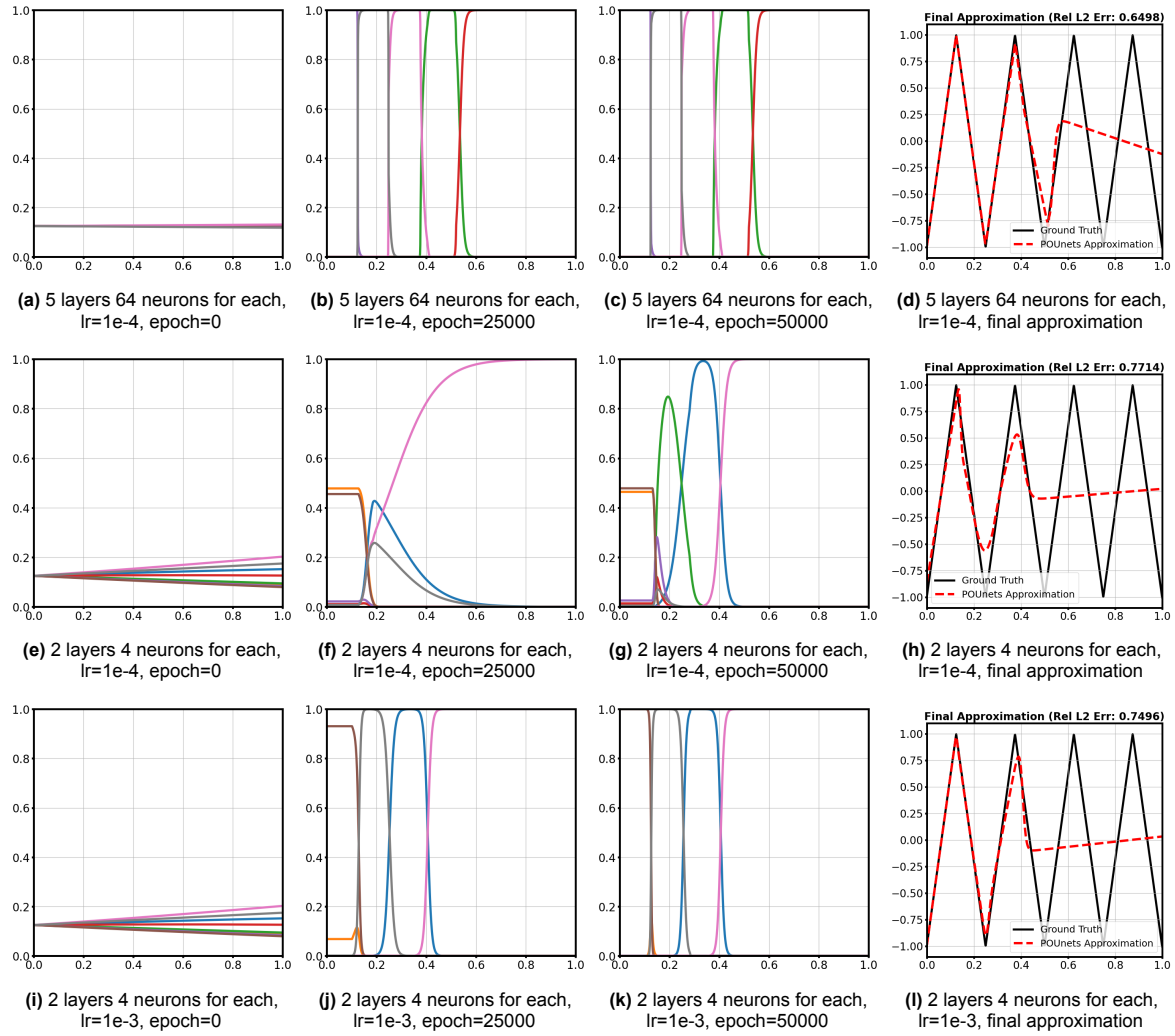
[67] Ziqi Liu Ziqi Liu, Wei Cai Wei Cai, and Zhi-Qin John Xu Zhi-Qin John Xu. “Multi-Scale Deep Neural Network (MscaleDNN) for Solving Poisson-Boltzmann Equation in Complex Domains”. In: *Communications in Computational Physics* 28.5 (Jan. 2020), pp. 1970–2001. issn: 1815-2406. doi: 10.4208/cicp.oa-2020-0179. url: http://dx.doi.org/10.4208/cicp.OA-2020-0179.

[68] Naturforschende Gesellschaft in Zürich. “Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich”. In: vol. v.15 (1870). https://www.biodiversitylibrary.org/bibliography/8297. Zürich Fäsi & Beer, 1870, pp. 272–286. url: https://www.biodiversitylibrary.org/page/8379779.

A

Additional Results for POUnets

A.1. MLP-POUnets on 1D triangular wave function



(a) 5 layers 64 neurons for each, lr=1e-4, epoch=0 **(b)** 5 layers 64 neurons for each, lr=1e-4, epoch=25000 **(c)** 5 layers 64 neurons for each, lr=1e-4, epoch=50000 **(d)** 5 layers 64 neurons for each, lr=1e-4, final approximation

(e) 2 layers 4 neurons for each, lr=1e-4, epoch=0 **(f)** 2 layers 4 neurons for each, lr=1e-4, epoch=25000 **(g)** 2 layers 4 neurons for each, lr=1e-4, epoch=50000 **(h)** 2 layers 4 neurons for each, lr=1e-4, final approximation

(i) 2 layers 4 neurons for each, lr=1e-3, epoch=0 **(j)** 2 layers 4 neurons for each, lr=1e-3, epoch=25000 **(k)** 2 layers 4 neurons for each, lr=1e-3, epoch=50000 **(l)** 2 layers 4 neurons for each, lr=1e-3, final approximation

Figure A.1: Training progression of RBF-POUnet on the triangular wave function for $p = 4$ and training for 50000 epochs. The first three columns show the evolution of the learned partitions over 50000 epochs. The final column displays the resulting function approximation.(First row): 5 layers 64 neurons for each layer, learning rate: $1\mathrm{e}{-4}$. (Second row): 2 layers 4 neurons for each layer, learning rate: $1\mathrm{e}{-4}$. (Third row): 2 layers 4 neurons for each layer, learning rate: $1\mathrm{e}{-4}$