Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2024, Article ID 3328977, 22 pages
https://doi.org/10.1155/2024/3328977

Research Article

Exploring Physics-Informed Neural Networks for the Generalized Nonlinear Sine-Gordon Equation

Alemayehu Tamirie Deresse and Tamirat Temesgen Dufera

Department of Applied Mathematics, School of Applied Natural Science, Adama Science and Technology University, Adama 5118, Oromia, Ethiopia

Correspondence should be addressed to Tamirat Temesgen Dufera; tamirat.temesgen@astu.edu.et

Received 22 December 2023; Revised 12 March 2024; Accepted 16 April 2024; Published 30 April 2024

Academic Editor: Yiming Tang

The nonlinear sine-Gordon equation is a prevalent feature in numerous scientific and engineering problems. In this paper, we propose a machine learning-based approach, physics-informed neural networks (PINNs), to investigate and explore the solution of the generalized non-linear sine-Gordon equation, encompassing Dirichlet and Neumann boundary conditions. To incorporate physical information for the sine-Gordon equation, a multiobjective loss function has been defined consisting of the residual of governing partial differential equation (PDE), initial conditions, and various boundary conditions. Using multiple densely connected independent artificial neural networks (ANNs), called feedforward deep neural networks designed to handle partial differential equations, PINNs have been trained through automatic differentiation to minimize a loss function that incorporates the given PDE that governs the physical laws of phenomena. To illustrate the effectiveness, validity, and practical implications of our proposed approach, two computational examples from the nonlinear sine-Gordon are presented. We have developed a PINN algorithm and implemented it using Python software. Various experiments were conducted to determine an optimal neural architecture. The network training was employed by using the current state-of-the-art optimization methods in machine learning known as Adam and L-BFGS-B minimization techniques. Additionally, the solutions from the proposed method are compared with the established analytical solutions found in the literature. The findings show that the proposed method is a computational machine learning approach that is accurate and efficient for solving nonlinear sine-Gordon equations with a variety of boundary conditions as well as any complex nonlinear physical problems across multiple disciplines.

1. Introduction

Differential equations provide a powerful framework for describing a wide range of engineering, mathematical, and scientific phenomena. They are particularly valuable in capturing heat transfer processes, fluid dynamics, wave propagation in electronic circuits, and mathematical modeling of chemical reactions. One notable example of a nonlinear hyperbolic partial differential equation (PDE) is the nonlinear sine-Gordon equation (NLSGE), which dates back to the nineteenth century and originally emerged in the study of surfaces with constant negative curvature [1–3]. This equation finds extensive application in simulating and describing various physical phenomena across engineering and scientific disciplines, including nonlinear waves, the propagation of fluxons in Josephson junctions, and the dislocation behavior of metals [4–9].

NLSGE has found numerous applications in various scientific and engineering domains. In the field of condensed matter physics, this equation has been used to study phenomena such as solitons and topological defects [10]. In the realm of nonlinear optics, the equation is used to model the propagation of optical pulses in nonlinear media, particularly in the context of optical fibers [11]. Furthermore, in the study of superconductivity, the NLSGE is used to describe the behavior of Josephson junctions, which are key components in superconducting devices [12]. The equation has also found application in surface science, where it describes the dynamics of atoms and molecules on surfaces, including the propagation of surface waves [13]. Furthermore, NLSGE has been applied in biophysics to model phenomena such as nerve impulse propagation and protein dynamics [14]. These are just a few examples of the wide-ranging applications of the NLSGE in diverse scientific and engineering problems. Readers interested in additional information should consult the monographs [15–24].

The NLSGE has recently been the subject of extensive computational and analytical analysis due to its significance in non-linear physics. For example, Babu and Asharaf [25] used a differential quadrature technique based on a modified set of cubic B-splines to numerically solve non-linear SGEs in one and two dimensions, as well as their coupled form. The modification employed in this approach achieves optimal accuracy of order four in the spatial domain. Spatial derivatives are approximated using the differential quadrature technique, and weight coefficients are calculated using the set of modified cubic B-splines. In a different study, Shiralizadeh et al. [26] implemented the numerical method of the rational radial basis function to solve the perturbed and unperturbed NLSGEs with Dirichlet or Neumann boundary conditions. This method is particularly suitable for cases where the solution exhibits a steep front or sharp gradients. Furthermore, Babu and Asharaf [27] employed the Daftardar-Gejji and Jafari method to obtain an approximate analytical solution for the NLSGE. They compared the obtained solution with the variational iteration method to assess its accuracy.

In 2022, Deresse [28] achieved successful integration of the double Sumudu transform with an iterative approach to obtain an approximate analytical solution for the one-dimensional coupled NLSGE. The double Sumudu transform alone is insufficient to solve this particular equation. As a result, the linear component of the problem was addressed using the double Sumudu transform, while the non-linear part was handled through an additional iterative approach. The two-dimensional stochastic time fractional NLSGE was investigated by the authors of the paper [29] in 2023. To find the numerical solution, they employ the clique polynomial approach. The clique polynomial is regarded as a fundamental function for operational matrices in this method. For more details, refer to the following references: [30–36].

These recent developments highlight the growing interest in tackling the challenges posed by the NLSGE, and researchers use various numerical and analytical techniques to explore its solutions and properties. This paper aims to introduce a deep learning-based method called a physics-informed neural network (PINN), to acquire the solution of NLSGE with Dirichlet and Neumann boundary conditions. PINNs are a scientific machine learning technique used to solve problems involving PDEs [37]. By training an ANN to minimize a loss function, PINNs approximate PDEs. This loss function incorporates various terms, including the initial and boundary conditions along the boundary of the space-time domain, as well as the PDE residual evaluated at specific points within the domain, known as collocation points. This approach allows PINNs to capture the essential physics of the problem and provide accurate solutions throughout the domain [38–40].

A parallel information-processing system, known as an ANN, shares similarities with certain brain functions. Comprised of neurons and synaptic weights, an ANN learns to perform complex computations [41]. By emulating the functioning of the human brain, the network receives inputs from various sources, combines them, applies non-linear operations, and produces an output [42–44]. The architecture of an ANN consists of three types of layers: input, hidden, and output, with neurons or units in each layer [45–47]. The architecture of the ANN processor is scalable, allowing for an infinite number of layers and neurons in each layer. It can also implement feedforward and dynamic recurrent networks [46, 48].

Approximating highly non-linear functions has become an attractive application of NNs due to their inherent capabilities. However, in low to moderate dimensions, PDE solvers based on NNs or deep NNs typically fall short when compared to classical numerical solution methods. This is primarily because solving an algebraic equation is generally easier than dealing with the highly non-linear, large-scale optimization problems associated with NN training [49, 50].

Furthermore, traditional numerical approaches have developed sophisticated error analysis techniques, which is an area where NN-based solvers currently lag. Consequently, specialized techniques have emerged over time to tackle specific issues, often incorporating constraints or underlying physical assumptions directly into the approximations [51]. One notable technique in this domain is PINNs, which have gained popularity for rapid prototyping when efficiency and high accuracy are not the primary concerns. PINNs can be applied to virtually any differential equation, making them versatile tools for approximation [52].

The authors of the research presented in [53] demonstrated promising results that indicate the ability of PINNs to achieve good prediction accuracy, provided that the given PDE is well posed and a sufficient number of collocation points are available. PINNs seek to identify an NN within a specific class of NNs that minimizes the loss function, resulting in an approximation of the PDE's solution [53]. Unlike the classic variational concept, which minimizes an energy function, PINNs have introduced modifications to this approach. A notable distinction between PINNs and variational methods is that not all PDEs satisfy a variational principle. However, the formulation of PINNs allows their application to a wide range of PDEs, regardless of whether the PDE possesses a variational principle [54].

In their work, Shin et al. [54] provide a theoretical justification for PINNs in the context of linear second-order elliptic and parabolic-type PDEs. They demonstrate that the sequence of minimizers strongly converges to the PDE solution in the set of continuous functions. Moreover, they argue that when each minimizer satisfies the initial/boundary conditions, the convergence mode becomes the Sobolev space of order one.

Recently, the repertoire of scientific publications on PINNs has increased rapidly, which confirms the effectiveness of PINNs. For example, Beck et al. [55] obtained the solution of stochastic differential equations, and Kolmogorov PDEs suffer from the curse of dimensionality employing deep learning. The authors derived and proposed a numerical approximation method that aims to overcome the related drawbacks. They solved examples including the heat equation, the Black-Scholes model, the stochastic Lorenz equation, and the Heston model, and showed that the proposed approximation algorithm is effective in high dimensions in terms of both accuracy and speed.

In the paper [37], the authors introduced an innovative approach that combines the power of NNs with the knowledge of physics to tackle complex problems related to non-linear PDEs. The authors propose a framework where NNs are trained to approximate the solution of these equations while incorporating physical principles as constraints. This approach enables the accurate and efficient solution of both forward and inverse problems, offering great potential for applications in various scientific and engineering fields. The study contributes to the growing field of physics-informed machine learning, providing a promising avenue for advancing the understanding and solving of non-linear systems.

Blechschmidt and Ernst [40] provided a comprehensive overview of recent approaches to solving PDEs using NNs. They discuss the taxonomy of informed deep learning, present a literature review in the field, and highlight the potential of using machine learning frameworks to accelerate numerical computations of time-dependent PDEs. The authors used the PINN to solve a high-dimensional linear heat equation as an illustration and suggested that PINNs can offer attractive approximation capabilities for highly non-linear and high-dimensional problems.

In the paper [56], the authors presented a novel approach to solving PDEs in complex geometries using deep feedforward NNs. The paper explores the application of deep NNs in approximating solutions to PDEs and demonstrates their effectiveness in solving systems of ordinary differential equations. The authors provide insights into the architecture of the NN and discuss the weight connections between the neurons in different layers. The research contributes to the field of computational mathematics by introducing a unified framework that combines deep learning techniques with the solution of PDEs, paving the way for more accurate and efficient numerical methods in complex geometries. To effectively solve differential equations, the authors of the paper [57] presented DeepXDE, a potent deep learning library that combines the advantages of deep NN and PINN.

Furthermore, Schäfer [58] applied Dirichlet boundary conditions to a PINN solution of the one-dimensional heat equation. To solve a single instance of the PDE, the authors compared a PINN to a NN with defined beginning and boundary conditions. It turned out that PINNs are more accurate than NNs for a limited number of training samples. However, it should be noted that a PINN uses more computation time than a NN because each iteration includes a gradient evaluation. As the runtime grows exponentially for an increasing number of input features, this can be a serious bottleneck for higher-dimensional issues.

More recently, [59] presented two novel PINN architectures that satisfy various invariance conditions for constructing robust and efficient deep learning-based subgrid-scale turbulence models for use in large Eddy simulation procedures widely used in various fluid engineering applications. The first architecture is called tensor basis neural networks (TBNN) and the second architecture is a Galilean invariance embedded neural network (GINN) that incorporates the Galilean invariance and takes as input the independent components of the integrity basis tensors in addition to the invariant inputs in a single input layer. A deep learning-accelerated computational framework based on PINN is presented by the investigator of the paper [60] for the solution of the linear continuum elasticity equation. The authors suggested a multi-objective loss function that included terms that fit data-driven physical knowledge across randomly chosen collocation points in the problem domain, constitutive relations derived from the governing physics, terms corresponding to the residual of the governing PDE, and different boundary conditions. In a different study, a multi-objective loss function-based PINN is used by the authors of the monograph [61] to obtain the solution to the data-driven elastoplastic solid mechanics problem.

Even though many studies are conducted to use PINN to solve a variety of problems, many of them focus on elliptic and parabolic DEs. There are very few research papers on the use of PINNs to solve hyperbolic PDEs. This is due to hyperbolic PDEs like the NLSGE involving both second-order time derivatives and spatial derivatives. Such a problem contained an initial condition involving time-derivative that adds an extra layer of complexity to the solution process, as the solution must satisfy the dynamics of the PDE while also matching the specified initial data. In research published in the journal [62], the PINN method was used to solve linear hyperbolic PDEs while taking into account forward and inverse issues. Examples considered by the author are homogeneous linear wave equations. The author did not, however, investigate the PINN for the non-linear, hyperbolic PDEs that are inhomogeneous. In the present work, we use PINNs to solve NLSGE (1), which is the inhomogeneous non-linear class of hyperbolic wave equation containing a derivative of second order in time, taking inspiration from the work of the paper's author [62]. We focus on exploring two boundary condition categories: Dirichlet and Neumann. To minimize the loss function of the residuals of the governing equation, initial conditions, and boundary conditions, a PINN technique with a multi-objective loss function is employed. In addition, we conducted experimental simulations to assess the impact of different neural architectures on the performance of the model. Subsequently, we implement the algorithm developed using the Python-based software library, DeepXDE, as a computational tool [57].

The remaining parts of this manuscript are organized as follows: In Section 2, the governing problem is presented with some preliminary descriptions. Fundamental ideas, theorems, definitions, and an algorithm for PINNs are addressed in Section 3 for the specified issues. The method is validated in Section 4 using a numerical experiment for Dirichlet and Neumann boundary conditions, and finally, concluding remarks are drawn in Section 5.

2. The Governing Equation

The generalized Cauchy-type NLSGE employed in this paper is given by [63]:

$$\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2}+\beta\frac{\partial u(\mathbf{x},t)}{\partial t}=\alpha\Delta u(\mathbf{x},t)-\phi(\mathbf{x})\sin(u(\mathbf{x},t))+f(\mathbf{x},t), \tag{1}$$

where $\mathbf{x}=(x_1,x_2,\ldots,x_d)\in\Omega\in\mathbb{R}^d$ and $d=1,2,\ldots,n$. Here, Δ represents the Laplacian operator and n the dimension of the space variable $\mathbf{x}$. The function $\phi(\mathbf{x})$ can be interpreted as the Josephson current density, while the parameters α and β are real numbers with $\alpha,\beta\geq 0$. The dissipative term, denoted by β, characterizes the presence of damping in the equation. When $\beta>0$, (1) reduces to the damped SGE, while $\beta=0$, equation (1) reduces to undamped SGE

$$u_{tt}(\mathbf{x},t)=\alpha\Delta u(\mathbf{x},t)-\phi(\mathbf{x})\sin(u(\mathbf{x},t))+f(\mathbf{x},t). \tag{2}$$

If $f(\mathbf{x},t)=0$ the undamped SGE (2) has the conservation for the energy defined by

$$E(t)=\frac{1}{2}\int_\Omega\left[|u_t|^2+|\nabla u|^2+\phi(1-\cos u)\right]dV, \tag{3}$$

which is not valid for the damped system (1) [64]. Here $dV=d^n\mathbf{x}$ is the Euclidean $n-$ dimensional volume differential.

In the case of $d=1$, with $\mathbf{x}=x$ and $\triangle u=\partial^2 u(x,t)/\partial x^2=u_{xx}$, (1) represents the NLSGE in one dimension. The equation is subject to initial conditions:

$$\begin{aligned}u(x,0)&=f_1(x),\\ u_t(x,0)&=f_2(x), x\in[a,b],\end{aligned} \tag{4}$$

along with either Dirichlet boundary conditions:

$$\begin{aligned}u(a,t)&=k_1(t),\\ u(b,t)&=k_2(t),\end{aligned} \tag{5}$$

or Neumann boundary conditions:

$$\begin{aligned}u_x(a,t)&=k_3(t),\\ u_x(b,t)&=k_4(t).\end{aligned} \tag{6}$$

In this study, our aim is to address the solution of this equation using PINNs [37]. PINNs employ NNs specifically designed for solving PDEs by minimizing a loss function that incorporates the given PDE and both initial and boundary conditions. We develop a PINN algorithm and implement it using the Python-based software library, DeepXDE. Additionally, we conduct various deep experiments to identify the optimal neural architecture for our purposes.

3. Physics-Informed Neural Networks

3.1. *The Mathematical Description of Neural Network*

Definition 1 (see [65, 66]). Let $d\in\mathbb{N}$. We define an artificial neuron v: $\mathbb{R}^d\longrightarrow\mathbb{R}$ as a mapping with weight $\mathbf{w}\in\mathbb{R}^d$, bias $b\in\mathbb{R}$, and activation function σ: $\mathbb{R}\longrightarrow\mathbb{R}$. The neuron's output is given by the expression

$$v(\mathbf{x})=\sigma(\mathbf{w}\cdot\mathbf{x}+b), \text{where } \mathbf{x}\in\mathbb{R}^{\mathrm{d}}. \tag{7}$$

The role of the activation function σ is to produce the output from a set of input values fed to a node (or a layer). There are benefits and drawbacks to each activation function. Note that there is no set rule regarding the selection of an activation function for a particular activity. In machine learning, the most commonly used activation functions with PINN are the sigmoid function $\sigma(x)=1/1+e^{-x}$, the tan hyperbolic function $\sigma(x)=\tanh(x)$ and the ReLU function $\sigma(x)=\max\{0,x\}$ [67].

Definition 2. A deep feedforward neural network is defined as a function of the form

$$\widehat{Y}:=F_{W,b}(X)=\left(f^{(L)}_{W^{(L)},b^{(L)}}\cdots\circ f^{(1)}_{W^{(1)},b^{(1)}}\right)(X), \tag{8}$$

where it consists of multiple layers. Each layer is represented by a semi-affine function

$$f^{(l)}_{W^{(l)},b^{(l)}}:=\sigma^{(l)}\left(W^{(l)}X+b^{(l)}\right), \tag{9}$$

incorporating a univariate and continuous non-linear activation function $\sigma^{(l)}$. The weight matrices $W=(W^{(1)},\ldots,W^{(L)})$ and the offsets (biases) $b=(b^{(1)},\ldots,b^{(L)})$ define the parameters of the network. This deep feedforward NN is designed to process input data X and produces output $\widehat{Y}$, representing predictions or results of the network computation [66].

3.2. *The PINNs Algorithm for 1D NLSGE with Dirichlet BCs.*

In this subsection, we present the PINN approach for approximating the solution u: $[0,T]\times\Omega\longrightarrow\mathbb{R}$ of the one-dimensional problem (10) with Dirichlet boundary conditions. The problem can be stated as follows:

$$\frac{\partial^2 u(x,t)}{\partial t^2} + \beta \frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2} - \phi(x)\sin(u(x,t)) + f(x,t), \tag{10}$$

subject to the conditions:

$$\begin{aligned} u(a,t) &= k_1(t), \quad t \in [0,T],\\ u(b,t) &= k_2(t), \quad t \in [0,T],\\ u(x,0) &= f_1(x), \quad x \in \Omega,\\ u_t(x,0) &= f_2(x), \quad x \in \Omega, \end{aligned} \tag{11}$$

where $(x,t) \in (0,T] \times \Omega, \Omega = \{x\colon a \le x \le b\} \subset \mathbb{R}$ represents a bounded domain, and T denotes the final time. The PINN method combines the supplied PDE with physical constraints placed on the network to ensure the answer respects the physics of the problem. In the PINNs method, a NN is used to approximate the solution, and a set of nodal points is where the equations are imposed in the least-squares sense.

The literature provides the following four well-known steps for utilizing the proposed method to solve a PDE [37, 40, 62, 68–70].

(i) Construct an ANN $\hat{u}(x,t;P)$ to serve as an approximation of the true solution $u(x,t)$.

(ii) Set up a training set that will be used to train the NN.

(iii) Formulate an appropriate loss function that considers residuals of the PDE, initial, boundary, and final conditions.

(iv) Train the NN by minimizing the cost function established in the previous step.

3.3. Step 1: Deep Neural Network.

3.3. Step 1: Deep Neural Network. We employ the following notations: The superscript (i) denotes the ith data point (collocation) or training example, while superscript (l) represents the lth layer in the network. The input size is denoted as n_x, and the output size as n_y. Additionally, n^l refers to the number of neurons in the lth layer, and L signifies the total number of layers in the network. The input is denoted by X, which is the set of collocation points comprising points from the interior and boundary of the domain. The weight matrix for the lth layer is denoted as $W^{(l)} \in \mathbb{R}^{n^{l+1}\times n^l}$, and the bias vector in the l^{th} layer is represented as $b^{(l)} \in \mathbb{R}^{n^{l+1}}$. The predicted output vector is denoted as $\hat{u} \in \mathbb{R}^{n_y}$ or equivalently written as $a^{(L)}$, where L indicates the total number of layers in the network. Figure 1 displays a demonstration of a sketch-deep NN diagram. The structure shown is an advancement of the NN structure in papers [48, 71] designed for the systems of ordinary differential equations.

To solve the one-dimensional NLSGE, our input data will have the form $(x^i, t^i) \in \mathbb{R}^{1+1}$. That is, according to the notations described above, $n_x = 2$. Furthermore, $n_y = 1$ since we have only one network output $\hat{u}(x,t;P)$, where P represents the parameter consisting of weights and biases. We selected the DNN scheme to have two nodes in the input layer and one node in the output layer that contains the value of $\hat{u}(x,t)$ to generate $u(x,t)$ that solves (7) using PINN. There were four hidden layers in the structure, and each layer contained fifty units (neurons). We consider a deep-feedforward NN, whose main objective is to approximate a function, in this case $u(x,t)$ for any input (x,t), among other options.

In our case, the solution $\hat{u}(x,t;P)$, which corresponds to the output of the NN, is constructed as described in [72], mainly:

$$\begin{cases} \text{Input layer: } \mathbf{a}^{(0)} = (x,t) \in \mathbb{R}^{1+1},\\ \text{Hidden layers: } \mathbf{a}^{(l)} = \sigma\left(W^l \mathbf{a}^{(l-1)} + \mathbf{b}^l\right) \in \mathbb{R}^{n^l}, l = 1,\ldots,L-1, \text{and,}\\ \text{Output layer: } \hat{u}(\mathbf{x},t;P) = \mathbf{a}^{(L)} = W^L \mathbf{a}^{(L-1)} + \mathbf{b}^L \in \mathbb{R}, \end{cases} \tag{12}$$

where:

(i) $\mathbf{a}^l\colon \mathbb{R}^{d_{in}} \longrightarrow \mathbb{R}^{d_{out}}$ is the l layer with n_l nodes,

(ii) $\mathbf{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\mathbf{b}^l \in \mathbb{R}^{n_l}$ are the weights and the biases and $\boldsymbol{\theta} = \left\{\mathbf{W}^l, \mathbf{b}^l\right\}_{l=1}^{L-1}$ are the parameters of the NNs, and

(iii) σ is an activation function which acts componentwise.

3.4. Step 2: Training Dataset. When using a PINN to solve a PDE, it is important to properly split collocation points into two disjoint sets: training and test data to ensure accurate model evaluation [73]. Training data will be used to train the PINN, while test data will be used to evaluate the model's performance. These data are typically split into ratios of 20% for testing and 80% for training in machine learning [74]. This division ratio is sometimes referred to as the $80-20$ rule. In this study, we used 500 for training and 125 for testing. The training data $X \subset \overline{\Omega}$ is the union of the set $X^{\Omega} \subset \overline{\Omega}$ which contains points selected from the interior domain and the set $X^{\Gamma} \subset \overline{\Omega}$ which contains points taken from the boundary. The general training set X of the PINN model for the initial/boundary value problem is a union of the following:

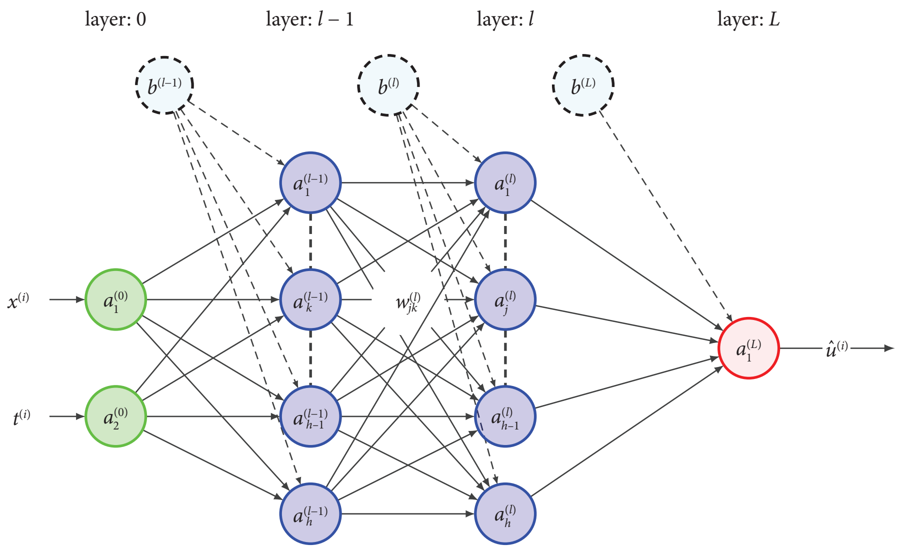


FIGURE 1: Illustration of a fully connected deep feedforward neural network with two nodes, (x^i, t^i), in the input layer and one node, $\widehat{u}(x^i, t^i)$, in the output layer. Layer 0 is the input layer, the layers $l-1$ & l are the hidden layers with associated weight matrix functions w^l & bias vectors $b^{(l-1)}$ & $b^{(l)}$, and the layer L is the output layer. Input data passes through the network by following (8).

(i) The interior domain $X^{\Omega} \subset (a,b) \times (0,T)$,

(ii) The boundaries $X^{\Gamma} \subset \{a,b\} \times [0,T]$ and,

(iii) $X^0 \subset (a,b) \times \{0\}$.

Thus,

$$X = X^{\Omega} \cup X^{\Gamma} \cup X^0. \tag{13}$$

3.5. Step 3: Loss Function. The total loss function $J(X;P)$ is the contributions of the losses due to: the residual of a given NN approximation $\widehat{u}$: $[0,T] \times \Omega \longrightarrow \mathbb{R}$ of the solution u, that is,

$$r(x,t;P) \coloneqq \frac{\partial^2 \widehat{u}}{\partial t^2} + \beta \frac{\partial \widehat{u}}{\partial t} - \alpha \frac{\partial^2 \widehat{u}}{\partial x^2} + \phi(x) \sin(\widehat{u}) - f(x,t). \tag{14}$$

(i) Differences from network approximations on the initial collocation points.

Similar to the originally proposed approach by authors of the paper [37], the PINN approach for the solution of the initial and boundary value problem now proceeds by minimization of the loss function of parameter P which is given by

$$J(X;P) = J_{\Omega}\left(X^{\Omega};P\right) + J_{\Gamma}\left(X^{\Gamma};P\right) + J_0\left(X^0;P\right), \tag{15}$$

where

$$J_{\Omega}\left(X^{\Omega};P\right) \coloneqq \frac{1}{N_{\Omega}} \sum_{(x^i,t^i)\in X^{\Omega}} \left| r\left(x^i,t^i\right)\right|^2, \tag{16}$$

$$J_{\Gamma}\left(X^{\Gamma};P\right) \coloneqq \frac{1}{N_{\Gamma_1}} \sum_{(x^i,t^i)\in X^{\Gamma_1}} \left|\widehat{u}\left(x^i,t^i\right) - k_1\left(t^i\right)\right|^2 + \frac{1}{N_{\Gamma_2}} \sum_{(x^i,t^i)\in X^{\Gamma_2}} \left|\widehat{u}\left(x^i,t^i\right) - k_2\left(t^i\right)\right|^2, \tag{17}$$

$$J_0\left(X^0;P\right) \coloneqq \frac{1}{N_0} \sum_{(x^i,t^i)\in X^0} \left|\widehat{u}\left(x^i,t^i\right) - f_1\left(x^i\right)\right|^2 + \frac{1}{N_0} \sum_{(x^i,t^i)\in X^0} \left|\widehat{u}_t\left(x^i,t^i\right) - f_2\left(x^i\right)\right|^2. \tag{18}$$

Thus, the optimal parameters P^* of the network satisfy

$$P^* = \underset{P}{\operatorname{argmin}}\ J(X; P). \tag{19}$$

3.6. Step 4: Training Process. The final step in the PINN algorithm amounts to minimizing (10). Therefore, we apply the loss function given by (15) on the training samples (parts of the domain and the boundary, see Figure 2), and we get the blue line in Figure 3, which implies that the loss function of the train decreases with respect to the training time. At the same time, we calculate the loss function.

$$J(X; P) = J_{\text{test},\Omega}\left(X^{\Omega}; P\right) + J_{\text{test},\Gamma}\left(X^{\Gamma}; P\right) + J_{\text{test},0}\left(X^{0}; P\right), \tag{20}$$

on the test samples.

3.6.1. The Combined Adam and L-BFGS-B Optimization Algorithms. Like NNs, the training process for PINNs corresponds to the minimization problem $\min_{P} J(X; P)$. Training of network parameters $\mathbf{P}$ is carried out using a gradient descent approach such as Adam [75] or L-BFGS-B (limited memory algorithm Broyden-Fletcher-Goldfarb-Shanno) [76]. However, the required number of iterations depends highly on the problem "(e.g., smoothness of the solution)" see [57]. The partial derivatives are necessary at every stage of the training process. Therefore, it is computationally difficult to calculate the PINN loss in each iteration if the interior domain contains a significant number of points. Lu et al. [57] proposed a method called residual-based adaptive refinement to increase the effectiveness of the training procedure. To validate the efficacy of these optimization techniques and enable their reuse, we conduct three separate experiments in this paper: one for the Adam optimization algorithm, one for L-BFGS-B optimization, and a final one for the combination of both Adam and L-BFGS-B optimization algorithms.

3.6.2. Weight Initialization. Due to the randomness of the initial weight state in deep learning, each training can produce a distinct set of outcomes. The variance of the input signal decreases as it moves through each layer of the network if the weights are set too close to zero. If the weights are excessively large, the network either approaches a vanishing gradient problem or the variance of the signal tends to amplify as it moves through the network layers. Therefore, choosing weights that are either too high or too small is not a feasible initialization since in both circumstances, the initialization is outside the optimization procedure's right-hand basin of attraction. There are several well-known randomized weight initialization techniques, including uniform, Gaussian, Glorot uniform, and Glorot normal initialization over time. When used in conjunction with symmetric activation functions, the Glorot uniform weight initializer offers a systematic method of weight initialization that can aid in training stability, gradient flow, and convergence in NNs [77, 78]. Taking this inquiry into account, Glorot uniform initialization was used for the demands of this article with a learning rate of 0.001.

3.6.3. Weakness and Limitation of the PINN Model. The PINNs model, while powerful, has several limitations [79]. A fair weakness and limitation of the PINNs model is the requirement of a large amount of labeled data for training. To enforce physical constraints, PINNs usually rely on solving PDEs, which calls for a good understanding of the underlying physics. However, it can be difficult to get labeled data that faithfully capture the physical system, particularly in situations where access to experimental data is expensive or limited. This restriction may make the PINN model less useful and less generalizable [80]. To address this weakness, one possible improvement is to incorporate transfer learning techniques. Through transfer learning, performance on a target task with limited data can be improved by utilizing pre-trained models on related tasks or domains. Explicitly integrating domain knowledge into the model design is another way to enhance PINNs. One can direct the model to produce more accurate predictions by feeding it with prior knowledge in the form of physical principles, equations, or constraints. Additionally, an ensemble-based approach can be used to enhance the predictive capacity of PINNs. Instead of relying on a single neural network, multiple networks with diverse architectures or initializations can be trained. In this paper, we also consider various networks with distinct architectures to effectively solve the NLSGE using the PINN algorithm as presented in Algorithm 1.

4. Implementation

In the following section, we use Python code to build the PINN algorithm to solve the NLSGE (1) in one dimension. As an illustration, we take into account both Dirichlet and Neumann boundary conditions to validate the effectiveness of the models.

4.1. 1D NLSGE with Dirichlet BCs. Consider the following one-dimensional NLSGE:

$$\frac{\partial^2 u}{\partial t^2} = \frac{1}{\pi^2}\frac{\partial^2 u}{\partial x^2} - \sin(u) + \sin(\cos(\pi x)\cos t), 0 \le x \le 1, 0 < t < 2, \tag{21}$$

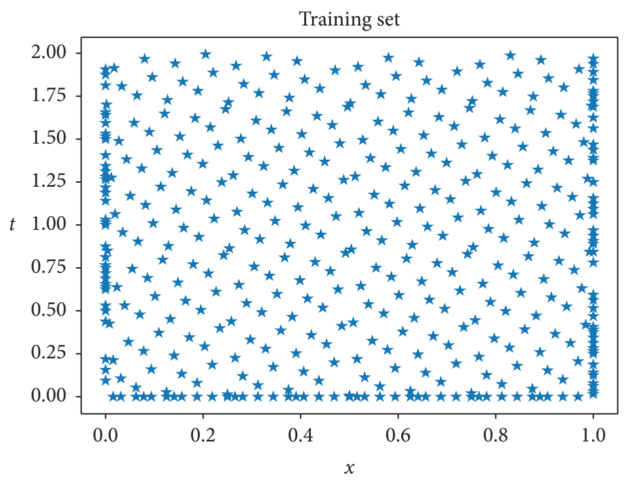
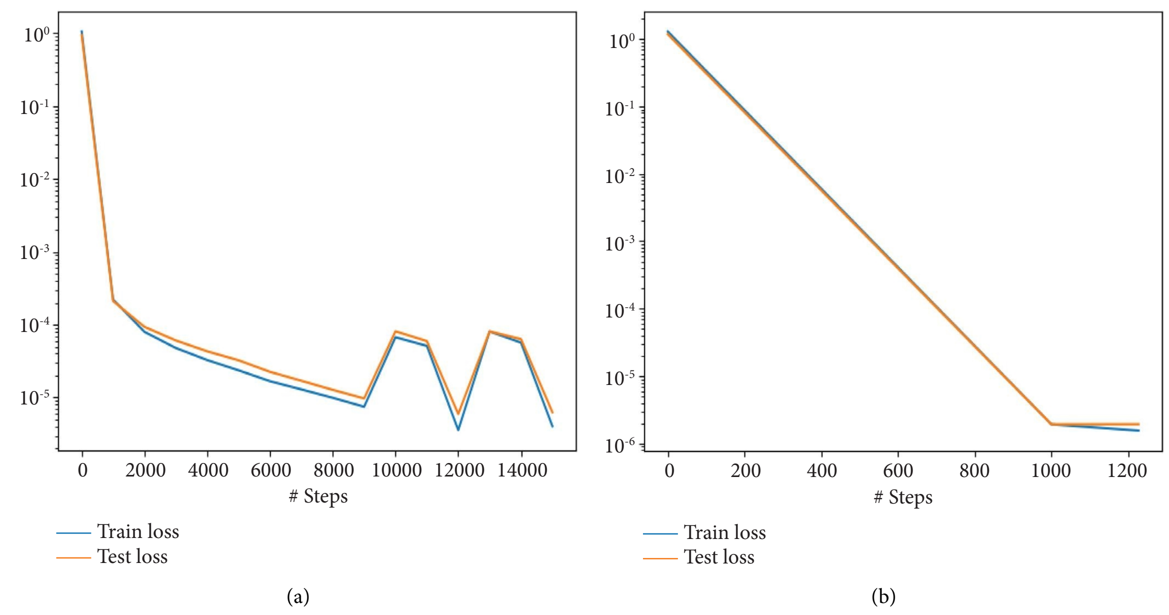


Figure 2: Illustration of the training dataset.



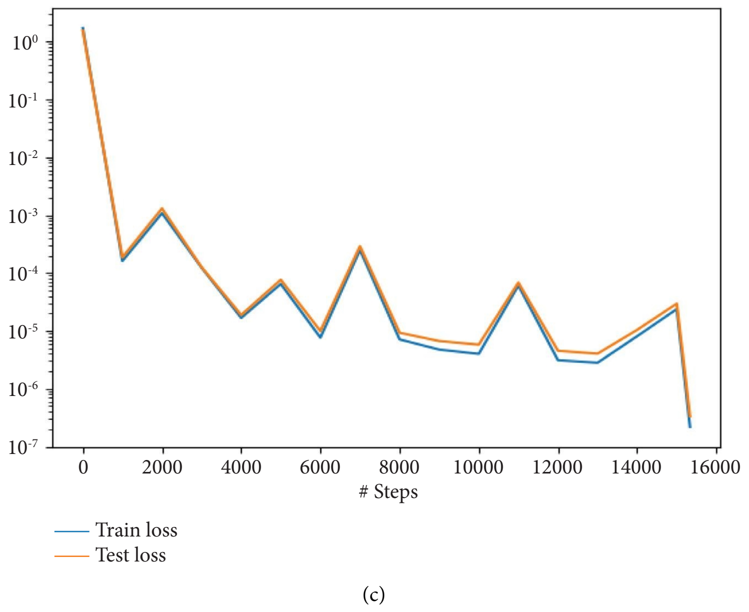


Figure 3: Train and test loss of PINN process for 15000 epochs (training iterations) for the dirichlet BCs case by using (a) adam, (b) L-BFGS-B, and (c) combined adam and L-BFGS-B optimization algorithms.

```
    Require: Training data, collocation points $X = X^{\Omega} \cup X^{\Gamma} \cup X^{0}$, contains interior and boundary points.
      Initial condition, boundary condition, and the NLSGE.
(1)   Define network architecture (input layer, hidden layers, output layer, activation function, and optimizer).
(2)   Initialize weights $W$ and biases $b$, $P = \{W, b\}$.
(3)   for all epochs do
(4)     apply forward propagation: $\hat{u}(x, t; X) \leftarrow F_P(X; P)$
(5)     compute the residual: $r(x, t; P)$
(6)     compute loss: $J(X; P)$
(7)     apply the optimizer: $\text{argmin}_P J(X; P)$.
(8)   end for
```

Algorithm 1: PINN algorithm for NLSGE.

with Dirichlet boundary conditions

$$u(0,t) = \cos t, \quad 0 \le t \le 2, \tag{22}$$

$$u(1,t) = -\cos t, \quad 0 \le t \le 2, \tag{23}$$

and initial conditions

$$u(x,0) = \cos(\pi x), \quad 0 < x < 1, \tag{24}$$

$$u_t(x,0) = 0, \quad 0 < x < 1. \tag{25}$$

The exact solution of the IBVP is given by $u(x,t) = \cos(\pi x)\cos t$ [32].

4.1.1. The PINNs Algorithm

(1) Step 1: Neural Network. To obtain $u(x,t)$ that solves (21) using the proposed method, we chose the structure of the NN to have two nodes in the input layer (x,t) and one node in the output layer that contains the prediction for the value of $u(x,t)$. The structure had four hidden layers, each of which contained 50 nodes (neurons).

(2) Step 2: Training Dateset. The general training set X of this model is selected in the interior domain $X^{\Omega} \subset (0,1) \times (0,2)$ and on the boundaries $X^{\Gamma_1} \subset \{0\} \times [0,2]$, $X^{\Gamma_2} \subset \{1\} \times [0,2]$, $X^0 \subset (0,1) \times \{0\}$. Thus

$$X = X^{\Omega} \cup X^{\Gamma_1} \cup X^{\Gamma_2} \cup X^{0}. \tag{26}$$

The training set we used consisted of 500 samples $\{(x^i, t^i); u(x^i, t^i)\}_{i=1}^{500}$ where $u(x^k, t^k)$ is the solution of (21) at (x^k, t^k). 300 training samples were chosen from $(0,1) \times (0,2)$ and the rest was taken from the boundary of the domain (see Figure 2).

(3) Step 3: Loss Function. The loss function used to train the PINN with the parameter P is given by (15) where

$$J_{\Omega}\left(X^{\Omega}; P\right) = \frac{1}{N_{\Omega}} \sum_{(x^i,t^i) \in X^{\Omega}} \left| r\left(x^i, t^i\right) \right|^2, \tag{27}$$

where

$$r\left(x^i, t^i\right) = \hat{u}_{tt}\left(x^i, t^i\right) - \frac{1}{\pi^2}\hat{u}_{xx}\left(x^i, t^i\right) + \sin\left(\hat{u}\left(x^i, t^i\right)\right) - \sin\left(\cos\left(\pi x^i\right)\cos t^i\right), \tag{28}$$

$$J_{\Gamma}\left(X^{\Gamma}; P\right) = \frac{1}{N_{\Gamma_1}} \sum_{(x^i,t^i) \in X^{\Gamma_1}} \left| \hat{u}\left(x^i, t^i\right) - \cos\left(t^i\right) \right|^2 + \frac{1}{N_{\Gamma_2}} \sum_{(x^i,t^i) \in X^{\Gamma_2}} \left| \hat{u}\left(x^i, t^i\right) + \cos\left(t^i\right) \right|^2, \tag{29}$$

$$J_0\left(X^0; P\right) = \frac{1}{N_0} \sum_{(x^i,t^i) \in X^0} \left| \hat{u}\left(x^i, t^i\right) - \cos\left(\pi x^i\right) \right|^2 + \frac{1}{N_0} \sum_{(x^i,t^i) \in X^0} \left| \hat{u}_t\left(x^i, t^i\right) - 0 \right|^2. \tag{30}$$

(4) Step 4: Training Process. With the training samples, we apply the loss function (10) to obtain the blue line in Figure 3, which indicates that the train loss function decreases with the number of model training repetitions. At the same time, we calculate the loss function on the test samples using (20).

The number of steps in Figure 3 (also known as the number of epochs) indicates the number of iterations used to train the model and thus the number of times the weights of the network are updated. In our case, we used 15000 epochs, which indicates that the NN was trained for 15000 passes over the training dataset. The loss in train and test decreases as the number of epochs increases, as the figure illustrates. As a result, using more training iterations results in smaller train and test losses, indicating that the suggested strategy produced a better solution. Additionally, the L-BFGS-B optimization algorithm produces fewer train and test losses than the Adam optimization algorithm, and combining the two optimization algorithms results in smaller train and test losses. Therefore, it is preferable to use both optimizations simultaneously rather than one of them alone.

Figures 4–6 present the precise solution and result of problem (21) using the suggested method. The graphs of the 2D and 3D solution plots for the model optimizations proposed in step 4 of Subsection 4.1 allow for a comparison of the two solutions. Furthermore, Figure 7 and Table 1 are used to compare the estimated solution error for the Adam, L-BFGS-B, and combined Adam and L-BFGS-B optimization algorithms.

The solution to NLSGE (16), depicted in 3D Figures 4–6, shows that there is not much difference between the precise solution and the solution produced using the suggested technique PINN. However, the result obtained using the L-BFGS-B optimization algorithm is relatively better than that obtained using the Adam optimization algorithm, and the result obtained using the Adam and L-BFGS-B mixture is better than that obtained by both optimizers, as we can observe from Figure 7.

The 2D line plot in Figure 8 shows comparisons of the solution of the suggested method with the exact solution at $x = 0.5$ with their corresponding absolute error by different optimization algorithms. As we can see in Figures 8(a), 8(c), and 8(e), the line plots of the two solutions overlap, suggesting that they are possibly much related. Observing the result for the selected optimization algorithm, as we can see from Figures 8(b), 8(d), and 8(f), the result obtained utilizing the L-BFGS-B optimization approach is relatively more successful than the one obtained using the Adam optimization technique, and the result produced using the combination of Adam and L-BFGS-B is of higher quality than both of them.

The precise answer and the suggested method are compared in Table 1, and the results are explained using L_2, L_∞, relative and mean square error. This comparison also shows that the PINN approach with the L-BFGS-B optimization algorithm yields a better solution than the one with the Adam optimization algorithm and that the solution resulting from the combination of Adam and L-BFGS-B is better than the individual algorithms with the least amount of absolute error. It takes a longer time for the model to compile when both techniques are used simultaneously.

4.1.2. Error Analysis and Computational Time

(1) Training Error. The training error provides insight into how well the predicted outputs of the training inputs fit the training outputs, i.e., how the model performs in the training set.

The training error varies as the number of training samples increases, as seen in Figure 9. It shows that the training error increases for the first few training samples before gradually decreasing for the remaining training trials. This finding indicates that using few samples results in high error rates and using more training samples is preferable to getting good results with low error rates.

(2) Error on a Validation Set. The training error is important to find out whether our model can be applied to any input data and still produce accurate results, even if it performs exceptionally well on training data (the error is small). According to this method, $\tau = (x_i, t_i; u_i)$ should be randomly divided into two disjoint sets; a training set and a validation set, where τ represents the set of all available data.

Figure 10 illustrates how, for a given number of training samples, the error initially reduces. Still, we find that even for a relatively modest collection of additional training examples, the error is close to 0. As the training set size grows, the error remains consistently insignificant.

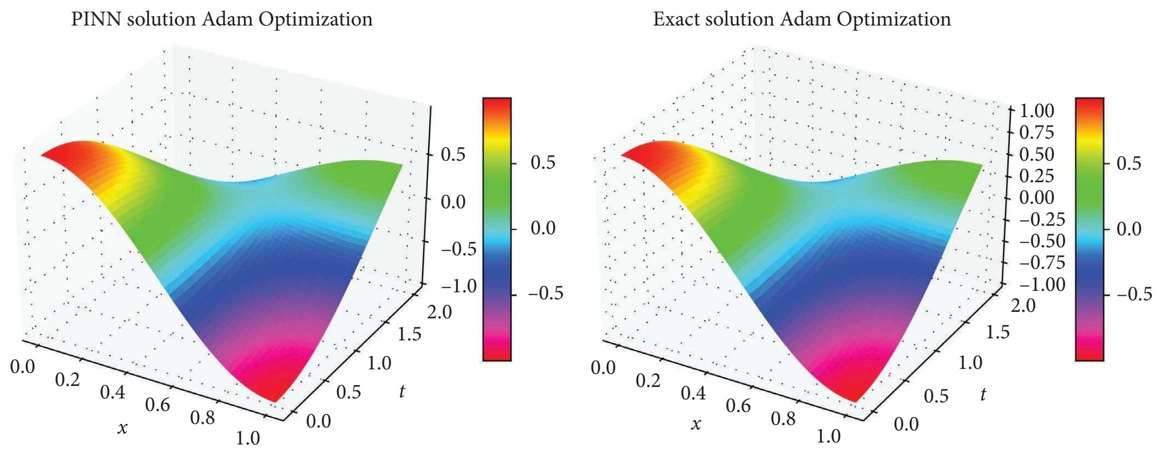
(3) Computational Time. Costly computations are involved when the number of training samples is increased. When a large number of training samples were taken into account, the code execution was incredibly slow. This is shown in Figure 11 below (time is given in seconds). We take into account eight different training samples, each having the following sizes: 5, 15, 25, 55, 90, 185, and 350. We can see that the compilation time increases with the size of the training set size differences.

The link between the size of training samples and the amount of time needed for compilation or model training is depicted in Figure 11. The training samples that were mentioned have specified sizes of 5, 15, 25, 55, 90, 185, and 350. The figure shows that the amount of time needed for model compilation or training increases with the number of training samples. This implies that the amount of time required for these procedures and the quantity of training samples are positively correlated.

(4) Test Error vs. Computational Time. We can examine the performance of our machine learning model in further depth thanks to the plot that depicts the dependence of the test error on the computational time required. Our goal is to create a model that performs well (test loss is minor) and that can be completed in a reasonable amount of time.

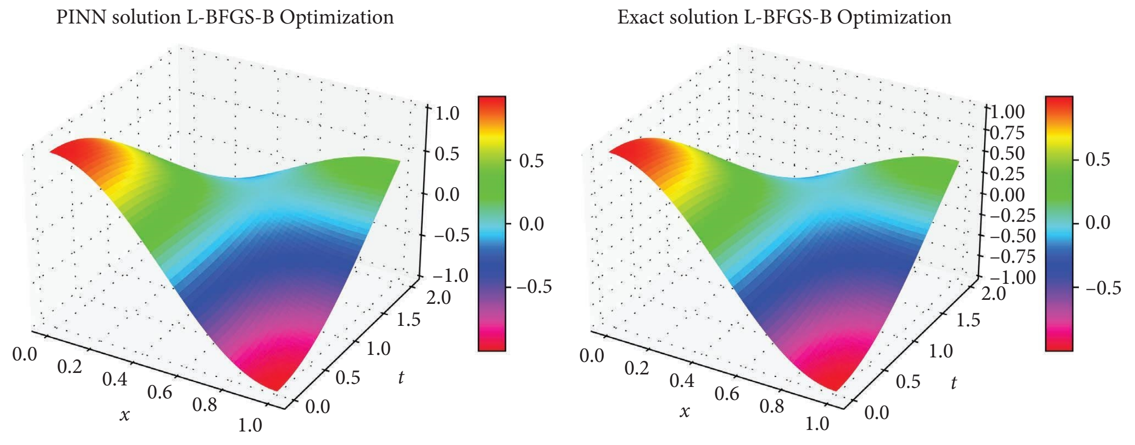
The decrease in test loss is initially accompanied by an increase in processing time, as seen in Figure 12. Even when the model takes longer to run, we see that this pattern disturbs and that the test loss is essentially constant.

(5) Discussion on the Number of Nodes in the Neural Network. We investigate how the size of the NN affects our model's performance by using five different NN layouts in the model construction and test loss collection. We fix our NN structure having four hidden layers and conduct experiments for 30, 50, 100, 150, and 200 nodes of the NN architectures. The test loss vs. computing time for the aforementioned NN structures for the NLSGE Dirichlet BCs example is depicted in Figure 13. The graph shows how the test error changes as the number of iterations (i.e., the processing time) rises for these five different NN settings. Figure 14 illustrates the absolute errors between the results of the proposed model and the exact solution for various nodes of the NN structures. The error for the NN architecture containing 50 is very close to zero relative to others,



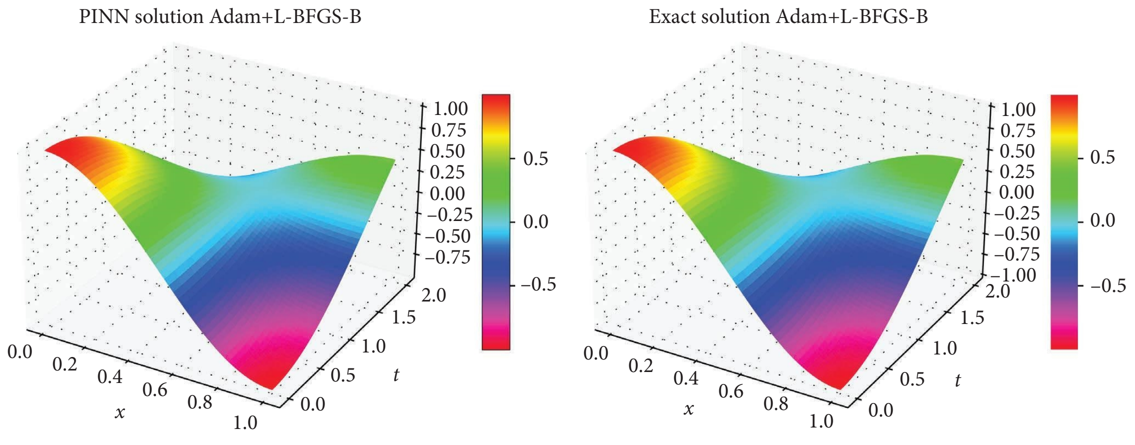
(a) (b)

Figure 4: 3D plots of the (a) PINN solution and (b) true solution for the dirichlet BCs using the adam optimization algorithm.



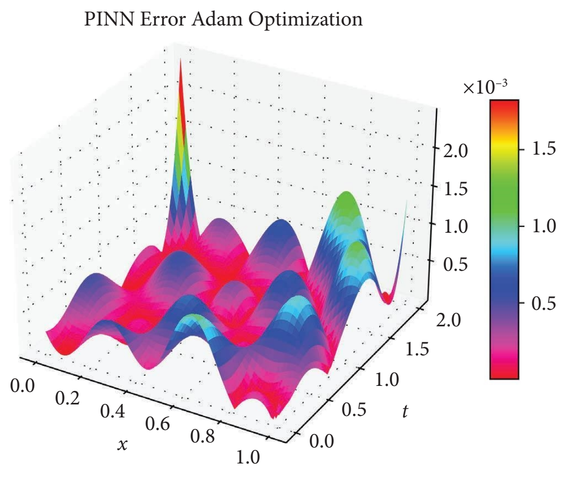
(a) (b)

Figure 5: 3D plots of the (a) PINN solution and (b) true solution for the dirichlet BCs using the L-BFGS-B optimization algorithm.

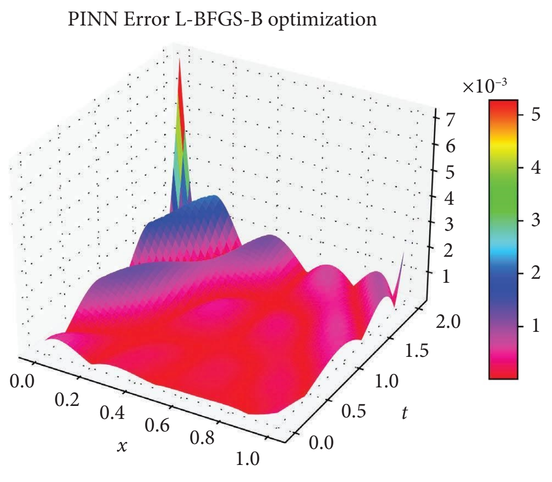


(a) (b)

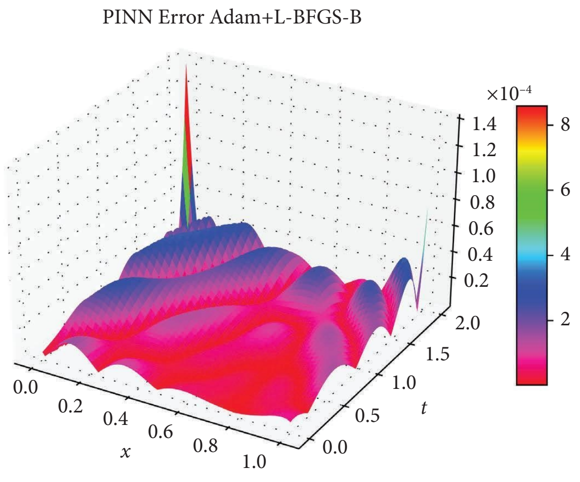
Figure 6: 3D plots of the (a) PINN solution and (b) true solution for the dirichlet BCs using combined adam and L-BFGS-B optimization algorithms.



(a)



(b)



(c)

Figure 7: 3D plots of the PINN point-wise absolute error (the difference of exact and PINN solution) for the dirichlet BCs using (a) adam (b) L-BFGS-B, and (c) combined adam & L-BFGS-B optimization algorithms.

Table 1: The comparisons of error approximation for different model optimizers.

Optimizer	L_2 error	L_∞ error	Relative error	MSE	Time in seconds
Adam	$6.659841e-02$	$4.862466e-03$	$1.039621e-03$	$3.329921e-04$	30.022
L-BFGS	$5.103039e-02$	$3.959396e-03$	$7.965995e-04$	$2.551519e-04$	58.996
Combined	$2.930245e-02$	$1.861282e-03$	$4.574200e-04$	$1.465123e-04$	70.341

indicating that our model showed the greatest performance improvement when the number of nodes is 50. Furthermore, a comparison between the solution produced using the suggested method and the exact one, based on L_∞, relative and mean square error for the five distinct nodes, is presented in Table 2.

As we can see from the table, NNs with node counts of 30, 100, 150, and 200, along with the corresponding hidden layers, have a nearly uniform pattern, and the NN with node 50 gives smaller L_∞, relative and mean squared error, indicating that the suggested approach is efficient for the NN architecture with 50 nodes.

(6) Discussion on the Selection of Activation Function. When using PINNs to solve PDEs, the choice of activation functions have an impact on the performance and convergence of the model. We provided a comparison between a few well-known activation functions in to determine which activation best minimized the loss function of our suggested model.

In Table 3 the approximation error of the proposed model based on L_∞, relative and mean square error is presented for the activation functions of Tanh, sigmoid, and ReLu. According to the findings, applying the sigmoid activation function yields a better result than the ReLu

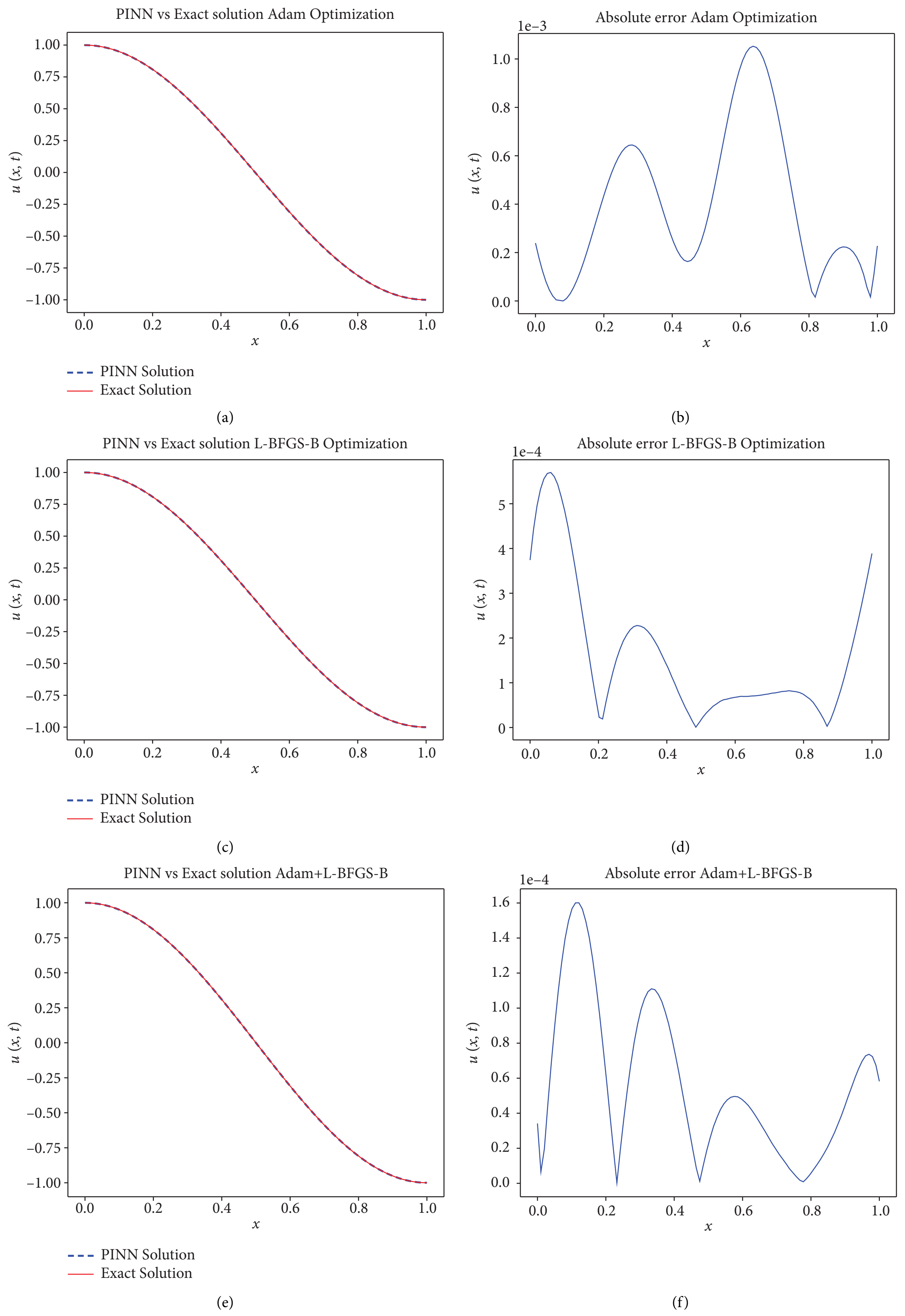


FIGURE 8: The line plots of the comparison of PINN-predicted with the exact solution and the corresponding absolute error for NLSGE dirichlet BCs case at $t = 0.5$ using the adam optimization algorithm (the first row (a) and (b)), the L-BFGS-B optimization algorithm (the second row (c) and (d)), and the combined adam and L-BFGS-B optimization algorithms (the third row (e) and (f)).

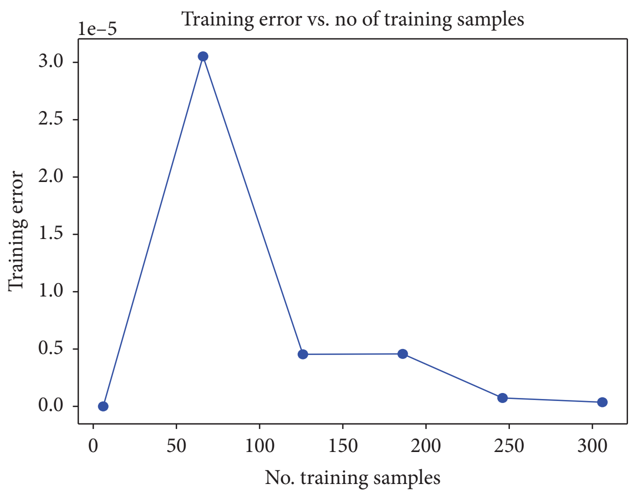


Figure 9: Training error dirichlet BCs case.

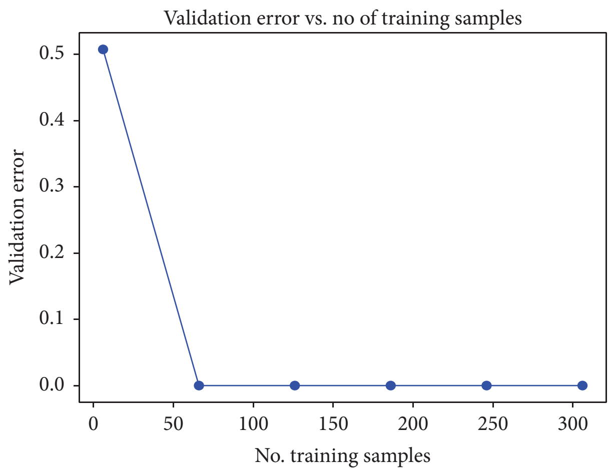


Figure 10: Error on a validation set dirichlet BCs case.

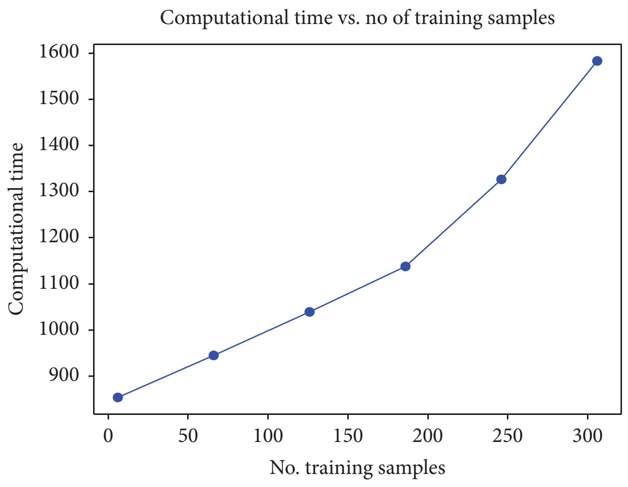


Figure 11: Computational time dirichlet BCs case.

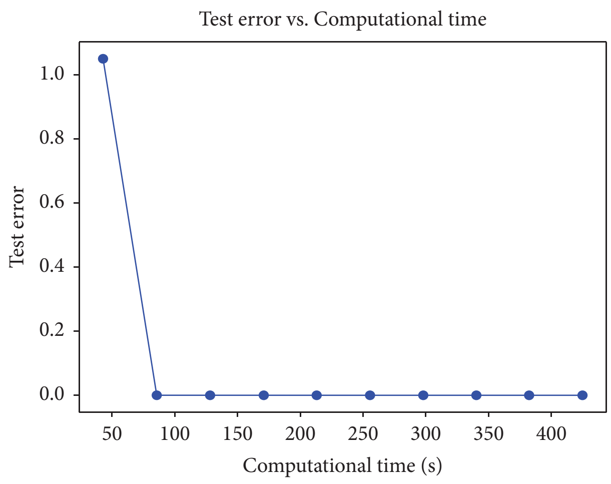


FIGURE 12: Test loss vs. computational time dirichlet BCs case.

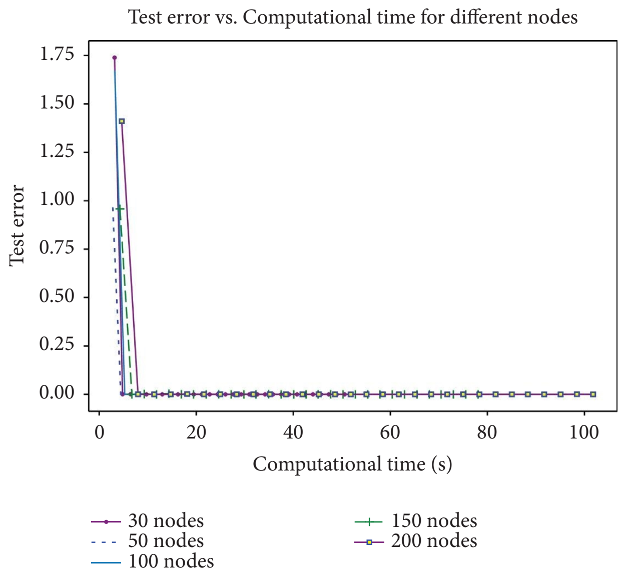


FIGURE 13: Test loss against processing time for PINN to handle the dirichlet BCs case for nodes 30, 50, 100, 150, and 200 at $t = 0.5$.

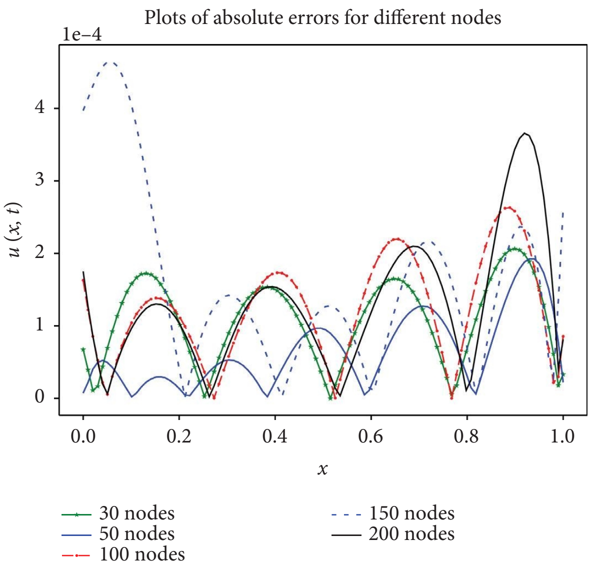


FIGURE 14: Plots of the absolute errors between PINN prediction and the real solution for the dirichlet BCs issue (16) considering different nodes of NNs at $t = 0.5$.

Table 2: The comparisons of PINN approximation errors for different nodes.

Nodes	L_∞ error	Relative error	MSE
30	$3.582307e-03$	$5.617903e-04$	$1.799422e-04$
50	$1.580038e-03$	$2.676965e-04$	$8.574355e-05$
100	$3.517338e-03$	$6.090592e-04$	$1.950825e-04$
150	$3.645369e-03$	$5.776454e-04$	$1.850206e-04$
200	$3.912070e-03$	$5.346607e-04$	$1.712526e-04$

Table 3: The comparisons of PINN error approximation for different activation functions.

Activation functions	L_∞ error	Relative error	MSE
Tanh	$1.580038e-03$	$2.676965e-04$	$8.574355e-05$
Sigmoid	$4.186728e-03$	$8.194311e-04$	$2.083582e-04$
ReLu	$2.937392e-02$	$4.524396e-03$	$1.449170e-03$

activation function. However, compared to the other methods, using the tangent hyperbolic function (tanh) yields the best approximation with the least amount of error.

Figure 15 demonstrates a comparison of the suggested approximate error line plots for these three different activation functions. The graph shows that the inaccuracy of the tanh activation function is nearly zero compared to the other lines, which also indicates that the tangent hyperbolic function is appropriate for our proposed model.

4.2. 1D NLSGE with Neumann BCs. Consider the one-dimensional NLSGE

$$\frac{\partial^2 u}{\partial t^2} = \frac{1}{\pi^2}\frac{\partial^2 u}{\partial x^2} - \sin(u) + \sin(\cos(\pi x)\cos t), \quad 0 \le x \le 1, 0 < t < 2, \tag{31}$$

with the Neumann boundary conditions

$$u_x(0,t) = 0, \quad 0 \le t \le 2, \tag{32}$$

$$u_x(1,t) = 0, \quad 0 \le t \le 2, \tag{33}$$

and initial conditions

$$u(x,0) = \cos(\pi x), \quad 0 < x < 1, \tag{34}$$

$$u_t(x,0) = 0, \quad 0 < x < 1. \tag{35}$$

The function $u(x,t) = \cos(\pi x)\cos(t)$ satisfies the (31) and conditions (26)–(29) [32]. The NN with two nodes in the input layer (x,t), one node in the output layer (value of $u(x,t)$)), and four hidden layers, each with 50 nodes, produces $u(x,t)$), which solves (25), for the given input (x,t)). We set this model's epoch count to 15000.

4.2.1. Training Dataset. The training set we used in this example consisted of 500 samples $\{(x^i,t^i); u(x^i,t^i)\}_{i=1}^{500}$ where $u(x^k,t^k)$ is the solution of (31) at (x^k,t^k) found by a PDE solver python offers deepxde. 300 training samples were chosen from $(0,1)\times(0,2)$ and the rest was taken from the domain boundary.

4.2.2. Loss Function. Similarly to the above example, the loss function is expressed as the summation of the square of the difference corresponding to each of the equations in (31). The loss function used to train the PINN with the parameter P is given by (15) where

$$J_\Omega(X^\Omega; P) = \frac{1}{N_\Omega}\sum_{(x^i,t^i)\in X^\Omega}\left|r(x^i,t^i)\right|^2, \tag{36}$$

where

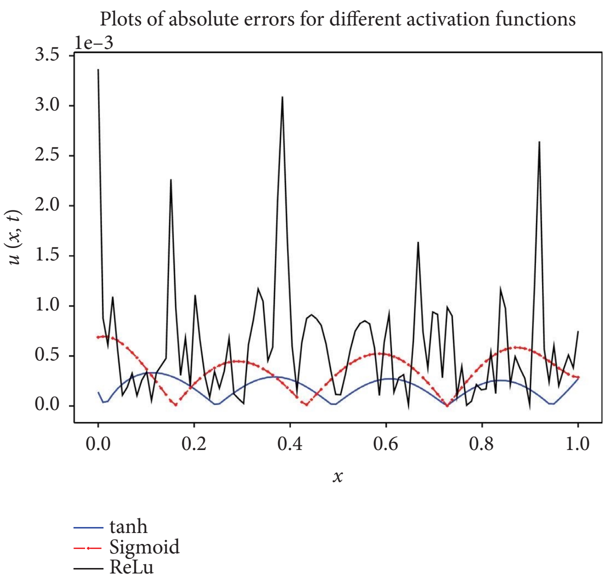


FIGURE 15: The comparisons of PINN approximation error to handle the dirichlet BCs case for tanh, sigmoid, and ReLu activation functions at $t = 0.5$.

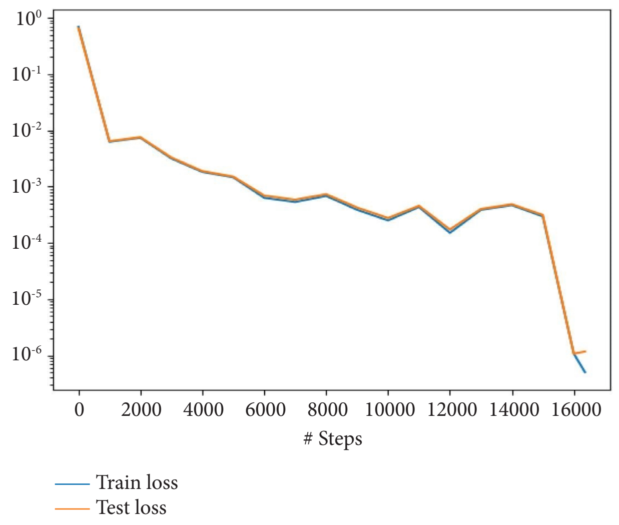


FIGURE 16: Train and test loss of PINN process for 15000 epochs (training iterations) for the neumann BCs case by using combined adam and L-BFGS-B optimization algorithms.

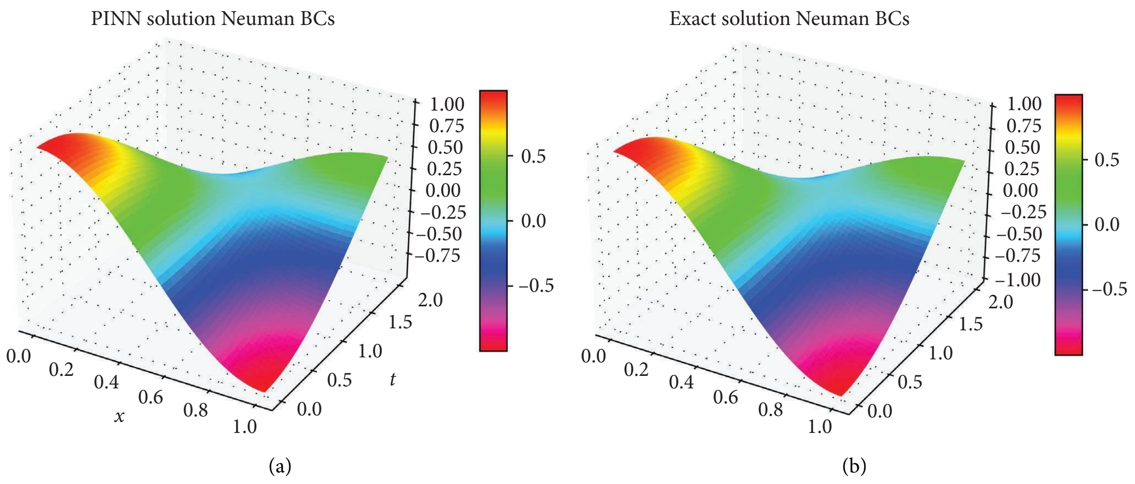
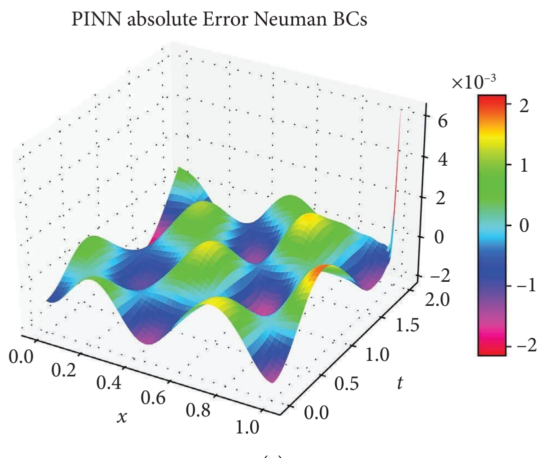


FIGURE 17: Continued.



(c)

FIGURE 17: 3D plots of the (a) PINN solution, (b) true solution, and (c) PINN point-wise absolute error (the difference of exact and PINN solution) to the problem (25) with the neumann BCs.

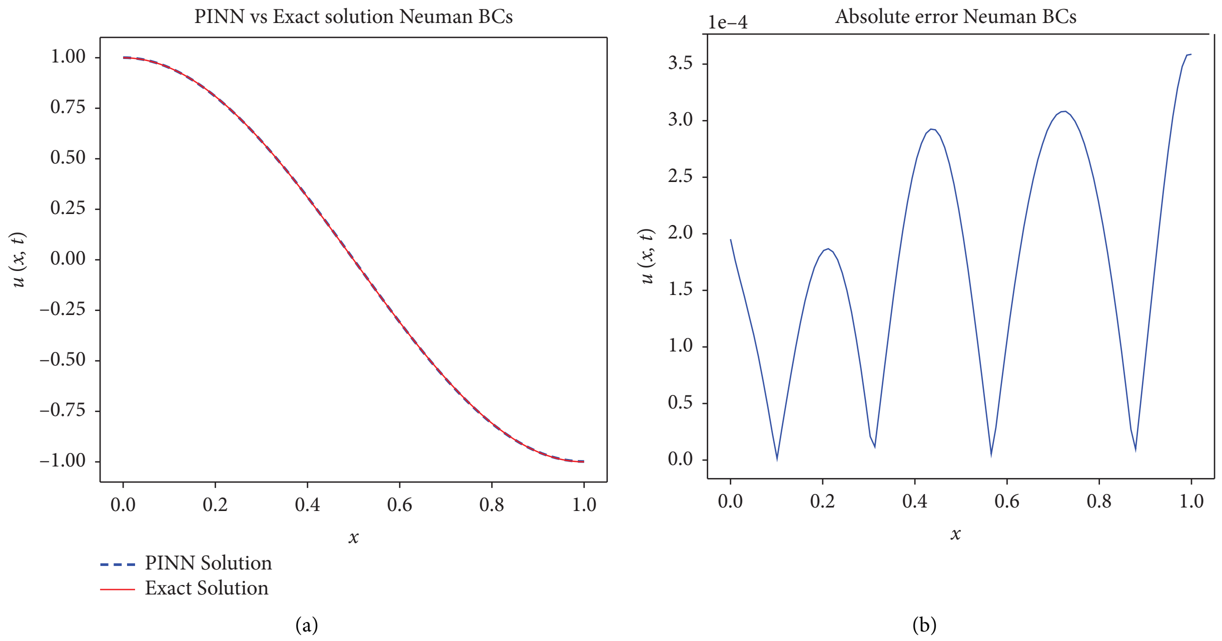


FIGURE 18: The line plots of (a) PINN-predicted and exact solution and (b) absolute error to SGE equation (25) with neumann BCs case for $t = 0.5$.

$$r\left(x^i, t^i\right) = \widehat{u}_{tt}\left(x^i, t^i\right) - \frac{1}{\pi^2}\widehat{u}_{xx}\left(x^i, t^i\right) + \sin\left(\widehat{u}\left(x^i, t^i\right)\right) - \sin\left(\cos\left(\pi x^i\right)\cos t^i\right), \tag{37}$$

$$J_\Gamma\left(X^\Gamma; P\right) = \frac{1}{N_{\Gamma_1}}\sum_{\left(x^i, t^i\right)\in X^{\Gamma_1}}\left|\widehat{u}_x\left(x^i, t^i\right) - 0\right|^2 + \frac{1}{N_{\Gamma_2}}\sum_{\left(x^i, t^i\right)\in X^{\Gamma_2}}\left|\widehat{u}_x\left(x^i, t^i\right) - 0\right|^2, \tag{38}$$

$$J_0\left(X^0; P\right) = \frac{1}{N_0}\sum_{\left(x^i, t^i\right)\in X^0}\left|\widehat{u}\left(x^i, t^i\right) - \cos\left(\pi x^i\right)\right|^2 + \frac{1}{N_0}\sum_{\left(x^i, t^i\right)\in X^0}\left|\widehat{u}_t\left(x^i, t^i\right) - 0\right|^2. \tag{39}$$

The train and test loss of this model is shown in Figure 16. Since we have previously demonstrated that, for example one, the combined Adam and L-BFGS-B optimization algorithm is the optimal optimization for our model, we employed this mixed optimization technique to minimize the loss function of the problem (31). Similarly, we used the tanh as an activation function to predict the solution of the suggested instances given by (31) by PINN.

Figures 17 and 18 show the exact solution and the resulting PINN solution with the corresponding absolute error to the problem (31). The graphs of the 2D and 3D solution plots for the combined model optimization Adam and L-BFGS-B allow a comparison of the two solutions.

The distinction between the precise and PINN solutions is seen in Figure 17(c), and we observe that the difference between these solutions equals mostly zero, which suggests a reasonable match between these two solutions. The overlap between the line plots representing the precise solution and the predicted solution, as seen in Figure 18(a), indicates that our suggested model provides an excellent approximation with the least amount of error, as demonstrated by the corresponding error plots in Figure 18(b).

5. Conclusions and Outlook

In this paper, we have presented a deep learning framework-based approach known as PINNs for the solution of nonlinear SGE with source terms. To solve efficiently the proposed problem, we provided PINN with a multi-objective loss function that incorporates the initial condition, Dirichlet/Neumann boundary conditions, and governing PDE residual over randomly selected collocation points in the problem domain. We used a feedforward deep neural network with two input layers, four hidden layers, and one output layer to train the PINN model. The weights of the feedforward NNs were initialized using a Glorot uniform initialization, also called a uniform Xavier initialization, which is the most appropriate when employing a symmetric activation function, such as the tanh or sigmoid. We looked at the NLSGE with Dirichlet and Neumann boundary conditions as benchmark examples to demonstrate how well the suggested model performed. We conducted several experiments and utilized graphs and tables to simulate the results using the Python DeepXDE software module. The PINN model's train and test loss for both the Dirichlet and Neumann boundary conditions are decreasing with respect to training iterations; this suggests that the model is making progress in resolving the given problem by improving its approximation of the NLSGE solution. The experiment on choosing the optimal optimization method for the proposed problem shows that the L-BFGS-B model optimization algorithm yields better results than the Adam optimization strategy. However, integrating the two gives the best result, but compiling the model takes more time. Furthermore, three activation functions ReLU, Sigmoid, and hyperbolic tangent (tanh) function are examined to determine the best choice of activation function to utilize with the suggested model. Results indicate that the tanh activation function produces the most accurate results, whereas the ReLU activation function produces the least accurate results (see Table 3 and Figure 15). Graphs and tables are used to depict the simulation for comparison between the exact solution and the PINN-predicted solution. The results show that the method can accurately capture the solution for the NLSGE, with the difference being extremely close to zero. To further strengthen the foundation of PINN for solving different classes of physical phenomena involving PDEs, further investigation must be performed in future work. More research is required to examine the stability, convergence, and robustness of the suggested method to solve NLSGE. Furthermore, investigating higher-order and multidimensional variants of the SGE can improve PINNs' ability to represent the complex dynamics and behavior of non-linear waves. Moreover, real-time simulations and greatly increased computational efficiency can be attained via the implementation of adaptive and parallelizable PINN architectures, such as Extended PINNs, Bayesian PINNS, Multi-fidelity PINNS, and Adaptive PINNs.

Data Availability

The literature listed in this article provides all of the data necessary for this research report.

Conflicts of Interest

Regarding the development of this manuscript, the authors have not disclosed any conflicts of interest.

Authors' Contributions

The submitted version of the article was approved by all authors who contributed equally.

Acknowledgments

The authors acknowledge the financial support provided by the Adama Science and Technology University for conducting this research. The authors of this publication would like to express their gratitude to the School of Natural Sciences at Adama Science and Technology University, especially the Department of Applied Mathematics, for providing crucial research resources.

References

[1] E. Bour, "Theorie de la deformation des surfaces," *de l'École impériale polytechnique*, vol. 22, no. 39, pp. 1–148, 1861.

[2] F. Pedit and H. Wu, "Discretizing constant curvature surfaces via loop group factorizations: the discrete sine-and sinh-gordon equations," *Journal of Geometry and Physics*, vol. 17, no. 3, pp. 245–260, 1995.

[3] J. Rubinstein, "Sine-gordon equation," *Journal of Mathematical Physics*, vol. 11, no. 1, pp. 258–266, 1970.

[4] E. G. Ekomasov, K. Y. Samsonov, A. M. Gumerov, and R. Kudryavtsev, "Nonlinear waves of the sine-gordon equation in the model with three attracting impurities," *Izvestiya VUZ. Applied Nonlinear Dynamics*, vol. 30, no. 6, pp. 749–765, 2022.

[5] M. El Tawil and H. El Zoheiry, "Stochastic propagation of fluxons on josephson lines," *Chaos, Solitons & Fractals*, vol. 8, no. 1, pp. 45–50, 1997.

[6] J. J. Mazo and A. V. Ustinov, "The sine-gordon equation in josephson-junction arrays," *Nonlinear Systems and Complexity*, pp. 155–175, 2014.

[7] H. Susanto, "Josephson junctions with phase shifts: stability analysis of fractional fluxons," *PROEFSCHRIFT*, vol. 68, no. 2, 2006.

[8] M. Arakelyan, "Analysis of the motion of frenkel-kontorova dislocations in single crystals of aluminum with allowance for the peierls barrier," *OALib*, vol. 05, no. 3, pp. 1–11, 2018.

[9] M. De Angelis, "Mathematical contributions to the dynamics of the josephson junctions: state of the art and open problems," 2015, https://arxiv.org/abs/1509.03054.

[10] L. M. Alonso, "Soliton classical dynamics in the sine-gordon equation in terms of the massive thirring model," *Physical Review D*, vol. 30, no. 12, pp. 2595–2601, 1984.

[11] G. Agrawal, *Nonlinear Fiber Optics*, academic, Cambridge, MA, USA, 5th edition, 2013.

[12] M. Tinkham and V. Emery, *Introduction to Superconductivity*, University of California, California, Irvine, 1996.

[13] V. G. Bykov, "Sine-gordon equation and its application to tectonic stress transfer," *Journal of Seismology*, vol. 18, no. 3, pp. 497–510, 2014.

[14] A. Scott, *Nonlinear Science*, Oxford University Press, Oxford, UK, 1999.

[15] S. P. Joseph, "Traveling wave exact solutions for general sine-gordon equation," *Advances in Mathematics: Scientific Journal*, vol. 9, no. 4, pp. 2293–2298, 2020.

[16] G. Chen, Z. Ding, C.-R. Hu, W.-M. Ni, and J. Zhou, "A note on the elliptic sine-gordon equation," *Contemporary Mathematics*, vol. 357, pp. 49–68, 2004.

[17] S. Watanabe, H. S. van der Zant, S. H. Strogatz, and T. P. Orlando, "Dynamics of circular arrays of josephson junctions and the discrete sine-gordon equation," *Physica D: Nonlinear Phenomena*, vol. 97, no. 4, pp. 429–470, 1996.

[18] J. D. Gibbon, I. N. James, and I. M. Moroz, "The sine-gordon equation as a model for a rapidly rotating baroclinic fluid," *Physica Scripta*, vol. 20, no. 3-4, pp. 402–408, 1979.

[19] J. E. Macías-Díaz, "Numerical study of the transmission of energy in discrete arrays of sine-gordon equations in two space dimensions," *Physical Review A*, vol. 77, no. 1, Article ID 016602, 2008.

[20] M. Hairer and H. Shen, "The dynamical sine-gordon model," *Communications in Mathematical Physics*, vol. 341, no. 3, pp. 933–989, 2016.

[21] O. Goubet, "Remarks on some dissipative sine-gordon equations," *Complex Variables and Elliptic Equations*, vol. 65, no. 8, pp. 1336–1342, 2020.

[22] Q. Zhou, M. Ekici, M. Mirzazadeh, and A. Sonmezoglu, "The investigation of soliton solutions of the coupled sine-gordon equation in nonlinear optics," *Journal of Modern Optics*, vol. 64, no. 16, pp. 1677–1682, 2017.

[23] M. Wu, G. Chen, and S. Luo, "Generalized sine–gordon equation and dislocation dynamics of superlattice," *Superlattices and Microstructures*, vol. 59, pp. 163–168, 2013.

[24] L. Q. English, "Experimental results for the sine-gordon equation in arrays of coupled torsion pendula," *Nonlinear Systems and Complexity*, pp. 111–129, 2014.

[25] A. Babu and N. Asharaf, "Numerical solution of nonlinear sine-gordon equation using modified cubic b-spline-based differential quadrature method," *Computational Methods for Differential Equations*, vol. 11, no. 2, pp. 369–386, 2023.

[26] M. Shiralizadeh, A. Alipanah, and M. Mohammadi, "Numerical solution of one-dimensional sine-gordon equation using rational radial basis functions," *Journal of Mathematical Modeling*, vol. 10, no. 3, pp. 387–405, 2022.

[27] B. Batiha, "New solution of the sine-gordon equation by the daftardar-gejji and jafari method," *Symmetry*, vol. 14, no. 1, p. 57, 2022.

[28] A. T. Deresse, "Double sumudu transform iterative method for one-dimensional nonlinear coupled sine-gordon equation," *Advances in Mathematical Physics*, vol. 2022, Article ID 6977692, 15 pages, 2022.

[29] Z. Eidinejad, R. Saadati, J. Vahidi, and C. Li, "Numerical solutions of 2d stochastic time-fractional sine–gordon equation in the c aputo sense," *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 36, no. 6, Article ID e3121, 2023.

[30] J. Fang, M. Nadeem, M. Habib, S. Karim, and H. A. Wahash, "A new iterative method for the approximate solution of klein-gordon and sine-gordon equations," *Journal of Function Spaces*, vol. 2022, Article ID 5365810, 9 pages, 2022.

[31] X. Xu, X. Luo, and H. Rabitz, "Numerical meshless solution of high-dimensional sine-gordon equations via fourier hdmr-hc approximation," *Journal of Mathematical Chemistry*, vol. 57, no. 7, pp. 1683–1699, 2019.

[32] A. T. Deresse, Y. O. Mussa, and A. K. Gizaw, "Analytical solution of two-dimensional sine-gordon equation," *Advances in Mathematical Physics*, vol. 2021, Article ID 6610021, 15 pages, 2021.

[33] F. Mirzaee, S. Rezaei, and N. Samadyar, "Solution of time-fractional stochastic nonlinear sine-gordon equation via finite difference and meshfree techniques," *Mathematical Methods in the Applied Sciences*, vol. 45, no. 7, pp. 3426–3438, 2022.

[34] A. Kamchatnov, "Modulation theory for the sine-gordon equation," 2023, https://arxiv.org/abs/2301.04360.

[35] A. T. Deresse, Y. O. Mussa, and A. K. Gizaw, "Solutions of two-dimensional nonlinear sine-gordon equation via triple laplace transform coupled with iterative method," *Journal of Applied Mathematics*, vol. 2021, Article ID 9279022, 15 pages, 2021.

[36] K. Raslan, A. Soliman, K. K. Ali, M. Gaber, and S. R. Almhdy, "Numerical solution for the sin-gordon equation using the finite difference method and the non-stander finite difference method," *Applied Mathematics*, vol. 17, no. 2, pp. 253–260, 2023.

[37] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[38] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[39] L. P. Aarts and P. Van Der Veer, "Neural network method for solving partial differential equations," *Neural Processing Letters*, vol. 14, no. 3, pp. 261–271, 2001.

[40] J. Blechschmidt and O. G. Ernst, "Three ways to solve partial differential equations with neural networks—a review," *GAMM-mitteilungen*, vol. 44, no. 2, Article ID e202100006, 2021.

[41] N. Yadav, A. Yadav, and M. Kumar, *An Introduction to Neural Network Methods for Differential Equations*, Springer, Berlin, Germany, 2015.

[42] I. A. Basheer and M. Hajmeer, “Artificial neural networks: fundamentals, computing, design, and application,” *Journal of Microbiological Methods*, vol. 43, no. 1, pp. 3–31, 2000.

[43] S. Chakraverty and S. Mall, *Artificial Neural Networks for Engineers and Scientists: Solving Ordinary Differential Equations*, CRC Press, Boca Raton, FL, USA, 2017.

[44] I. E. Lagaris, A. Likas, and D. I. Fotiadis, “Artificial neural networks for solving ordinary and partial differential equations,” *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[45] C. Bland, L. Tonello, E. Biganzoli, D. Snowdon, P. Antuono, and M. Lanza, “Advances in artificial neural networks,” *Advances in Artificial Neural Networks*, p. 119, 2020.

[46] D. Valencia, S. F. Fard, and A. Alimohammad, “An artificial neural network processor with a custom instruction set architecture for embedded applications,” *IEEE Transactions on circuits and systems I: Regular Papers*, vol. 67, no. 12, pp. 5200–5210, 2020.

[47] M. Puri, A. Solanki, T. Padawer, S. M. Tipparaju, W. A. Moreno, and Y. Pathak, “Introduction to artificial neural network (ann) as a predictive tool for drug design, discovery, delivery, and disposition: basic concepts and modeling,” in *Artificial Neural Network for Drug Design, Delivery and Disposition*, pp. 3–13, Elsevier, Amsterdam, Netherlands, 2016.

[48] T. T. Dufera, Y. C. Seboka, and C. Fresneda Portillo, “Parameter estimation for dynamical systems using a deep neural network,” *Applied Computational Intelligence and Soft Computing*, vol. 2022, Article ID 2014510, 10 pages, 2022.

[49] A. Jooya, B. Keshavarz, N. Dimopoulos, and J. S. Oberoi, “Accelerating neural network ensemble learning using optimization and quantum annealing techniques,” in *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*, pp. 1–7, New York, NY, USA, November 2017.

[50] L. Daolun, S. Luhang, Z. Wenshu, L. Xuliang, and T. Jieqing, “Physics-constrained deep learning for solving seepage equation,” *Journal of Petroleum Science and Engineering*, vol. 206, Article ID 109046, 2021.

[51] E. Small, “An analysis of physics-informed neural networks,” 2023.

[52] K. Tang, X. Wan, and C. Yang, “Das-pinns: a deep adaptive sampling method for solving high-dimensional partial differential equations,” *Journal of Computational Physics*, vol. 476, Article ID 111868, 2023.

[53] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations,” 2017, https://arxiv.org/abs/1711.10561.

[54] Y. Shin, J. Darbon, and G. E. Karniadakis, “On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes,” *Communications in Computational Physics*, vol. 28, no. 5, pp. 2042–2074, 2020.

[55] C. Beck, S. Becker, P. Grohs, N. Jaafari, and A. Jentzen, “Solving the Kolmogorov pde by means of deep learning,” *Journal of Scientific Computing*, vol. 88, no. 3, pp. 73–28, 2021.

[56] J. Berg and K. Nyström, “A unified deep artificial neural network approach to partial differential equations in complex geometries,” *Neurocomputing*, vol. 317, pp. 28–41, 2018.

[57] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, “Deepxde: a deep learning library for solving differential equations,” *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[58] V. Schäfer, *Generalization of physics-informed neural networks for various boundary and initial conditions*, Ph.D. thesis, Technische Universität Kaiserslautern, Kaiserslautern, Germany, 2022.

[59] R. Bose and A. M. Roy, “Invariance embedded physics-infused deep neural network-based sub-grid scale models for turbulent flows,” *Engineering Applications of Artificial Intelligence*, vol. 128, Article ID 107483, 2024.

[60] A. M. Roy, R. Bose, V. Sundararaghavan, and R. Arróyave, “Deep learning-accelerated computational framework based on physics informed neural network for the solution of linear elasticity,” *Neural Networks: The Official Journal of the International Neural Network Society*, vol. 162, pp. 472–489, 2023.

[61] A. M. Roy and S. Guha, “A data-driven physics-constrained deep learning computational framework for solving von mises plasticity,” *Engineering Applications of Artificial Intelligence*, vol. 122, Article ID 106049, 2023.

[62] D. Sana, “Approximating the wave equation via physics informed neural networks: various forward and inverse problems,” 2022, https://dcn.nat.fau.eu/wp-content/uploads/FAUMoD_DaniaSana-InternReport_PINN.pdf.

[63] J. Li and J. Qu, “Barycentric Lagrange interpolation collocation method for solving the sine–gordon equation,” *Wave Motion*, vol. 120, Article ID 103159, 2023.

[64] L. T. K. Nguyen and N. F. Smyth, “Modulation theory for radially symmetric kink waves governed by a multi-dimensional sine-gordon equation,” *Journal of Nonlinear Science*, vol. 33, no. 1, p. 11, 2023.

[65] O. Calin, *Deep Learning Architectures*, Springer, Heidelberg, Germany, 2020.

[66] M. F. Dixon, I. Halperin, and P. Bilokon, *Machine Learning in Finance*, Springer, Heidelberg, Germany, 2020.

[67] D. V. Dung, N. D. Song, P. S. Palar, and L. R. Zuhal, “On the choice of activation functions in physics-informed neural network for solving incompressible fluid flows,” in *Proceedings of the AIAA SCITECH 2023 Forum*, p. 1803, Heidelberg, Germany, January 2023.

[68] C. Beck, S. Becker, P. Grohs, N. Jaafari, and A. Jentzen, “Solving stochastic differential equations and Kolmogorov equations by means of deep learning,” 2018, https://arxiv.org/abs/1806.00421.

[69] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck, “An overview on deep learning-based approximation methods for partial differential equations,” 2020, https://arxiv.org/abs/2012.12348.

[70] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, “Scientific machine learning through physics-informed neural networks: where we are and what’s next,” 2022, https://arxiv.org/abs/2201.05624.

[71] T. T. Dufera, “Deep neural network for system of ordinary differential equations: vectorized algorithm and simulation,” *Machine Learning with Applications*, vol. 5, Article ID 100058, 2021.

[72] C. J. García-Cervera, M. Kessler, and F. Periago, “Control of partial differential equations via physics-informed neural networks,” *Journal of Optimization Theory and Applications*, vol. 196, no. 2, pp. 391–414, 2023.

[73] Z. Reitermanova, “Data Splitting,” *WDS, Matfyzpress Prague*, pp. 31–36, 2010.

[74] V. R. Joseph, “Optimal ratio for data splitting,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 15, no. 4, pp. 531–538, 2022.

[75] A. Williams, N. Walton, A. Maryanski, S. Bogetic, W. Hines, and V. Sobes, “Stochastic gradient descent for optimization

for nuclear systems," *Scientific Reports*, vol. 13, no. 1, p. 8474, 2023.

[76] T.-D. Guo, Y. Liu, and C.-Y. Han, "An overview of stochastic quasi-Newton methods for large-scale machine learning," *Journal of the Operations Research Society of China*, vol. 11, no. 2, pp. 245–275, 2023.

[77] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings*, pp. 249–256, Quebec, Canada, January 2010.

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, Santiago, Chile, December 2015.

[79] S. Wang, S. Sankaran, and P. Perdikaris, "Respecting causality is all you need for training physics-informed neural networks," 2022, https://arxiv.org/abs/2203.07404.

[80] N. Doumèche, G. Biau, and C. Boyer, "Convergence and error analysis of pinns," 2023, https://arxiv.org/abs/2305.01240.