Physics-Informed Neural Networks for Approximately Solving Partial Differential Equations

Thi Thanh Mai Ta[1], Thi Huong Giang Nguyen[1,*]

[1]*Hanoi University of Science and Technology, Hanoi, Vietnam*

[*]*giang.nth227043@sis.hust.edu.vn*

Abstract

This study explores the use of Physics-Informed Neural Networks to approximate solutions for Partial Differential Equations problems. Physics-Informed Neural Networks leverage the nonlinear approximation capability of deep neural networks and incorporate physical constraints directly into the training process. The proposed model is based on a fully connected feedforward neural network, trained using a composite loss function that accounts for Partial Differential Equation residuals, initial conditions, and boundary conditions. Optimization is performed using a combination of Adam and L-BFGS algorithms. To validate the approach, we consider benchmark heat transfer and linear elasticity problems with regular geometries and known analytical solutions. Experimental results demonstrate that Physics-Informed Neural Networks achieve low error and fast convergence, even with limited training data. The findings highlight the potential of Physics-Informed Neural Networks as a reliable method for solving Partial Differential Equations, while also pointing out challenges that remain in extending the approach to high-dimensional and non-smooth problems.

Keywords: Physics-Informed Neural Networks, Partial Differential Equations, Feed forward neural network, Heat equation, Elasticity equation.

1. Introduction

Partial Differential Equations (PDEs) are fundamental in modeling numerous physical phenomena such as heat conduction and elasticity. These problems are typically defined over a spatial domain with initial and boundary conditions that reflect physical constraints of the system. [1] The PDE problem considered in this work takes the general form:

$$\mathcal{L}(u(t,x)) = f, \quad (t,x) \in [0,T]\times\Omega, \quad \Omega \subset \mathbb{R}^d, \quad (1)$$

subject to the initial and boundary conditions:

$$u(t_0,x) = u_0(x), \quad x \in \Omega,$$

$$u(t,x) = g(t,x), \quad (t,x) \in [0,T]\times\partial\Omega.$$

Classical numerical methods such as the Finite Element Method (FEM) and the Finite Difference Method (FDM) have been extensively utilized to approximate solutions of PDEs. However, these methods face significant computational challenges when addressing high-dimensional problems or complex geometries, a difficulty commonly referred to as the ”curse of dimensionality” [2, 3] .

To address such challenges, deep learning techniques have been proposed to approximate PDE solutions without relying on dense meshes. A notable example is the Deep Galerkin Method (DGM) introduced by Sirignano and Spiliopoulos, which trains deep neural networks to minimize the residual of PDE at randomly sampled points instead of employing traditional mesh grids [2] . In addition, other approaches use neural networks in a data-driven manner, learning mappings from input coordinates to known PDE solutions [4] . However, these conventional neural network-based methods suffer from critical limitations. First, they often exclude physical laws from the learning process, making them prone to overfitting and reliant on large quantities of training data for generalization [5] .

Moreover, traditional deep learning models struggle to impose initial and boundary conditions precisely. These constraints are usually enforced indirectly through penalty terms in the loss function, which can result in unstable or inaccurate solutions [2] . This becomes particularly problematic in heat transfer problems that require energy conservation or in models of elasticity that require equilibrium laws. Furthermore, neural network models can be difficult to train and may fail to converge in problems involving multiscale behaviors or oscillatory solutions, such as heat conduction in heterogeneous materials or stress analysis in composite structures [6].

To overcome these limitations, PINNs have been proposed as a potential alternative. PINNs incorporate physical laws, expressed as differential equations, directly into the loss function of the neural network. This formulation leverages automatic differentiation to compute PDE residuals and naturally enforces both the governing equations and boundary conditions across the domain. As a result, PINNs are capable of solving forward and inverse PDE problems with high accuracy, even in settings with sparse or noisy data. Recent studies have shown that PINNs can achieve physically consistent and accurate solutions in heat transfer and elasticity problems while requiring significantly fewer data than conventional methods [5, 7].

In this paper, we solve the aforementioned PDE using

PINN approach. Specifically, the solution $u(t, x)$ is approximated by a neural network

$$\hat{u}(t, x) = \mathrm{NN}(t, x; \theta),$$

where θ denotes the network parameters. The network is trained by minimizing a composite loss function that combines the error from data (e.g., initial and boundary conditions) and the residual of the PDE. This approach enables the network to learn an approximate solution while maintaining consistency with the physical structure of the problem.

The remainder of this paper is organized as follows. In Section 2, we present the formulation of PINNs, including the feedforward neural network architecture, backpropagation algorithm. Section 3 introduces our problem-solving framework: overview, loss function design, dataset and training algorithm. Section 4 presents numerical experiments with two types of PDEs: the heat equation and the linear elasticity equation. Finally, Section 5 provides concluding remarks and discusses potential future directions.

2. Background

2.1. Feedforward Neural Network

A Feedforward Neural Network (FNN) is a type of artificial neural network where information moves in one direction from the input layer, through one or more hidden layers, to the output layer without forming cycles or loops. The structure of an FNN is illustrated in Figure 1.

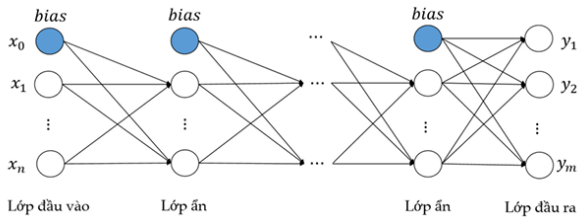

Fig. 1. Structure of a Feedforward Neural Network [8]

The main components of an FNN include:

- **Input Layer**: Receives the input data.
- **Hidden Layers**: One or more intermediate layers that transform the input through nonlinear computations.
- **Output Layer**: Provides the final prediction or approximation.

Each neuron computes its output as a nonlinear transformation of the weighted sum of its inputs. Mathematically, the operation is given by:

$$z_j = g\left(\sum_i w_{ji} x_i + b_j\right), \tag{2}$$

where x_i are the inputs, w_{ji} are the weights, b_j is the bias, and $g(\cdot)$ is the activation function. Figure 2 illustrates this processing unit.

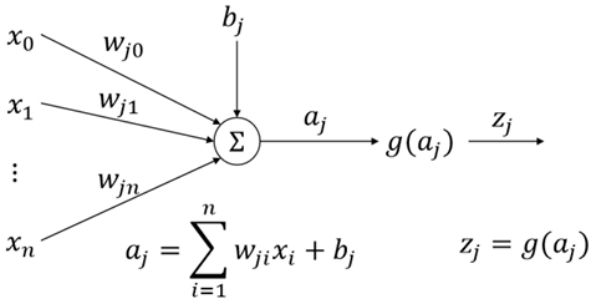

Fig. 2. Computation in a Single Neuron [8]

2.2. Backpropagation

Backpropagation is the core algorithm for training neural networks. It computes the gradient of the loss function with respect to the network's parameters using the chain rule of calculus. These gradients are then used to update weights in the direction that minimizes the loss, typically via gradient descent. This enables the network to learn mappings from inputs to outputs over time.

3. Physics-Informed Neural Networks

3.1. Overview

Physics-Informed Neural Networks (PINNs) are a class of deep learning models that leverage known physical laws - typically expressed as partial differential equations (PDEs) - to guide the training process. Unlike traditional data-driven models, which rely heavily on large datasets, PINNs incorporate domain knowledge by embedding the governing equations directly into the loss function. This hybrid learning approach enhances model generalization, particularly in scenarios with scarce or noisy data.

The core idea of PINNs is to approximate the solution $u(t, x)$ of a PDE by a neural network $u_\theta(t, x)$, parameterized by θ. The network is trained not only to fit any available data but also to satisfy the underlying PDE and its associated boundary and initial conditions. By minimizing a composite loss function that includes physics-based residuals, PINNs effectively enforce physical consistency across the input domain.

PINNs are particularly useful in scientific machine learning tasks involving forward modeling, inverse problems, and system identification. Their compatibility with automatic differentiation frameworks (e.g., TensorFlow, PyTorch) allows for the efficient computation of derivatives, making them a flexible and powerful modeling tool across diverse application domains.

3.2. Loss Function Formulation

To train a Physics-Informed Neural Network, we define a composite loss function that incorporates both data fidelity and physical law enforcement. The total loss is expressed as:

$$L = \lambda_p L_{\text{PDE}} + \lambda_b L_{\text{BC/IC}} + \lambda_d L_{\text{data}}, \tag{3}$$

where λ_p , λ_{b}, and λ_{d} are non-negative scalar weights that balance the contributions of each loss term. [9]

- **PDE Loss** $L_{\mathbf{PDE}}$: This term enforces that the neural network approximately satisfies the PDE over a set of collocation points $\{(t_i, x_i)\}$. Specifically, with (1) the residual is computed as:

$$L_{\text{PDE}} = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathcal{L}(u_\theta(t_i, x_i)) - f(t_i, x_i)|^2,$$

  where N_r is the number of collocation points in the domain.

- **Boundary/Initial Condition Loss** $L_{\mathbf{BC/IC}}$: This term enforces the satisfaction of boundary and initial conditions by penalizing deviations between the network output and prescribed conditions. For example:

$$L_{\text{BC/IC}} = \frac{1}{N_b} \sum_{i=1}^{N_b} |u_\theta(t_i, x_i) - g(t_i, x_i)|^2,$$

  where $g(t, x)$ represents the exact boundary or initial values, and N_b is the number of such sample points.

- **Data Loss** $L_{\mathbf{data}}$: When observational or simulated data is available, this term measures the discrepancy between network predictions and known data points:

$$L_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left| u_\theta(t_i, x_i) - u^{\text{data}}(t_i, x_i) \right|^2.$$

The loss function is optimized using gradient-based methods. Automatic differentiation tools are employed to compute all required derivatives of $u_\theta(t, x)$ with respect to time and space variables, enabling the accurate evaluation of the differential operator $\mathcal{L}$. The choice of weights λ_i may be fixed or dynamically adjusted (e.g., via curriculum learning or adaptive weighting) to balance convergence across all loss terms. In this work, for simplicity and to avoid additional hyperparameter tuning, we set all weights to 1, i.e., $\lambda_{\text{PDE}} = \lambda_{\text{BC/IC}} = \lambda_{\text{data}} = 1$.

This formulation provides a strong potential framework for learning physically consistent solutions, even in the absence of dense labeled datasets.

3.3. Data Sampling and Preprocessing

To ensure accurate modeling of both the solution and its derivatives, data points are sampled from the interior domain and its boundary. A uniform grid is employed to balance point density, enabling stable learning of gradients and effective enforcement of boundary conditions [10]. This project uses a square grid with balanced distribution between interior and boundary collocation points.

3.4. Training Algorithm

The training process of a PINNs involves minimizing a composite loss that enforces agreement with physical laws and observed data. The procedure is illustrated in figure 3 and can be summarized as follows:

1) **Initialize** the neural network parameters θ using standard initialization methods (e.g., Xavier initialization).
2) **Sample training points** from the spatial-temporal domain:
   - *Collocation points* $\{(t_R^j, x_R^j)\}_{j=1}^{N_R} \in [0, T] \times \Omega$, used to enforce the PDE.
   - *Initial condition points* $\{(t_i^j, x_i^j)\}_{j=1}^{N_i}$, typically taken at $t = 0$.
   - *Boundary condition points* $\{(t_b^j, x_b^j)\}_{j=1}^{N_b} \in [0, T] \times \partial\Omega$.
3) **Evaluate the network** $\hat{u}_\theta(t, x)$ at all sampled points to obtain the predicted solution.
4) **Construct the loss function**: In this paper, we focus solely on solving PDE problem. Therefore, data related to the inverse problem, L_{Data} in equation (3), is excluded. Our goal is to ensure that the solution satisfies both the PDE and the boundary/initial conditions.

$$L_{\text{BC/IC}} = \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \hat{u}_\theta(t_b^i, x_b^i) - u_b^i \right|^2 + \frac{1}{N_i} \sum_{j=1}^{N_i} \left| \hat{u}_\theta(t_i^j, x_i^j) - u_i^j \right|^2,$$

$$L_{\text{PDE}} = \frac{1}{N_R} \sum_{i=1}^{N_R} \left| \mathcal{L}(\hat{u}_\theta)(t_R^i, x_R^i) - f(t_R^i, x_R^i) \right|^2,$$

$$L = L_{\text{PDE}} + L_{\text{BC/IC}}.$$

5) **Compute gradients** $\nabla_\theta L$ using automatic differentiation provided by the deep learning framework.
6) **Update the parameters** θ using a gradient-based optimizer such as Adam (for first-order updates) or L-BFGS (for more precise convergence).
7) **Iterate** the process until convergence criteria are satisfied (e.g., loss threshold, maximum epochs, or validation error).

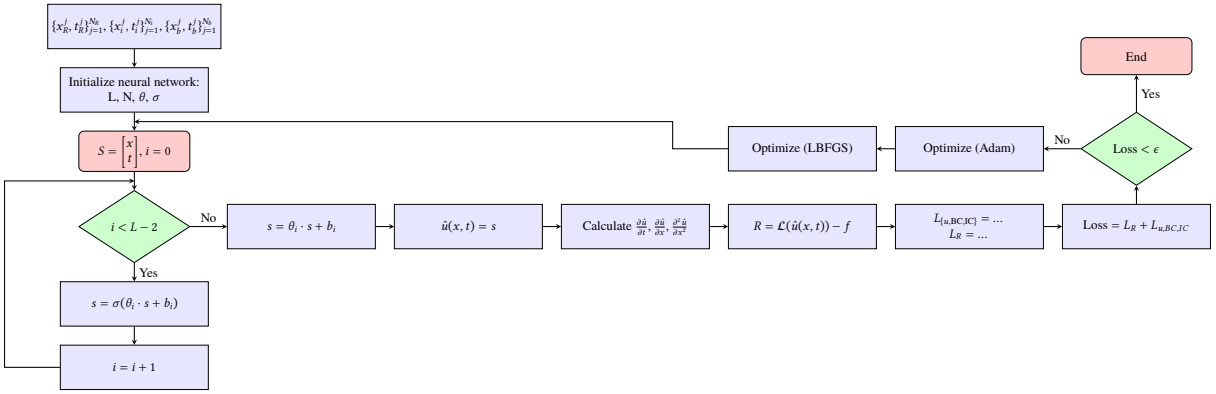

Fig. 3. Flowchart of the training process of the physics-informed neural network

4. Experimental results

4.1. Heat Equation

The heat equation is a fundamental partial differential equation that describes the distribution of temperature in a medium over time.

4.1.1. Steady-State Heat Equation

The steady-state heat equation describes the situation in wich the temperature in a medium does not change over time. [1] In the general case with multiple spatial dimensions, the equation takes the form:

$$-\Delta u(x) = f(x), \quad x \in \Omega \subset \mathbb{R}^n$$

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

Where:

$u(x)$ The temperature distribution.

$f(x)$ The source term.

Ω The spatial domain.

$\Delta = \sum_{i=1}^{n} \frac{\partial^2 u}{\partial x_i^2}$ Laplace operator.

The specific problem we investigate involves the domain $\Omega = [0, 1]^2$ with:

$$f(x) = 5\pi^2 \sin(2\pi x_1) \sin(\pi x_2)$$
$$g(x) = 0$$

The analytical solution of the problem is:

$$u(x) = \sin(2\pi x_1) \sin(\pi x_2)$$

Training data:

- Number of points in the domain Ω: 1521 points (evenly distributed on the grid)
- Number of points on the boundary $\partial\Omega$: 400 points

Testing data:

- Number of points in the domain Ω: 350 points
- Number of points on the boundary $\partial\Omega$: 150 points

The networks use the `tanh()` activation function in each hidden layer.
Due to the advantages of both the Adam and L-BFGS optimizers, we employ both in the training process. First, the Adam optimizer is used to reduce the loss to a reasonable level, after which L-BFGS is applied for further training. [11]

Table 1. Numerical results for the steady-state Poisson problem

L	**N**	**Loss**	**L_2 Error**	**Δt**
3	16	4.556×10^{-4}	8.615×10^{-6}	6.05×10^{-3}
3	32	6.464×10^{-5}	1.241×10^{-6}	8.47×10^{-3}
3	64	6.172×10^{-5}	1.834×10^{-6}	1.20×10^{-2}
5	16	1.24×10^{-4}	2.146×10^{-5}	1.45×10^{-2}
5	32	3.236×10^{-6}	4.27×10^{-6}	1.88×10^{-2}
5	64	1.798×10^{-5}	1.668×10^{-6}	3.88×10^{-2}
7	16	7.602×10^{-5}	1.106×10^{-5}	1.93×10^{-2}
7	32	4.502×10^{-5}	3.719×10^{-6}	3.36×10^{-2}
7	64	1.404×10^{-5}	1.711×10^{-6}	6.95×10^{-2}

L Number of layers in the neural network

N Number of neurons in each hidden layer

Δt Time per epoch

Remarks from Table 1:

Effect of network depth (L): Increasing the depth L of the neural network (with a fixed number of hidden nodes N) generally improves the solution quality in terms of both Loss and L_2 error. Effect of network width (N): With fixed L, increasing the number of hidden nodes N also leads to better results, although the improvements tend to diminish. Trade-off between

accuracy and training time: Larger models (with higher L and N) typically yield more accurate results, but require significantly longer training time Δt.

Diminishing performance gain: Beyond a certain network size, improvements in accuracy become marginal or may even decline due to overfitting or training instability. Balanced model: Overall, the model with $L = 5$, $N = 64$ provides a good balance between high accuracy and reasonable training time.

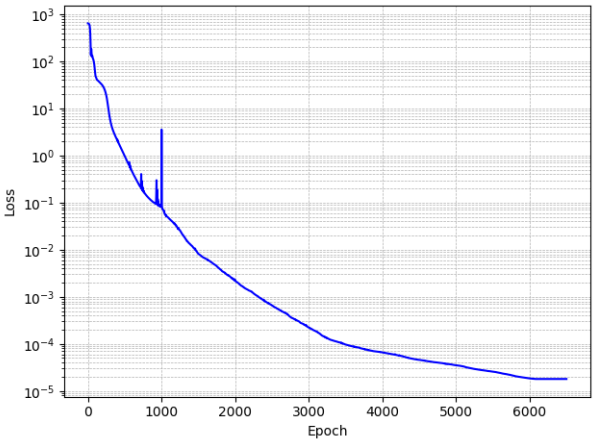

Fig. 4. Best case: L = 5, N = 64

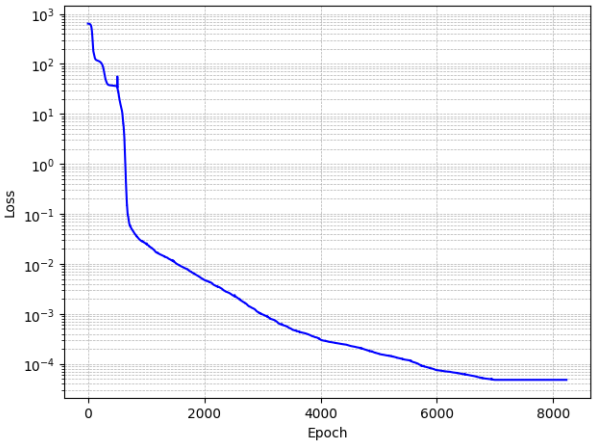

Fig. 5. Worst case: L=5, N=16

4.2. Elasticity Equation

We investigate the static deformation of a two-dimensional, linearly elastic beam occupying the rectangular domain $\Omega = [0, 1] \times [0, 0.15]$, subjected to vertical body forces. The material is assumed to be homogeneous and isotropic, and it follows the constitutive laws of linear elasticity. The Young's modulus is set to $E = 1$, and the Poisson's ratio is $\nu = 0.3$. From these values, the Lamé parameters are computed as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}, \tag{4}$$

where λ and μ denote the first and second Lamé parameters, respectively.

The governing equations are those of static linear elasticity, expressed as the equilibrium condition in the absence of inertial effects:

$$-\nabla \cdot \boldsymbol{\sigma} = \mathbf{f}, \tag{5}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{f}$ is the body force per unit volume. The stress-strain relationship, according to Hooke's law, is given by

$$\boldsymbol{\sigma} = \lambda \operatorname{tr}(\boldsymbol{\varepsilon})\mathbf{I} + 2\mu\boldsymbol{\varepsilon}, \tag{6}$$

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^\top), \tag{7}$$

where $\boldsymbol{\varepsilon}$ is the linearized strain tensor and $\mathbf{u} = (u, v)^\top$ is the displacement vector field. [12]

In this paper, we examine two distinct cases of boundary conditions. In both cases, the boundary conditions are incorporated into the framework of Physics-Informed Neural Networks (PINNs). Dirichlet conditions are directly enforced on the predicted displacement field, whereas Neumann conditions are imposed by computing the corresponding stress components through automatic differentiation of the learned displacement field. To represent the displacement vector $\mathbf{u}$, two separate neural networks are employed: one for approximating the horizontal component u and another for the vertical component v. Both networks take the spatial coordinates (x, y) as input and are trained jointly to ensure consistency across the displacement field.

4.2.1. Elasticity Equation with One Fixed End

In the first case, the beam is modeled as a cantilever structure subjected to a localized shear load. The body force is specified as $\mathbf{f} = (0, -0.1)^\top$. The left boundary at $x = 0$ is fully clamped, enforcing zero displacement in both directions ($u = v = 0$). The right boundary at $x = 1$ is mostly traction-free, i.e., $\sigma_{xx} = \sigma_{xy} = 0$, except for a narrow vertical strip located at the center of the edge, where a localized shear load is applied with $\sigma_{xx} = 0$ and $\sigma_{xy} = 0.1$. Along the top and bottom boundaries, $y = 0.15$ and $y = 0$, the beam is stress-free, meaning that $\sigma_{yy} = \sigma_{xy} = 0$. This setup corresponds to a classical cantilever beam subjected to a concentrated shear force at its free end. [12]

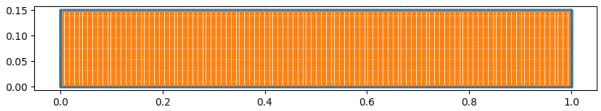

Fig. 6. Initial domain (case 1)

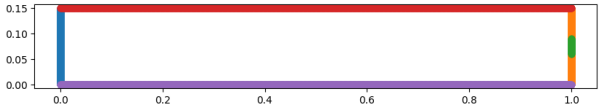

Fig. 7. Simulation of the geometric domain of a 2D rectangular beam with boundary conditions (case 1).

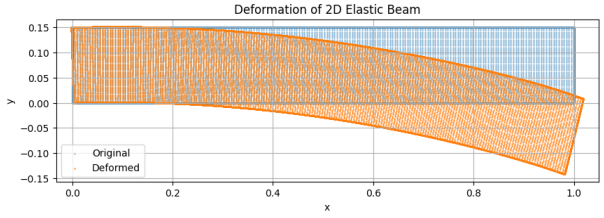

Fig. 8. Deformed domain (case 1)

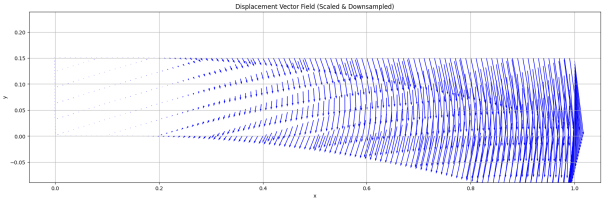

Fig. 9. Deformation vector field (case 1)

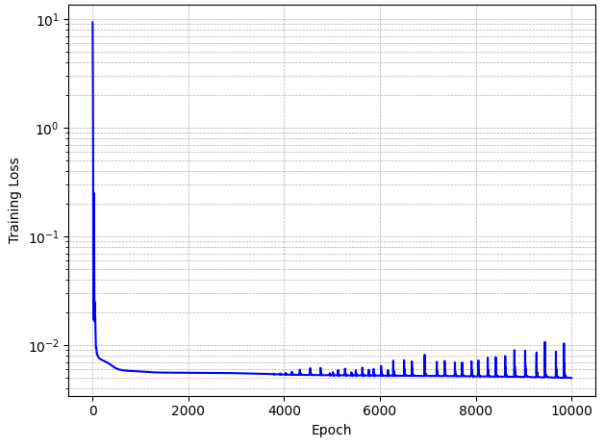

Fig. 10. Loss function plot (case 1)

The geometric domain represents a rectangular beam clamped at the left end and subjected to a concentrated downward force at the free end. The resulting deformation illustrates significant deflection at the free end, while the clamped end remains nearly stationary characteristic of a classical cantilever beam. The displacement vector field increases in magnitude toward the free end, consistent with the expected stress and displacement distribution. The loss function converges rapidly toward zero, indicating that the deep learning model successfully approximated the PDE solution.

4.2.2. Elasticity Equation with Two Fixed Ends

In the second case, the beam is clamped at both ends and subjected to a uniform vertical traction on its top surface. The body force is defined as $\mathbf{f} = (0, -0.2)^\top$. Both the left and right boundaries ($x = 0$ and $x = 1$) are fully clamped, enforcing $u = v = 0$. A uniform vertical traction is applied along the top boundary ($y = 0.15$), with $\sigma_{yy} = -0.2$ and $\sigma_{xy} = 0$, while the bottom boundary ($y = 0$) remains stress-free with $\sigma_{yy} = \sigma_{xy} = 0$. This configuration simulates a beam fixed at both ends and subjected to a uniform downward force along its top edge. [12]

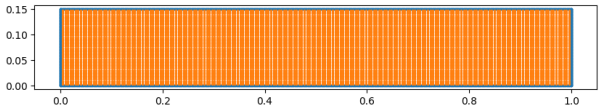

Fig. 11. Initial domain (case 2)

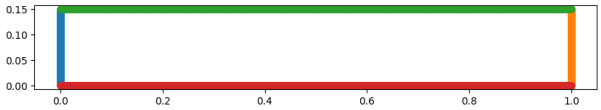

Fig. 12. Simulation of the geometric domain of a 2D rectangular beam with boundary conditions (case 2).

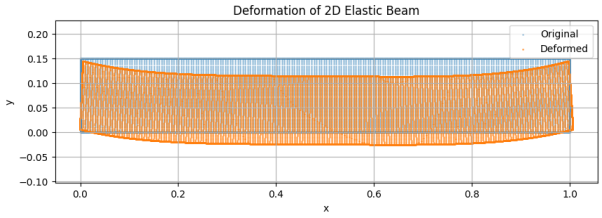

Fig. 13. Deformed domain (case 2)

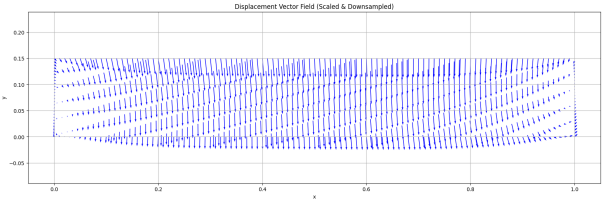

Fig. 14. Deformation vector field (case 2)

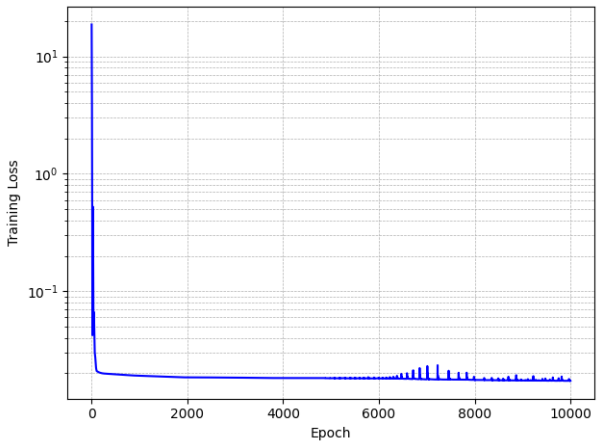

Fig. 15. Loss function plot (case 2)

In this case, the beam is clamped at both ends and subjected to a uniformly distributed load on the top surface. The deformation shows a symmetric deflection downward at the center, with both ends remaining fixed consistent with theoretical expectations.

The displacement vector field is symmetric about the beam's midline, reflecting the uniform load distribution. The loss function also shows stable and rapid convergence, demonstrating the accuracy of the learned solution.

5. Conclusion

This paper demonstrates positive results when applying Physics-Informed Neural Networks (PINNs) to solve partial differential equations (PDEs), with two representative types of equations: the steady-state heat equation and the elasticity equation. The results highlight the powerful approximation capabilities of neural networks while leveraging the physical information embedded in the PDEs to enhance accuracy and efficiency in the solution process. Thus, PINNs not only exhibit the ability to learn and model physical phenomena but also open up a new approach to solving complex problems.

In the future, we plan to continue developing and expanding the capabilities of PINNs to solve time-dependent PDEs, in order to simulate dynamic physical phenomena. Future research will focus on improving the convergence of the algorithm, optimizing the network architecture, and applying it to problems with more complex boundary and initial conditions.

References

[1] L. C. Evans, Partial Differential Equations, American Mathematical Society, 2010.

[2] J. Sirignano and K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics, vol. 375, pp. 1339–1364, 2018.

[3] R. J. LeVeque, Finite difference methods for ordinary and partial differential equations, SIAM, 2007.

[4] J. Han, A. Jentzen, and W. E, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences, vol. 115, no. 34, pp. 8505–8510, 2018.

[5] R. Maziar, P. Paris, and E. K. George, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, vol. 378, pp. 686–707, 2019.

[6] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering, vol. 360, pp. 112789, 2020.

[7] X. Meng *et al.*, PPINN: Parareal physics-informed neural network for time-dependent PDEs, Computer Methods in Applied Mechanics and Engineering, vol. 370, pp. 113250, 2020.

[8] P. N. Duy, Giai xap xi phuong trinh dao ham rieng bang mang hoc sau, Ph.D. dissertation, Hanoi University of Science and Technology (HUST), 2021.

[9] R. J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems, SIAM, 2007.

[10] G. E. Karniadakis *et al.*, Physics-informed machine learning, Nature Reviews Physics, vol. 3, pp. 422–440, 2021.

[11] D. P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, International Conference on Learning Representations (ICLR), 2015.

[12] A. Ern and J.-L. Guermond, Theory and Practice of Finite Elements, vol. 159, Springer, 2004, 525–530 pp.