Literature Review Internship on Broad Learning System, Physics-Informed Extreme Learning Machine, and Bayesian Physics-Informed Extreme Learning Machine

A Report Submitted in Partial Fulfillment

of the Requirements for Summer Internship

by

Megha P

(Roll No. IMS22165)

SCHOOL OF DATA SCIENCE

IISER THIRUVANANTHAPURAM

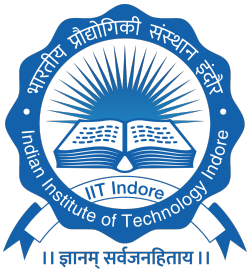
Under the Supervision of

Prof. M. Tanveer

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY, INDORE

July 2025



DECLARATION

I, **Megha P** (Roll No: **IMS22165**), hereby declare that this report entitled **"Literature Review Internship on Broad Learning System, Physics-Informed Extreme Learning Machine, and Bayesian Physics-Informed Extreme Learning Machine"**, submitted to the **Department of Mathematics**, Indian Institute of Technology Indore, is a faithful account of the reading internship I undertook under the supervision of **Prof. M. Tanveer**.

This internship involved a detailed study of existing literature on the **Broad Learning System**, **Physics-Informed Extreme Learning Machine (PIELM)**, and **Bayesian Physics-Informed Extreme Learning Machine (BPIELM)**. The work presented in this report is based solely on the analysis and understanding of previously published research and does not contain any original model proposals by me.

I affirm that I have maintained academic integrity throughout this work. All external sources of information, results, or statements have been duly acknowledged and properly cited.

IIT Indore **Megha P**

July 2025

CERTIFICATE

This is to certify that **Megha P** (Roll No: **IMS22165**), a student of the **School of Data Science** at the **Indian Institute of Science Education and Research Thiruvananthapuram**, has successfully completed a Summer Internship titled **Literature Review Internship on Broad Learning System, Physics-Informed Extreme Learning Machine, and Bayesian Physics-Informed Extreme Learning Machine**, under my supervision at the **Department of Mathematics, Indian Institute of Technology Indore**, from **22 May 2025 to 5 July 2025**.

During the internship, the student demonstrated sincerity and enthusiasm for learning. The work involved reviewing and analyzing machine learning models, particularly the **Broad Learning System (BLS)**, **Physics-Informed Extreme Learning Machine (PIELM)**, and **Bayesian Physics-Informed Extreme Learning Machine (BPIELM)**. The outcomes of this study are presented in the attached report.

IIT Indore
July 2025

Prof. M. Tanveer
Project Supervisor

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Prof. M. Tanveer**, Department of Mathematics, Indian Institute of Technology Indore, for providing me with the opportunity to undertake this reading internship during the Summer of 2025. I am highly indebted to him for his constant support, insightful guidance, and invaluable suggestions throughout the course of this work.

I also extend my heartfelt thanks to the **Department of Mathematics**, IIT Indore, for facilitating an intellectually stimulating environment that allowed me to explore and understand advanced machine learning methodologies such as the **Broad Learning System**, **Physics-Informed Extreme Learning Machine (PIELM)**, and **Bayesian Physics-Informed Extreme Learning Machine (BPIELM)**.

I am grateful to my peers, friends, and well-wishers for their moral support and encouragement during this internship. Their thoughtful discussions and consistent motivation helped me stay focused and committed to this academic endeavor.

Lastly, I would like to thank my family for their unwavering belief in my abilities, and for their emotional support, which was indispensable throughout this journey.

IIT Indore **Megha P**

July 2025

ABSTRACT

This reading internship was centered on an in-depth exploration of three advanced machine learning frameworks : Broad Learning System (BLS), Physics-Informed Extreme Learning Machine (PIELM), and Bayesian Physics-Informed Extreme Learning Machine (BPIELM). The objective was to understand their theoretical underpinnings, training methodologies, and application to solving partial differential equations.

The internship involved a structured literature review, critical analysis of key concepts, and documentation of comparative insights. The report encapsulates academic engagement with these models and highlights important developments in the field of data-driven and physics-informed learning.

Keywords:

Broad Learning System, BLS, PIELM, BPIELM, Physics-Informed Learning, Machine Learning, PDE Solvers

Contents

List of Figures

List of Tables

Chapter 1

Broad Learning System

1.1 Broad Learning System Architecture

1.1.1 Overview of the BLS Model Structure

The Broad Learning System (BLS) is a flat, non-deep neural architecture designed to efficiently learn complex nonlinear functions without relying on the hierarchical depth that characterizes conventional deep learning models. Instead of increasing depth through multiple stacked layers, BLS increases model capacity by expanding horizontally through the addition of feature and enhancement nodes. This design significantly simplifies training and improves computational efficiency while maintaining strong predictive performance.

Challenges with Deep Learning Models: Deep neural networks have been widely adopted due to their ability to extract hierarchical feature representations

from data. However, they suffer from several fundamental challenges:

- **Computational Cost**: Deep models require extensive iterative optimization using backpropagation. This results in long training times, high memory usage, and the need for specialized hardware (e.g., GPUs).
- **Vanishing and Exploding Gradients**: As depth increases, gradients propagated through the layers can vanish or explode, making training unstable and convergence difficult.
- **Hyperparameter Sensitivity**: Deep networks often require careful tuning of learning rates, initialization strategies, dropout rates, and batch sizes, which can be time-consuming and empirically driven.
- **Difficult Model Expansion**: Once trained, modifying the architecture (e.g., adding layers or nodes) typically requires retraining from scratch, which is impractical for real-time or online scenarios.
- **Lack of Interpretability**: The internal representations learned by deep models are often opaque, making it challenging to understand or debug their decision-making process.

How BLS Addresses These Challenges: BLS avoids these issues through its architectural simplicity and training method:

- **No Backpropagation**: Instead of optimizing layer-wise through gradient descent, BLS computes output weights via closed-form ridge regression. This eliminates the need for backpropagation entirely.

- **Fast One-Shot Learning**: Training is reduced to matrix computations, which are deterministic and efficient, especially when compared to iterative gradient updates.
- **Modular Expansion**: Nodes and data can be incrementally added without affecting the rest of the architecture. This allows the model to grow dynamically without retraining from scratch.
- **Stability and Robustness**: By applying regularization inherently, ridge regression mitigates overfitting and enhances generalization without the need for complex heuristics.
- **Interpretability and Simplicity**: Each component of the BLS architecture performs a specific, mathematically defined transformation, making the system more transparent than deep models.

The Broad Learning System is designed to replicate the expressive power of deep networks without inheriting their associated difficulties. Its horizontally expandable, non-iterative learning framework provides a scalable and practical alternative for a wide range of applications.

The architecture consists of the following components:

1. **Feature Mapping Layer**: Transforms the input data into multiple nonlinear subspaces through random projections.
2. **Enhancement Layer**: Applies additional nonlinear transformations to the feature mappings to increase representational complexity.

3. **Output Layer**: Computes a linear mapping from the transformed features to the target outputs using ridge regression.

The primary goal of this design is to maintain the expressive capacity of deep networks while avoiding their computational burden and complexity.

1.1.2 Feature Mapping Layer and Random Projections

Let:

- $X \in \mathbb{R}^{N \times d}$ be the input data with N samples and d features.
- n be the number of feature mapping groups.
- k be the number of nodes in each group.

Each group performs a nonlinear transformation of the input:

$$Z_i = \phi(XW_{e_i} + \beta_{e_i}), \quad i = 1, \ldots, n$$

where:

- $W_{e_i} \in \mathbb{R}^{d \times k}$ is a randomly initialized weight matrix.
- $\beta_{e_i} \in \mathbb{R}^{1 \times k}$ is a randomly generated bias vector.
- $\phi(\cdot)$ is a nonlinear activation function such as sigmoid, tanh, or ReLU.

The full feature matrix is:

$$Z = [Z_1, Z_2, \ldots, Z_n] \in \mathbb{R}^{N \times nk}$$

Intuition and Mathematical Justification:

- Random nonlinear projections help to separate input patterns in higher-dimensional space.
- This strategy is similar to the kernel trick used in Support Vector Machines, where data becomes more linearly separable after transformation.
- The use of random features is backed by the universal approximation theorem for randomized basis functions, which states that such projections can approximate any continuous function with sufficient nodes.

1.1.3 Enhancement Layer for Representation Enrichment

The enhancement nodes apply additional nonlinearity to the already transformed feature space:

$$H_j = \xi(ZW_{h_j} + \beta_{h_j}), \quad j = 1, \ldots, m$$

where:

- $W_{h_j} \in \mathbb{R}^{nk \times q}$ is a random weight matrix.
- $\beta_{h_j} \in \mathbb{R}^{1 \times q}$ is the associated bias.

- $\xi(\cdot)$ is another nonlinear activation function.

The full enhancement matrix is:

$$H = [H_1, H_2, \ldots, H_m] \in \mathbb{R}^{N \times mq}$$

Purpose and Functional Role:

- The enhancement nodes enrich the nonlinear feature space by enabling interactions between different parts of the feature mappings.
- They simulate the role of additional layers in deep learning by increasing the representational complexity without stacking.
- This additional stage enhances the model's ability to fit more complex functions with a relatively simple structure.

1.1.4 Formation of the Augmented Design Matrix

After feature mapping and enhancement, the complete transformed input becomes:

$$A = [Z \mid H] \in \mathbb{R}^{N \times (nk+mq)}$$

The output labels $Y \in \mathbb{R}^{N \times c}$ are predicted as:

$$Y = AW$$

where $W \in \mathbb{R}^{(nk+mq)\times c}$ is the output weight matrix to be learned.

This design matrix A captures a rich, nonlinearly transformed version of the input data and is the basis for learning the final linear mapping.

1.1.5 Computation of Output Weights via Ridge Regression

The weight matrix W is computed by solving a regularized least-squares problem:

$$W = \arg\min_{W} \|AW - Y\|_2^2 + \lambda \|W\|_2^2$$

This leads to the closed-form ridge regression solution:

$$W = (A^T A + \lambda I)^{-1} A^T Y$$

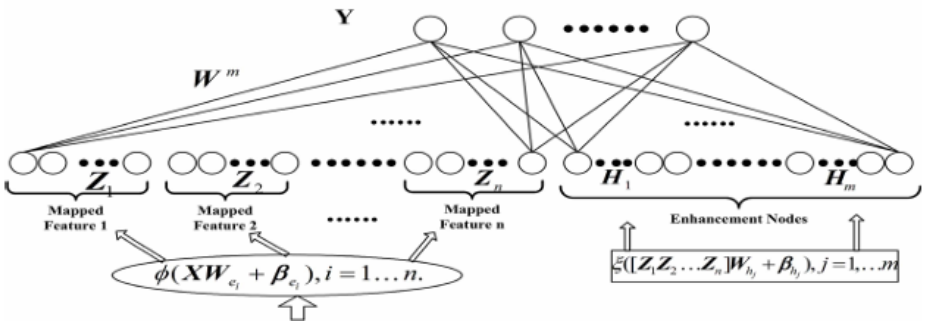


Figure 1.1: Illustration of the Broad Learning System (BLS) architecture. The input X is first passed through multiple nonlinear feature mapping groups Z_i and enhancement nodes H_j. These are concatenated into a broad design matrix $A = [Z \,|\, H]$, and the output weights W are computed using ridge regression.

Explanation and Motivation:

- Ridge regression adds an L_2 penalty to the weights, which improves numerical stability and reduces overfitting.
- It is especially important in high-dimensional settings, such as BLS, where A may be ill-conditioned or overcomplete.
- The solution is obtained without iterative training, allowing for efficient one-shot learning.

Geometric Interpretation:

- The solution W corresponds to projecting Y onto the column space of A, adjusted to minimize overfitting through regularization.
- The added λI term ensures invertibility and penalizes large weights.

1.1.6 Advantages of the BLS Design

The architecture of BLS offers several theoretical and practical benefits:

- **Expressive Power**: The combination of random feature mappings and nonlinear enhancements increases the model's capacity to approximate complex functions.
- **Training Efficiency**: Closed-form ridge regression provides a fast and deterministic training process without backpropagation.

- **Modular Expansion**: Feature and enhancement nodes can be added incrementally as needed.
- **Interpretability**: Each part of the model performs a well-defined transformation, making the system easier to understand and debug.

1.1.7 Comparative Perspective

In comparison to deep neural networks:

- BLS avoids vanishing gradients, complex optimization, and layer-wise backpropagation.
- It maintains comparable accuracy in many tasks while offering significantly faster training and simpler deployment.
- Unlike kernel methods, BLS does not require computation of a full Gram matrix and scales better to large datasets.

1.2 Incremental Learning in the Broad Learning System

1.2.1 Motivation and Setting

In real-world applications, machine learning systems often encounter new data or require additional capacity over time. Unlike deep neural networks, which typically require retraining from scratch when modifying architecture or data, the Broad

Learning System (BLS) supports efficient, algebraic updates. This is enabled by the use of ridge regression and pseudoinverse computations, which allow new components to be added while maintaining consistency with the existing model.

Let the original model be trained using the augmented feature matrix:

$$A = [Z \mid H] \in \mathbb{R}^{N \times p}$$

where:

- Z: Concatenated feature mapping from n groups of random projections.
- H: Concatenated enhancement mappings.
- $p = nk + mq$: Total number of columns in A (from n feature groups and m enhancement groups).
- $Y \in \mathbb{R}^{N \times c}$: Target labels for c output classes.

The output weights $W \in \mathbb{R}^{p \times c}$ are computed using ridge regression:

$$W = (A^T A + \lambda I)^{-1} A^T Y$$

This minimizes the regularized least-squares cost:

$$\mathcal{L}(W) = \|AW - Y\|_2^2 + \lambda \|W\|_2^2$$

1.2.2 Incremental Enhancement Node Addition

Suppose we add a new enhancement group $H_{\text{new}} \in \mathbb{R}^{N \times q'}$, where q' is the number of newly added enhancement nodes. The new feature matrix becomes:

$$A' = [A \,|\, H_{\text{new}}] \in \mathbb{R}^{N \times (p+q')}$$

Rather than solving a new ridge regression problem from scratch using A', BLS uses the following incremental weight update:

$$W' = \begin{bmatrix} W - DB^T Y \\ B^T Y \end{bmatrix}$$

where:

- $D = A^+ H_{\text{new}} \in \mathbb{R}^{p \times q'}$: Projection of new features onto existing feature space.
- A^+ is the Moore–Penrose pseudoinverse of A.
- $B = (I - H_{\text{new}}^T D)^+ \in \mathbb{R}^{q' \times N}$: Pseudoinverse of the residual projection.

Interpretation of Terms:

- D answers the question: "How much of the new nodes are already explained by existing nodes?"
- $I - H_{\text{new}}^T D$ measures the residual, i.e., the part of the new nodes orthogonal to A.

- $B^T Y$ projects the output targets onto this residual direction.
- $W - DB^T Y$ corrects the existing weights to prevent redundant explanation.

Why this update works: This formula results from block matrix inversion of:

$$W' = (A'^T A' + \lambda I)^{-1} A'^T Y$$

which is costly for large A'. Instead, the recursive update reuses A^+ and handles the residual using projection operators.

Key Intuition: We are extending the feature space. Rather than recomputing the regression, we ask: - How "new" are the new features (via projection)? - What extra information do they provide for predicting Y?

The weight update aligns these answers algebraically and efficiently.

1.2.3 Incremental Addition of Feature Groups and Enhancements

Let new random feature nodes $Z_{\text{new}} \in \mathbb{R}^{N \times k'}$ and corresponding enhancement nodes $H_{\text{new}} \in \mathbb{R}^{N \times q'}$ be added.

The updated design matrix is:

$$A' = [A \,|\, Z_{\text{new}} \,|\, H_{\text{new}}]$$

This is treated as a stacked enhancement node update — with two separate groups. The same pseudoinverse projection technique is used to compute:

$$W' = \begin{bmatrix} W - DB^T Y \\ B^T Y \end{bmatrix}$$

with D and B now corresponding to the combined block $[Z_{\text{new}} \mid H_{\text{new}}]$.

When is this necessary?

- If initial features are insufficient for the desired performance.
- When new sensors, data types, or input dimensions become available.
- To prevent saturation or underfitting.

1.2.4 Incremental Addition of Data Samples

Assume that N_a new data samples $X_a \in \mathbb{R}^{N_a \times d}$ arrive, with corresponding targets $Y_a \in \mathbb{R}^{N_a \times c}$. The data is passed through the same trained feature and enhancement mappings to compute:

$$A_a = [Z_a \mid H_a] \in \mathbb{R}^{N_a \times p}$$

Let W be the current output weights. The prediction error on new data is:

$$E_a = Y_a - A_a W$$

To incorporate the new data into training without modifying the full system, the

weight update is:

$$W' = W + E_a B$$

with:

$$B = A_a^T (A_a A_a^T + \lambda I)^{-1}$$

Interpretation:

- E_a is the residual — the part of Y_a not explained by the current model.
- B determines how to correct the weights so that predictions on A_a better match Y_a.
- The update is minimal in ℓ_2 norm and maintains regularization.

1.2.5 Theoretical Foundations

These updates are grounded in:

- **Ridge regression duality**: All updates solve the same cost function with new data or features.
- **Projection geometry**: Updates compute residuals orthogonal to current span.
- **Matrix inverse identities**: Blockwise inversion and Sherman–Morrison–Woodbury formula allow recursive updates.

1.3 Structural Simplification Using Singular Value Decomposition

1.3.1 Need for Structural Simplification in BLS

The Broad Learning System (BLS) expands its capacity by increasing the number of feature node groups and enhancement node groups. While this lateral growth enhances model expressiveness, it may lead to:

- Redundancy in feature representations,
- High-dimensional design matrices ($A = [Z \mid H]$),
- Increased computational burden in both training and inference,
- Overfitting to spurious or noisy features.

To address these issues, the BLS framework employs **Singular Value Decomposition (SVD)** as a principled technique for dimensionality reduction, allowing the model to retain most of its predictive power while eliminating redundant components.

1.3.2 Mathematical Foundation of SVD

Given a real matrix $A \in \mathbb{R}^{m \times n}$ (typically Z_i or H_j in the BLS context), SVD factorizes it into three matrices:

$$A = U \Sigma V^T$$

where:

- $U \in \mathbb{R}^{m \times m}$ is a matrix of orthonormal **left singular vectors**,
- $V \in \mathbb{R}^{n \times n}$ is a matrix of orthonormal **right singular vectors**,
- $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing the **singular values** $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$.

Each singular value σ_i represents the "energy" or variance captured by the corresponding singular vector direction.

1.3.3 Low-Rank Approximation of Feature and Enhancement Groups

Let A be either a feature group Z_i or enhancement group H_j. After computing its SVD, we approximate it by retaining only the top r singular values:

$$A_r = U_r \Sigma_r V_r^T$$

where:

- U_r contains the first r columns of U,
- Σ_r contains the top r singular values,
- V_r contains the first r columns of V.

Why this approximation is optimal: By the **Eckart–Young–Mirsky Theorem**, A_r is the best rank-r approximation of A under both the spectral and Frobenius norms:

$$\min_{\text{rank}(B)=r} \|A - B\|_F = \|A - A_r\|_F = \left(\sum_{i=r+1}^{\min(m,n)} \sigma_i^2 \right)^{1/2}$$

Thus, removing small singular values discards directions with minimal variance (typically noise or redundant features), while retaining the informative structure.

1.3.4 Application of SVD in BLS

In the BLS architecture, each group of features Z_i and enhancements H_j can be independently simplified:

1. Compute $Z_i = \phi(XW_{e_i} + \beta_{e_i})$.
2. Perform SVD: $Z_i = U_i \Sigma_i V_i^T$.
3. Choose the number of retained singular values r_i based on a threshold:

$$\text{Retain } \sigma_1, \ldots, \sigma_{r_i} \text{ such that } \frac{\sigma_{r_i}}{\sigma_1} > \eta$$

   where η is a user-defined threshold (e.g., $\eta = 0.01$).
4. Form the compressed version: $Z_i^{\text{compressed}} = U_{i,r} \Sigma_{i,r} V_{i,r}^T$.
5. Use $Z_i^{\text{compressed}}$ in the final matrix A instead of the full Z_i.

The same process is applied to each H_j.

1.3.5 Steps Involved

- **Step 1 (Nonlinear Transformation)**: Converts input data into a high-dimensional feature space.
- **Step 2 (SVD)**: Decomposes the data into orthogonal directions ordered by information content.
- **Step 3 (Thresholding)**: Ensures that only directions contributing meaningfully to variance are retained.
- **Step 4 (Reconstruction)**: Generates a simplified representation that preserves essential data geometry.
- **Step 5 (Integration)**: Replaces the original node output with its simplified form to build a compact design matrix A.

1.3.6 Benefits of SVD-Based Simplification

- **Reduces computational cost**: Smaller A matrices lead to faster training and inference.
- **Improves generalization**: Eliminating low-variance (noisy) directions acts as a form of regularization.
- **Enhances interpretability**: The retained singular vectors represent the most significant directions of variation.

- **Preserves accuracy**: The use of SVD guarantees minimal information loss under Frobenius norm.
- **Modularity**: Each Z_i and H_j can be processed independently, simplifying implementation and tuning.

1.4 Limitations and Challenges of the Broad Learning System

While the Broad Learning System (BLS) offers significant advantages in terms of simplicity, speed, and adaptability, it also presents several limitations and open challenges that merit consideration.

1.4.1 Randomness in Feature Generation

Randomly generated weights and biases in the feature mapping and enhancement layers lead to variability in performance. Although this is mitigated by generating sufficiently large feature sets, it can lead to:

- Inconsistent accuracy across different runs,
- Reduced reproducibility compared to deterministic architectures,
- Dependence on manual trial-and-error for selecting the number of feature and enhancement nodes.

1.4.2 Lack of Learned Hierarchical Representations

BLS lacks the hierarchical structure that is characteristic of deep neural networks. While it mimics multi-level transformations through enhancement nodes, it cannot automatically extract abstract features layer-by-layer from raw data. This can be limiting for:

- Highly structured data such as images, text, or audio,
- Transfer learning or fine-tuning across tasks,
- Interpretability of intermediate representations.

1.4.3 Sensitivity to Parameter Choices

BLS requires careful tuning of architectural hyperparameters, such as:

- Number of feature groups and enhancement groups,
- Number of nodes per group,
- Regularization parameter λ in ridge regression,
- SVD truncation threshold η for structural simplification.

There is no universally optimal choice, and improper tuning can lead to overfitting, underfitting, or inefficient models.

1.4.4 Scalability to Ultra-High Dimensions

Although BLS is computationally efficient, when the number of nodes becomes extremely large (e.g., tens of thousands), the design matrix A can grow to a size that exceeds available memory. Matrix inversion or SVD then becomes expensive, reducing the model's scalability in certain environments.

1.4.5 Limited Integration with End-to-End Tasks

Since BLS operates on precomputed features using random projections, it lacks integration with end-to-end learning frameworks. This makes it less suitable for tasks requiring:

- Gradient-based joint optimization across multiple components,
- Differentiable pipelines, such as those in reinforcement learning or generative models.

1.5 Experimental Results

This section presents the experimental evaluation of the Broad Learning System (BLS) on two datasets: MNIST (handwritten digit recognition) and Breast Cancer (binary classification). All experiments were conducted using Google Colab (Free Tier) with an Intel Xeon CPU @ 2.20GHz, 13.61 GB RAM, and Python 3.11.13. Performance is assessed based on training time, testing time, and classification accuracy. The architectural details used in each case are also documented.

1.5.1 Dataset 1: MNIST

Dataset Description:

- **Input dimension:** 784 (28×28 pixels)
- **Number of output classes:** 10 (digits 0 to 9, one-hot encoded)

BLS Configuration:

- **Number of feature mapping groups (n):** 10
- **Nodes per group (k):** 10
- **Total feature nodes:** $n \times k = 100$
- **Number of enhancement nodes (m):** 11,000
- **Total design matrix size:** $A \in \mathbb{R}^{N \times 11,100}$
- **Activation function:** Sigmoid (for enhancement layer)
- **Regularization parameter (λ):** 1×10^{-8}
- **Batch size:** 5000 (incremental batch training)

Performance Results:

Table 1.1: BLS Performance on MNIST Dataset

Metric	**Value**
Training Time (seconds)	245.83
Testing Time (seconds)	1.88
Test Accuracy (%)	96.24

BLS achieved strong performance on MNIST, with an accuracy of 96.24%. Although training time was relatively high due to the large number of enhancement nodes, the model demonstrated its ability to effectively learn nonlinear representations from high-dimensional input data using a flat architecture.

1.5.2 Dataset 2: Breast Cancer

Dataset Description:

- **Input dimension:** 30 features (standardized)
- **Number of output classes:** 2 (malignant vs. benign, one-hot encoded)

BLS Configuration:

- **Number of feature mapping groups (n):** 2
- **Nodes per group (k):** 8
- **Total feature nodes:** $n \times k = 16$
- **Number of enhancement nodes (m):** 4
- **Total design matrix size:** $A \in \mathbb{R}^{N \times 20}$
- **Activation function:** Sigmoid (for enhancement layer)
- **Regularization parameter (λ):** 1×10^{-6}
- **Train-test split:** 70% training, 30% testing

Performance Results:

Table 1.2: BLS Performance on Breast Cancer Dataset

Metric	**Value**
Training Time (seconds)	0.0036
Testing Time (seconds)	0.0019
Test Accuracy (%)	97.66

On the Breast Cancer dataset, BLS trained and tested almost instantly while achieving excellent classification accuracy. This showcases the model's adaptability and computational efficiency, especially in low-dimensional, small-sample scenarios.

1.5.3 Cross-Dataset Summary

Table 1.3: Summary of BLS Results on MNIST and Breast Cancer

Metric	**MNIST**	**Breast Cancer**
Training Time (s)	269.09	0.0036
Testing Time (s)	1.72	0.0019
Test Accuracy (%)	96.24	97.66

1.5.4 Key Observations

- BLS achieved high classification accuracy on both datasets, confirming its strong generalization capability.

- The architecture scaled well across different problem sizes, from a small medical dataset to a large image classification task.
- Although training time was higher on MNIST due to the high number of enhancement nodes, the accuracy gain justifies the computation.
- On Breast Cancer, the model demonstrated efficiency and robustness with minimal computational effort.

1.6 Conclusion

The Broad Learning System (BLS) provides a compelling alternative to deep learning architectures by leveraging a flat, modular design. Through the use of random nonlinear transformations, enhancement nodes, and ridge regression, BLS avoids many of the limitations associated with deep networks, such as high training time, backpropagation complexity, and inflexibility in adapting to new data.

Three key strengths of BLS have been highlighted in this report:

1. **Architectural Efficiency**: The broad expansion of feature and enhancement nodes enables expressive function approximation without depth, supported by fast closed-form training.
2. **Incremental Learning**: BLS supports dynamic model updates, allowing new data and nodes to be added without retraining the entire system.
3. **Structural Simplification**: SVD-based pruning removes redundant features while preserving accuracy, leading to more compact and generalizable models.

These properties make BLS particularly well-suited for:

- Online and streaming learning environments,
- Applications requiring fast adaptation and minimal computational resources,
- Systems where interpretability, modularity, and retrainability are essential.

However, BLS is not without limitations. It does not learn hierarchical representations, relies heavily on random initialization, and requires manual tuning of architectural parameters. In its current form, it serves best as a complementary tool rather than a universal replacement for deep models.

In summary, BLS bridges the gap between shallow neural networks and deep architectures by combining expressive power, training speed, and adaptability into a single, elegant framework. With further research into automatic node tuning, hybrid architectures, and integration with end-to-end pipelines, BLS holds promise as a foundational method for efficient and scalable machine learning.

Chapter 2

Physics-Informed Extreme Learning Machine (PIELM)

2.1 Introduction

Partial Differential Equations (PDEs) are central to modeling physical systems such as fluid flow, heat transfer, electromagnetic waves, and structural deformation. While exact solutions exist for a few special cases, most real-world problems involve complex geometries and boundary conditions that require numerical approximation.

Traditional numerical methods such as the Finite Difference Method (FDM), Finite Element Method (FEM), and Finite Volume Method (FVM) rely on discretization and meshing of the computational domain. However, these methods can be computationally expensive, particularly in high dimensions or for irregular geometries.

To address these challenges, neural-network-based approaches, particularly Physics-Informed Neural Networks (PINNs), have emerged as mesh-free solvers. PINNs embed the governing physics, in the form of differential operators, directly into the loss function of the neural network. However, they require iterative gradient-based training, which can be slow and sensitive to initialization.

The Physics-Informed Extreme Learning Machine (PIELM) is a fast, mesh-free, and training-free alternative to PINNs. It combines:

- The speed and simplicity of Extreme Learning Machines (ELMs),
- The physical consistency of PINNs by enforcing residuals of the governing equations.

2.2 Extreme Learning Machine (ELM)

2.2.1 Architecture Overview

The Extreme Learning Machine (ELM) is a feedforward neural network with a single hidden layer, designed to achieve fast learning by solving for output weights analytically. The key idea is to randomly initialize the weights of the hidden layer and solve a linear system for the output weights.

2.2.2 Function Approximation Capability

According to the Universal Approximation Theorem, a single hidden-layer neural network with a non-linear activation function can approximate any continuous function on a compact subset of $\mathbb{R}^m$, provided it has a sufficient number of neurons. ELM exploits this by keeping the hidden layer weights fixed and training only the output weights.

2.2.3 Network Formulation

Let:

- N: Number of training samples.
- m: Input dimension (e.g., $m = 2$ for (x, t)).
- N^*: Number of neurons in the hidden layer.

Each input $\vec{x} \in \mathbb{R}^{m\times 1}$ is processed by N^* hidden neurons:

$$h_j(\vec{x}) = \varphi(\vec{w}_j^\top \vec{x} + b_j)$$

where $\vec{w}_j \in \mathbb{R}^{m\times 1}$ is the input weight vector and $b_j \in \mathbb{R}$ is the bias for neuron j.

The output of all neurons for one input is:

$$\vec{h}(\vec{x}) = \begin{bmatrix} \varphi(\vec{w}_1^\top \vec{x} + b_1) \\ \vdots \\ \varphi(\vec{w}_{N^*}^\top \vec{x} + b_{N^*}) \end{bmatrix} \in \mathbb{R}^{N^*\times 1}$$

For all N training samples, the hidden layer output matrix becomes:

$$H = \begin{bmatrix} \vec{h}(\vec{x}_1)^\top \\ \vdots \\ \vec{h}(\vec{x}_N)^\top \end{bmatrix} \in \mathbb{R}^{N \times N^*}$$

Let $\vec{T} \in \mathbb{R}^{N \times 1}$ be the target output. The network prediction is:

$$\vec{y} = H\vec{c}$$

where $\vec{c} \in \mathbb{R}^{N^* \times 1}$ are the output weights.

2.2.4 Analytical Computation of Output Weights

The least-squares problem for training is defined as:

$$\min_{\vec{c}} \|H\vec{c} - \vec{T}\|_2^2$$

Using the Moore–Penrose pseudo-inverse, the minimum-norm solution is:

$$\vec{c} = \text{pinv}(H)\vec{T}$$

2.2.5 Justification for Pseudo-Inverse Solution

The pseudo-inverse solution is theoretically justified as follows:

- The Moore–Penrose pseudo-inverse exists for any matrix H, regardless of rank or shape.
- It provides a least-squares solution when the system is overdetermined and the minimum-norm solution when underdetermined.
- It guarantees a unique minimizer of $\|\vec{c}\|_2$ among all solutions.

2.3 Physics-Informed Neural Networks (PINNs)

2.3.1 Conceptual Overview

Traditional neural networks are trained using datasets containing input-output pairs. In contrast, many scientific problems governed by Partial Differential Equations (PDEs) often lack dense measurement data. Instead, the underlying physical laws themselves can serve as the training signal.

Physics-Informed Neural Networks (PINNs) incorporate these governing PDEs directly into the training loss. This approach allows the neural network to approximate the solution $u(x,t)$ using only the physics of the system and a limited set of initial and boundary conditions.

2.3.2 General PDE Formulation

The generic time-dependent linear PDE under consideration is given by:

$$\frac{\partial u(x,t)}{\partial t} + \mathcal{N}[u(x,t)] = R(x,t), \quad (x,t) \in \Omega$$

subject to the following constraints:

- **Initial Condition (IC):**

$$u(x, 0) = F(x), \quad x \in [x_L, x_R]$$

- **Boundary Condition (BC):**

$$u(x, t) = B(x, t), \quad (x, t) \in \partial\Omega$$

Where:

- $u(x, t) \in \mathbb{R}$ is the true scalar-valued solution,
- $\mathcal{N}[u]$ is a spatial differential operator (e.g., advection or diffusion),
- $R(x, t) \in \mathbb{R}$ is a known source term,
- $\Omega \subset \mathbb{R}^2$ denotes the spatio-temporal domain.

2.3.3 Neural Network Approximation

Let $f(x, t; \theta)$ represent the neural network output approximating $u(x, t)$, where θ includes all trainable parameters.

The training process in PINNs aims to minimize residuals of the governing PDE and its constraints by evaluating the network at selected collocation points.

2.3.4 Residual Definitions

Three types of residuals are defined, all of which are scalar-valued and computed pointwise:

- **PDE Residual at Interior Point** (x_i^f, t_i^f)**:**

$$\xi_f^i = \left[\frac{\partial f}{\partial t}(x_i^f, t_i^f) + \mathcal{N}[f](x_i^f, t_i^f) - R(x_i^f, t_i^f) \right] \in \mathbb{R}$$

- **Initial Condition Residual at** x_j^{ic}**:**

$$\xi_{ic}^j = f(x_j^{ic}, 0) - F(x_j^{ic}) \in \mathbb{R}$$

- **Boundary Condition Residual at** (x_k^{bc}, t_k^{bc})**:**

$$\xi_{bc}^k = f(x_k^{bc}, t_k^{bc}) - B(x_k^{bc}, t_k^{bc}) \in \mathbb{R}$$

Each residual quantifies the discrepancy between the predicted solution and the expected physical behavior.

2.3.5 Residual Loss Function

Let the following sets denote different types of collocation points:

- $\mathcal{S}_f = \{(x_i^f, t_i^f)\}_{i=1}^{N_f}$: interior points for enforcing the PDE,
- $\mathcal{S}_{ic} = \{x_j^{ic}\}_{j=1}^{N_{ic}}$: spatial points at $t = 0$,

- $\mathcal{S}_{bc} = \{(x_k^{bc}, t_k^{bc})\}_{k=1}^{N_{bc}}$: boundary points.

The total loss function is defined as:

$$\mathcal{L} = \underbrace{\frac{1}{N_f}\sum_{i=1}^{N_f}(\xi_f^i)^2}_{\text{PDE residual}} + \underbrace{\frac{1}{N_{ic}}\sum_{j=1}^{N_{ic}}(\xi_{ic}^j)^2}_{\text{Initial condition}} + \underbrace{\frac{1}{N_{bc}}\sum_{k=1}^{N_{bc}}(\xi_{bc}^k)^2}_{\text{Boundary condition}}$$

2.3.6 Optimization and Training Method

Unlike traditional supervised learning where training relies on labeled data, PINNs are trained by minimizing the total residual loss $\mathcal{L}$. This is achieved using gradient-based optimization algorithms such as Adam or L-BFGS.

All derivatives required in residual computation are obtained using automatic differentiation frameworks. However, the training process may exhibit:

- Sensitivity to initialization,
- Slow convergence for stiff PDEs,
- Performance degradation with increasing network depth or complexity.

Challenges like these highlight the need for training-free alternatives such as the PIELM.

2.4 Physics-Informed Extreme Learning Machine (PIELM)

2.4.1 Overview and Motivation

The Physics-Informed Extreme Learning Machine (PIELM) is a non-iterative method for approximating the solution of linear partial differential equations (PDEs). It constructs a single-hidden-layer feedforward neural network whose parameters are computed analytically by enforcing physical consistency with the PDE and associated constraints.

Unlike PINNs, PIELM avoids gradient-based optimization and achieves fast computation by solving a single linear system. It is mesh-free, training-free, and well-suited for problems involving irregular domains.

2.4.2 Governing Equation and Boundary Conditions

Let $u(\vec{x}) \in \mathbb{R}$ denote a scalar-valued function defined on a domain $\Omega \subset \mathbb{R}^d$. The general form of the governing linear PDE is:

$$\mathcal{L}[u](\vec{x}) = R(\vec{x}), \quad \vec{x} \in \Omega$$

with Dirichlet boundary conditions:

$$u(\vec{x}) = B(\vec{x}), \quad \vec{x} \in \partial\Omega$$

Where:

- $\vec{x} = (x_1, x_2, \ldots, x_d)^\top \in \mathbb{R}^{d\times 1}$,
- $\mathcal{L}[\cdot]$ is a linear differential operator,
- $R(\vec{x}) \in \mathbb{R}$ is the source term,
- $B(\vec{x}) \in \mathbb{R}$ is the prescribed boundary value function.

The objective is to construct a network $f(\vec{x})$ that satisfies:

$$\mathcal{L}[f](\vec{x}) \approx R(\vec{x}), \quad \vec{x} \in \Omega; \quad f(\vec{x}) \approx B(\vec{x}), \quad \vec{x} \in \partial\Omega$$

2.4.3 Network Architecture

Input Representation:

Let the total number of input points be N. Define the input matrix:

$$X = \begin{bmatrix} \vec{x}^{(1)} \\ \vec{x}^{(2)} \\ \vdots \\ \vec{x}^{(N)} \end{bmatrix} \in \mathbb{R}^{N\times d}$$

where each row $\vec{x}^{(i)} \in \mathbb{R}^{1\times d}$ is a spatial point in the domain.

Hidden Layer:

The hidden layer consists of N^* neurons, defined by:

$$W \in \mathbb{R}^{d \times N^*}, \quad b \in \mathbb{R}^{1 \times N^*}$$

Here:

- W is the input weight matrix,
- b is the bias row vector,
- Both W and b are fixed and randomly initialized.

The pre-activation is computed as:

$$Z = XW + b \in \mathbb{R}^{N \times N^*}$$

Activation Layer:

Apply a non-linear activation function φ elementwise:

$$H = \varphi(Z) \in \mathbb{R}^{N \times N^*}$$

where φ is typically chosen as a smooth function such as $\tanh(z)$, allowing for analytical differentiation.

Output Layer and Prediction:

Let $\vec{c} \in \mathbb{R}^{N^* \times 1}$ denote the vector of output weights. The predicted output at all input points is:

$$f(X) = H\vec{c} \in \mathbb{R}^{N \times 1}$$

For an individual input $\vec{x}^{(i)}$, the predicted scalar output is:

$$f(\vec{x}^{(i)}) = \sum_{k=1}^{N^*} c_k \cdot \varphi(\vec{w}_k^\top \vec{x}^{(i)} + b_k)$$

This linearity in $\vec{c}$ allows analytical enforcement of physical constraints.

2.4.4 Differentiation of the Network Output

To enforce the governing PDE, the network output $f(\vec{x})$ must be differentiated with respect to spatial variables. This section presents the computation of the first and second derivatives required to evaluate the operator $\mathcal{L}[f]$.

Gradient of the Output:

The gradient of $f(\vec{x})$ with respect to $\vec{x} \in \mathbb{R}^{d \times 1}$ is computed using the chain rule:

$$\nabla f(\vec{x}) = \sum_{k=1}^{N^*} c_k \cdot \varphi'\left(\vec{w}_k^\top \vec{x} + b_k\right) \cdot \vec{w}_k$$

Dimensional Justification:

- $\varphi'(z_k) \in \mathbb{R}$,
- $\vec{w}_k \in \mathbb{R}^{d\times 1}$,
- Thus, $\varphi'(z_k)\vec{w}_k \in \mathbb{R}^{d\times 1}$,
- The sum over k yields $\nabla f(\vec{x}) \in \mathbb{R}^{d\times 1}$.

Second Derivative (Hessian or Laplacian):

The second-order derivatives follow from differentiating the gradient. For the (i, j)-th component:

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \sum_{k=1}^{N^*} c_k \cdot \varphi''(z_k) \cdot w_{ki} w_{kj}$$

The diagonal elements yield:

$$\frac{\partial^2 f}{\partial x_i^2} = \sum_{k=1}^{N^*} c_k \cdot \varphi''(z_k) \cdot w_{ki}^2$$

For problems governed by the Laplacian operator $\Delta f(\vec{x})$, the expression simplifies to:

$$\Delta f(\vec{x}) = \sum_{k=1}^{N^*} c_k \cdot \varphi''(z_k) \cdot \|\vec{w}_k\|^2$$

Batch Derivative Evaluation:

Let $X \in \mathbb{R}^{N\times d}$ denote a batch of input points. The following intermediate matrices are used:

- $Z = XW + b \in \mathbb{R}^{N \times N^*}$,
- $\varphi'(Z), \varphi''(Z) \in \mathbb{R}^{N \times N^*}$: elementwise derivatives.

Let:

$$W^2 = \left[\|\vec{w}_1\|^2, \ldots, \|\vec{w}_{N^*}\|^2\right] \in \mathbb{R}^{1 \times N^*}$$

Then:

$$H_f = \varphi''(Z) \circ W^2 \in \mathbb{R}^{N \times N^*}$$

where $\circ$ denotes the Hadamard (elementwise) product.

Linearity of Derivatives in Output Weights:

All derivatives of $f(\vec{x})$ are linear with respect to $\vec{c}$, i.e.,

$$\mathcal{L}[f](X) = H_{\mathcal{L}} \cdot \vec{c}$$

for some matrix $H_{\mathcal{L}}$ constructed from $\varphi, \varphi', \varphi''$, and W. This linear structure enables physics to be enforced via a linear system.

2.4.5 Construction of the Residual System

Collocation Points:

Let:

- $X_f \in \mathbb{R}^{N_f \times d}$: interior collocation points,
- $X_{bc} \in \mathbb{R}^{N_{bc} \times d}$: boundary points,
- $N = N_f + N_{bc}$: total number of evaluation points.

PDE Residual Matrix:

For interior points:

$$Z_f = X_f W + b, \quad \varphi''(Z_f) \in \mathbb{R}^{N_f \times N^*}$$

$$H_f = \varphi''(Z_f) \circ W^2 \in \mathbb{R}^{N_f \times N^*}$$

Boundary Residual Matrix:

For boundary points:

$$Z_{bc} = X_{bc} W + b, \quad H_{bc} = \varphi(Z_{bc}) \in \mathbb{R}^{N_{bc} \times N^*}$$

Combined Residual Matrix:

The full residual system is:

$$H_{\text{total}} = \begin{bmatrix} H_f \\ H_{bc} \end{bmatrix} \in \mathbb{R}^{(N_f + N_{bc}) \times N^*}$$

Right-Hand Side Vector:

$$\vec{K}_{\text{total}} = \begin{bmatrix} R(X_f) \\ B(X_{bc}) \end{bmatrix} \in \mathbb{R}^{(N_f+N_{bc})\times 1}$$

Linear System and Solution:

The system to solve is:

$$H_{\text{total}} \cdot \vec{c} = \vec{K}_{\text{total}}$$

If overdetermined, the minimum-norm solution is:

$$\vec{c} = \text{pinv}(H_{\text{total}}) \cdot \vec{K}_{\text{total}}$$

The network prediction at any input matrix X is then given by:

$$f(X) = H(X) \cdot \vec{c}$$

2.5 Limitations and Challenges of PIELM

While the Physics-Informed Extreme Learning Machine (PIELM) provides significant advantages in terms of speed, simplicity, and analytical solvability, it also exhibits several theoretical and practical limitations. These are categorized below based on architectural constraints, numerical conditioning, and PDE generalizability.

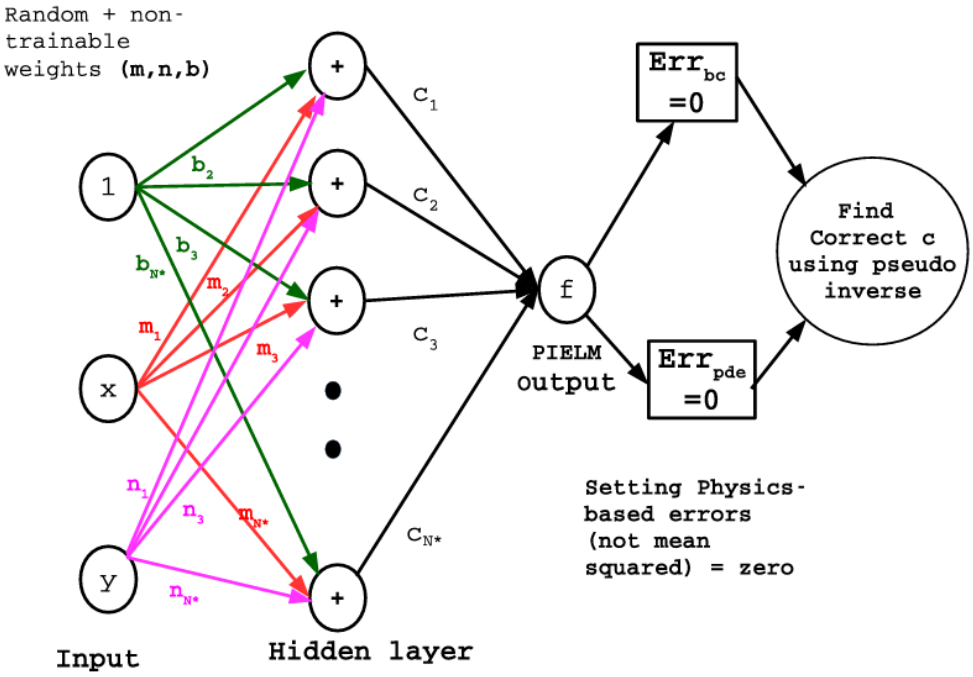


Figure 2.1: Schematic diagram of PIELM for 1D unsteady problems

2.5.1 Fixed Random Weights

PIELM relies on random initialization of the input weights and biases in the hidden layer. These parameters are not updated through training.

- The hidden layer basis functions may fail to sufficiently span the solution space, depending on the random seed.
- As the hidden layer is fixed, the model cannot adapt to local residuals or improve accuracy through weight updates.
- Poor initializations may yield underfitting and reduced generalization.

2.5.2 Expressiveness of Hidden Basis

Accurate PDE solution requires that the function space spanned by $\varphi(Z)$ be rich enough. If this space is insufficient, the residual system:

$$\min_{\vec{c}} \|H\vec{c} - \vec{K}\|_2^2$$

may have a large minimum, indicating poor approximation even at the optimal $\vec{c}$.

- Increasing the number of hidden neurons N^* can improve expressiveness.
- However, larger N^* also increases the risk of numerical instability and ill-conditioning.

2.5.3 Lack of Regularization

The least-squares pseudo-inverse does not include any form of regularization by default.

- Solutions may overfit collocation points while oscillating in between.
- Output weights $\vec{c}$ can have large magnitudes.
- Regularization terms such as ℓ_2 penalties must be added manually to mitigate instability.

2.5.4 Restriction to Linear PDEs

PIELM requires the residual system to be linear in the output weights $\vec{c}$. Consequently, it is applicable only to linear PDEs.

- Nonlinear operators (e.g., u^2, $\sin(u)$) render the residuals nonlinear in $\vec{c}$.
- In such cases, the pseudo-inverse cannot be used directly.
- Nonlinear PDEs require iterative schemes, linearization, or alternate network formulations.

2.5.5 Scalability Limitations in High Dimensions

Despite being mesh-free, PIELM is not immune to the curse of dimensionality.

- Higher-dimensional domains require denser sampling for accurate residual enforcement.
- The size of the system $H\vec{c} = \vec{K}$ grows with both input dimension d and neuron count N^*.
- The computational cost of evaluating derivatives and computing the pseudo-inverse increases significantly.

2.5.6 Handling of Irregular Geometries

For domains with curved or non-rectangular boundaries:

- Boundary collocation must account for geometry-specific sampling and parametrization.
- PIELM does not inherently support mesh adaptivity or error-driven refinement.
- Enforcement of boundary conditions becomes more complex and problem-dependent.

2.6 Experimental Results

This section presents the application of the Physics-Informed Extreme Learning Machine (PIELM) to a set of five benchmark test cases (TC-1 to TC-5), which encompass steady and unsteady linear partial differential equations (PDEs) in both one and two spatial dimensions. Each test case is characterized by a known governing PDE

and an associated analytical solution, enabling rigorous validation of the method. For each test case, the governing equation, computational domain, and boundary or initial conditions are clearly specified.

Four architectural variants of the PIELM framework are evaluated, and their performances are assessed in terms of approximation accuracy and computational efficiency. Specifically, the Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and CPU time are computed for each model. In addition, solution plots and corresponding error plots are generated to visualize predictive accuracy. For clarity and brevity, only the plots corresponding to the best-performing model (based on MAE and RMSE) are included for each test case.

2.6.1 Experimental Environment

All experiments were conducted using Google Colab (Free Tier) with an Intel Xeon CPU @ 2.20GHz, 13.61 GB RAM, and Python 3.11.13.

2.6.2 PIELM Model Configurations

Four variants of the PIELM architecture were evaluated across all test cases:

- **Model 1:** Standard PIELM using the Moore–Penrose pseudo-inverse with fixed number of neurons (Number of neurons = Number of Collocation points + Number of Boundary Condition points + Number of Initial Condition points).
- **Model 2:** PIELM with ridge regularization, where the output weights are computed using Tikhonov regularized least-squares.

- **Model 3:** PIELM with hyperparameter tuning over the number of hidden neurons.
- **Model 4:** PIELM with both neuron count and ridge regularization jointly optimized via nested grid search.

Each model uses a single-hidden-layer neural network. The weights and biases in the hidden layer are randomly initialized and fixed, while the output weights are solved analytically using a non-iterative approach.

2.6.3 Test Cases

TC-1: 1D Steady Linear Advection

Governing PDE:

$$\frac{du}{dx} = R(x), \quad x \in (0, 1)$$

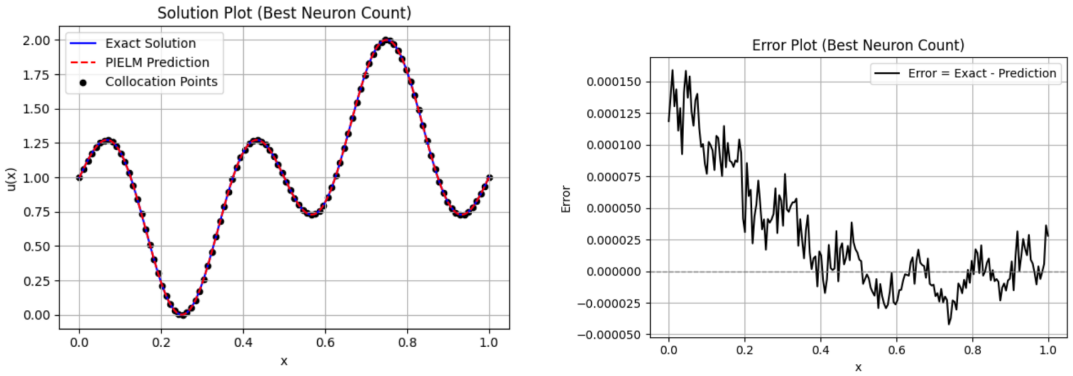
Exact Solution:

$$u(x) = \sin(2\pi x)\cos(4\pi x) + 1$$

Boundary Condition: Dirichlet boundary at $x = 0$ and $x = 1$ i.e., $u(0) = 1$ and $u(1) = 1$

Table 2.1: Results for TC-1: 1D Steady Linear Advection

Model	**Neurons**	$\boldsymbol{\lambda}$	**MAE**	**RMSE**	**Tcpu (s)**
Model 1	102	–	3.28e-03	3.79e-03	0.0154
Model 2	102	0.00e+00	3.28e-03	3.79e-03	0.0270
Model 3	590	–	3.77e-05	5.51e-05	11.86
Model 4	590	0.00e+00	3.77e-05	5.51e-05	42.30



(a) PIELM vs. Exact Solution (b) Prediction Error

Figure 2.2: Solution and Error Plots for TC-1

Model 1 and Model 2 yield identical performance due to the negligible regularization effect at $\lambda = 0$. While computationally efficient, they show higher error compared to the tuned models. Model 3 significantly improves accuracy by optimizing the neuron count. Model 4 achieves the best trade-off between regularization and capacity, but at a greater computational cost. Overall, neuron tuning substantially benefits the approximation quality for this smooth 1D problem.

TC-2: 1D Steady Linear Diffusion

Governing PDE:

$$\frac{d^2u}{dx^2} = R(x), \quad x \in (0,1)$$

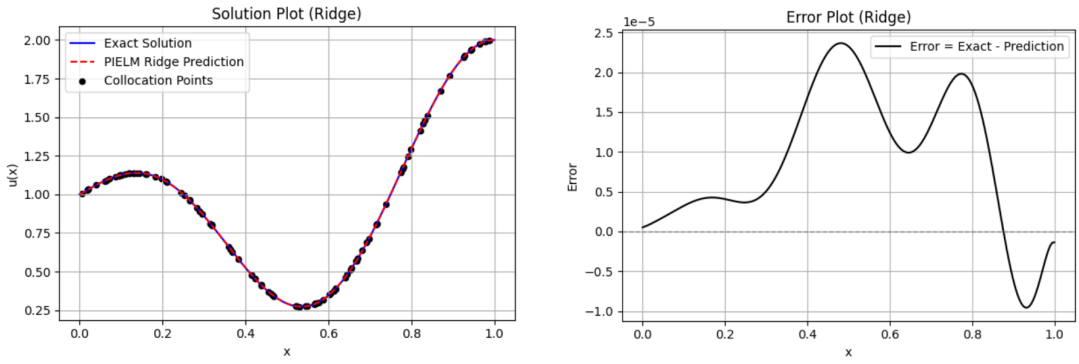
Exact Solution:

$$u(x) = \sin\left(\frac{\pi x}{2}\right)\cos(2\pi x) + 1$$

Boundary Conditions: Dirichlet boundaries at $x = 0$ and $x = 1$, imposed using the exact solution.

Table 2.2: Results for TC-2: 1D Steady Linear Diffusion

Model	Neurons	λ	MAE	RMSE	Tcpu (s)
Model 1	102	–	2.78e-01	3.71e-01	0.0666
Model 2	102	1.00e-10	1.04e-05	1.25e-05	0.0209
Model 3	135	–	2.38e-01	2.97e-01	9.113
Model 4	970	1.00e-02	1.23e-01	1.44e-01	44.08



(a) PIELM vs. Exact Solution (b) Prediction Error

Figure 2.3: Solution and Error Plots for TC-2

For the 1D steady linear diffusion problem, Model 2 with ridge regularization yielded the lowest error metrics, demonstrating superior numerical stability and accuracy. Model 1 and Model 3 suffered from relatively large errors, indicating sensitivity to conditioning and neuron count. While Model 4 slightly improved accuracy over Model 1 and Model 3, its performance did not surpass that of Model 2 and incurred substantially higher computational time.

TC-3: 1D Steady Linear Advection–Diffusion

Governing PDE:

$$\frac{du}{dx} - \nu \frac{d^2u}{dx^2} = R(x), \quad x \in (0,1), \quad \nu = 0.2$$

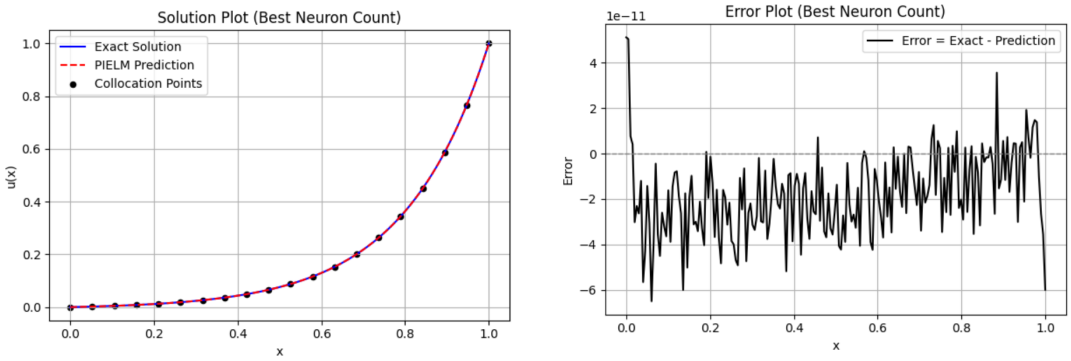
Exact Solution:

$$u(x) = \frac{e^{x/\nu} - 1}{e^{1/\nu} - 1}$$

Boundary Conditions: Dirichlet boundaries: $u(0) = 0$, $u(1) = 1$

Table 2.3: Results for TC-3: 1D Steady Linear Advection–Diffusion

Model	**Neurons**	**λ**	**MAE**	**RMSE**	**Tcpu (s)**
Model 1	22	–	2.79e-10	3.60e-10	0.0055
Model 2	22	0.00e+00	2.79e-10	3.60e-10	0.0054
Model 3	765	–	2.12e-11	2.57e-11	4.03
Model 4	30	1.00e-01	3.60e-01	3.85e-01	30.39



(a) PIELM vs. Exact Solution (b) Prediction Error

Figure 2.4: Solution and Error Plots for TC-3

TC-3, representing a coupled advection-diffusion scenario, was solved with very high accuracy by Models 1, 2, and 3. Model 3 slightly outperformed the others in error metrics due to optimized neuron count. Surprisingly, Model 4 underperformed, likely due to suboptimal interplay between regularization and neuron selection in this stiff PDE case. Model 2 remains a strong balance of speed and accuracy.

TC-4: 2D Steady Linear Advection

Governing PDE:

$$a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} = R(x, y), \quad (x, y) \in \Omega$$

$$a = 1$$

and

$$b = 0.5$$

Exact Solution:

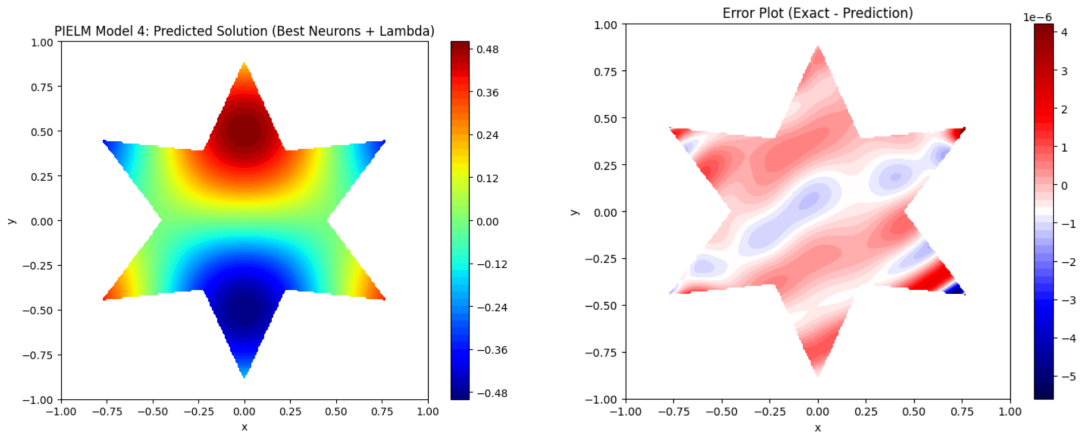
$$u(x, y) = 0.5cos(\pi x)\sin(\pi y)$$

Domain: Star-shaped polygon, rescaled to unit square.

Boundary Condition: Dirichlet conditions on $\partial\Omega$, imposed using the exact solution.

Table 2.4: Results for TC-4: 2D Steady Linear Advection

Model	**Neurons**	$\boldsymbol{\lambda}$	**MAE**	**RMSE**	**Tcpu (s)**
Model 1	2000	–	1.17e-03	1.32e-03	27.92
Model 2	2000	1.00e-10	5.86e-07	7.24e-07	8.996
Model 3	1250	–	1.29e-04	1.83e-04	2314.26
Model 4	920	1.00e-10	5.17e-07	6.73e-07	1407.20



(a) PIELM vs. Exact Solution (b) Prediction Error

Figure 2.5: Solution and Error Plots for TC-4

In the two-dimensional advection scenario with complex domain geometry, Model 2 demonstrated superior performance in both accuracy and computation time, achieving sub-micro MAE and RMSE. Model 4 performed comparably in error, but with significantly higher computational cost. Model 3 struggled due to the intensive neu-

ron search in 2D space. Model 1 was stable but relatively inaccurate, reaffirming the advantage of regularization in higher dimensions.

TC-5: 2D Steady Linear Diffusion

Governing PDE:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = R(x, y), \quad (x, y) \in \Omega$$

Exact Solution:

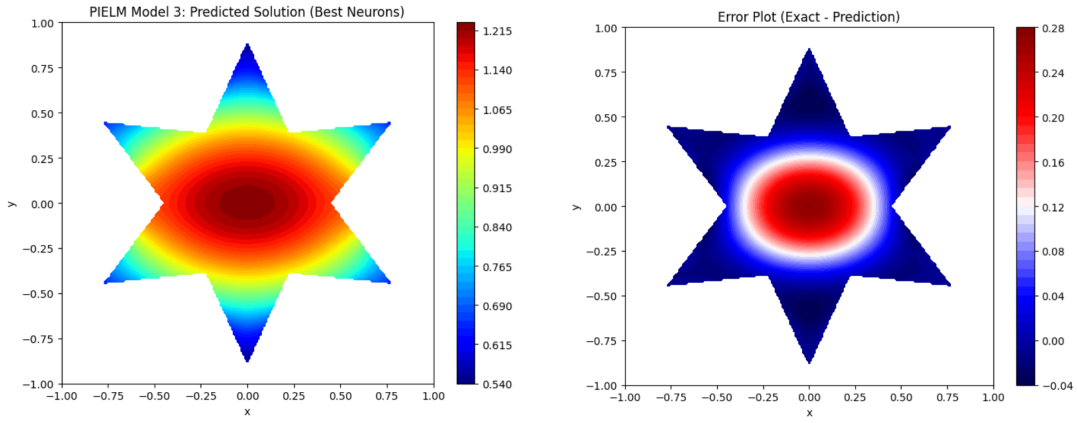
$$0.5 + e^{-(2x^2+4y^2)}$$

Domain: Star-shaped polygon, rescaled to unit square (same as TC-4)

Boundary Condition: Dirichlet conditions on $\partial\Omega$, imposed using the exact solution.

Table 2.5: Results for TC-5: 2D Steady Linear Diffusion

Model	**Neurons**	$\boldsymbol{\lambda}$	**MAE**	**RMSE**	**Tcpu (s)**
Model 1	2000	–	7.74e-02	1.13e-01	29.47
Model 2	2000	0.00e+00	7.78e-02	1.14e-01	13.75
Model 3	1110	–	7.36e-02	1.06e-01	2648.25
Model 4	1110	0.00e+00	7.36e-02	1.06e-01	9959.71



(a) PIELM vs. Exact Solution (b) Prediction Error

Figure 2.6: Solution and Error Plots for TC-5

In contrast to the advection case, all models struggled to achieve high accuracy in solving the 2D diffusion equation. Even with regularization or increased neurons, errors remained above 7% MAE. Model 3 showed a slight improvement with more neurons but came at a massive computational cost. Notably, Model 4 failed to outperform Model 2 in accuracy, despite its extended runtime, indicating diminishing returns in hyperparameter tuning for this case.

2.7 Key Improvement

While the Physics-Informed Extreme Learning Machine (PIELM) effectively integrates physical laws into the learning process by embedding differential equation residuals into the loss formulation, it remains a deterministic framework. This fixed-weight approach does not account for uncertainty in predictions, which can be critical in scientific and engineering applications where model robustness and confidence estimation are essential. To address this limitation, we adopt the Bayesian extensions of

PIELM, wherein the output weights are treated as random variables endowed with prior distributions. By adopting a probabilistic perspective, the Bayesian PIELM (BPIELM) framework enables not only point predictions but also quantification of predictive uncertainty, thus enhancing interpretability and reliability. The subsequent section discusses the Bayesian formulation, outlining its theoretical underpinnings and demonstrating its effectiveness.

Chapter 3

Bayesian Physics-Informed Extreme Learning Machine (BPIELM)

3.1 Introduction

The Bayesian Physics-Informed Extreme Learning Machine (BPIELM) is a fast and robust machine learning framework designed to solve linear partial differential equations (PDEs) under noisy and uncertain data conditions. It combines the efficiency of Extreme Learning Machines (ELMs) with the physical modeling capabilities of physics-informed approaches, while also incorporating Bayesian inference to quantify uncertainty in predictions.

Traditional methods like Physics-Informed Neural Networks (PINNs) have been

effective in embedding physical laws into neural networks by penalizing the PDE residuals during training. However, PINNs rely on computationally expensive gradient-based optimization and typically provide only point estimates, with limited or no capability to quantify uncertainty. On the other hand, Physics-Informed ELMs (PIELMs) offer a fast alternative by fixing the hidden-layer parameters randomly and solving a linear system to determine the output weights. While efficient, PIELMs are prone to overfitting when the training data is noisy and offer no probabilistic interpretation of the solution.

BPIELM addresses these limitations by adopting a Bayesian perspective. It treats the output-layer weights (and any unknown physical parameters in inverse problems) as random variables with a Gaussian prior. This enables the computation of a full posterior distribution over the solution space, yielding both the most probable solution (mean) and a measure of uncertainty (variance). The resulting predictions are not just single-valued but come with confidence intervals, making BPIELM particularly useful in real-world scenarios where data may be sparse, noisy, or uncertain.

In essence, BPIELM offers a unified, non-iterative framework for solving both forward and inverse linear PDE problems. It retains the speed and simplicity of ELMs while introducing principled uncertainty quantification through Bayesian linear regression. This makes it an appealing alternative to existing PINN-based methods, especially when balancing accuracy, robustness, and computational efficiency is critical.

3.2 Problem Setup

In this work, the authors aim to solve **linear partial differential equations (PDEs)** using the BPIELM framework. The problems considered fall under two main categories: *forward problems* and *inverse problems*, both of which may involve noisy data.

3.2.1 General Form of the PDE

The target PDE is assumed to have the following structure:

$$\mathcal{L}[u(x)] = f(x;\lambda), \quad x \in \Omega \tag{3.1}$$

$$\mathcal{B}[u(x)] = g(x), \quad x \in \partial\Omega \tag{3.2}$$

- $\mathcal{L}$: known linear differential operator (e.g., ∇^2, ∂_t, etc.)
- $u(x)$: unknown solution field to be estimated
- $f(x;\lambda)$: known source term, possibly dependent on unknown parameter(s) λ
- $\mathcal{B}$: boundary or initial condition operator
- $g(x)$: known boundary data

3.2.2 Measurement Model

The noisy data is modeled using Gaussian noise:

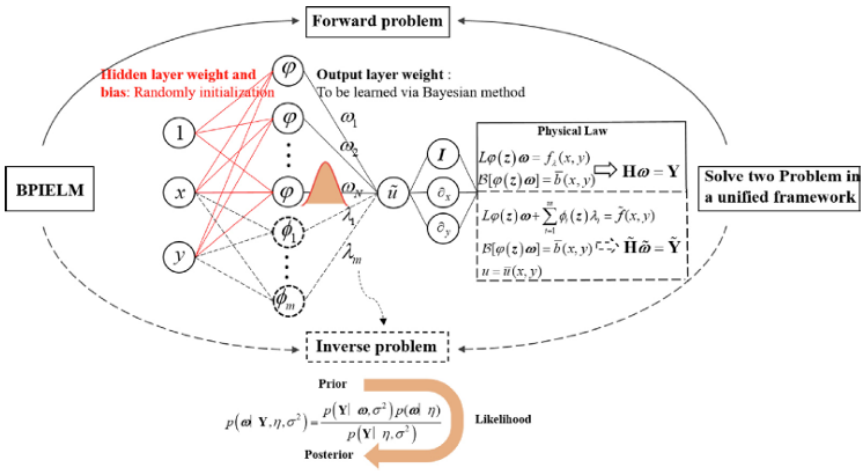


Figure 3.1: Conceptual flow of BPIELM with noisy data, where $p(\mathbf{w}, \boldsymbol{\theta})$ is the prior over output weights and problem parameters, $p(\mathbf{Y} \mid \mathbf{X}, \sigma^2)$ is the likelihood of measurements, and $p(\mathbf{w}, \boldsymbol{\theta} \mid \mathbf{Y}, \gamma, \sigma^2)$ is the posterior distribution.

- For boundary observations: $\hat{b}_i = b(x_i) + \varepsilon_i^b, \quad \varepsilon_i^b \sim \mathcal{N}(0, \sigma^2)$
- For interior solution measurements (inverse problem): $\hat{u}_i = u(x_i) + \varepsilon_i^u, \quad \varepsilon_i^u \sim \mathcal{N}(0, \sigma^2)$

3.2.3 Data Definitions

- **Boundary data:** $D_b = \left\{(x_i^b, \hat{b}_i)\right\}_{i=1}^{N_b}$
- **Interior data (for inverse problems):** $D_u = \{(x_i^u, \hat{u}_i)\}_{i=1}^{N_u}$

This formulation provides the foundation for applying the BPIELM methodology to efficiently and robustly solve PDE problems with uncertainty-aware solutions.

3.3 Methodology

This section outlines the core methodology used in the paper, with an emphasis on the development and implementation of the Bayesian Physics-Informed Extreme Learning Machine (BPIELM). The authors build upon existing techniques such as Physics-Informed Neural Networks (PINNs) and Physics-Informed Extreme Learning Machines (PIELMs) to formulate a fast and uncertainty-aware solver for linear PDEs.

3.3.1 Physics-Informed Neural Networks (PINNs)

PINNs solve PDEs by embedding the physical laws directly into the loss function of a neural network. The network attempts to approximate the solution $u(x)$ with parameters x, where:

$$Y = Hx \tag{3.3}$$

Here,

- Y: vector of known values (e.g., forcing term, boundary data, interior measurements)
- H: design matrix from PDE residuals, boundary constraints, or observed data
- x: weights and biases of the neural network (optimized via backpropagation)

PINNs minimize the residuals over N_f collocation points, N_b boundary points, and N_u observed data points via:

$$\mathcal{L}_{\text{PINN}}(x) = \frac{1}{N_f}\sum_{i=1}^{N_f}|H_f x - Y_f|^2 + \frac{1}{N_b}\sum_{i=1}^{N_b}|H_b x - Y_b|^2 + \frac{1}{N_u}\sum_{i=1}^{N_u}|H_u x - Y_u|^2 \quad (3.4)$$

PINNs rely on iterative optimization (e.g., gradient descent) and do not provide uncertainty estimates.

3.3.2 Physics-Informed Extreme Learning Machines (PIELMs)

PIELMs are a special case of single-layer feedforward networks where input weights and biases are randomly fixed, and only the output weights x are learned. The network's output is linear in x:

$$Y = Hx \quad (3.5)$$

Where:

- $H \in \mathbb{R}^{N^\star \times N}$: matrix formed from hidden-layer activations evaluated at input points and differentiated to encode PDEs and BCs
- Y: target vector containing the right-hand sides of the PDE, boundary values, or data
- x: output weights to be determined

PIELM solves for x using the Moore–Penrose pseudoinverse:

$$x = H^{\dagger}Y \tag{3.6}$$

PIELM is fast, but it is sensitive to noise and lacks any probabilistic interpretation.

3.3.3 Bayesian Physics-Informed Extreme Learning Machine (BPIELM)

Bayesian Physics-Informed Extreme Learning Machine (BPIELM) is an extension of the Physics-Informed Extreme Learning Machine (PIELM) that has a **Bayesian probabilistic framework** to estimate the output weights and quantify predictive uncertainty. While PIELM provides a deterministic solution, BPIELM captures the **uncertainty** due to limited or noisy data through **posterior distributions** over model parameters. This approach is especially useful for solving forward and inverse partial differential equation (PDE) problems where uncertainty quantification is crucial.

Bayesian Linear Model Setup

We begin by reformulating the PIELM setup into a **Bayesian linear regression** model. The goal is to model the target output data Y as a linear transformation of the hidden-layer features H, with additive noise:

$$Y = Hx + \varepsilon \tag{3.7}$$

Where:

- $Y \in \mathbb{R}^{N^*}$ is the observed data vector. This includes:
  - Physics residuals at collocation points
  - Boundary/initial condition values
  - Measurements (if any)

  All these are stacked into a single column vector of dimension $N^* \times 1$.
- $H \in \mathbb{R}^{N^* \times N}$ is the **design matrix**, constructed by combining:
  - Hidden layer activation outputs for the input data
  - Additional rows that encode physics-based constraints such as residuals from PDEs and boundary conditions

  Each row of H corresponds to a physical constraint or data point, and each column corresponds to one of the N hidden layer neurons.
- $x \in \mathbb{R}^N$ is the vector of unknown **output weights** (also called model parameters), which connect the hidden-layer neurons to the output. These are the parameters to be inferred.
- $\varepsilon \sim \mathcal{N}(0, r^2 I)$ is the additive Gaussian noise term with zero mean and **variance** r^2, where I is the identity matrix of size $N^* \times N^*$. This models the randomness or error in the observed data due to measurement noise or model discrepancy.

The formulation $Y = Hx + \varepsilon$ is equivalent to a standard **Bayesian linear regression model**, where the design matrix H is fixed, and inference is done on the weights x, given noisy observations Y.

Prior Distribution

Since we are working in a Bayesian framework, we place a **prior distribution** over the unknown weights x to encode our beliefs about them before seeing any data. A common and analytically convenient choice is a **zero-mean isotropic Gaussian prior**:

$$p(x|\gamma) = \mathcal{N}(0, \gamma^{-1}I) \tag{3.8}$$

Here:

- γ is the **precision** of the prior, which is the reciprocal of the variance. That is, if the prior variance is σ^2, then $\gamma = \frac{1}{\sigma^2}$.
- The identity matrix $I \in \mathbb{R}^{N\times N}$ implies that all weights are assumed a priori to be independent and identically distributed (i.i.d.).
- This prior reflects the belief that, before seeing the data, all weights are centered around 0 with equal uncertainty, and no weight is preferred over another.

The prior acts as a regularizer in the Bayesian learning process. Large values of γ correspond to stronger regularization (shrinking weights toward zero), while smaller values allow more flexible fitting.

Likelihood

Given the Bayesian model setup and the Gaussian noise assumption $\varepsilon \sim \mathcal{N}(0, r^2I)$, the **likelihood function** describes the probability of observing data Y, given weights

x and noise variance r^2. Since the noise is Gaussian, the likelihood also follows a Gaussian distribution:

$$p(Y|x, r^2) = \mathcal{N}(Hx, r^2 I) \tag{3.9}$$

This means that, conditioned on the weights x, the output vector Y is normally distributed around the mean Hx with covariance $r^2 I$.

- Hx represents the deterministic prediction from the model.
- $r^2 I$ captures the uncertainty due to measurement noise or unmodeled dynamics.

This likelihood expresses how well a particular choice of weights x explains the observed data Y, under the noise model.

Posterior Distribution

Having defined both the **prior** and the **likelihood**, we now apply **Bayes' Theorem** to obtain the posterior distribution over the weights x. This posterior captures our updated knowledge of the weights after observing data.

Bayes' Theorem

$$p(x \mid Y, \gamma, r^2) \propto p(Y \mid x, r^2) \cdot p(x \mid \gamma) \tag{3.10}$$

Where:

- $p(x \mid \gamma) = \mathcal{N}(0, \gamma^{-1} I)$ is the prior on the weights.
- $p(Y \mid x, r^2) = \mathcal{N}(Hx, r^2 I)$ is the likelihood of the observed data.

Since both the prior and the likelihood are Gaussian distributions, their product—and hence the posterior—is also Gaussian:

$$p(x \mid Y, \gamma, r^2) = \mathcal{N}(\mu, \Sigma) \tag{3.11}$$

We now derive the expressions for the posterior mean μ and covariance Σ.

Derivation of the Posterior

Step 1: Expand the log-posterior (up to a constant) We take the log of the unnormalized posterior:

$$\log p(x \mid Y) = \log p(Y \mid x) + \log p(x) + \text{const}$$

Insert the Gaussian forms:

$$\log p(Y \mid x) = -\frac{1}{2r^2} \|Y - Hx\|^2 + \text{const}, \quad \log p(x) = -\frac{\gamma}{2} x^T x + \text{const}$$

So the log-posterior becomes:

$$\log p(x \mid Y) = -\frac{1}{2r^2} (Y - Hx)^T (Y - Hx) - \frac{\gamma}{2} x^T x + \text{const}$$

Step 2: Expand the quadratic terms

$$(Y - Hx)^T(Y - Hx) = Y^TY - 2x^TH^TY + x^TH^THx$$

$$\Rightarrow \log p(x \mid Y) = -\frac{1}{2r^2}\left(Y^TY - 2x^TH^TY + x^TH^THx\right) - \frac{\gamma}{2}x^Tx + \text{const}$$

Drop constant terms and rearrange:

$$\log p(x \mid Y) = -\frac{1}{2}x^T\left(\frac{1}{r^2}H^TH + \gamma I\right)x + \frac{1}{r^2}x^TH^TY + \text{const}$$

Step 3: Recognize Gaussian canonical form This is a quadratic in x, and matches the log of a Gaussian:

$$\log p(x \mid Y) = -\frac{1}{2}(x^TAx - 2x^Tb) + \text{const}$$

Comparing terms:

$$A = \frac{1}{r^2}H^TH + \gamma I, \quad b = \frac{1}{r^2}H^TY$$

From the standard identity for Gaussian distributions:

$$\log p(x) \propto -\frac{1}{2}(x^TAx - 2x^Tb) \Rightarrow p(x) = \mathcal{N}(A^{-1}b, A^{-1})$$

We therefore obtain:

$$\Sigma = \left(\gamma I + \frac{1}{r^2} H^T H\right)^{-1} \tag{3.12}$$

$$\mu = \Sigma \cdot \left(\frac{1}{r^2} H^T Y\right) = \frac{1}{r^2} \Sigma H^T Y \tag{3.13}$$

Interpretation

- $\mu \in \mathbb{R}^N$ is the posterior **mean** of the weight vector. It gives the most probable value of x after observing the data.
- $\Sigma \in \mathbb{R}^{N \times N}$ is the posterior **covariance matrix**, quantifying uncertainty in the inferred weights.
- Larger values of the prior precision γ enforce stronger regularization (shrinking the weights toward zero).
- Smaller values of the noise variance r^2 make the model trust the data more and reduce posterior uncertainty.
- The prior serves as a stabilizer in case $H^T H$ is ill-conditioned or underdetermined.

This posterior distribution not only provides a point estimate of the weights but also enables uncertainty quantification, which is essential in applications involving noisy data or ill-posed PDEs.

Prediction at a New Point

To make a prediction at a new input point $\mathbf{x}_* \in \mathbb{R}^d$, we utilize the posterior distribution over the weights $x \sim \mathcal{N}(\mu, \Sigma)$, obtained during the training phase of BPIELM.

Let $h_* \in \mathbb{R}^{1\times N}$ be the hidden-layer feature vector evaluated at $\mathbf{x}_*$, computed as:

$$h_* = \left[\sigma(w_1^T\mathbf{x}_* + b_1),\, \sigma(w_2^T\mathbf{x}_* + b_2), \ldots, \sigma(w_N^T\mathbf{x}_* + b_N)\right]$$

where:

- $\sigma(\cdot)$ is the nonlinear activation function (e.g., `tanh` or `sigmoid`),
- $w_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$ are the randomly generated input weights and biases of the i-th hidden neuron,
- N is the total number of hidden neurons in the network.

The model assumes the predictive output at $\mathbf{x}_*$ to follow:

$$y_* = h_* x + \varepsilon_*, \quad \varepsilon_* \sim \mathcal{N}(0, r^2)$$

Since both x and ε_* are Gaussian random variables, the resulting predictive distribution of y_* is also Gaussian:

$$y_* \sim \mathcal{N}(\mu_*, \sigma_*^2) \tag{3.18}$$

The closed-form expressions for the predictive mean and variance are given by:

$$\mu_* = h_* \mu \tag{3.7}$$

$$\sigma_*^2 = h_* \Sigma h_*^T + r^2 \tag{3.19}$$

Interpretation:

- μ_* represents the predictive mean at $\mathbf{x}_*$, i.e., the most probable output according to the Bayesian model.
- σ_*^2 quantifies the predictive uncertainty, incorporating both model (epistemic) uncertainty via $h_* \Sigma h_*^T$ and observational noise via r^2.
- A smaller value of σ_*^2 indicates a high-confidence prediction, typically corresponding to regions well-covered by the training data.
- A higher value of σ_*^2 suggests greater model uncertainty due to sparse training data or significant noise.

The uncertainty-aware prediction facilitates the construction of pointwise confidence intervals:

$$y_* \in [\mu_* - 2\sigma_*,\ \mu_* + 2\sigma_*] \quad \text{(approximately 95\% confidence interval)}$$

This formulation is particularly useful in scientific and engineering applications where knowing the confidence in model outputs is as important as the predictions themselves.

Advantages of BPIELM

- **Computational Efficiency:** BPIELM retains the rapid training capabilities of Extreme Learning Machines by reducing the training process to closed-form matrix operations, avoiding costly iterative optimization.
- **Uncertainty Quantification:** Unlike traditional ELMs or deterministic PIELMs, BPIELM provides a principled way to quantify predictive uncertainty through posterior variance, allowing the generation of confidence intervals and error bounds.
- **Noise Robustness:** The Bayesian treatment of weights and noise enables BPIELM to handle noisy data more gracefully, leading to improved generalization performance and reduced overfitting in ill-posed or underdetermined problems.
- **General Applicability:** BPIELM is versatile and can be applied to a wide range of PDE problems, including both forward and inverse settings, with or without noisy measurements.
- **Physics-Consistency:** By incorporating physical constraints via PDE residuals and boundary conditions into the likelihood, BPIELM ensures that the learned solutions are not only data-driven but also physically meaningful.

Bayesian Physics-Informed Extreme Learning Machines (BPIELMs) thus synergize the computational efficiency of ELMs, the expressiveness of physics-informed learning, and the uncertainty quantification of Bayesian inference. This makes BPIELM particularly well-suited for scientific machine learning tasks involving partial differential equations and noisy data.

3.4 Numerical Results

This section presents the application of the Bayesian Physics-Informed Extreme Learning Machine (BPIELM) on a set of benchmark partial differential equations (PDEs) for solving forward problems. The selected equations include the two-dimensional Poisson equation, the advection equation in one space–one time domain, and the diffusion equation in one space–one time domain. The performance of BPIELM is compared against the Physics-Informed Extreme Learning Machine (PIELM) and Physics-Informed Neural Network (PINN) in terms of predictive accuracy and robustness to noisy boundary conditions.

3.4.1 Poisson Equation

The two-dimensional Poisson equation considered is:

$$u_{xx} + u_{yy} = \left(16x^2 + 64y^2 - 12\right) e^{-2x^2-4y^2}, \quad (x, y) \in \Omega \tag{3.8}$$

with the exact solution:

$$u(x, y) = \frac{1}{2} + e^{-2x^2-4y^2} \tag{3.9}$$

The domain Ω is a butterfly-shaped region described parametrically using $q(\theta) = 1 + \cos(\theta)\sin(4\theta)$, where $\theta \in [0, 2\pi]$.

Only boundary measurements with Gaussian noise are available. The governing equation and boundary data are encoded into a linear system using both the PIELM and BPIELM frameworks. BPIELM assumes a Gaussian prior over the output weights and employs Bayesian inference to estimate posterior distributions and

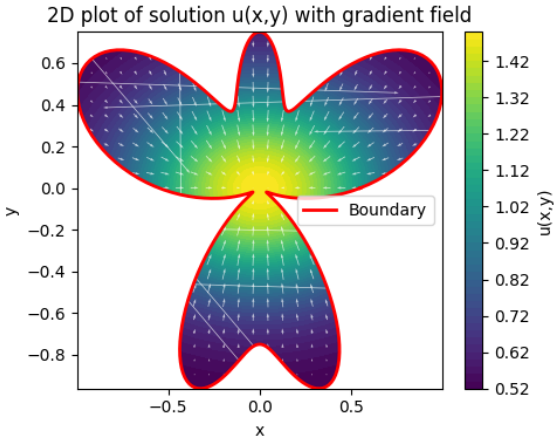


Figure 3.2: BPIELM Solution plot

predictive uncertainty.

BPIELM produces more accurate results than PIELM, particularly in sensor-sparse regions. The standard deviation maps from BPIELM align well with error distributions, indicating reliable uncertainty quantification. BPIELM remains stable across varying noise levels and sensor counts, whereas PIELM suffers from instability and overfitting.

Table 3.1: Comparison of BPIELM Models for TC-1 (2D Poisson Equation)

Model	Neurons	Error (MAE)	RMSE	Tcpu (seconds)
Model 1 (Base)	440	4.804464×10^{-3}	5.726068×10^{-3}	1.616797
Model 2 (Ridge)	440	6.731859×10^{-4}	1.142675×10^{-3}	1.528648
Model 3 (Neuron Tuning)	175	1.591316×10^{-5}	3.496201×10^{-5}	41.575
Model 4 (Neuron + Ridge)	975	3.545034×10^{-5}	6.744117×10^{-5}	279.9179

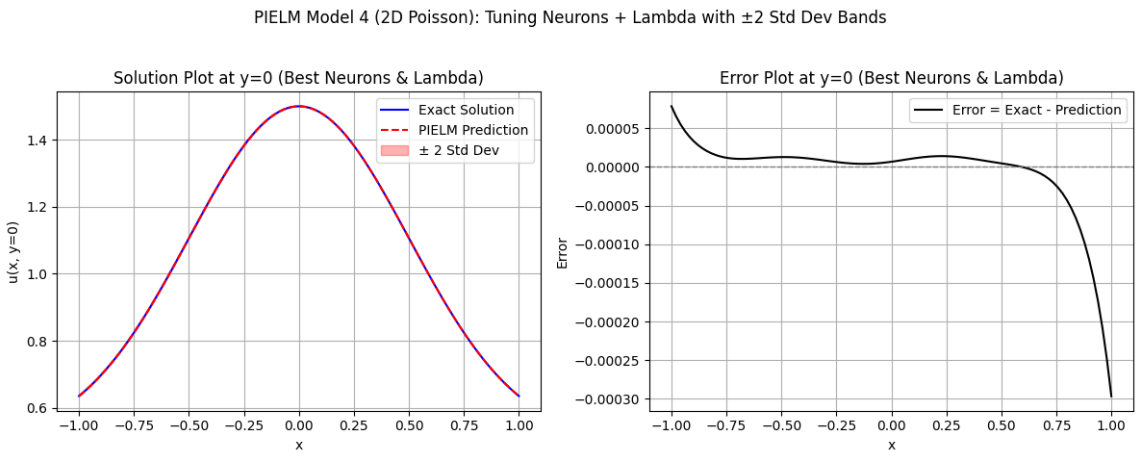


Figure 3.3: BPIELM predicted vs exact solution , Error plot

3.4.2 Advection Equation

The one-dimensional advection equation with periodic boundary conditions is:

$$\frac{\partial u}{\partial t} - c\frac{\partial u}{\partial x} = 0, \quad (x,t) \in [0,1] \times [0,1], \quad c = -2 \tag{3.10}$$

with conditions:

$$u(x_1,t) = u(x_2,t), \quad u(x,0) = h(x) \tag{3.11}$$

The exact solution is a periodic traveling wave given by:

$$u(x,t) = 2\,\mathrm{sech}\left(3\left(\frac{1}{2}+n\right)\right), \quad n = \mathrm{mod}\left(\frac{x - x_0 + ct + L/2}{L}, 1\right) \tag{3.12}$$

where $L = x_2 - x_1$ and $x_0 = 0.5$.

Boundary measurements are assumed to be noisy and sparse. BPIELM constructs a physics-constrained system and applies Bayesian inference to estimate the solution and associated uncertainty.

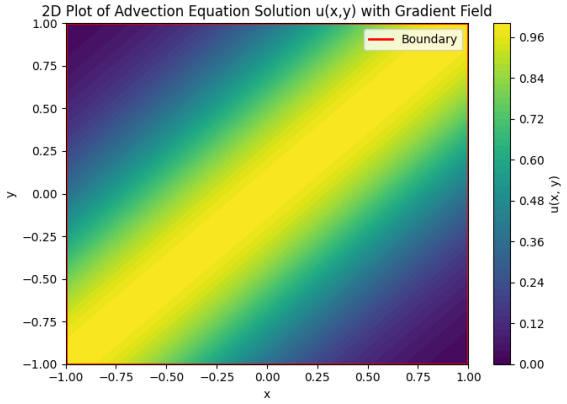


Figure 3.4: BPIELM Solution plot

Under both low and high noise conditions, BPIELM outperforms PIELM in predictive accuracy. Its uncertainty estimates correlate with regions of higher error. BPIELM is also less sensitive to neuron count and sensor sparsity, whereas PIELM shows instability and overfitting tendencies.

Table 3.2: Comparison of BPIELM Models for TC-2 (2D Steady Diffusion)

Model	Neurons	Error (MAE)	RMSE	Tcpu (seconds)
Model 1 (Base)	440	1.067236×10^{-4}	1.37819×10^{-4}	2.940964
Model 2 (Ridge)	440	1.213544×10^{-3}	1.265754×10^{-3}	0.991076
Model 3 (Neuron Tuning)	125	5.284994×10^{-7}	9.739365×10^{-7}	38.06
Model 4 (Neuron + Ridge))	350	7.309398×10^{-7}	8.936025×10^{-7}	59.50

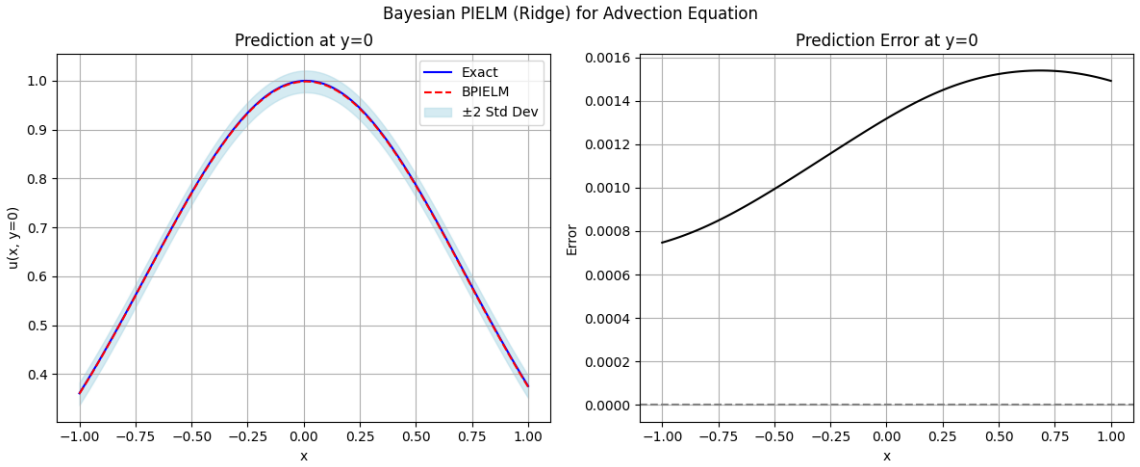


Figure 3.5: BPIELM predicted vs exact solution , Error plot

3.4.3 Diffusion Equation

The time-dependent one-dimensional diffusion equation is:

$$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} = f(x,t), \quad (x,t) \in [0,1] \times [0,2], \quad \nu = 0.01 \tag{3.13}$$

The source term $f(x,t)$ is designed such that the exact solution is:

$$u(x,t) = \left(2\cos\left(\pi x + \frac{\pi}{5}\right) + \frac{3}{2}\cos\left(2\pi x - \frac{3\pi}{5}\right)\right) \times \left(2\cos\left(\pi t + \frac{\pi}{5}\right) + \frac{3}{2}\cos\left(2\pi t - \frac{3\pi}{5}\right)\right) \tag{3.14}$$

Noisy measurements of boundary and initial conditions are assumed. The problem is discretized in both space and time, and solved using BPIELM by encoding the governing equations and constraints into a linear system with Bayesian output weight inference.

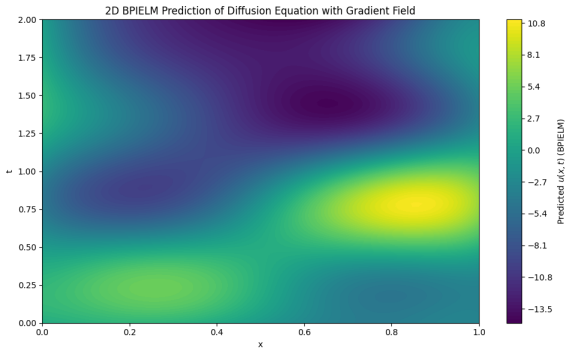


Figure 3.6: BPIELM Solution plot

BPIELM significantly outperforms PIELM in accuracy, especially under high noise. The uncertainty quantification provided by BPIELM reflects prediction confidence across the domain. Unlike PIELM, which becomes unstable at high neuron counts, BPIELM remains robust and its posterior weight distributions remain consistent between noisy and noiseless settings.

Table 3.3: Comparison of BPIELM Models for TC-3 (2D Steady Advection)

Model	Neurons	Error (MAE)	RMSE	Tcpu (seconds)
Model 1 (Base)	180	2.356142×10^{-2}	3.493180×10^{-2}	0.228198
Model 2 (Ridge)	180	2.371090×10^{-2}	3.530149×10^{-2}	0.476903
Model 3 (Neuron Tuning)	450	7.032718×10^{-3}	8.900169×10^{-3}	3.227
Model 4 (Neuron + Ridge)	950	5.288599×10^{-3}	7.198373×10^{-3}	21.475872

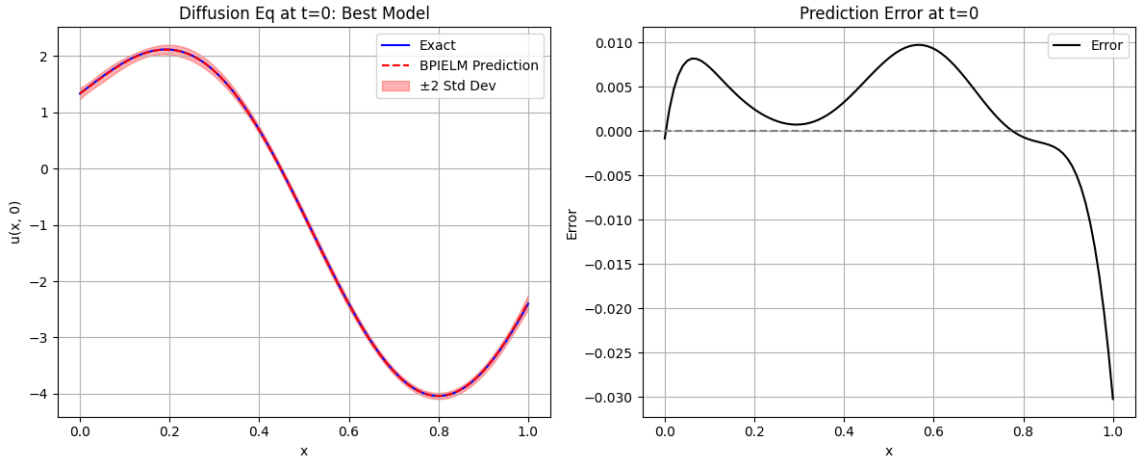


Figure 3.7: BPIELM predicted vs exact solution , Error plot

Code Availability

For a detailed implementation of the methodologies discussed, the complete source code for both PIELM and BPIELM has been made available in our public GitHub repository: github.com/MEGHAPANKAJ/Literature-Review.

Bibliography

Chen, C. and Zhulin Liu. "Broad Learning System: An Effective and Efficient Incremental Learning System Without the Need for Deep Architecture". In: *IEEE Transactions on Neural Networks and Learning Systems* PP (July 2017), pp. 1–15. DOI: 10.1109/TNNLS.2017.2716952.

Dwivedi, Vikas and Balaji Srinivasan. *Physics Informed Extreme Learning Machine (PIELM) – A rapid method for the numerical solution of partial differential equations*. July 2019. DOI: 10.48550/arXiv.1907.03507.

Liu, Xu et al. "Bayesian Physics-Informed Extreme Learning Machine for Forward and Inverse PDE Problems with Noisy Data". In: *Neurocomputing* 549 (June 2023), p. 126425. DOI: 10.1016/j.neucom.2023.126425.