BOHR International Journal of Smart Computing and Information Technology
2023, Vol. 4, No. 1, pp. 55–61
DOI: 10.54646/bijscit.2023.37
www.bohrpub.com

RESEARCH

Artificial neural network with the Levenberg-Marquardt algorithm for numerical solution of two-dimension Poisson's equation

Anup Kumar Thander[1]* and Dwaipayan Bhowmik[2]

[1]Department of Applied Science and Humanities Guru Nanak Institute of Technology, Kolkata, India
[2]Department of Computer Science and Engineering Dr. Sudhir Chandra Sur Institute of Technology and Sports Complex, Kolkata, India

***Correspondence:**
Anup Kumar Thander,
anup.thander@gnit.ac.in

Received: 13 October 2023; **Accepted:** 21 October 2023; **Published:** 15 December 2023

This study introduces an Artificial Neural Network (ANN) framework to address the two-dimensional Poisson's equation within a rectangular domain. It places a focus on the training process of a neural network with three layers, incorporating hidden neurons. The feedforward ANN is trained using MATLAB, which calculates weights for all neurons within the network structure. These acquired weights are subsequently applied in the trained network model to make predictions for the desired output of a specific partial differential equation. The architecture of the ANN consists of three layers: one input layer, one hidden layer, and one output layer. In this study, we specifically employ an ANN configuration with 50 hidden neurons. The training process is executed using MATLAB, utilizing the Levenberg–Marquardt algorithm (LMA) for optimization. Furthermore, the study encompasses the development of surface and contour plots that illustrate the computational solution of the partial differential equation. Additionally, error functions are graphed to assess the effectiveness of the ANN model.

Keywords: artificial neural network, Levenberg–Marquardt algorithm, Poisson's equation, optimization algorithms, numerical solution

1. Introduction

Differential equations hold fundamental importance in modeling a wide range of scientific issues across disciplines like physics, chemistry, biology, and economics (1). While analytical tools can be used to formulate equations based on well-established principles of physics, discovering closed-form solutions is often a challenging, if not impossible, task (1–23). As a result, various methods have been proposed in the literature to address the solution of these equations. Several key numerical techniques (1, 13) employed in solving partial differential equations (PDEs) include the Finite Difference Method (FDM) (1, 8, 11, 13), the Finite Element Method (FEM) (1, 11), the Finite Volume Method (FVM), and the Boundary Element Method (BEM). These methods typically require discretizing the problem domain into finite grid points (8).

In contrast, neural networks can be seen as approaches for approximating strategies to solve differential equations. The solutions obtained through neural networks are differentiable, expressed in a closed analytical format (1), and easily usable in subsequent computations. This sets them apart from many other techniques, which often yield discrete solutions with limited differentiability. Several recent research articles (1–7, 9–18, 20–23) and even a book have been dedicated to this topic. The core aim of this study is to employ a Feed-Forward neural network to solve Poisson's equation within a specific domain. Subsequently, the numerical solution obtained through this neural network will be compared with the exact solution. This investigation aims to assess the efficacy of neural networks in solving such

problems and understand how they perform in comparison to traditional numerical methods.

The paper is structured as the following sections: Section II covers the details of the partial differential equation and domain discretization. In Section III, an overview of the feedforward neural network and its properties is presented. Section IV delves into the discussion of the Levenberg–Marquardt Algorithm (LMA). Section V provides a comprehensive presentation of detailed numerical results. Finally, Section VI offers concluding remarks.

2. The Poisson equation within a rectangular region

Two-dimensional Poisson's Equation within rectangular domain where $x \in [0, A]$ and $y \in [0, B]$.

Such that

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = f(x, y) \tag{1}$$

Here the forcing function is,

$$f(x, y) = x(x - A) + y(y - B)$$

We prescribe Dirichlet Boundary Conditions on all four sides of the rectangle as

$$v(x, 0) = 0, v(x, A) = 0, \quad \forall x \in (0, A)$$
$$v(0, y) = 0, v(A, y) = 0, \quad \forall x \in (0, B)$$

For numerical resolution of Equation (1), we have decomposed the rectangular solution region into mesh points (11, 13). Here h_x and h_y are step length along the x and y directions, respectively. Let (x_i, y_j) be a mesh point in the region. Then $x_i = x_0 + ih_x$, $y_j = y_0 + jh_y$. We have considered (x_0, y_0) as (0,0). A = 3, B = 2, $h_x = 0.03$, $h_y = 0.02$

The exact solution of the PDE mentioned in equation (1) is,

$$v_{exact} = \frac{x(A - x)y(B - y)}{2}$$

This exact solution is used to compute the error function ($\|v_{exact} - v_{ann}\|_p$) numerically in $L_p$space (p = 2). Here v_{ann} is the solution of equation (1) using ANN.

3. Feed-forward neural network

One of the two primary types of artificial neural networks, a feedforward neural network, is distinguished by the way information is processed and sent between its various layers. Feedforward neural networks are structured as a sequence of interconnected layers (1–7). The initial layer is connected to the network's input, and each subsequent layer is linked to the one preceding it. Ultimately, the last layer generates the network's output. These networks are versatile and can be applied to map inputs to outputs across various problem domains. A feedforward neural network, particularly one equipped with a single hidden layer containing an adequate number of neurons, possesses the capability to approximate and fit any finite input-output mapping problem. In essence, it can adapt to a wide range of tasks where there is a need to transform inputs into corresponding outputs. Additionally, specialized variations of feedforward networks are available, including networks designed for fitting purposes and those tailored for pattern recognition tasks (6, 7, 17, 18). These variations allow for the network's adaptation to specific problem types, enhancing its applicability across various domains.

Equation (2) provides a mathematical representation that describes an individual neuron within Feed Forward neural network architecture in general sense.

$$a_k^l = \sum_{p=1}^{N_{l-1}} w_{k,p}^l X_p + b_{k,0}^l, \quad k = 1, 2, \ldots\ldots, N_l$$
$$y_k^l = F_l(a_k^l) \tag{2}$$

Here, N_l represents the quantity of neurons in the l layer (14). Every concealed neuron, labeled as "k," is supplied with the outcome of every input neuron "p" originating from the input layer, which is scaled by weight ($w_{k,p}^l$). The total of weighted inputs is then fed into activation function F_l to determine the output of concealed layer neuron. This information is subsequently moved ahead to the output layer, and a comparable process involving weighted sums takes place for each output neuron. The term $b_{k,0}^l$ signifies the bias of the neuron indexed as "k" in the "lth" layer. Bias values are incorporated to introduce a degree of randomness to the initial conditions, which ultimately enhances the network's likelihood of reaching convergence. In Equation (3), the weight matrix is defined as the connector between the ("$l-1$")-th layer and the "lth" layer (14).

$$\omega^l = \begin{pmatrix} w_{1,1} & w_{1,2} & \ldots. & w_{1,l-1} \\ w_{2,1} & w_{2,2} & \ldots. & w_{2,l-1} \\ \ldots.. & \ldots.. & \ldots.. & \ldots.. \\ w_{l,1} & w_{l,2} & \ldots.. & w_{l,l-1} \end{pmatrix}_{lX(l-1)} \tag{3}$$

4. The Levenberg–Marquardt algorithm (LMA)

The Levenberg–Marquardt algorithm (LMA) (1–6, 14, 16) is a widely used trust region method designed to locate a minimum of a function, be it linear or non-linear, within a parameter space. It essentially builds an internal model

of the objective function, often quadratic, to establish a trusted region. When a good fit is achieved, the trust region expands. However, like many numerical techniques, LMA can be sensitive to the initial parameter values. In traditional implementations of the Levenberg–Marquardt method, finite differences are employed to approximate the Jacobian matrix. Within the realm of artificial neural networks, this method is well-suited for training small to medium-sized problems.

It combines elements from both gradient descent and Gauss–Newton methods, resulting in an adaptive and reliable optimization tool. LMA often assures successful problem-solving due to its adaptable nature. However, when we represent the back-propagation (BP) method as gradient descent, the algorithm tends to slow down and may not achieve an optimal solution. Conversely, if we express BP as Gauss–Newton, the algorithm substantially increases the likelihood of reaching an optimal solution. Within this algorithm, an approximation for calculating the Hessian matrix (H_a) is presented in Eqn (4), while the gradient (G) computation is expressed in Eqn (5).

$$H_a = J_a^t J_a \tag{4}$$

$$G = J_a^t \, E_r \tag{5}$$

Here, the Jacobian matrix is denoted as "J_a" and "E_r" represents a vector representing the network's error. In this context, the Levenberg–Marquardt Algorithm (LMA) exhibits behavior akin to the Newton method. This can be articulated via the subsequent convergence procedure:

$$Z_{k+1} = Z_k - \left[J_a^t J_a + \mu I\right]^{-1} J_a^t \, E_r \tag{6}$$

In this context, "Z_{k+1}" denotes a new weight value, which is computed by applying the gradient function to the current weight "Z_k" using the Newton algorithm. Here I is the identity matrix and μ is the learning factor.

Notably, it can successfully converge even when the error landscape is notably more intricate than a straightforward quadratic scenario. Core concept behind the Levenberg–Marquardt algorithm involves a hybrid training approach: in regions with intricate curvature, the steepest descent technique is employed until the local curvature is suitable for a quadratic estimation. Subsequently, this transition leads to an approximation akin to the Gauss–Newton algorithm, notably expediting the convergence.

5. Design and training ANN

Training process of an ANN involves adjusting weights and biases based on input data to reduce the discrepancy between the network's outputs and the intended target results. For comprehensive overview of neural networks, please refer to (12, 14). To train a multi-layer ANN, the Backpropagation (BP) algorithm is commonly utilized to iteratively update the weights and biases. Precision of ANN heavily depends on the availability of substantial training dataset.

Typically, the training data are segmented into three separate parts: training, validation, and testing sets. Each of these divisions is utilized separately to assess the training's effectiveness. This approach allows for the assessment of the entire dataset's training results and facilitates comparisons between various training algorithms and ANN architectures.

In MATLAB, we have generated a substantial dataset consisting of 100 × 100 samples for solving Poisson's equation. Throughout the training phase, these samples are categorized into three distinct subsets: Seventy percent is designated for training, fifteen percent is set aside for validation, and the remaining fifteen percent is reserved for testing. Training of the ANN is conducted using the Levenberg–Marquardt algorithm.

We have completed three training cycles for each network configuration using this method, with each training cycle consisting of 1,000 epochs (14). "Epoch" denotes the frequency of occurrences that all the training data samples are utilized once for updating the network's weights.

Figure 1 illustrates the artificial neural network architecture designed for the numerical solution of the partial differential equation outlined in Eqn (1).

Figures 2 and **3** depict the surface and contour plots, respectively, showcasing the numerical solution obtained using the current method for the partial differential equation.

The evaluation of the network training's effectiveness is done by quantifying the discrepancy between the computed output (y^{nn}) of the neural network and the intended target output for training (y^t). In essence, we set a threshold error value that is deemed sufficiently small for us to consider the output with precision. The assessment of the network training operation depends on the speed and efficiency with which this error approaches the predefined cutoff point. Usually, the most widely employed approach for measuring the output error involves Mean Squared Error (MSE), as shown in equations (7, 14).

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i^{nn} - y_i^t)^2 \tag{7}$$

Here, N denotes the quantity of outputs.

In the MATLAB simulation environment, we use 64-bit floating-point data representation to represent weights, biases, and training data within the ANN model.

Figure 4 through **Figure 5** display training performance results of the Levenberg–Marquardt algorithm (LMA).

In **Figure 4**, we observe the performance curve where MSE decreases as the number of epochs increases. It is noteworthy that the error in the test set and the error in the validation set show analogous patterns. Importantly, there is no prominent overfitting issue observed up to epoch 1000, which corresponds to the point where the best validation performance is achieved, and the MSE reaches a remarkably

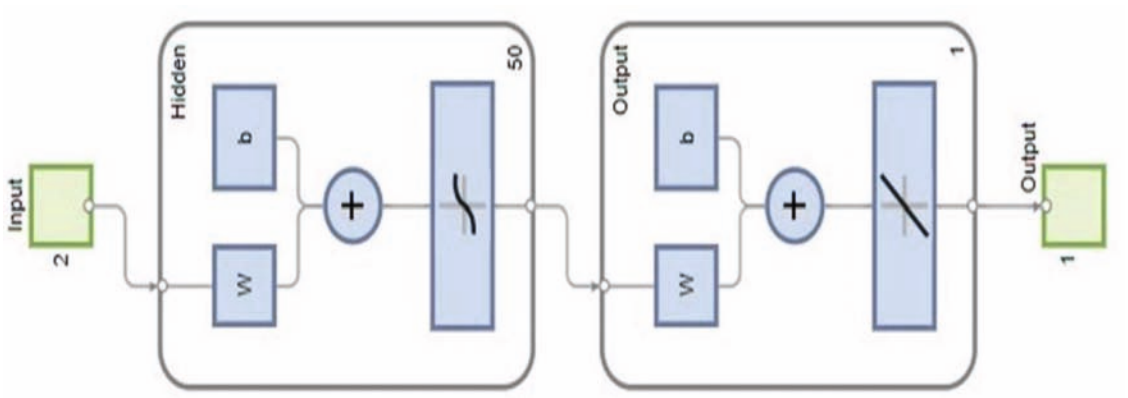

FIGURE 1 | ANN Architecture with 50 hidden layers for the PDE.

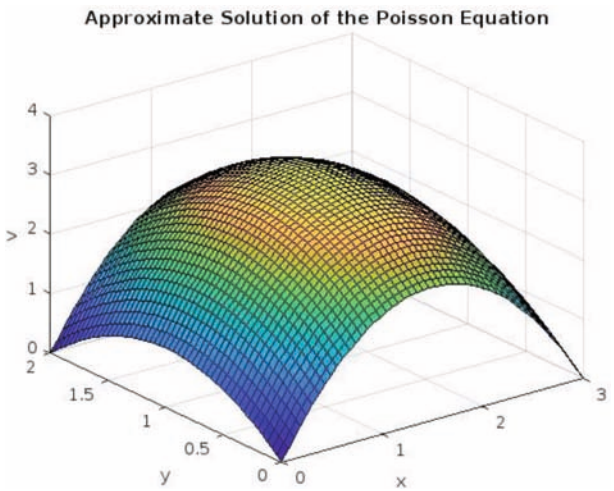

FIGURE 2 | Surface plot of solution of the PDE.

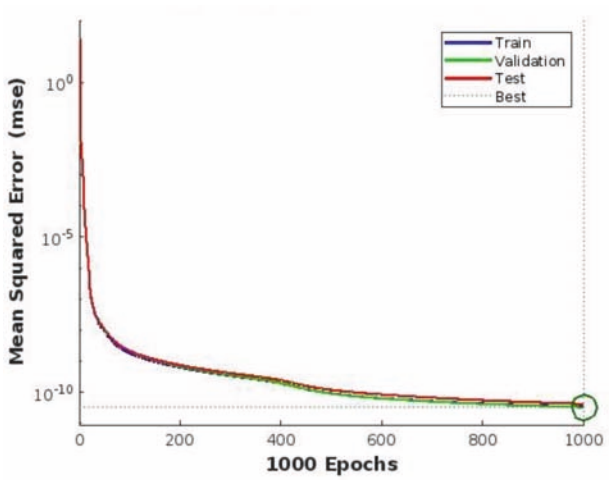

FIGURE 4 | Mean Squared Error (MSE) values for different Epochs.

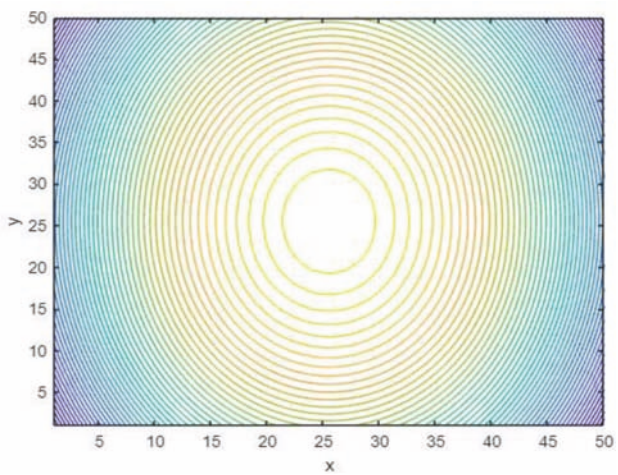

FIGURE 3 | Contour plot of solution of the PDE.

low value of 3.4856e-11. **Figure 6**, which displays the gradient plot, offers insights into the optimization techniques employed to attain global solutions.

In **Figure 7**, we have an error histogram divided into 20 bins within the ANN model, representing the training, validation (check), and test data. The errors are determined by subtracting the predicted output values from the actual target values for each specimen. In this figure, the yellow line serves as a reference for zero error, and we can observe that there are 175 instances in the training phase that achieve these zero errors.

In **Figure 5**, we depict the connection between the measured target values and the predicted output values, separately for both the training and test phases. R-squared (R^2) is a statistical metric that quantifies the proximity of the data points to the fitted regression line. Across all four sub-figures, we observe that the model yields favorable results, as evidenced by the R values. It is noteworthy that in **Figure 5**, the values for the slope (m) and intercept (b) are roughly 1 and 0, respectively, across all cases, indicating a strong fit. Furthermore, the R-value consistently hovers around 100%, signifying excellent overall performance for the entire dataset, as depicted in **Figure 5** across all scenarios.

By plotting the discrepancy between the computed solution and the analytical solution, we can find that the error falls within range of a specific order 10^{-6}.

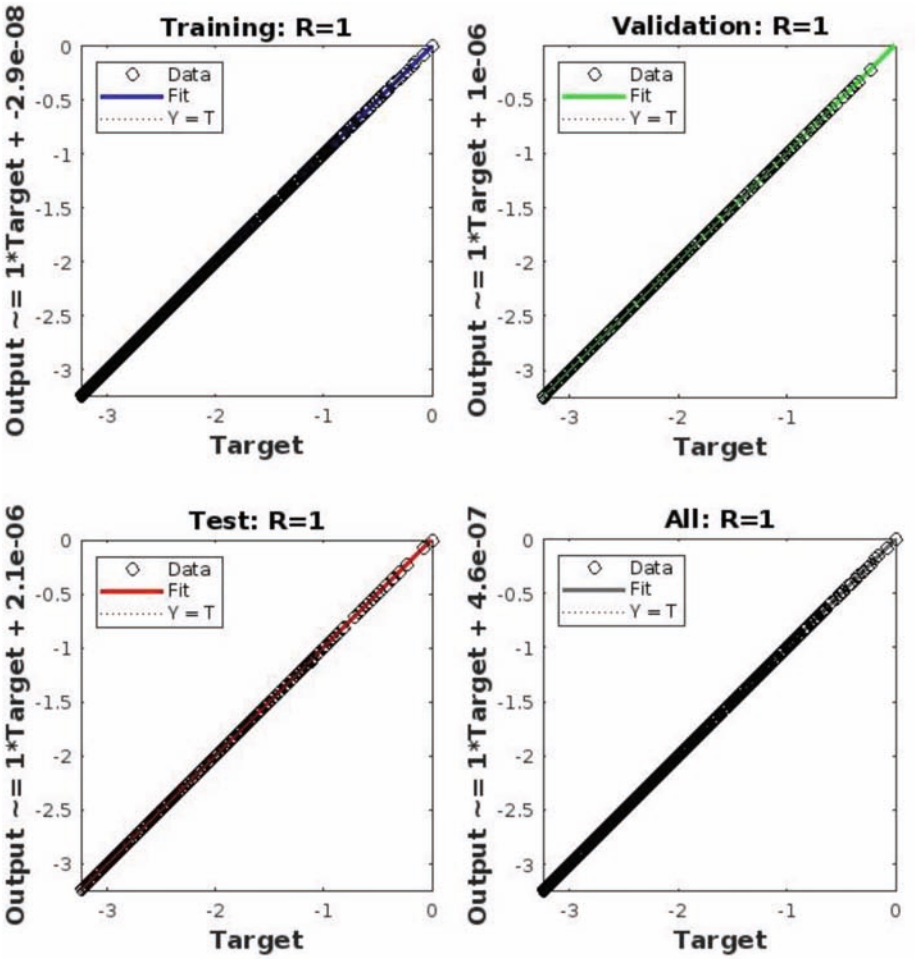

FIGURE 5 | Neural Network Training Regression Analysis.

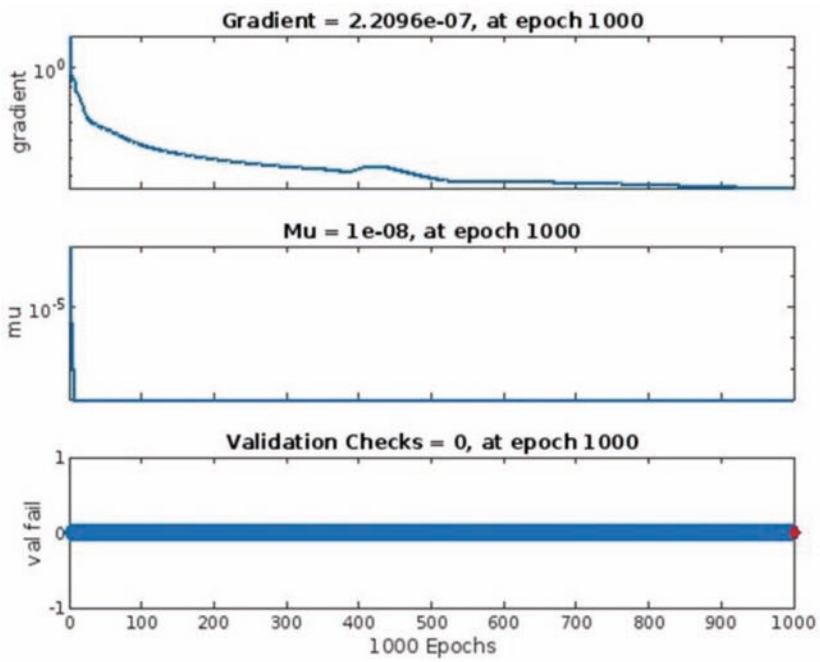

FIGURE 6 | Neural Network Training State at different epochs.

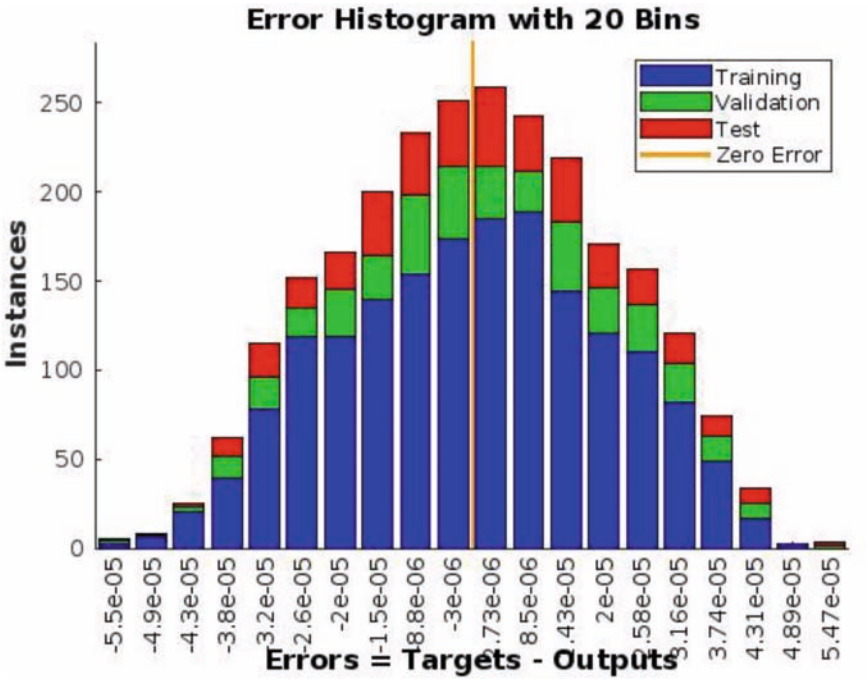

FIGURE 7 | Neural Network Training Error Histogram with 20 bins.

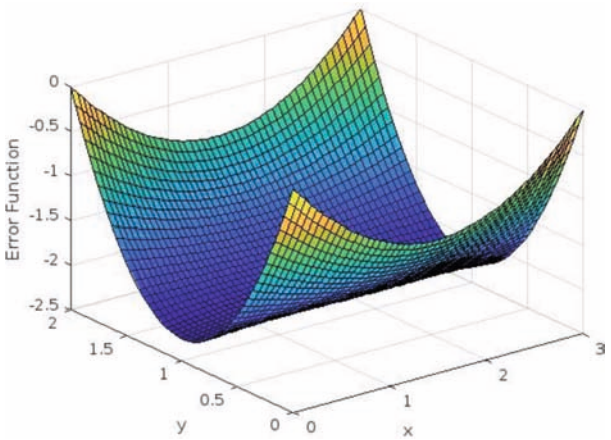

FIGURE 8 | A 3-dimensional surface representation of the error function for the PDE.

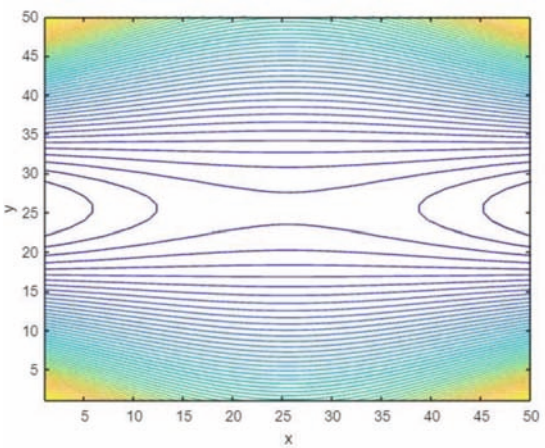

FIGURE 9 | A contour map depicting the error function for the solution of the PDE.

Figures 8 and 9 display the surface and contour plots of the error function associated with the PDE. This error function is computed in Euclidean space, comparing the analytical solution of the PDE with the numerical solution obtained using the current method.

Liu et al. (23) employed a similar approach to obtain a numerical solution for elliptic partial differential equations (PDEs) using an artificial neural network (ANN)-based radial basis function (RBF) collocation method. In their approach, the training data encompass the prescribed boundary values of the dependent variable and the radial distances between exterior fictitious sources and the boundary points of the solution domain. This technique is suitable when dealing with Dirichlet boundary conditions.

In contrast, our method is versatile, as it can handle both Dirichlet and Neumann boundary conditions. Nonetheless, a drawback of our method involves the necessity for discretizing the solution domain, a requirement that differs from Liu et al.'s approach (23). Furthermore, our proposed method has limitations when it comes to addressing elliptic PDEs with complex geometries where employing finite-difference meshes with a uniform grid size is unfeasible.

6. Conclusion

This paper is centered on the development and refinement of a specialized Artificial Neural Network (ANN) design

for addressing Poisson's equation. In particular, a 3-layer ANN structure was trained using optimization techniques, including the Levenberg–Marquardt algorithm within the MATLAB environment. Notably, the hidden layer of the ANN consisted of 50 neurons. The numerical findings demonstrate that this ANN-based method can achieve an error below a certain threshold. As part of future research endeavors, our aim is to extend this work to address the numerical solution of the Helmholtz wave equation within the context of specific rib-structured waveguides, utilizing artificial neural networks as a promising approach.

References

1. Yadav N, Yadav A, Kumar M. *An Introduction to Neural Network Methods for Differential Equations. Springer Briefs in Applied Sciences and Technology: Computational Intelligence*. Berlin: Springer (2015).
2. Jiang Z, Jiang J, Yao Q, Yang G. A neural network-based PDE solving algorithm with high precision. *Sci Rep.* (2023) 13:4479.
3. Althubiti S, Kumar M, Goswami P, Kumar K. Artificial neural network for solving the nonlinear singular fractional differential equations. *Appl Math Sci Eng.* (2023) 31:2187389.
4. Basir S, Senocak I. Physics and equality constrained artificial neural networks: application to forward and inverse problems with multi-fidelity data fusion. *J Comput Phys.* (2022) 463:111301.
5. Seo J. A pretraining domain decomposition method using artificial neural networks to solve elliptic PDE boundary value problems. *Sci Rep.* (2022) 12:13939.
6. Sun Y, Zhang L, Schaeffer H. NeuPDE: Neural Network Based Ordinary and Partial Differential Equations for Modeling Time-Dependent Data. *Proc Mach Learn Res.* (2020) 107:352–72.
7. Blechschmidt J, Ernst O. Three ways to solve partial differential equations with neural networks — A review. *GAMM-Mitteilungen.* (2021) 44:e202100006.
8. Thander AK, Bhattacharyya S. Optical confinement study of different semiconductor rib wave guides using higher order compact finite difference method. *Optik.* (2016) 127:2116–20.
9. Li Y, Hu X. Artificial neural network approximations of Cauchy inverse problem for linear PDEs. *Appl Math Comput.* (2022) 414:126678.
10. Bhattacharya K, Hosseini B, Kovachki NB, Stuart AM. Model reduction and neural networks for parametric PDEs. *SMAI J Comput Math.* (2021) 7:121–57.
11. Thander AK, Bhattacharyya S. Study of optical modal index for semiconductor rib wave guides using higher order compact finite difference method. *Optik.* (2017) 131:775–84.
12. Zhang L. Artificial neural networks model design of Lorenz chaotic system for EEG pattern recognition and prediction. *Proceedings of the 2017 IEEE Life Sciences Conference (LSC)*. London: (2017).
13. Thander AK, Mandal G. Optical waveguide analysis using alternative direction implicit (ADI) method in combination with successive over-relaxation (SOR) algorithm. *J Optics.* (2023).
14. Zhang L. Artificial Neural Network model design and topology analysis for FPGA implementation of Lorenz chaotic generator. *Proceedings of the 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*. New York, NY (2017).
15. Yadav AK, Chandel SS. Artificial neural network based prediction of solar radiation for Indian Stations. *Int J Comput Applic.* (2012) 50:975–8887.
16. Szczuka M, Slezak D. Feedforward neural networks for compound signals. *Theor Comput Sci.* (2011) 412:5960–73.
17. Sunny J, Schmitz J, Zhang L. Artificial neural network modelling of Rossler's and Chua's chaotic systems. *Proceedings of the 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*. New York, NY (2018).
18. Zhang L. Chaotic system design based on recurrent artificial neural network for the simulation of EEG Time Series. *Int J Cogn Inform Natl Intell.* (2019) 13:103.
19. Bhattacharyya S, Thander A. Study of H-field using higher-order compact (HOC) finite difference method (FDM) in semiconductor rib waveguide structure. *J Optics.* (2019) 48:345–56.
20. Dua V, Dua P. A simultaneous approach for parameter estimation of a system of ordinary differential equations, using artificial neural network approximation. *Ind Eng Chem Res.* (2012) 51:1809–14.
21. Dua V. An artificial neural network approximation-based decomposition approach for parameter estimation of system of ordinary differential equations. *Comput Chem Eng.* (2011) 35:545–53.
22. Pratama DA, Bakar MA, Ismail NB, Mashuri M. ANN-based methods for solving partial differential equations: a survey. *Arab J Basic Appl Sci.* (2022) 29:233–48.
23. Liu C, Ku C. A novel ANN-based radial basis function collocation method for solving elliptic boundary value problems. *Mathematics.* (2023) 11:3935.