



Paper Type: Original Article

Artificial Neural Networks to Investigate Mathematical Models: A Concise Review

Pathipati Lakshmi Durga^{1,*}, Sukanta Nayak¹

¹Department of mathematics, School of advanced sciences, VIT-AP university, Inavolu, Beside AP secretariat, Amaravati, AP, 522237, India; Lakshmipathipati1993@gmail.com, Sukantgacr@gmail.com.

Citation:

Received: 26 March 2025

Revised: 23 May 2025

Accepted: 21 June 2025

Lakshmi Durga, P., & Nayak, S. (2025). Artificial neural networks to investigate mathematical models: A concise review. *Systemic Analytics*, 3(3), 177-192.

Abstract

Mathematical modelling plays an essential role in understanding numerous physical phenomena, often described by Differential Equations (DEs). However, handling these models, particularly when inundated with extensive data and inputs, poses challenges for both analytical and numerical approaches. Consequently, alternative techniques are sought. In this context, Artificial Neural Networks (ANNs) become valuable tool. Researchers have dedicated efforts to enhance ANNs capability in handling various types of DE's such as Ordinary Differential Equations (ODE's), Partial Differential Equations (PDE's), and Nonlinear Partial Differential Equations (NPDE's). This pursuit has led to advancements in learning algorithms tailored for ANNs. Currently, the Levenberg-Marquardt (LM) learning technique stands out as a popular method for training ANN models. Traditional learning techniques, marked by complexities such as the size of the Jacobian matrix, error stability, and computational intensity, have spurred the development of advanced learning techniques aimed at simplifying the training process. These advancements necessitate a thorough analysis of various ANN techniques, providing insights into selecting the most suitable approach for a given system. Consequently, the objective of this paper is systematically exploring the ANNs in solving mathematical models, shedding light on their application in diverse domains.

Keywords: Artificial neural network architecture, Activation functions, Levenberg marquardt algorithm, Modified levenberg marquardt algorithms.

1 | Introduction

Over the time with the evolution of science it has been seen that mathematical models are helpful to understand real-world problems effectively. Mostly, the mathematical models are formulated in terms of (Differential Equations) DEs. Based on the nature of the problems, one can get different types of DEs viz. ODEs, coupled DEs, PDEs, and Nonlinear DEs (NDEs). To investigate the field variables various analytical and numerical methods can be employed. In this regard, it is observed that the complexity, and nonlinearity nature of the problems motivate to adopt numerical approaches to solve the same. As such, researchers use

✉ Corresponding Author: Lakshmipathipati1993@gmail.com

doi <https://doi.org/10.31181/sa33202554>



Licensee System Analytics. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0>).

few renowned approaches like shooting method, Finite Difference Method (FDM), and Finite Element Method (FEM), Finite Volume Method (FVM) to solve governing DEs numerically [1, 2, 11–13, 3–10]. The advantages of these methods include the transformation of DEs into algebraic equations that makes simple to identify the unknown field variables. Whereas, the presence of computational complexity of the system, real time data sets, variations in input parameters, and training data sets finds difficult to apply only numerical approaches. Hence, there is an essence of alternate technique for the same.

In this context, Artificial Neural Networks (ANNs) offer a promising solution to the governing DEs. The concepts of ANN are inspired from the biological neural network. Initially, ANNs was proposed by Walter Pitts and Warren McCulloch [14] in 1943. ANN was first introduced in biological science and then adopted in various other fields. The beauty of ANN is its architecture and computational structure that is mimicked as the human brain. Current trends show majority of academic publication based on AI, particularly Neural Networks. As such, the applications of ANN can be seen in various fields viz. banking and finances [15], health care [16], [17], safety [18], and entertainment [19] etc.

Furthermore, deep study on ANN and its applications reveals that ANN can be used to solve various mathematical models governed by DE that can give reliable solutions for the involved field variables. In this regard, [20] introduced a new method that approximates the solution of the DEs by a trial function. Here, the trial solution is a hybridization of an approximation to a neural network that would not depend on the boundary or initial conditions, with the other that satisfies the boundary or initial values. This approach is applicable for single ODE to systems of coupled ODEs as well as PDEs. In [21], the author presented an innovative approach for solving PDEs with complex boundary geometries using ANN. It utilizes a combination of both Radial Basis Function (RBF) networks and Multilayer Perceptron (MLP) to accurately satisfy the boundary conditions.

In [22], the author introduces a new approach for solving DEs using unsupervised Feed Forward Neural Networks (FFNs). The method has been successfully solved to a chemical engineering such as non-steady fixed bed non-catalytic solid-gas reactor problem. In [23] the authors introduce a new method that integrates optimization techniques and neural networks to address high-order ODEs. By combining FFNs with the Nelder-Mead optimization technique, this method achieves improved approximations to ODE solutions without the need for predefined discretized points. The method presented in [24] solves an extensive range of ODEs and PDEs using neural networks constructed through linguistic based evolution of the variables. The method combines evolutionary algorithms with local optimization to create and enhance the defined trial solutions. Whereas, in [25], authors investigate an evolutionary computational technique that employs neural networks as universal approximators to solve initial value problems in ODEs. It offers the advantage of delivering approximate solutions across a continuous finite time domain, potentially reducing time and space complexity. In [26], The researcher introduced the length factor ANN method with the aim of tackling complex PDEs. This method was subsequently expanded to handle coupled systems of PDEs like the Navier-Stokes equations. This approach offers advantages over traditional continuous approximate solutions while automatically satisfying boundary conditions.

In [27], authors introduced a Regression-Based Algorithm (RBA) for solving ODEs with initial conditions using a FFN approach. This technique combines arbitrary (Random) and regression-based initial weights for the nodes within the hidden layer. The number of hidden nodes is determined by the polynomial degree utilized during the regression fitting process. This method is straightforward, computationally efficient, and shows promise for a variety of applications. In [28], the authors introduce a method to solve ODEs using a Regression-Based Neural Network (RBNN) approach. This method takes a trial solution combining a fixed term satisfying initial or boundary conditions and an RBNN-based term with adjustable parameters. It demonstrates higher accuracy and faster training compared to traditional ANN methods. In [29] the researchers mentioned that there is a limited literature on the selection of the loss function and network hyperparameters and their impact on the quality of numerical solutions. Consequently, they conducted a series of numerical experiments to evaluate the effectiveness of various loss functions in solving DEs. Authors [30] have introduced a novel method to address second-order nonlinear ODEs of Lane-Emden type, which are

recognized as singular initial value problems. This approach employs Chebyshev Neural Network (ChNN) models, utilizing a single-layer network. The network parameters are adjusted to reduce the loss function without employing optimization methods, through the utilization of an unsupervised error backpropagation method. This approach is applied to both homogeneous and non-homogeneous Lane-Emden equations, which hold significance in astrophysics and mathematical physics. Then the authors [31] introduced a novel approach employing a single-layer ChNN with regression-based weights for ODEs. In this ChNN model, the hidden layer is substituted with a functional expansion block utilizing Chebyshev polynomials. In [32], the authors introduce a stochastic approach for solving nonlinear fractional order Riccati DEs employing a combination of FFN and heuristic computational algorithms based on Particle Swarm Optimization (PSO). Whereas in [33], researchers discussed a new approach for solving initial value problems of fractional order DEs. This approach utilizes neural networks constructed from adjustable cosine basis functions. This method is applied for both single and coupled fractional DEs, giving numerical solutions through iterative training of the networks. In addition, many researchers [34–48] gave their contributions in solving fractional DEs, and PDEs by using neural networks.

The above literature reveals various approaches that can be used to investigate DEs through ANN. But, further studies on this silver lining area witness many developments in connection with various optimization techniques that provide computational efficient solutions. The common idea of implementing optimization in ANN is to represent the loss function in terms of objective function that need to be minimized. Researchers have significantly contributed potential approaches for the same. Few of the research works are discussed in [49–54]. Here primarily authors have used gradient approaches viz. steepest descent and Newton method to the minimize the loss function with ANN. The main contribution of optimization techniques lies in the training process for ANN as a powerful first-order and second-order algorithm. The first-order algorithm [55] is the Error Back Propagation (EBP) algorithm, which was introduced in 1986. However, EBP has low training efficiency and is iteratively most expensive in real applications. Second-order algorithms like the Levenberg-Marquardt (LM) algorithm and the Gauss-Newton algorithm [45], [46] offer advantages in terms of execution time and computational efficiency.

The LM algorithm stands out as one of the most effective training algorithms for ANN. It integrates elements of both the Steepest Descent and Newton's algorithms, particularly effective for small and medium-sized patterns. In this regard, many researchers have been modified the LM algorithm for better results and accuracy. The authors [56] introduced a new method for optimizing the LM algorithm. This technique divides a Jacobian matrix and distributes it to all neurons. By using this method, the computational complexity of the traditional variants is reduced and parallel computations are made possible. As a result, it reduces the huge computational complexity that arises when employing large networks and training sets. The author [57] introduced a technique to enhance the stability and boundedness of the errors and weights during the learning process of ANN. It improves the performance of neural network models across various applications. The authors [58] introduced an improved computation LM method for training neural networks. By eliminating the need to store and multiply large Jacobian matrices, it reduces memory requirements and computation time. As mentioned earlier, a detail study of ANN with mathematical modelling and basic architecture of the same is presented here.

The rest of the paper is structured as follows. Section 2 discussed the fundamentals of the ANN and their structures, including various activation functions, Section 3 analyses various research articles for solving DEs with neural networks, Section 4 discusses the various computational modifications of LM algorithm and finally, section 5 discusses the conclusion of it.

2 | Artificial Neural Network

ANN is computationally inspired by the function and structure of human neural networks in the brain. It is a subfield of Machine Learning (ML) models to recognize patterns and relationships within data and ML is a subpart of Artificial Intelligence (AI), as shown in *Fig. 1* [59]. The very essence of ANN lies in the use of AI

and automated systems with learning. To understand it one need to know the motivation behind the ANN and related benchmark works. As such, in the following subsections the insights of ANN are discussed.

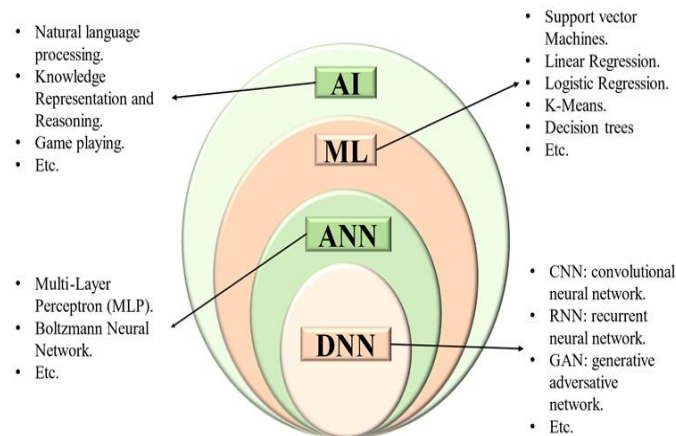


Fig. 1. The orientation of ANN in AI. AI: Artificial Intelligence, ML: Machine Learning, ANN: Artificial Neural Network, DNN: Deep Learning Neural Network.

ANNs began in the mid-20th century. Then significant developments have occurred over several decades in ANN. Initially, the concept of artificial neurons was introduced by Warren McCulloch & Walter Pitts in 1943 [14]. They proposed a neuron that is a simplified model of the biological neuron. As a result, it formed the basis for later developments in neural networks. The perceptron concept was developed by Frank Rosenblatt in the year 1957 [60]. The perceptron, featuring a single-layer neural network, showcases its capability to learn simple patterns and perform decision-making based on input data. It is capable of performing binary classification tasks. In this context, in 1957 Frank Rosenblatt proved the perceptron convergence theorem [61]. If the data is linearly separable it converges to a solution. However, it has limitation to handle nonlinearly separable data. Further, in timeline, the period from 1970s-1980s called neural network winter because the researchers were not very interested in neural networks to solve complex problems.

In 1986, D. Rumelhart, G. Hinton, and R. Williams reintroduced neural networks and introduced backpropagation algorithm [55]. It effectively calculates the gradients of the error function with respect to the network parameters. The development of backpropagation led to increased interest in neural networks for many researchers. Hence, various architectures, activation functions, and training methods were explored, leading to improved performance and scalability. As a result, the period from 1980s-1990s is known as the rebirth of neural networks [55, 62, 63]. In the 1990s, other ML techniques like kernel methods and Support Vector Machines (SVMs) also emerged as powerful alternatives to classification and regression tasks. From the 2000s to the present, advances in big data, powerful GPUs, and computational resources led to new interest in deep learning and neural networks. Researchers have developed deep neural networks with many layers. These advancements have propelled progress in natural language processing, computer vision, speech recognition, and various other fields. Recent years have developed advanced ANN architectures viz. recurrent neural network (RNN) for sequential data, Convolutional Neural Networks (CNN) for image processing, etc. The significant transformation of ANN is shown in Fig. 2.

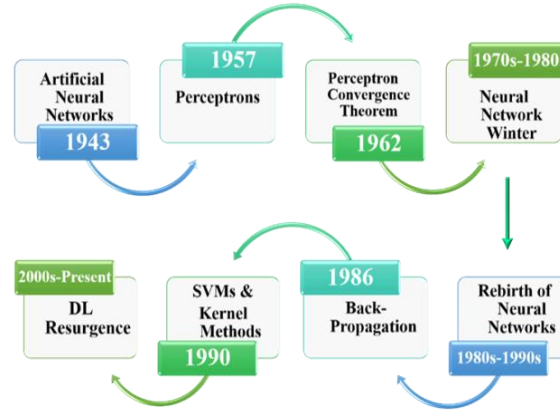


Fig. 2. Historical development of artificial neural network.

2.1| Structure of Artificial Neural Network

ANNs consist of interconnected neurons which is also known as nodes or units. It is organised by layers. The primary layers of ANN viz. input, hidden, and output layers. The input layer is the initial one that receives data and transmits it to subsequent layers within the network. The hidden layers lie between the output and input layers. Here most of the computation occurred in the ANNs. The number of hidden layers and their neurons in each hidden layer depend on the complexity of the given problem and the architecture of the ANN. The processing of inputs is performed in this layer. The final layer is the output layer. It produces final output of the ANN. The number of neurons depends on the nature of the problem. Different architectures are included in ANN like FFNs, RNNs, Generative Models (GM), CNN, Modular Neural Networks (MNNs), Hybrid Neural Networks (HNNs), autoencoders, and etc. [64–70]. Fig. 3 shows the different types of ANN.

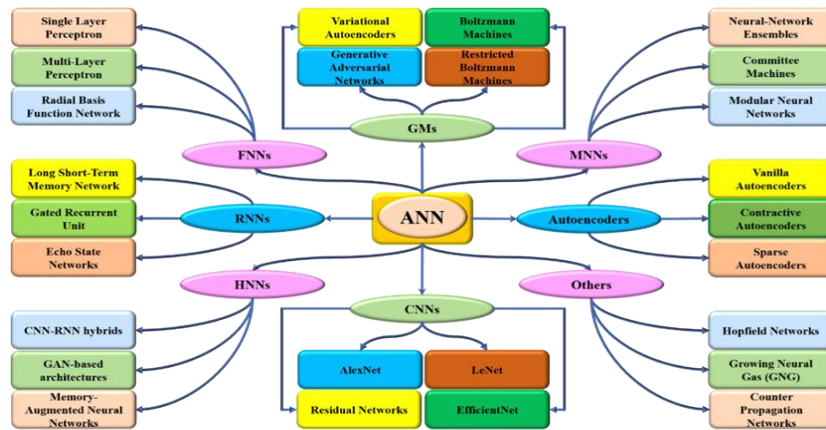


Fig. 3. Different types of artificial neural network.

The study shows ANN is best suitable for optimization problems. One of the types of ANN that is FFNs is well-known in researchers. The FFNs have consisted of various architectures such as MLP, Fully Connected Multilayer Perceptron (FCMLP), and Fully Connected Cascade networks (FCCs) as shown in Figs. 4-6 respectively. MLP is the most common architecture. Each layer of MLP is connected to the previous layer. FCMLP is similar to MLP but each layer of this network is interconnected with all preceding layers as well as the network inputs with the same neurons. It contains a greater number of weights. FCCs follow the same pattern as the FCMLP network but it has only a single neuron in each layer except the input network [56].

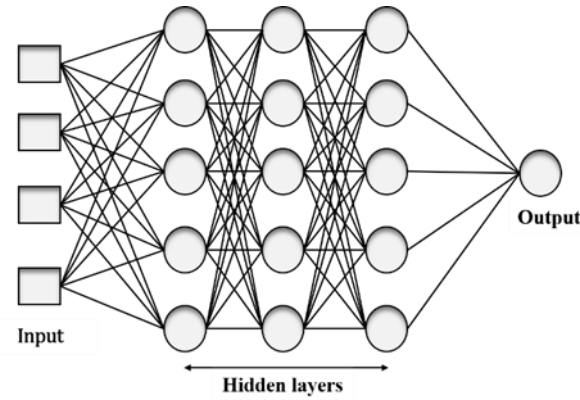


Fig. 4. Multilayer perceptron.

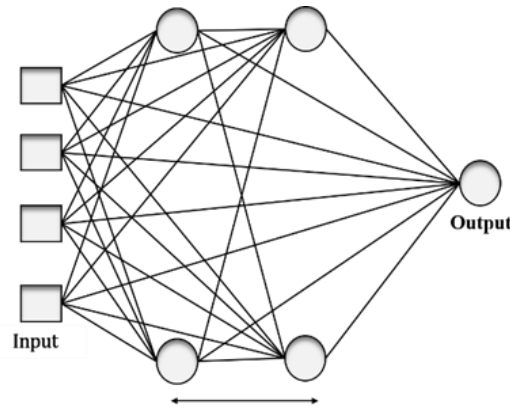


Fig. 5. Fully connected multilayer perceptron.

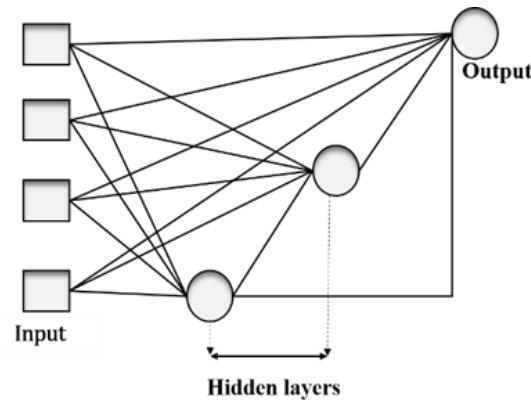


Fig. 6. Fully connected cascade networks.

2.2 | Mathematical Formation of Artificial Neural Network

In the FFN, the mathematical notation represents the information from the input through multiple hidden layers to generate the output. Each layer, except the input layer, which directly receives the data, involves a matrix multiplication and addition operation. The input data, denoted as $\mathcal{X} = x_j = \{x_1, x_2, x_3, \dots, x_n\}$, is passed through successive layers j with associated weight matrices $\mathcal{W}_j$ and bias vectors $\mathcal{b}_j = \{b_1, b_2, \dots, b_n\}$. The computations at each layer perform a linear transformation which involves calculating a weighted sum of inputs $z_j = (x_j * \mathcal{W}_j) + \mathcal{b}_j$ and applying an activation function $g(z_j)$ to generate the output. This process persists through the hidden layers until it reaches the output layer, which is producing the final result. During training, these weights and biases are iteratively modified using optimization techniques to minimize the difference between predicted and actual outputs. It can be expressed in Eq. (1).

$$e = y - d, \quad (1)$$

Where e is the loss function, y is the actual output and d is the predicted output of ANN.

2.3 | Activation Functions

In ANNs, the activation function play an important role. It introduces non-linear mappings between the initial and last layer of each neuron. This non-linearity enables ANNs to learn and represent complex relationships within the data. Activation functions also help in gradient-based optimization during the training of neural networks. They introduce derivatives that are used to update parameters through backpropagation, allowing the network to learn and modify its parameters to minimize the errors. In this regard, various activation functions used in ANNs based on the specific requirements and nature of the problem [71], [72] as shown in Fig. 7.

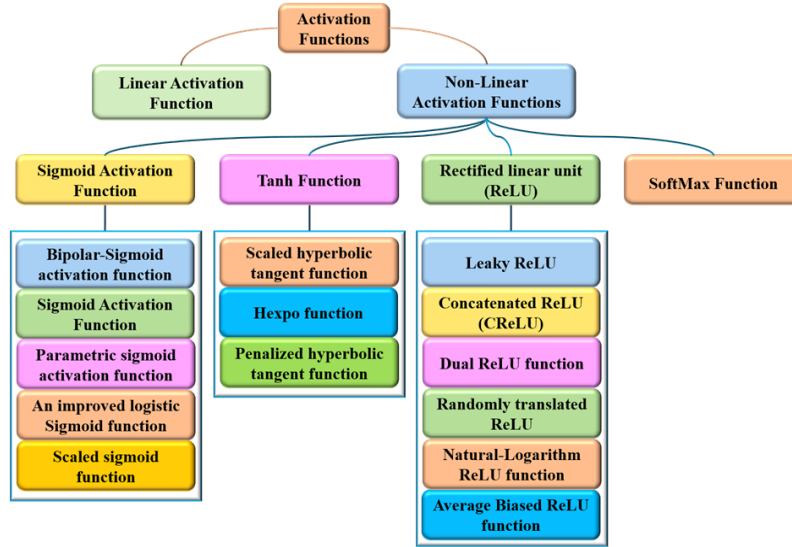


Fig. 7. Various activation functions.

We can see a few activation functions that are commonly used in the ANN as follows:

2.3.1 | Linear activation function

The linear activation function is the simplest activation function that is a straight-line. It is the direct proportional to the input data lies between $-\infty$ to ∞ . Mathematically, the linear activation function can be defined as.

$$f(z) = z. \quad (2)$$

The main important characteristic of the linear activation function is that it maintains linearity. This means that the output of the function grows linearly with the input. It is commonly used in the last layer of the regression models.

2.3.2 | Sigmoid activation function

The sigmoid function is also called the logistic function [73]. Sigmoid activation function is a mathematical formula that takes an input, which is often the weighted sum of inputs and it reduces into a range between 0 and 1. This is helpful in many ML models, particularly those involving binary classification and probability prediction. The formula for the sigmoid function can be defined in Eq. (3).

$$f(z) = \frac{1}{1+e^{-z}}. \quad (3)$$

Where e is Euler's number and the base of the natural logarithm, z is the input of the given function. It can be shown in Fig. 8.

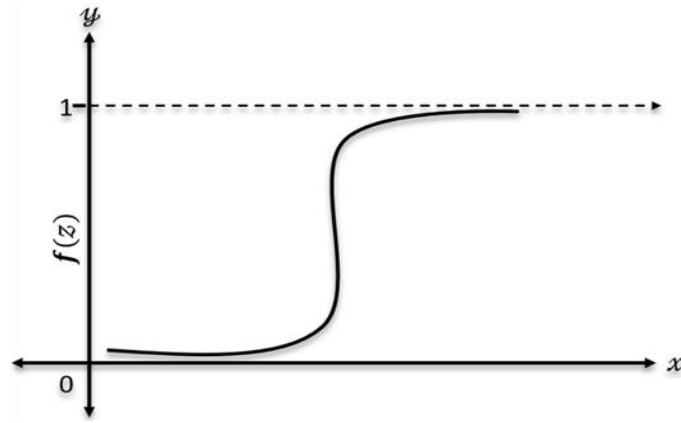


Fig. 8. Sigmoid activation function $f(z)$.

The primary benefit of the sigmoid function lies in its ability to normalize values within neural networks while ensuring boundedness. However, it has some drawbacks like the gradient vanishing issue, being computationally costly, and not centering its output on zero, making optimization harder, particularly in deep neural networks [74], [75]. A few improvements of sigmoid function is shown in the *Table 1*.

Table 1. Modifications in sigmoid function.

Name	Mathematical Formula	Range
Bipolar-sigmoid activation function [76], [77]	$F(z) = \frac{1-e^{-z}}{1+e^{-z}}$	$[-1,1]$
Elliott activation function [78], [79]	$f(z) = \frac{0.5z}{1+ z } + 0.5$	$[0,1]$
Parametric sigmoid activation function [80]	$f(z) = \left(\frac{1}{1+e^{-z}}\right)^m$	$m \in (0, \infty)$
An improved logistic sigmoid function [81]	$f(z) = \begin{cases} \alpha(z-a) + \text{Sigmoid}(a), & z \geq a, \\ \text{Sigmoid}(z), & -a < z < a, \\ \alpha(z+a) + \text{Sigmoid}(a), & z \leq -a, \end{cases}$	$[-\infty, \infty]$
Scaled sigmoid function [82]	$f(z) = \frac{4}{1+e^{-z}} - 2$	$[-2,2]$

2.3.3 | Tanh function (Hyperbolic tangent)

The tanh function is widely employed in ANNs. It resembles the sigmoid function but compresses the range between -1 and 1. Mathematically, it can be defined as follows:

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (4)$$

Tanh activation function is zero centric function which means that when one gives zero input, it gives zero output, and it is a smooth gradient converging function. However, it faces a similar problem of sigmoid function such as for extremely big or small numbers, it does not change much. In comparison to the sigmoid, the tanh function performs better, as shown in *Fig. 9*.

A few improvements of the tanh activation function are shown in the *Table 2*.

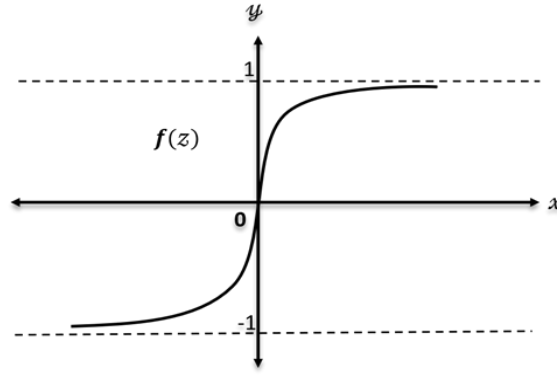
Fig. 9. Tanh Activation function $f(z)$.

Table 2. Modifications in the tanh activation function.

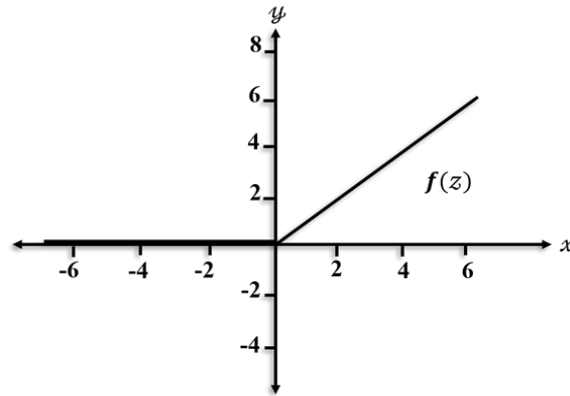
Name	Mathematical Formula	Range
Scaled hyperbolic tangent function [62]	$f(z) = \mathcal{A} \tanh(\mathcal{S}z)$, $\mathcal{A}$ = the amplitude of the function, $\mathcal{S}$ = the slope at the origin.	$[-\mathcal{A}, \mathcal{A}]$.
Hexpo function [83]	$f(z) = \begin{cases} -ae^{\left(-\frac{z}{b}\right)} - 1, & z \geq 0, \\ ce^{\left(-\frac{z}{d}\right)} - 1, & z < 0, \end{cases}$ where a, b, c , and d are the constants.	$[-c, a]$.
Penalized hyperbolic tangent function [84]	$f(z) = \begin{cases} \tanh z, & z \geq 0, \\ a \tanh z, & z < 0, \end{cases}$ where a is the constant.	$[-a, 1]$.

2.3.4 | Rectified linear unit activation function

The Rectified Linear Unit (ReLU) function is the commonly used neural network model. It can be defined as follows:

$$f(z) = \begin{cases} 0, & z \leq 0 \\ z, & z > 0 \end{cases} \quad (5)$$

The derivative of the ReLU function is zero when $z < 0$, unity when $z > 0$, and at $z = 0$, the derivative is undefined. The ReLU function ranges from $[0, \infty)$. Unlike tanh and sigmoid, the implementation of ReLU is very easy. Graphically it has been shown in Fig. 10.

Fig. 10. Rectified linear unit activation function $f(z)$.

The derivative of the ReLU function is not always fixed at 0 or 1. Instead, it depends on the input value. For negative inputs, the derivative is 0, and for positive inputs, it is 1. However, the function is not differentiable when the input is 0.

The ReLU function also faces certain limitations, with one of the primary concerns being the "dying ReLU" problem [85], [86] which means that neurons get stuck at a certain point. It causes the output to be zero for all inputs. Therefore, it makes the neurons "dead" and does not contribute to the learning process. Some improvements of the ReLU activation function are illustrated in the *Table 3*.

Table 3. Modifications in the rectified linear unit activation function.

Name	Mathematical Formula	Range
Leaky ReLU [85]	$f(z) = \begin{cases} \alpha z, & z \leq 0 \\ z, & z > 0 \end{cases}$, where α is a small positive constant.	$(-\infty, \infty)$.
Concatenated ReLU (CReLU) function [87]	$f(z) = [\text{ReLU}(z), \text{ReLU}(-z)]$.	$[0, \infty)$.
Dual ReLU function [88]	$f(z) = \max(0, a) - \max(0, b)$, where a and b are the inputs	$(-\infty, \infty)$.
Randomly translated ReLU function [89]	$f(z) = \begin{cases} z + \alpha, & z + \alpha > 0 \\ 0, & z + \alpha \leq 0 \end{cases}$, where α is a random sample.	$[0, \infty)$.
Natural-Logarithm ReLU function [90]	$f(z) = \log(\beta \max(0, z) + 1)$, where the β is a constant.	$[0, \infty]$.
Average Biased ReLU function [91]	$f(z) = \begin{cases} z - \beta, & z - \beta \geq 0 \\ 0, & \text{Otherwise} \end{cases}$, where the β is a constant.	$[0, \infty)$.

2.3.4 | Softmax activation function

The SoftMax activation function is also known as the normalized exponential function or soft argmax. It is often used in the output layer of the ANNs. It measures or converts the numbers or logits into probability distribution of possible outcomes. The SoftMax's output is a vector with the probability distribution of each possible outcome or class. Generally, it can be defined as follows:

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}, \quad (6)$$

where $i = 1, 2, \dots, n$.

3 | Comprehensive Survey of Artificial Neural Network for Differential Equations

ANNs can provide a promising solution for DEs by utilizing its capacity for function approximation and data-driven learning. ANNs can directly satisfy complex functions related to DEs, unlike traditional numerical methods. It can produce better approximate solutions for unseen inputs by training ANNs on datasets containing input-output of DE solutions. ANNs are a universal function approximator. Thus, they can effectively capture the complexity of DE solutions. Moreover, its adaptive nature allows them to adjust internal variables during training. Hybrid approaches provide additional opportunities for improving accuracy and efficiency in solving DEs. The various hybrid approaches are summarized in the *Tables 4-7*.

Table 4. Important literatures on ordinary differential equation based on artificial neural network.

Year	Author(s)	Model	Remark
1990	Hyuk Lee, In Seok Kang [92]	Neural minimization algorithms-continuous & discrete case	Using nonlinear thresholding and other neural network concepts, this technique can solve a variety of DEs which was achieved by using electronic processors for low-order DEs, but optical implementation is preferred for higher-order DEs due to increased complexity.

Table 4. Continued.

Year	Author(s)	Model	Remark
1994	Meade Jr and Fernandez [93]	FFNs-piece wise linear activation function	It can directly construct FFNs for approximating nonlinear ODEs without traditional training. The method utilizes a piecewise linear activation function, leading to a linear storage. It can monotonically decrease approximation errors by increasing more hidden layer neurons.
1994	Meade Jr and Fernandez [94]	FFNs-the Hard Limit Transfer Function (HLTF)	Authors constructed FFNs for approximating linear ODEs directly and noniteratively. This method utilizes a HLTF, leading to a linear storage and processing time, and quadratically decreasing errors with a greater number of hidden layer neurons.
1997	Lagaris et al. [95]	MLP	This article presents a novel approach using ANNs to solve eigenvalue problems of ODEs, PDEs, and integro DEs, primarily within the domain of Quantum Mechanics. By employing MLP for parametrization, collocation methods for error formulation, and optimization procedures, this method demonstrates high accuracy, efficiency, and applicability across various dimensionalities and complexities.
1998	Lagaris et al. [96]	MLP & RBF	This article presents a method for solving PDEs with Dirichlet boundary conditions on complex and irregular boundaries. This method employs a combination of MLP and RBF. The approach demonstrates the ability to produce accurate and differentiable solutions that satisfy boundary conditions at selected points. It presents a promising method for solving PDE problems with complex boundaries and has potential applications in various scientific fields.
2002	Valasoulis et al. [40]	MLP	Authors introduces a method using ANNs for solving ODEs and PDEs. This approach involves representing the trail solution as a combination of two parts that is one part satisfies initial/boundary conditions, while the other part utilizes a FFNs whose weights are adjusted to solve the equation. The use of neural networks offers advantages such as a differentiable and easily usable solution format, excellent generalization properties, and compact models requiring minimal memory. The focus of the work is on implementing this method on two digital signal processors, enabling real-time solutions to challenging differential equation problems in various engineering applications.
2003	Parisi et al. [22]	FFNs and sigmoid activation function.	In this article, the authors introduce a new approach for solving DEs using unsupervised FFNs. The technique has effectively been employed to address a non-steady fixed bed non-catalytic solid-gas reactor problem in chemical engineering. It offers advantages such as efficient evaluation at any point in the domain and compact mathematical expressions, making it suitable for optimization and control tasks. The comparison with the traditional method like Runge-Kutta method, shows equivalent solutions, but neural network produces better efficiency in terms of time and memory.
2006	Malek and Beidokhti [23]	Feed forward neural networks-Sigmoid activation function	It combines Nelder-Mead optimization method and neural networks for solving high ODEs without predefined discretized points. Also, this method is used to solve for complex mathematical problems.

Table 4. Continued.

Year	Author(s)	Model	Remark
2008	Fojdl et al. [45]	FFN and total least mean square error (TLMSE)	This study investigates the trade-off between standard numerical integration methods for solving ODEs and neural network-based approximations. It addresses the need for faster computational methods, especially in applications like real-time physics engines and weather forecasting. This research explores the error rates and resource requirements of both approaches, considering factors such as computation time and memory usage. The findings suggest that the choice between ODEs and neural networks depends on available resources and the specific problem at hand.
2009	Junaid et al. [26]	FFNs	This paper presents an evolutionary computing technique that employs neural networks as universal approximators to solve initial value problems in ODEs. The adaptive parameters within the neural networks are fine-tuned through optimization utilizing a Genetic Algorithm (GA). The approach offers a continuous grid of time for solution, in contrast to discrete methods used in classical numerical techniques.
2011	Otadi et al. [97]	Multi-layer FFNs with Sigmoid activation function	It combines neural networks and soft computing to solve quadratic Riccati DEs, especially within the initial neighbourhood. The advantage of this method lies in its ability to offer accurate and generalized solutions compared to existing numerical techniques.
2011	Ibraheem and Khalaf [98]	Combination of FFN & shooting method.	This method combines Neural Networks, integration, and interpolation techniques to tackle boundary value problems (BVPs) in ODEs, primarily emphasizing the principle of backpropagation.
2013	Mall and Chakraverty [27]	FFN, backpropagation & RBA	It introduced a RBA for solving ODEs with initial conditions using a FFN approach. It utilizes both arbitrary (random) and regression-based initial weights for the neural network's neurons in the hidden layer. The number of neurons is determined by the degree of the polynomial in the regression fitting.
2013	Tawfiq and Hussein [99]	FFN back propagation-various learning algorithms	This method is employed to address second-order singular boundary value problems within ODEs using FFN backpropagation, employing a range of learning algorithms including quasi-Newton, LM, and Bayesian Regulation. It has been observed that LM gives better results than others.
2014	Mall and Chakraverty [30]	ChNN, FFN backpropagation	A ChNN-based model was utilized to solve both homogeneous and non-homogeneous Lane-Emden equations. In this context, an unsupervised error backpropagation algorithm is employed to minimize the loss function and update the network weights without using optimization methods
2014	Chakraverty and Mall [28]	FFNs-unsupervised backpropagation.	The authors propose a technique for solving ODEs through a RBN methodology. The trial solution consists of two components: the first part boundary /initial conditions without adjustable weights and biases, while the second part involves an RBN-based component with adjustable parameters

Table 4. Continued.

Year	Author(s)	Model	Remark
2016	Mall and Chakraverty [100]	Functional Link ANN (FLANN), FFN backpropagation	The authors presented an innovative approach to address initial and boundary value problems associated with fractional DEs, employing a single-layer Legendre Neural Network (LeNN) model. This proposed method employs Legendre polynomial-based FLANNs to handle nonlinear singular boundary and initial value problems as well as systems of coupled ODEs. Through the expansion of the input pattern with Legendre polynomials, the hidden layer is eliminated. This method uses the error backpropagation algorithm for updating parameters.
2020	Chakraverty and Mall [31]	ChANN, FFN-Unsupervised backpropagation algorithm.	The authors introduced a novel approach using a regression-based weights and single-layer ChANN to solve ODEs. The ChANN model substitutes the hidden layer with a functional expansion block utilizing Chebyshev polynomials.
2021	Okereke et al. [101]	FFN through MLP	The author concentrates on employing ANNs to tackle the initial conditions of first- and second-order linear constant coefficient ODEs. A feed-forward multilayer perceptron is utilized; however, the standard backpropagation process for weight updates is avoided.

Given that PDEs form the foundation of numerous real-world problems, their thorough scientific examination amidst varying datasets is imperative. Consequently, researchers have documented diverse and intriguing methods and findings employing ANNs. *Table 5* presents a comprehensive discussion of noteworthy contributions in this field.

Table 5. Some of the important literatures on partial differential equation based on artificial neural network.

Year	Author and Reference	Model	Remark
1998	Lagaris et al. [96]	MLP & RBF	The article presents a method for solving PDEs with Dirichlet boundary conditions on complex and irregular boundaries using a combination of MLP and RBF. This approach is shown to yield accurate and differentiable solutions that satisfy boundary conditions at selected points. It presents a promising method for addressing a wide range of PDE problems characterized by complex boundaries, holding potential for applications across various scientific domains.
2000	Isaac et al. [21]	ANNs: MLP and RBF	This article introduces a novel method for tackling PDEs featuring complex boundary geometries, encompassing Dirichlet and Neumann boundary conditions, leveraging ANNs. It utilizes a combination of MLP and RBF networks to accurately satisfy the boundary conditions. This approach has significant advantages, including differentiability, superior interpolation capabilities, a small number of parameters, and the potential for efficient implementation on parallel computers or specialized hardware like neuro-processors. Hence, this method presents a valuable tool for solving PDE with complex geometries.
2001	Aarts and Van Der Veer [102]	FFNs	This article presents a technique for addressing PDEs along with their corresponding boundary or initial conditions, employing neural networks. This method utilizes the approximation capabilities of single-hidden-layer FFNs featuring a linear output layer, enabling the approximation of arbitrary functions and their derivatives.

Table 5. Continued.

Year	Author and Reference	Model	Remark
2002	Valasoulis et al. [40]	MLP	The authors introduce a method for solving ODEs and PDEs through ANNs. This approach involves representing the solution of two terms: first term satisfies initial/boundary conditions, while the other term utilizes a FFNs whose weights are adjusted to solve the equation. It offers some advantages such as a differentiability and easily usable solution format, excellent generalization properties, and compact models requiring minimal memory.
2009	Shekari Beidokhti and Malek, [103]	FFNs	It introduces a hybrid approach for tackling time-dependent PDEs alongside their initial and boundary conditions employing ANNs. This method combines ANNs, minimization methods, and collocation methods to determine an accurate approximate solution in closed analytical form. Unlike some previous approaches that use neural networks to approximate solutions, this method focuses on novel trial functions to simplify the problem and minimize the need for domain discretization. It demonstrates the ability to solve high-order two-dimensional PDEs and offers advantages such as generality, fast evaluation, and applicability to initial and boundary value problems.
2015	Rudd and Ferrari [104]	Constrained Integration (CINT) method-ANN and Galerkin method	The authors present a novel CINT method for solving boundary and initial value PDEs through the utilization of ANNs and the Galerkin method. The CINT method offers advantages such as applicability to irregular domains, high convergence rates, and reduced computational time when compared to traditional FEM. This method is especially effective for hyperbolic PDEs with Dirichlet and Neumann boundary conditions. However, it may not be suitable for problems with sharp gradients or discontinuities.
2018	Berg and Nystrom [105]	Deep feed forward ANNs	The authors propose a method employing deep feedforward ANNs to approximate solutions for complex PDEs within intricate geometries. This method adapts the backpropagation algorithm to compute partial derivatives, facilitating the approximation of the differential operator. This method is applicable to high-dimensional problems, complex domains and benefits from using DNNs with increased hidden layers to achieve faster convergence.
2019	C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk, [106]	ANNs	It gives an adaptive collocation strategy combined with ANNs to solve PDEs. The primary aim is to enhance the robustness of neural network approximations for PDE solutions, especially in cases involving non-smooth regions. The method is mesh-free and offers promising results for solving 2nd order boundary value problems like Helmholtz equations and Poisson's. However, further research is needed to explore convergence properties, parameter tuning, and the applicability to energy minimization formulations of DEs.

From the extensive study it has also been seen that many ANN based approaches that can deal both ODEs and PDEs. *Table 6* highlights a selection of pertinent studies focusing on ODEs and PDEs utilizing ANN methodologies.

Table 6. Some of the important literatures on ordinary differential equation and partial differential equation based on artificial neural network.

Year	Author and Reference	Model	Remark
1998	Lagaris et al. [20]	FFNs and sigmoid activation function	This article presents a technique for addressing initial and boundary value problems ranging from single ODEs to systems of coupled ODEs and PDE problems utilizing ANNs. The trial solution is expressed as the two parts. The first part satisfies the initial and boundary conditions while the second term satisfies the FFNs with adjustable parameters. The use of neural networks gives this method an advantage. The solution provided by ANNs is in a differentiable, closed analytic form, low memory space.
2003	Parisi et al. [22]	FFNs and sigmoid activation function.	In this article, the author introduces a novel approach for solving DEs utilizing unsupervised FFNs. The method is applied to a non-steady fixed bed non-catalytic solid-gas reactor problem in chemical engineering. This method offers advantages such as efficient evaluation at any point in the domain and compact mathematical expressions, making it suitable for optimization and control tasks. The neural network gives efficiency in terms of memory and time.
2019	Liu et al. [107]	ANN and automatic differentiation (AD).	This paper explores the use of ANNs and AD techniques for approximating and solving ODEs and PDEs. This approach is applied on various DEs, demonstrating high accuracy and universal applicability for ODEs and positive results for certain PDEs. This paper discusses the potential of NNs as universal solvers for ODEs and PDEs. This approach offers advantages such as handling complex geometries and providing differential information. Challenges such as handling boundary conditions and computational efficiency are acknowledged, with future work focused on improving accuracy and addressing these challenges. The potential for using NNs in fluid experiment data analysis is also discussed.
2021	Panghal and Kumar [108]	FFNs	The authors introduce an efficient approach using the extreme ML algorithm to solve ODE and PDEs with NNs. It eliminates the need for time-consuming optimization and trial function construction. The method provides analytic solutions, generalizes well with few data points, and handles various boundary conditions.

Besides ODEs and PDEs, it is observed that fractional order DEs often arise due to the inherent nature of certain problems. Therefore, *Table 7* presents significant research endeavours centred around fractional-order DEs employing ANN.

Table 7. Continued.

Year	Author and Reference	Model	Remark
2011	Raja et al. [32]	FFNs with heuristic computational algorithm.	In this research paper, the author introduces a stochastic technique for solving nonlinear fractional order Riccati DEs using a combination of FFNs and heuristic computational algorithms based on PSO and simulated annealing. This method is shown to effectively handle both integer and fractional order equations, offering improved optimization compared to existing stochastic solvers like Simulating Annealing (SA), GA, GA hybridized with SA (GA-SA), PSO, and PSO hybridized with SA (PSO-SA). This method proves reliable and applicable to complex nonlinear equations, showcasing its potential for solving challenging problems.
2015	Raja et al. [34]	ANNs through FFNs and Sequential Quadratic Programming (SQP)	The authors introduce a method using ANNs and SQP to solve nonlinear fractional order Riccati DEs. This method demonstrates on various initial value problems, showing its accuracy when compared to exact solutions and other numerical methods based on Adams–Bashforth–Moulton method with a higher degree of accuracy. This method is showcased across a range of initial value problems, illustrating its accuracy through comparison with exact solutions and other numerical methods employing the Adams–Bashforth–Moulton technique, offering higher degree precision. This method shows good results in terms of accuracy and computational efficiency
2015	Haidong Qu and Xuan Liu, [33]	FFNs based on cosine function.	This paper presents a new approach using ANNs constructed from adjustable cosine basis functions for solving initial value problems of fractional order DEs. This method is applied for both single and coupled fractional DEs, giving numerical solutions through iterative training of the networks. This technique proves useful for complex problems with applications in various interdisciplinary fields. This method shows better accuracy and convergence properties through computer graphics. This approach offers a powerful tool for fractional order DEs and able to fractional order PDEs as well.
2017	Jafarian et al. [35]	FFNs with backpropagation learning algorithm	This research paper introduces an approach for solving fractional-order initial value problems efficiently with the combination of ANNs and the power series method. This method transforms the DE into a minimization problem and uses a three-layer FFNs to find the coefficients of the power series. This technique demonstrates promise in approximating solutions for fractional order DEs. However, this methods efficiency depends on careful parameter selection. The approach is compared to existing methods and shows potential for broader applications in solving complex fractional-type problems.

Year	Author and Reference	Model	Remark
2017	M. Pakdaman, A. Ahmadian, S. Effati, S. Salahshour, D. Baleanu, [36]	ANNs for function approximation	The authors present a novel approach that utilizes ANNs to approximate solutions of Fractional DEs (FDEs). This method combines ANNs with optimization techniques to adjust network weights for accurate FDE solutions. The method is compared to existing numerical techniques, demonstrating its efficiency with high accuracy, fast convergence, and low memory usage.
2018	Jafarian et al. [109]	Three-layered FFN with cost function, supervised backpropagation algorithm.	This manuscript presents an approach using ANNs to solve linear Fractional Order Odes (FODE). This method uses a three-layer FFN with sigmoid activation functions and a supervised back-propagation learning algorithm. It converts the FODE into a minimization optimization problem and uses the least mean square error function to train the network parameters. Authors provide numerical examples and comparisons with standard methods, demonstrating the effectiveness of the ANN approach in solving FODEs.
2023	Admon et al. [110]	FFN with Vectorized Algorithm (FNNVA)	The authors discuss the vectorized algorithm to efficiently implement FFNs for solving fractional DEs and focuses on exploring the effectiveness and efficiency of optimization methods such as gradient descent, momentum method and adaptive moment estimation in training the FFNs for solving fractional DEs. It discusses the advantages of using FFNs for numerical approximation of fractional DEs and compares the performance of FFNs through various number of hidden layers, learning rates and number of neurons.

4 | Levenberg-Marquardt Algorithm with Neural Networks

The LM algorithm is a nonlinear optimization method and utilizing an iterative process that takes advantage of both Newton's method and gradient descent method. Its primary objective is to minimize the error function inherent in neural network predictions. It has a stable and very fast convergence. It is suitable for the training of small-sized problems in ANNs. It is designed to work with a loss function, which is the Sum of Squared Errors (SSE) as the objective function. The following Eq. (7) calculates the objective function for the training process.

$$\mathcal{E}(x, \boldsymbol{p}) = \frac{1}{2} \sum_{n=1}^{\mathcal{N}} \sum_{m=1}^{\mathcal{M}} e_{(n,m)}^2, \quad (7)$$

where x is the input vector, $\boldsymbol{p}$ is the weight vector and $e_{(n,m)}$ is the error function with $\mathcal{N}$ patterns and $\mathcal{M}$ outputs it is expressed in Eq. (8).

$$e_{(n,m)} = \boldsymbol{\psi}_{(n,m)} - \boldsymbol{d}_{(n,m)}, \quad (8)$$

where $\boldsymbol{\psi}_{(n,m)}$ is the actual output vector, and $\boldsymbol{d}_{(n,m)}$ is the desired output vector.

The weight update procedure in the LM algorithm is calculated [111] as follows:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \Delta \boldsymbol{w}(x, \boldsymbol{p}). \quad (9)$$

In Eq. (9), $\Delta \boldsymbol{w}(x, \boldsymbol{p})$ is computed as.

$$\Delta \boldsymbol{w}(x, \boldsymbol{p}) = -[\nabla^2 \mathcal{E}(x, \boldsymbol{p})]^{-1} \nabla \mathcal{E}(x, \boldsymbol{p}), \quad (10)$$

where $\nabla \mathcal{E}(x, p)$ is gradient vector which is evaluated as.

$$\nabla \mathcal{E}(x, p) = J(x, p)e(x, p). \quad (11)$$

The $\nabla^2 \mathcal{E}(x, p)$ approximates Hessian matrix that is.

$$\mathcal{H} \approx \nabla^2 \mathcal{E}(x, p) = J^T(x, p)J(x, p) + \mu J, \quad (12)$$

where J is the identity matrix, μ is the positive coefficient, and $J(x, p)$ is the Jacobian matrix which is defined as follows:

$$J(x, p) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \dots & \frac{\partial e_{1,1}}{\partial w_p} \\ \frac{\partial e_{1,2}}{\partial w_1} & \frac{\partial e_{1,2}}{\partial w_2} & \dots & \frac{\partial e_{1,2}}{\partial w_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{1,M}}{\partial w_1} & \frac{\partial e_{1,M}}{\partial w_2} & \dots & \frac{\partial e_{1,M}}{\partial w_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{N,1}}{\partial w_1} & \frac{\partial e_{N,1}}{\partial w_2} & \dots & \frac{\partial e_{N,1}}{\partial w_p} \\ \frac{\partial e_{N,2}}{\partial w_1} & \frac{\partial e_{N,2}}{\partial w_2} & \dots & \frac{\partial e_{N,2}}{\partial w_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{N,M}}{\partial w_1} & \frac{\partial e_{N,M}}{\partial w_2} & \dots & \frac{\partial e_{N,M}}{\partial w_p} \end{bmatrix}, e = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ \vdots \\ e_{1,M} \\ \vdots \\ e_{N,1} \\ e_{N,2} \\ \vdots \\ e_{N,M} \end{bmatrix}, \quad (13)$$

where P is the number of weights. The above given information, the Eq. (9) can be written as

$$w_{k+1} = w_k - [J^T J + \mu J]^{-1} J^T e. \quad (14)$$

The LM algorithm can be summarized for easy understanding in the following Fig. 11.

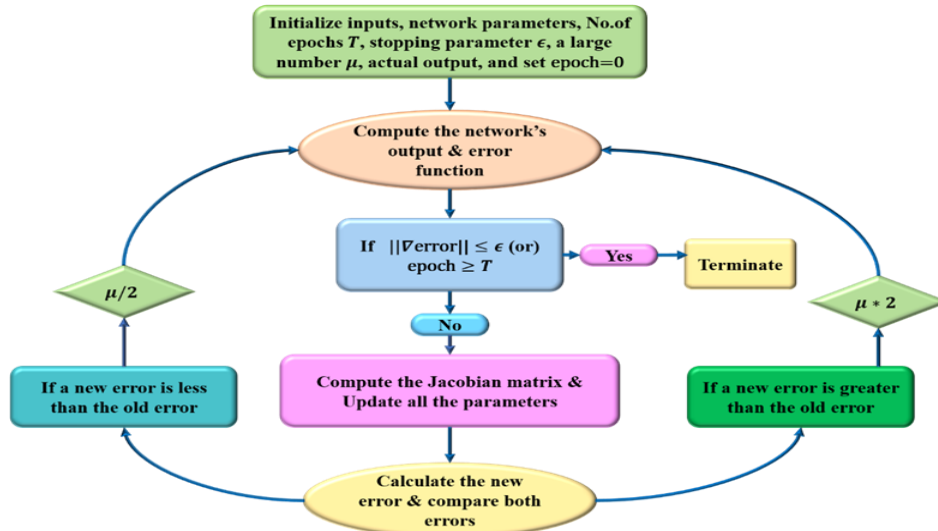


Fig. 11. Flow diagram of traditional levenberg-marquardt algorithm.

4.1 | Local Modification of Levenberg-Marquardt Algorithm

The traditional LM algorithm encountered computational complexity and memory consumption due to the size of the Jacobian matrix. Specifically, the structure of the Jacobian matrix was constructed only once per epoch. In this regard, [56] presented a method for optimizing the LM algorithm, i.e., Local Modification of LM (LLM). Consequently, it reduces the complexity associated with the size of the Jacobian matrix by partitioning the single Jacobian matrix into smaller ones and locally applying them to entire neurons within an epoch. As a result, the formation of Jacobian matrix of j^{th} neuron in n^{th} layer [56] is shown in the following Eq. (15).

$$J_j^n(x_j, p_j) = \begin{bmatrix} \frac{\partial e_j^n}{\partial w_{j(1)}} & \frac{\partial e_j^n}{\partial w_{j(2)}} & \dots & \frac{\partial e_j^n}{\partial w_{j(\mathcal{P})}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_j^n}{\partial w_{j(1)}} & \frac{\partial e_j^n}{\partial w_{j(2)}} & \dots & \frac{\partial e_j^n}{\partial w_{j(\mathcal{P})}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_j^n}{\partial w_{j(1)}} & \frac{\partial e_j^n}{\partial w_{j(2)}} & \dots & \frac{\partial e_j^n}{\partial w_{j(\mathcal{P})}} \end{bmatrix}. \quad (15)$$

The Eq. (15) calculates each element as mentioned in the Eq. (16).

$$\frac{\partial e_j^n}{\partial w_{ij}} = \delta_j^n x_j^n. \quad (16)$$

It can observe that each node has only one output, automatically reducing the number of outputs in given equation. The following Eq. (17) calculates the weight update rule of the LLM algorithm for the respective neuron.

$$\Delta_j^n w_j(x_j, p_j) = \left[J_j^{nT}(x_j, p_j) J_j^n(x_j, p_j) + \mu_j^n \mathcal{I} \right]^{-1} J_j^n(x_j, p_j)^T e_j^n(x_j, p_j). \quad (17)$$

Now, by using QR decomposition the above Eq. (17) can be converted into the matrix structure as follows:

$$\mathcal{B}_j^n = \left[J_j^{nT}(x_j, p_j) J_j^n(x_j, p_j) + \mu_j^n \mathcal{I} \right]. \quad (18)$$

$$\mathcal{G}_j^n = J_j^n(x_j, p_j)^T e_j^n(x_j, p_j). \quad (19)$$

From Eqs. (18) and (19) the weight update vector of the corresponding node of LLM is calculated as mentioned below in Eq. (20).

$$\Delta_j^n w_j(x_j, p_j) = \left[\mathcal{B}_j^n \right]^{-1} \mathcal{G}_j^n. \quad (20)$$

The Eq. (20) is the entry point to the QR decomposition. The performance of the QR decomposition is given in [56] that can be represented in Eqs. (21) and (22).

$$\mathcal{Q}_j^{nT} \left[\mathcal{B}_j^n \right] \Delta_j^n w_j(x_j, p_j) = \mathcal{Q}_j^{nT} \mathcal{G}_j^n. \quad (21)$$

$$\mathcal{R}_j^n \Delta_j^n w_j(x_j, p_j) = \mathcal{Q}_j^{nT} \mathcal{G}_j^n. \quad (22)$$

The following steps can summarize the LLM algorithm.

- I. Calculate the feedforward neural network's output by the given data.
- II. Perform the backpropagation algorithm and update the corresponding weights using Eqs. (15) and (16).
- III. Compute the error and go to the step IV. Then, compute the error criterion for Eq. (8).
- IV. Solve Eq. (11) to obtain updated weights of all nodes and compute the new error using updated weights also compute new error criteria for each node using Eq. (8).
- V. Compare both errors (new and old). If new < old, divide μ_j^n by β , then go to step I. If new > old, multiply μ_j^n by β . Then go to step I.
- VI. Continue this process until the target error is reached.

4.2 | Improved Computation for Levenberg-Marquardt Training

In the traditional LM algorithm, more memory is required for storing the Jacobian matrix and conducting its multiplications. Therefore, the proposed modification of LM enhances both memory storage efficiency and time efficiency in Jacobian matrix multiplication [58]. Consequently, this adapted LM algorithm directly computes the Quasi-Hessian matrix and gradient vector without the need for Jacobian matrix multiplication.

The Hessian matrix $\mathcal{H}$ of Eq. (1) of size $n \times n$ is derived as follows:

$$\mathcal{H} = \begin{bmatrix} \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_1^2} & \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_1 \partial w_n} \\ \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_2 \partial w_1} & \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_2^2} & \dots & \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_2 \partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_n \partial w_1} & \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_n \partial w_2} & \dots & \frac{\partial^2 \mathcal{E}(x,p)}{\partial w_n^2} \end{bmatrix}. \quad (23)$$

The elements of the Hessian matrix can be obtained from Eqs. (7) and (23) as follows:

$$\frac{\partial^2 \mathcal{E}(x,p)}{\partial w_i \partial w_j} = \sum_{n=1}^{\mathcal{N}} \sum_{m=1}^{\mathcal{M}} \frac{\partial e_{(n,m)}}{\partial w_i} \frac{\partial e_{(n,m)}}{\partial w_j} + \frac{\partial e_{(n,m)}}{\partial w_i \partial w_j} e_{(n,m)}. \quad (24)$$

From Eq. (24) is approximated as follows:

$$\frac{\partial^2 \mathcal{E}(x,p)}{\partial w_i \partial w_j} \approx \sum_{n=1}^{\mathcal{N}} \sum_{m=1}^{\mathcal{M}} \frac{\partial e_{(n,m)}}{\partial w_i} \frac{\partial e_{(n,m)}}{\partial w_j} = q_{(i,j)}, \quad (25)$$

where $q_{(i,j)}$ is the element of Quasi-Hessian matrix.

From Eqs. (13) and (25), Quasi-Hessian matrix 'Q' can be computed as an approximation of Hessian matrix.

$$\mathcal{H} \approx \mathcal{Q} = \mathcal{J}^T \mathcal{J}. \quad (26)$$

The gradient vector $\mathcal{g}$ of size $n \times 1$ is.

$$\mathcal{g} = \left[\frac{\partial \mathcal{E}(x,p)}{\partial w_1} \quad \frac{\partial \mathcal{E}(x,p)}{\partial w_2} \quad \dots \quad \frac{\partial \mathcal{E}(x,p)}{\partial w_n} \right]^T. \quad (27)$$

The elements of $\mathcal{g}$ can be computed by inserting Eqs. (7)-(27) as follows:

$$\mathcal{g}_i = \frac{\partial \mathcal{E}(x,p)}{\partial w_i} = \sum_{n=1}^{\mathcal{N}} \sum_{m=1}^{\mathcal{M}} \frac{\partial e_{(n,m)}}{\partial w_i} e_{(n,m)}. \quad (28)$$

From Eqs. (13) and (28), we obtain the following Eq. (29).

$$\mathcal{g} = \mathcal{J}^T e(x, p). \quad (29)$$

Finally, the modified process of the LM algorithm can be obtained by Eqs. (26), (29), and (10) as follows:

$$\Delta w = (\mathcal{Q} + \mu \mathcal{J})^{-1} \mathcal{g}. \quad (30)$$

4.3 | Stability and Boundedness Analysis of the Modified Levenberg-Marquardt Algorithm

In the traditional Modified LM algorithm (MLM), it is noticeable that a determinant value of zero in the Hessian matrix indicates a singularity point in the learning rates. As a result, the LM algorithm does not guarantee stable error during training. This stability is crucial for ensuring consistent and reliable learning in neural networks. Therefore, it is necessary to find a way to modify the LM algorithm to ensure its error is stable.

The updating weights of the MLM algorithm are defined as follows:

$$\begin{aligned} w_{ij}^{k+1} &= w_{ij}^k - \beta^k \frac{\partial \mathcal{E}(x,p)}{\partial w_{ij}^k} + \gamma^k \frac{\partial \mathcal{E}(x,p)}{\partial v_{ij}^k}, \\ v_{ij}^{k+1} &= v_{ij}^k - \beta^k \frac{\partial \mathcal{E}(x,p)}{\partial v_{ij}^k} + \gamma^k \frac{\partial \mathcal{E}(x,p)}{\partial w_{ij}^k}, \end{aligned} \quad (31)$$

Where

$$\beta^k = \frac{(\alpha + (\delta_3^k)^2)(\alpha + (\delta_4^k)^2)}{\det(\mathcal{H} + \alpha \mathcal{I})^k}, \quad (32)$$

where $(\mathcal{H} + \alpha \mathcal{I})^k$ is a Hessian matrix of MLM algorithm.

The determinant of the Hessian matrix can be defined as follows:

$$\det(\mathcal{H} + \alpha \mathcal{I})^k = \left([\alpha + (\delta_1^k)^2 + (\delta_2^k)^2] * [(\alpha + (\delta_3^k)^2)(\alpha + (\delta_4^k)^2) + (\delta_5^k)^2] \right), \quad (33)$$

Where

$$\begin{aligned} \delta_1^k &= \sum \frac{\left(\frac{\partial \mathcal{E}(x, p)}{\partial w_{ij}^k} \right)}{(d^k - y^k)}, \quad \delta_2^k = \sum \frac{\left(\frac{\partial \mathcal{E}(x, p)}{\partial v_{ij}^k} \right)}{(d^k - y^k)}, \\ \Delta_3^k &= \sum \frac{\partial^2 \mathcal{E}(x, p)}{\partial (w_{ij}^k)^2}, \quad \delta_4^k = \sum \frac{\partial^2 \mathcal{E}(x, p)}{\partial (v_{ij}^k)^2}, \end{aligned} \quad (34)$$

$$\delta_5^k = \sum \frac{\partial^2 \mathcal{E}(x, p)}{\partial w_{ij}^k \partial v_{ij}^k}, \gamma^k = 0,$$

And

$$\begin{aligned} \frac{\left(\frac{\partial \mathcal{E}(x, p)}{\partial w_{ij}^k} \right)}{(d^k - y^k)} &= v_{ij}^k g'(z^k) x^k, \quad \frac{\left(\frac{\partial \mathcal{E}(x, p)}{\partial v_{ij}^k} \right)}{(d^k - y^k)} = g(z^k), \\ \frac{\partial^2 \mathcal{E}(x, p)}{\partial (w_{ij}^k)^2} &= (x^k)^2 v_{ij}^k * [-2g(z^k)g'(z^k)(d^k - y^k) + g'(z^k)^2 v_{ij}^k], \end{aligned} \quad (35)$$

$$\frac{\partial^2 \mathcal{E}(x, p)}{\partial (v_{ij}^k)^2} = g(z^k)^2, \quad \frac{\partial^2 \mathcal{E}(x, p)}{\partial w_{ij}^k \partial v_{ij}^k} = x^k g'(z^k) [(d^k - y^k) + g(z^k) v_{ij}^k],$$

$$\frac{\partial \mathcal{E}(x, p)}{\partial w_{ij}^k} = (d^k - y^k) v_{ij}^k g'(z^k) x^k, \quad \frac{\partial \mathcal{E}(x, p)}{\partial v_{ij}^k} = g(z^k) (d^k - y^k).$$

Here, we use tanh in the hidden layer and a linear function in the output layer. w_{ij}^k , v_{ij}^k are the weights of the input to hidden layer and the hidden to output layer respectively, x^k are the input values, $g(z^k)$ is the activation function, β^k is learning rate, α is the learning factor, d^k is the neural networks output, and y^k is the data set actual values. The MLM algorithm of Eq. (31)-(35) is considered from [57].

According to [57], The MLM algorithm incorporates two modifications to ensure the stability of error and the boundedness of weights, as outlined below

- I. There is not a value of zero on the right side of Eq. (33) and there is not a singularity point of learning rate β^k of the MLM algorithm.
- II. The MLM has only one learning rate.

The above-mentioned LM modifications are successfully applied to various applications such as parity problem and Modified National Institute of Standards and Technology (MNIST) database, electric and brain signals dataset, various classification problems, and get better results as shown in Tables 8 and 9.

Table 8. Applications problems using modified levenberg-marquardt algorithm method in neural networks.

Base Algorithm	Modification	Application/Problem	Analysis
LM [58]	Without computing the Jacobian matrix	Parity problem and MINST database problem	Improved memory and time
LM [57]	Modification on weight updating and hessian matrix and their determinant	Electric and brain signals dataset	Error stability and weight boundedness
LM [56]	Local modification on Jacobian matrix	different classification problems	Computational complexity and memory usage

Table 9. Results compared between the traditional levenberg-marquardt and modified LM.

Applications	Comparing parameter	Method	
		Traditional LM	Modified LM
Parity problem	Run Time (Milliseconds (ms))	1.85ms	0.4ms
	Jacobian matrix size	11×35	11×8
	Run Time (Milliseconds (ms))	52.94	6.02
	Jacobian matrix size	50×133	50×16
	Memory used	79.21"Mb"	3.41"Mb"
	MNIST database problem		
MNIST database problem	Memory used	"385.68Mb"	15.67"Mb"
	Training data set	0.1304	0.0555
	Testing data set	0.0925	0.0227

5 | Conclusion

This paper presents an in-depth examination of the application of ANNs in exploring DEs. It discusses the evolution of ANN over time and its wide-ranging utility. ANNs have become essential due to limitations inherent in analytical and numerical methods, as well as challenges associated with accessing comprehensive system datasets. Numerical techniques such as FDM, FEM, and FVM can incur significant computational costs, particularly with large-scale problems. Moreover, certain types of DEs may exhibit singularities or discontinuities, posing stability and convergence issues for numerical methods. In response to these challenges, alternative approaches like ANNs are being embraced for solving DEs. Following are the major features of ANN.

- I. Flexibility and effortless tackling of nonlinear functions.
- II. ANN acts as a universal function approximator, capable of accurately approximating any continuous function to arbitrary precision when provided with a sufficient number of neurons and layers. This characteristic empowers them to represent diverse solutions to DEs across a broad spectrum.
- III. ANNs have the ability to learn from input-output data, discerning patterns and relationships within the dataset.

- IV. ANNs offer computational efficiency in solving DEs compared to numerical methods, especially when confronted with extensive datasets.

In summary, ANNs emerge as potent tools for tackling DEs, particularly in scenarios where numerical techniques encounter obstacles. Thus, ANNs find extensive applicability across various domains including science and engineering. This paper delves into the utilization of ANNs for solving DEs such as ODEs, PDEs. It underscores research efforts integrating ANNs with optimization techniques to enhance training efficiency and accuracy.

Furthermore, the paper explores the structure of ANNs along with their diverse types and activation functions like sigmoid, tanh, ReLU, and SoftMax, alongside their advancements. Through this examination, it becomes evident that the traditional (LM) algorithm stands out as a robust training algorithm for ANNs. Additionally, modifications to the LM algorithm within ANNs are discussed, showcasing enhancements in the learning process. Leveraging the LM algorithm and its refinements for training ANNs enhances algorithm accuracy, reduces execution time, and optimizes memory usage.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability

All data are included in the text.

Funding

This research received no specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

- [1] Iyengar, S. R. K., & Jain, R. K. (2009). *Numerical methods*. New Age International.
<https://books.google.com/books?hl=en&lr=&id=5p5jFxb16UEC&oi>
- [2] Nayak, S., & Chakraverty, S. (2018). *Interval finite element method with MATLAB*. Academic press.
<https://www.researchgate.net/publication/325195030>
- [3] Chapra, S. C., Canale, R. P., & others. (2011). *Numerical methods for engineers* (Vol. 1221). McGraw-hill New York. <https://www.researchgate.net/profile/Steven-Chapra/publication/44398580>
- [4] Gerald, C. F. (2004). *Applied numerical analysis*. Pearson Education India.
[https://www.cse.iitm.ac.in/~vplab/downloads/opt/Applied Numerical Analysis.pdf](https://www.cse.iitm.ac.in/~vplab/downloads/opt/Applied%20Numerical%20Analysis.pdf)
- [5] Atkinson, K. E. (2008). *An introduction to numerical analysis*. John Wiley & sons.
<https://math.science.cmu.ac.th/docs/qNA2556>
- [6] Reddy, J. N. (2006). *An introduction to the finite element method*. 27 New York (Vol. 27). McGraw-Hill Higher Education. <http://dx.doi.org/10.1115/1.3265687>
- [7] Radi, B., & El Hami, A. (2018). *Advanced numerical methods with matlab 2: Resolution of nonlinear, differential and partial differential equations*. John Wiley & Sons. <http://dx.doi.org/10.1002/9781119492238>
- [8] Hildebrand, F. B. (1987). *Introduction to numerical analysis*. Courier Corporation.
https://books.google.com/books?hl=en&lr=&id=f7We11dz0_kC&oi=fnd&pg
- [9] Burden, R. L., & Faires, J. D. (2011). *Student solutions manual and study guide*. Brooks/Cole, Cengage Learning. https://d1wqtxts1xzle7.cloudfront.net/51824818/9th_edition
- [10] Trefethen, L. N. (1996). *Finite difference and spectral methods for ordinary and partial differential equations*. Cornell University-Department of Computer Science and Center for Applied.
<http://www.math.hmc.edu/~dyong/math165/trefethenbook.pdf>

- [11] Priyadarshini, S., & Nayak, S. (2022). A numerical approach to study heat and mass transfer in porous medium influenced by uncertain parameters. *International communications in heat and mass transfer*, 139, 106411. <https://doi.org/10.1016/j.icheatmasstransfer.2022.106411>
- [12] Priyadarshini, S., & Nayak, S. (2023). Effects of imprecisely defined parameters on heat and mass transfer in a vertical annular porous cylinder. *International communications in heat and mass transfer*, 149, 107097. <https://doi.org/10.1016/j.icheatmasstransfer.2023.107097>
- [13] Priyadarshini, S., & Nayak, S. (2023). A new hybrid approach to study heat and mass transfer in porous medium influenced by imprecisely defined parameters. *Case studies in thermal engineering*, 51, 103619. <https://doi.org/10.1016/j.csite.2023.103619>
- [14] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 115–133. <https://doi.org/10.1007/BF02478259>
- [15] Moghaddam, A. H., Moghaddam, M. H., & Esfandyari, M. (2016). Stock market index prediction using artificial neural network. *Journal of economics, finance and administrative science*, 21(41), 89–93. <https://doi.org/10.1016/j.jefas.2016.07.002>
- [16] Shakeel, P. M., Tobely, T. E. El, Al-Feel, H., Manogaran, G., & Baskar, S. (2019). Neural network based brain tumor detection using wireless infrared imaging sensor. *IEEE access*, 7, 5577–5588. <https://doi.org/10.1109/ACCESS.2018.2883957>
- [17] Heydarpour, F., Abbasi, E., Ebadi, M. J., & Karbassi, S.-M. (2020). Solving an optimal control problem of cancer treatment by artificial neural networks. *IJIMAI*, 6(4), 18–25. <http://dx.doi.org/10.9781/ijimai.2020.11.011>
- [18] de Souza, G. B., da Silva Santos, D. F., Pires, R. G., Marana, A. N., & Papa, J. P. (2019). Deep features extraction for robust fingerprint spoofing attack detection. *Journal of artificial intelligence and soft computing research*, 9(1), 41–49. <http://dx.doi.org/10.2478/jaiscr-2018-0023>
- [19] Lam, M. W. Y. (2018). One-match-ahead forecasting in two-team sports with stacked Bayesian regressions. *Journal of artificial intelligence and soft computing research*, 8. <http://dx.doi.org/10.1515/jaiscr-2018-0011>
- [20] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5), 987–1000. <https://doi.org/10.1109/72.712178>
- [21] Isaac, El., Likas, A. C., & Papageorgiou, D. G. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE transactions on neural networks*, 11(5), 1041–1049. <https://doi.org/10.1109/72.870037>
- [22] Parisi, D. R., Mariani, M. C., & Laborde, M. A. (2003). Solving differential equations with unsupervised neural networks. *Chemical engineering and processing: Process intensification*, 42(8–9), 715–721. [https://doi.org/10.1016/S0255-2701\(02\)00207-6](https://doi.org/10.1016/S0255-2701(02)00207-6)
- [23] Malek, A., & Beidokhti, R. S. (2006). Numerical solution for high order differential equations using a hybrid neural network—optimization method. *Applied mathematics and computation*, 183(1), 260–271. <https://doi.org/10.1016/j.amc.2006.05.068>
- [24] Tsoulos, I. G., Gavrilis, D., & Glavas, E. (2009). Solving differential equations with constructed neural networks. *Neurocomputing*, 72(10–12), 2385–2391. <https://doi.org/10.1016/j.neucom.2008.12.004>
- [25] Zahoor Raja, M. A., Khan, J. A., & Qureshi, I. M. (2010). *Evolutionary computational intelligence in solving the fractional differential equations* [presentation]. Asian conference on intelligent information and database systems (pp. 231–240). https://doi.org/10.1007/978-3-642-12145-6_24
- [26] Junaid, A., Raja, M. A. Z., & Qureshi, I. M. (2009). Evolutionary computing approach for the solution of initial value problems in ordinary differential equations. *World academy of science, engineering and technology*, 55, 578–5581. https://doi.org/10.1007/978-3-642-12145-6_24
- [27] Mall, S., & Chakraverty, S. (2013). Comparison of artificial neural network architecture in solving ordinary differential equations. *Advances in artificial neural systems*, 2013(1), 181895. <https://doi.org/10.1155/2013/181895>
- [28] Chakraverty, S., & Mall, S. (2014). Regression-based weight generation algorithm in neural network for solution of initial and boundary value problems. *Neural computing and applications*, 25(3), 585–594. <https://doi.org/10.1007/s00521-013-1526-4%0A%0A>

- [29] Shi, E., & Xu, C. (2021). A comparative investigation of neural networks in solving differential equations. *Journal of algorithms & computational technology*, 15, 1748302621998605. <https://doi.org/10.1177/1748302621998605>
- [30] Mall, S., & Chakraverty, S. (2014). Chebyshev neural network based model for solving Lane--Emden type equations. *Applied mathematics and computation*, 247, 100–114. <https://doi.org/10.1016/j.amc.2014.08.085>
- [31] Chakraverty, S., & Mall, S. (2020). Single layer Chebyshev neural network model with regression-based weights for solving nonlinear ordinary differential equations. *Evolutionary intelligence*, 13(4), 687–694. <https://doi.org/10.1007/s12065-020-00383-y%0A%0A>
- [32] Raja, M. A. Z., Khan, J. A., & Qureshi, I. M. (2010). A new stochastic approach for solution of Riccati differential equation of fractional order. *Annals of mathematics and artificial intelligence*, 60(3), 229–250. <https://doi.org/10.1007/s10472-010-9222-x%0A%0A>
- [33] Qu, H., & Liu, X. (2015). A numerical method for solving fractional differential equations by using neural network. *Advances in mathematical physics*, 2015(1), 439526. <https://doi.org/10.1155/2015/439526>
- [34] Raja, M. A. Z., Manzar, M. A., & Samar, R. (2015). An efficient computational intelligence approach for solving fractional order Riccati equations using ANN and SQP. *Applied mathematical modelling*, 39(10–11), 3075–3093. <https://doi.org/10.1016/j.apm.2014.11.024>
- [35] Jafarian, A., Mokhtarpour, M., & Baleanu, D. (2017). Artificial neural network approach for a class of fractional ordinary differential equation. *Neural computing and applications*, 28(4), 765–773. <https://doi.org/10.1007/s00521-015-2104-8>
- [36] Pakdaman, M., Ahmadian, A., Effati, S., Salahshour, S., & Baleanu, D. (2017). Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Applied mathematics and computation*, 293, 81–95. <https://doi.org/10.1016/j.amc.2016.07.021>
- [37] Pratama Danang, A., A. Bakar, M., Ismail, N. B., & M, M. (2022). ANN-based methods for solving partial differential equations: a survey. *Arab journal of basic and applied sciences*, 29(1), 233–248. <https://doi.org/10.1080/25765299.2022.2104224>
- [38] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339–1364. <https://doi.org/10.1016/j.jcp.2018.08.029>
- [39] Knoke, T., & Wick, T. (2021). Solving differential equations via artificial neural networks: Findings and failures in a model problem. *Examples and counterexamples*, 1, 100035. <https://doi.org/10.1016/j.exco.2021.100035>
- [40] Valasoulis, K., Fotiadis, D. L., Lagaris, I. E., & Likas, A. (2002). *Solving differential equations with neural networks: implementation on a dsp platform* [presentation]. 2002 14th international conference on digital signal processing proceedings. dsp 2002 (cat. no. 02th8628) (Vol. 2, pp. 1265–1268). <https://doi.org/10.1109/ICDSP.2002.1028323>
- [41] Zang, C., & Wang, F. (2020). *Neural dynamics on complex networks* [presentation]. Proceedings of the 26th acm sigkdd international conference on knowledge discovery & data mining (pp. 892–902). <https://doi.org/10.1145/3394486.3403132>
- [42] Blechschmidt, J., & Oliver, E. (2021). Three ways to solve partial differential equations with neural networks—A review. *Gamm-mitteilungen*, 44(2), e202100006. <https://doi.org/10.1002/gamm.202100006>
- [43] Althubiti, S., Kumar, M., Goswami, P., & Kumar, K. (2023). Artificial neural network for solving the nonlinear singular fractional differential equations. *Applied mathematics in science and engineering*, 31(1), 2187389. <https://doi.org/10.1080/27690911.2023.2187389>
- [44] Venkatachalapathy, P., & Mallikarjunaiah, S. M. (2023). A deep learning neural network framework for solving singular nonlinear ordinary differential equations. *International journal of applied and computational mathematics*, 9(5), 1–68. <https://doi.org/10.1007/s40819-023-01563-x>
- [45] Fojdl, J., & Brause, R. W. (2008). *The performance of approximating ordinary differential equations by neural nets* [presentation]. 2008 20th IEEE international conference on tools with artificial intelligence (Vol. 2, pp. 457–464). <https://doi.org/10.1109/ICTAI.2008.44>
- [46] Michoski, C., Milosavljević, M., Oliver, T., & Hatch, D. R. (2020). Solving differential equations using deep neural networks. *Neurocomputing*, 399, 193–212. <https://doi.org/10.1016/j.neucom.2020.02.015>

- [47] Kumar, M., & Yadav, N. (2011). Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. *Computers & mathematics with applications*, 62(10), 3796–3811. <https://doi.org/10.1016/j.camwa.2011.09.028>
- [48] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 1–13. https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6
- [49] Nayak, S. (2020). *Fundamentals of optimization techniques with algorithms*. Academic Press. <https://shop.elsevier.com/books/fundamentals-of-optimization-techniques-with-algorithms/nayak/978-0-12-821126-7>
- [50] Kalyanmoy, D. (2004). *Optimization for engineering design: Algorithms and examples*. Prentice-Hall Of India Pvt. Limited. <https://www.semanticscholar.org/paper/Optimization-for-Engineering-Design%3A-Algorithms-and-Deb-Deb/48556b84dd1bf424a45d4f6acd39e29a05cc9158>
- [51] Rao, S. S. (2019). *Engineering optimization: theory and practice*. John Wiley & Sons. <https://doi.org/10.1002/9781119454816>
- [52] Panigrahi, P. K., & Nayak, S. (2024). Numerical approach to solve imprecisely defined systems using Inner Outer Direct Search optimization technique. *Mathematics and computers in simulation*, 215, 578–606. <https://doi.org/10.1016/j.matcom.2023.08.025>
- [53] Panigrahi, P. K., & Nayak, S. (2023). Numerical investigation of non-probabilistic systems using Inner Outer Direct Search optimization technique. *AIMS mathematics*, 8(9), 21329–21358. <https://doi.org/10.3934/math.20231087%0D>
- [54] Nayak, S., & Pooja, J. (2022). Numerical optimisation technique to solve imprecisely defined nonlinear system of equations with bounded parameters. *International journal of mathematics in operational research*, 23(3), 394–411. <https://doi.org/10.1504/IJMOR.2022.127381>
- [55] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533–536. <https://doi.org/10.1038/323533a0>
- [56] Bilski, J., Kowalczyk, B., Marchlewska, A., & Zurada, J. M. (2020). Local Levenberg-Marquardt algorithm for learning feedforward neural networks. *Journal of artificial intelligence and soft computing research*, 10(4), 299–316. <https://doi.org/10.2478/jaiscr-2020-0020>
- [57] de Jesús Rubio, J. (2020). Stability analysis of the modified Levenberg–Marquardt algorithm for the artificial neural network training. *IEEE transactions on neural networks and learning systems*, 32(8), 3510–3524. <https://doi.org/10.1109/TNNLS.2020.3015200>
- [58] Wilamowski, B. M., & Yu, H. (2010). Improved computation for Levenberg–Marquardt training. *IEEE transactions on neural networks*, 21(6), 930–937. <https://doi.org/10.1109/TNN.2010.2045657>
- [59] Li, S., Deng, Y. Q., Zhu, Z. L., Hua, H. L., & Tao, Z. Z. (2021). A comprehensive review on radiomics and deep learning for nasopharyngeal carcinoma imaging. *Diagnostics*, 11(9), 1523. <https://doi.org/10.3390/diagnostics11091523>
- [60] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386–408. <https://psycnet.apa.org/doi/10.1037/h0042519>
- [61] Rosenblatt, F., & others. (1962). *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms* (Vol. 55). Spartan books Washington, DC. <https://www.scirp.org/reference/referencespapers?referenceid=1501842>
- [62] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. <https://doi.org/10.1109/5.726791>
- [63] Holyoak, K. J. (1987). A connectionist view of cognition: parallel distributed processing. explorations in the microstructure of cognition. David E. Rumelhart, James L. McClelland, and the PDP Research Group. MIT Press, Cambridge, MA, 1986. In two volumes. Vol. 1, Foundations. . *Science*, 236(4804), 992–996. <https://www.science.org/doi/abs/10.1126/science.236.4804.992>
- [64] Shanmuganathan, S. (2016). Artificial neural network modelling: An introduction. In *Artificial neural network modelling* (pp. 1–14). Springer. https://doi.org/10.1007/978-3-319-28495-8_1
- [65] Chakraverty, S., & Mall, S. (2017). *Artificial neural networks for engineers and scientists: solving ordinary differential equations*. CRC Press. <https://doi.org/10.1201/9781315155265>

- [66] Han, J., Pei, J., & Tong, H. (2022). *Data mining: Concepts and techniques*. Morgan kaufmann. <https://myweb.sabanciuniv.edu/rdehkharghani/files/2016/02>
- [67] Zurada, J. (1992). *Introduction to artificial neural systems*. West Publishing Co. <https://dl.acm.org/doi/abs/10.5555/131393>
- [68] Bonaccorso, G. (2017). Machine learning algorithms.[SI]. https://books.google.com/books/about/Machine_Learning_Algorithms.html?id=_ZDDwAAQBAJ
- [69] Yadav, N., Yadav, A., & Kumar, M. (2015). An introduction to neural network methods for differential equations. *SpringerBriefs in applied sciences and technology*. <https://doi.org/10.1007/978-94-017-9816-7>
- [70] Demuth, H., Beale, M., & Hagan, M. (1992). *Neural network toolbox*. , 2000 For Use with MATLAB. The MathWorks Inc (Vol. 2000). The MathWorks. https://books.google.com/books/about/Machine_Learning_Algorithms.html?id=_ZDDwAAQBAJ
- [71] Karlik, B., Olgac, A. V., & others. (2011). Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International journal of artificial intelligence and expert systems*, 1(4), 111–122. <https://www.cscjournals.org/manuscript/Journals/IJAE/Volume1/Issue4/IJAE-26.pdf>
- [72] Jagtap, A. D., & Karniadakis, G. E. (2023). How important are activation functions in regression and classification? A survey, performance comparison, and future directions. *Journal of machine learning for modeling and computing*, 4(1), 21–75. <https://55.10.1615/JMachLearnModelComput.2023047367>
- [73] Han, J., & Moraga, C. (1995). *The influence of the sigmoid function parameters on the speed of backpropagation learning* [presentation]. International workshop on artificial neural networks (pp. 195–201). https://doi.org/10.1007/3-540-59497-3_175
- [74] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., ... & Others. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE signal processing magazine*, 29(6), 82–97. <https://doi.org/10.1109/MSP.2012.2205597>
- [75] Nair, V., & Hinton, G. E. (2010). *Rectified linear units improve restricted boltzmann machines* [presentation]. Proceedings of the 27th international conference on machine learning (icml-10) (pp. 807–814). <https://dl.acm.org/doi/10.5555/3104322.3104425>
- [76] Kwan, H. K. (1992). Simple sigmoid-like activation function suitable for digital hardware implementation. *Electronics letters*, 28(15), 1379–1380. /doi/abs/10.1049/el:19920877
- [77] Mansor, M. A., & Sathasivam, S. (2016). *Activation function comparison in neural-symbolic integration* [presentation]. AIP conference proceedings (Vol. 1750, p. 20013). <https://doi.org/10.1063/1.4954526>
- [78] Elliott, D. L. (1993). *A better activation function for artificial neural networks*. <https://api.drum.lib.umd.edu/server/api/core/bitstreams/fbe46c3c>
- [79] Farzad, A., Mashayekhi, H., & Hassanpour, H. (2019). A comparative performance analysis of different activation functions in LSTM networks for classification. *Neural computing and applications*, 31(7), 2507–2521. <https://doi.org/10.1007/s00521-017-3210-6%0A%0AKeywords>
- [80] Pravin, C., & Singh, Y. (2004). An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing*, 61, 429–437. <https://doi.org/10.1016/j.neucom.2004.04.001>
- [81] Qin, Y., Wang, X., & Zou, J. (2018). The optimized deep belief networks with improved logistic sigmoid units and their application in fault diagnosis for planetary gearboxes of wind turbines. *IEEE transactions on industrial electronics*, 66(5), 3814–3824. <https://doi.org/10.1109/TIE.2018.2856205>
- [82] Steffen, E., Youssef, P., & Gurevych, I. (2019). Is it time to swish? Comparing deep learning activation functions across NLP tasks. *ArXiv preprint arxiv:1901.02671*. <https://doi.org/10.48550/arXiv.1901.02671>
- [83] Kong, S., & Takatsuka, M. (2017). *Hexpo: A vanishing-proof activation function* [presentation]. 2017 international joint conference on neural networks (IJCNN) (pp. 2562–2567). <https://doi.org/10.1109/IJCNN.2017.7966168>
- [84] Eger, S., Youssef, P., & Gurevych, I. (2019). Is it time to swish? Comparing deep learning activation functions across NLP tasks. *ArXiv preprint arxiv:1901.02671*. <https://doi.org/10.48550/arXiv.1901.02671>
- [85] Maas, A. L., Hannun, A. Y., Ng, A. Y., & others. (2013). *Rectifier nonlinearities improve neural network acoustic models* [presentation]. Proceedings of the 30th international conference on machine learning. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

- [86] Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2019). Dying relu and initialization: Theory and numerical examples. *ArXiv preprint arxiv:1903.06733*. <https://doi.org/10.4208/cicp.OA-2020-0165>
- [87] Shang, W., Sohn, K., Almeida, D., & Lee, H. (2016). *Understanding and improving convolutional neural networks via concatenated rectified linear units* [presentation]. International conference on machine learning (pp. 2217–2225). <http://dx.doi.org/10.48550/arXiv.1603.05201>
- [88] Godin, F., Degraeve, J., Dambre, J., & De Neve, W. (2018). Dual rectified linear units (DReLU): A replacement for tanh activation functions in quasi-recurrent neural networks. *Pattern recognition letters*, 116, 8–14. <https://doi.org/10.1016/j.patrec.2018.09.006>
- [89] Cao, J., Pang, Y., Li, X., & Liang, J. (2018). Randomly translational activation inspired by the input distributions of ReLU. *Neurocomputing*, 275, 859–868. <https://doi.org/10.1016/j.neucom.2017.09.031>
- [90] Liu, Y., Zhang, J., Gao, C., Qu, J., & Ji, L. (2019). Natural-logarithm-rectified activation function in convolutional neural networks. 2019 *IEEE 5th international conference on computer and communications (ICCC)* (pp. 2000–2008). IEEE. <https://doi.org/10.1109/ICCC47050.2019.9064398>
- [91] Dubey, S. R., & Chakraborty, S. (2021). Average biased ReLU based CNN descriptor for improved face retrieval. *Multimedia tools and applications*, 80(15), 23181–23206. <https://doi.org/10.1007/s11042-020-10269-x%0A%0A>
- [92] Lee, H., & Kang, I. S. (1990). Neural algorithm for solving differential equations. *Journal of computational physics*, 91(1), 110–131. [https://doi.org/10.1016/0021-9991\(90\)90007-N](https://doi.org/10.1016/0021-9991(90)90007-N)
- [93] Meade Jr, A. J., & Fernandez, A. A. (1994). Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and computer modelling*, 20(9), 19–44. [https://doi.org/10.1016/0895-7177\(94\)00160-X](https://doi.org/10.1016/0895-7177(94)00160-X)
- [94] Meade Jr, A. J., & Fernandez, A. A. (1994). The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and computer modelling*, 19(12), 1–25. [https://doi.org/10.1016/0895-7177\(94\)90095-7](https://doi.org/10.1016/0895-7177(94)90095-7)
- [95] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1997). Artificial neural network methods in quantum mechanics. *Computer physics communications*, 104(1–3), 1–14. [https://doi.org/10.1016/S0010-4655\(97\)00054-4](https://doi.org/10.1016/S0010-4655(97)00054-4)
- [96] Lagaris, I. E., Likas, A., & Papageorgiou, D. G. (1998). Neural network methods for boundary value problems defined in arbitrarily shaped domains. *ArXiv preprint cs/9812003*. <https://doi.org/10.48550/arXiv.cs/9812003>
- [97] Otadi, M., Mosleh, M., & others. (2011). Numerical solution of quadratic Riccati differential equation by neural network. <https://www.sid.ir/paper/322546/en>
- [98] Ibraheem, K. I., & Khalaf, B. M. (2011). Shooting neural networks algorithm for solving boundary value problems in ODEs. *Applications and applied mathematics: An international journal (aam)*, 6(1), 15. <https://digitalcommons.pvamu.edu/aam/vol6/iss1/15/>
- [99] Tawfiq, L. N. M., & Hussein, A. A. T. (2013). Design feed forward neural network to solve singular boundary value problems. *International scholarly research notices*, 2013(1), 650467. <https://doi.org/10.1155/2013/650467>
- [100] Mall, S., & Chakraverty, S. (2016). Application of Legendre neural network for solving ordinary differential equations. *Applied soft computing*, 43, 347–356. <https://doi.org/10.1016/j.asoc.2015.10.069>
- [101] Okereke, R. N., Maliki, O. S., & Oruh, B. I. (2021). A novel method for solving ordinary differential equations with artificial neural networks. *Applied mathematics*, 12(10), 900–918. <https://doi.org/10.4236/am.2021.1210059>
- [102] Aarts, L. P., & Van Der Veer, P. (2001). Neural network method for solving partial differential equations. *Neural processing letters*, 14(3), 261–271. <https://doi.org/10.1023/A:1012784129883>
- [103] Beidokhti, R. S., & Malek, A. (2009). Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. *Journal of the franklin institute*, 346(9), 898–913. <https://doi.org/10.1016/j.jfranklin.2009.05.003>
- [104] Rudd, K., & Ferrari, S. (2015). A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. *Neurocomputing*, 155, 277–285. <https://doi.org/10.1016/j.neucom.2014.11.058>

- [105] Berg, J., & Nyström, K. (2018). A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317, 28–41.
<https://doi.org/10.1016/j.neucom.2018.06.056>
- [106] Anitescu, C., Atroshchenko, E., Alajlan, N., & Rabczuk, T. (2019). Artificial neural network methods for the solution of second order boundary value problems. *Computers, materials & continua*, 59(1), 345–359.
<http://dx.doi.org/10.32604/cmc.2019.06641>
- [107] Liu, Z., Yang, Y., & Cai, Q. (2019). Neural network as a function approximator and its application in solving differential equations. *Applied mathematics and mechanics*, 40(2), 237–248.
<https://doi.org/10.1007/s10483-019-2429-8>
- [108] Panghal, S., & Kumar, M. (2021). Optimization free neural network approach for solving ordinary and partial differential equations. *Engineering with computers*, 37(4), 2989–3002. <https://doi.org/10.1007/s00366-020-00985-1>
- [109] Jafarian, A., Measoomy Nia, S., Khalili Golmankhaneh, A., & Baleanu, D. (2018). On artificial neural networks approach with new cost functions. *Applied mathematics and computation*, 339, 546–555.
<https://doi.org/10.1016/j.amc.2018.07.053>
- [110] Admon, M. R., Senu, N., Ahmadian, A., Abdul Majid, Z., & Salahshour, S. (2023). A new efficient algorithm based on feedforward neural network for solving differential equations of fractional order. *Communications in nonlinear science and numerical simulation*, 117, 106968.
<https://doi.org/10.1016/j.cnsns.2022.106968>
- [111] Hagan, M. T., & Menhaj, M. B. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE transactions on neural networks*, 5(6), 989–993. <https://doi.org/10.1109/72.329697>