Research Paper
Computational Methods for Differential Equations
http://cmde.tabrizu.ac.ir
Vol. *, No. *, *, pp. 1-13
DOI:10.22034/cmde.2024.58774.2486

Solving initial value problems using multilayer perceptron artificial neural networks

Fatemeh Ahmadkhanpour[1], **Hossein Kheiri**[2,*], **Nima Azarmir**[1], **and Farzin Modarres Khiyabani**[1]

[1] Department of Mathematics, Faculty of Science, Tabriz Branch, Islamic Azad University, Tabriz, Iran.

[2] Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran.

Abstract

This research introduces a novel approach using artificial neural networks (ANNs) to tackle ordinary differential equations (ODEs) through an innovative technique called enhanced back-propagation (EBP). The ANNs adopted in this study, particularly multilayer perceptron neural networks (MLPNNs), are equipped with tunable parameters such as weights and biases. The utilization of MLPNNs with universal approximation capabilities proves to be advantageous for ODE problem-solving. By leveraging the enhanced back-propagation algorithm, the network is fine-tuned to minimize errors during unsupervised learning sessions. To showcase the effectiveness of this method, a diverse set of initial value problems for ODEs are solved and the results are compared against analytical solutions and conventional techniques, demonstrating the superior performance of the proposed approach

Keywords. Artificial neural networks, Ordinary differential equations, Back-propagation algorithm.

2010 Mathematics Subject Classification. 92B20, 97R40, 34E13.

1. Introduction

By using deep neural networks (NNs) in solving differential equations, we can enjoy the benefits of both numerical methods and machine learning [7, 15]. These networks have the ability to approximate the unknown solutions of differential equations with high accuracy, while also being able to deal effectively with noise in the data. This is especially useful in real-world applications where the data is often noisy and incomplete.

One of the key advantages of artificial neural networks is their effectiveness in solving nonlinear problems without the need for auxiliary assumptions [6, 17]. Traditional numerical methods often require simplifications or linearizations of the problem in order to find a solution. Deep neural networks, on the other hand, are capable of directly approximating the nonlinear function of the problem [16, 21]. This opens up a wide range of applications in fields such as physics, engineering, and finance where problems are inherently nonlinear.

Another advantage of artificial neural networks is their ability to handle high-dimensional problems. Traditional numerical methods often struggle with the computational complexity that comes with higher dimensions. Deep neural networks, however, are able to preserve their speed and efficiency even when dealing with high-dimensional problems [2, 10]. This makes them a powerful tool for solving complex problems in fields such as image processing, natural language processing, and medical diagnostics.

Overall, deep neural networks have become a reliable and efficient tool for predicting the future of situations and patterns. Their ability to handle nonlinear problems, deal with noisy data, and solve problems in higher dimensions makes them a valuable asset in a wide range of fields. As research and development in deep neural networks continue to progress, we can expect even more advancements and applications in the future [4, 8, 20]. Some of the important advantages of neural networks are as follows:

Received: 09 October 2023 ; Accepted: 27 April 2024.
∗ Corresponding author. Email: h-Kheiri@tabrizu.ac.ir.

- Neural networks have demonstrated the ability to meet the universal function approximation theorem. This theorem suggests that by having an ample amount of neurons in the hidden layer, a neural network can effectively model any continuous function with remarkable precision. This remarkable trait of neural networks sets them apart as potent instruments for tackling a wide range of problems, from linear to intricate non-linear differential equations. The utilization of neural networks and deep learning techniques prove to be successful strategies for addressing mathematical models entailing non-linear differential equations [6, 18, 22, 23].
- Neural networks are like infinite cosmic beings in the world of mathematics, requiring the magical essence of automatic differentiation to reveal their mystical derivatives. This is in contrast to other techniques like Runge-Kutta and finite elements methods, which offer solutions with limited differentiability or discrete solutions. The solutions provided by ANNs are in a closed analytic form [7, 12, 17].
- Solutions with remarkable interpolation accuracy showcase the versatile generalization abilities of neural networks, making them highly effective [15, 19].
- The number of precise parameters (weights and biases) required in ANNs methods is significantly lower compared to other methods such as traditional numerical methods. As a result, complex models can be achieved with a compact solution, requiring minimal memory space [14, 16].

Therefore, it is possible to approximate any function to a high degree of accuracy using a multi-layer perceptron neural network (MLPNN) with just one hidden layer. This makes MLPNN with one hidden layer an ideal model for solving ODEs, which is one of its key advantages [16, 21]. Exploring the synergy between wavelet theory and neural networks, Sabir and his team engineered a Morlet wavelet neural network that exhibited remarkable efficacy in solving second-order non-linear differential equations [21]. Additionally, novel architectures like a single-layer orthogonal neural network tailored to leverage fractional order Legandre functions have emerged, proving instrumental in navigating the intricacies of non-linear differential equations [16].

In his paper, we present a novel approach that harnesses the power of ANNs, specifically MLPNNs, to address ODEs using an innovative technique known as enhanced back-propagation (EBP). The key idea behind this approach lies in enhancing the back-propagation algorithm by incorporating a momentum component, which allows the network to adapt and learn more efficiently during unsupervised learning sessions. One of the distinguishing features of the MLPNNs employed in this study is their universal approximation capabilities, enabled by tunable parameters such as weights and biases. These tunable parameters play a crucial role in enabling the networks to iteratively refine their predictions and converge towards accurate solutions for ODEs. Through a series of experiments involving a diverse set of initial value problems for ODEs, we aim to showcase the effectiveness and reliability of the proposed approach. The results obtained will be compared against analytical solutions and other techniques to highlight the superior performance and efficiency offered by the EBP-enhanced MLPNNs in solving ODEs. The results in the tables and figures show the efficiency of our method well.

The organization of the paper is as follows: In section 2, the concept of physics-informed neural networks is discussed. In section 3, the formulas for computing the gradient of the Loss function are derived, and the general formulation is described. In section 4, the appropriate form of the trial solution is described, and some classes of problems are presented in which the proposed method can be applied. In section 5, several numerical examples are given, and the accuracy of the obtained solution and details concerning the implementation of the approach are provided. The results obtained by other methods for the examined ODE problems are also compablue. Finally, in section 6, some future research directions and a conclusion are presented.

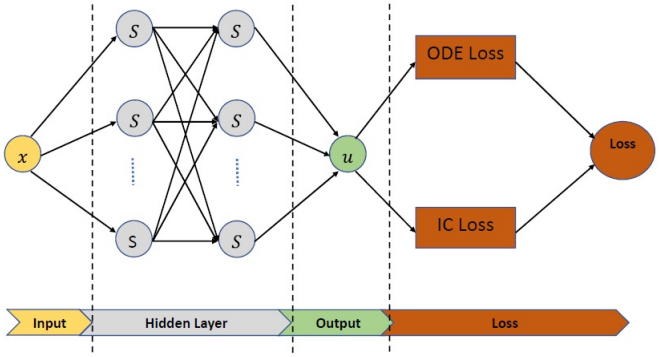


FIGURE 1. A schematic of a PINN for solving ODEs with initial condition (IC).

2. Physics-Informed Artificial Neural Networks

Generally, a deep neural network (DNN) for solving ODEs starts by considering the weights w_{ij} to form a linear combination with the bias b_i and the input X. This is mathematically represented as a specific compositional function. The fundamental type of neural network used for this purpose is the Feedforward Neural Network (FNN), also known as a perceptron. The FNN consists of multiple hidden layers that recursively apply non-linear and linear transformations to the inputs. Figure 1 provides a general schematic of a Physics-Informed Neural Network (PINN). However, prior to the development of the back-propagation algorithm, training a perceptron for improved pblueiction was challenging.

The main approach to solving ODEs using neural networks is to minimize the residual of the differential equations by reformulating the problem as an optimization problem. Over the past decades, various neural network architectures have been developed, including recurrent neural networks and convolutional neural networks.

In this research, we primarily focus on the use of the FNN, which is sufficient for implementing the residual neural network (ResNet) and effectively solving most ODE problems. The proposed method demonstrates ease in training deep networks. However, other neural network architectures can be easily incorporated into the problem-solving process.

Let $N^L(x)$ denote a mapping from $R^{d_{in}}$ to $R^{d_{out}}$, which represents an L-layer neural network. The lth layer of this network has N_l neurons, where $N_0 = d_{in}$ and $N_L = d_{out}$. The bias vector and weight matrix in the lth layer are represented by $b^l \in R^{N_l}$ and $p^l \in R^{N_l \times N_{l-1}}$, respectively.

In this paper, we utilize a MLP with the activation function S, which is applied element-wise. The recursively feed-forward neural network can be defined as follows:

$$\begin{aligned}
&\text{input layer:} && \mathcal{N}^0(x) = x \in R^{d_{in}} \\
&\text{hidden layers:} && \mathcal{N}^l(x) = S(p^l \mathcal{N}^{l-1}(x) + b^l) \in R^{N_l} \text{ for } 1 \le l \le L-1 \\
&\text{output layer:} && \mathcal{N}^L(x) = p^L \mathcal{N}^{L-1}(x) + b^L \in R^{d_{out}}.
\end{aligned}$$

An illustration of a neural network structure is shown in Figure 1. In multi-class classification problems, commonly used activation functions include hyperbolic tangent, logistic sigmoid, and ReLU functions. Additionally, the Leaky ReLU (Rectified Linear Unit) function can be utilized to address the "dying ReLU" problem by allowing a small slope for negative values. The choice of activation functions has a considerable impact on the performance and training of a neural network.

3. Numerical solution of ODEs using multilayer perceptron

Let the following ODE is given[3]:

$$F(x, y, y', ..., y^{(n)}) = 0. \tag{3.1}$$

Let $y_{\text{trial}}(x, P)$ be the obtained solution after training an ANN, where P represents the adjustable weights of the network. This solution can be formulated as [10]:

$$y_{\text{trial}}(x, p) = A(x) + g(x, N(x, P)), \tag{3.2}$$

where, $A(x)$ satisfies the initial or boundary conditions, $N(x, P)$ is the single output of the ANN, and P represents the biases and weights in the ANN. The function $g(x, N(x, P))$ in Equation (3.2) is a function of the feed-forward neural network output, which can be expressed as [3, 13]:

$$\begin{aligned} &N(x, P) = \sum_{i=1}^{N} P_i S(z_i) \quad \text{where } z_i = \sum_{j=1}^{m} \tilde{P}_{ij} x_j + b_i, \\ &\text{or} \\ &N(x, P) = \sum_{i=1}^{N} \sum_{j=1}^{m} P_i S(\tilde{P}_{ij} x_j + b_i), \end{aligned} \tag{3.3}$$

where, $P = \{P_i, \tilde{P}_{ij}\}_{i=1}^{m}$ represents the weights of the ANN, $b_i, i = 1 \cdots m$ represents the bias of the ANN, S represents the activation function of the hidden layer neurons, and $\tilde{P}_{ij} x_j + b_i$ represents the input value of the corresponding neuron. The adjustable weights P approximate the solution with an initial distribution, and by defining a loss function and minimizing it, better weights can be obtained. Therefore, the solution defined in Equation (3.2) is substituted into Equation (3.1), resulting in:

$$G(x, y_{\text{trial}}(x, p), y_{\text{trial}}(x, p)', ..., y^{(n)}) = R(x, P).$$

The loss function can be defined as:

$$\text{Loss} := \|R(x, P)\|, \tag{3.4}$$

where, $\| \, . \, \|$ represents a norm, such as the $\mathcal{L}_1$, $\mathcal{L}_2$, and etcetera. For example, if the $\mathcal{L}_2$ norm is used, the problem of solving the ODE in Equation (3.1) can be transformed into an optimization problem [3, 13]:

$$\min_{P} \|G(x, y_{\text{trial}}(x, p), y_{\text{trial}}(x, p)', ..., y^{(n)})\|_2. \tag{3.5}$$

Hence, the goal is to minimize the loss function in Equation (3.4).

3.1. Gradient Computation.

The efficient minimization of Equation (3.5) can be considerable as a procedure of training the neural network, where the error corresponding to each input vector x_i (i.e., the value $G(x_i)$ must be minimized. To compute this error value, both the derivatives of the output with respect to any of its inputs and the network output (as is the case in conventional training) are computed. Thus, to compute the derivative of the error with respect to the network weights, both the gradient of the network derivatives with respect to its inputs and outputs are taken into account.

In general, a MLPNN with n input nodes, one hidden layer with m nodes, and a linear output unit is considered. The extension can also be obtained in the case of multiple hidden layers. To train the ANN and compute the derivative of y_{trial} with respect to x and hyper-parameters in Equation (3.5), Equation (3.3) is used:

$$\frac{\mathrm{d}y_{\text{trial}}}{\mathrm{d}x} = \frac{\mathrm{d}A(x)}{\mathrm{d}x} + \frac{\mathrm{d}g(x, N(x, P))}{\mathrm{d}x}.$$

The k^{th} order derivative of $N(x,P)$ with respect to x is:

$$\frac{\mathrm{d}^k N}{\mathrm{d}x_j^k} = \sum_{i=1}^{m} P_i \tilde{P}_{ij}^k S_i^{(k)}, \tag{3.6}$$

where, $\vec{x} = (x_1, ..., x_n)$ is a given input vector, $S_i = S(z_i)$ and $S(z)$ denotes the sigmoid transfer function and $S^{(k)}$ denotes the k^{th} order derivative of the sigmoid transfer function. In implementation the method, equation

$$\frac{\mathrm{d}^{\lambda_1}}{\mathrm{d}x_1^{\lambda_1}} \frac{\mathrm{d}^{\lambda_2}}{\mathrm{d}x_2^{\lambda_2}} \cdots \frac{\mathrm{d}^{\lambda_n}}{\mathrm{d}x_n^{\lambda_n}} N = \sum_{i=1}^{n} P_i \bar{P}_i S_i^{(\Lambda)}, \tag{3.7}$$

is calculated from automatic derivation (AD) [10], where,

$$\bar{P}_i = \prod_{k=1}^{n} \tilde{P}_{ik}^{\lambda_k}, \quad \Lambda = \sum_{i=1}^{n} \lambda_i. \tag{3.8}$$

Equation (3.7) demonstrates that the derivative of the network with respect to any of its inputs is equivalent to a feed-forward neural network, denoted as $N_g(\vec{x})$, with one sigmoid hidden unit. This equivalence is achieved by using the same values for the thresholds b_i and weights $\tilde{P}_{ij}$ and replacing each weight P_i with $P_i\bar{P}_i$. Furthermore, the Λ^{th} order derivative of the sigmoid function is employed as the transfer function for each hidden layer. In the context of PINNs, this allows for updating the parameters of training. In order to obtain the derivative of N_g with respect to the parameters of the original network using back-propagation, one can easily follow the given formula.

$$\begin{aligned} \frac{\mathrm{d}N_g}{\mathrm{d}P_i} &= \bar{P}_i S_i^{(\Lambda)}, \\ \frac{\mathrm{d}N_g}{\mathrm{d}b_i} &= P_i \bar{P}_i S_i^{(\Lambda+1)}, \end{aligned} \tag{3.9}$$

$$\frac{\mathrm{d}N_g}{\mathrm{d}\tilde{P}_{ij}} = x_j P_i \bar{P}_i S_i^{(\Lambda+1)} + P_i \lambda_j \tilde{P}_{ij}^{\lambda_j - 1} \left(\prod_{k=1, k\neq j} \tilde{P}_{ik}^{\lambda_k} \right) S_i^{(\Lambda)}. \tag{3.10}$$

Once the error derivative with respect to the network parameters (weights and biases) has been obtained, the choice of minimization technique depends on various factors such as the complexity of the problem, the availability of computational resources, and the preference of the researcher. The back-propagation algorithm, specifically the steepest descent variant, is one of the most commonly used techniques for minimizing the error derivative with respect to the network parameters. It works by iteratively adjusting the weights and biases in the network based on the gradient information obtained through back-propagation. Ultimately, the choice of minimization technique depends on the specific problem and the characteristics of the neural network being trained. Researchers often experiment with different optimization algorithms to find the one that provides the best performance and convergence properties for their particular task.

To minimize errors, we utilized standardized machine learning optimization packages, namely TensorFlow [1] and PyTorch [18], along with several open-source projects [6, 17, 21, 23]. In all of our experiments, we specifically employed the LBFGS method, which has demonstrated remarkable performance in similar problem domains due to its quadratic convergence. It is important to highlight that when working with parallel hardware, the gradient of each network (or derivatives network) with respect to the parameters for a given grid point can be obtained simultaneously. Additionally, when dealing with Mini-Batch Processing, weight updates can be performed either in batch or online mode.

4. Illustration of the method

In our unique approach, we creatively pair the resolution of a complex differential Equation (3.1) with the adaptable nature of an ANNs. Our technique harnesses the power of the universal approximation theorem in selecting the optimal starting point for tackling ODEs through ANNs. By transforming the original model function into a streamlined

quadratic loss function, our method elegantly encapsulates two crucial factors: the precise representation of the differential equation's residual (initial/boundary conditions) via sophisticated unconstrained optimization strategies aimed at fine-tuning the neural network's parameters (weights and biases), and the stringent fulfillment of the differential Equation (3.1). To illustrate the method, we consider the following second-order ODE [13]:

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} = f(x, y, \frac{\mathrm{d}y}{\mathrm{d}x}), \quad x \in [a, b].$$

with initial value conditions: $y(a) = A$ and $\frac{\mathrm{d}}{\mathrm{d}x}y(a) = B$. The trial solution can be written in the following form:

$$y_{\text{trial}}(x, P) = A + B(x - a) + (x - a)^2 N(x, P_i), \tag{4.1}$$

for the boundary conditions: $y(a) = A$ and $y(b) = B$. The trial solution can be written in the following form:

$$y_{\text{trial}}(x, P) = B\frac{a - x}{a - b} + A\frac{b - x}{b - a} + (x - a)(x - b)N(x, P_i). \tag{4.2}$$

The error quantity for this problem is as:

$$E(x, P) = \sum_i \frac{1}{2}\left(\frac{\mathrm{d}^2y_{\text{trial}}(x_i, P)}{\mathrm{d}x^2} - f(x_i, y_{\text{trial}}(x_i, P), \frac{\mathrm{d}y_{\text{trial}}(x_i, P)}{\mathrm{d}x})\right)^2. \tag{4.3}$$

By utilizing optimization techniques, one can effectively reduce the negative impacts of the traditional Momentum-based Backpropagation (MBP) algorithm. However, the conventional method has its own set of limitations such as susceptibility to getting trapped in local minima, sluggish learning pace, and delayed convergence. Hence, to address these challenges, a novel approach known as the MBP algorithm is introduced in this work, showcasing remarkable efficacy. It is noteworthy that the MBP algorithm is equipped with supplementary parameters designed to accelerate the optimization and convergence rates of the network while simultaneously diminishing network errors.

5. Examples

In this part, we present our approach to solving a set of three ordinary differential equations starting from given initial conditions. Our method involves employing a neural network architecture consisting of a single hidden layer comprised of 6 units, ultimately yielding a linear output for each equation. The objective is to compare the solutions obtained by training the neural network using a specialized technique called algorithmic differentiation (specifically, the MBPNN algorithm) with the analytic solutions of the test problems. The accuracy of the obtained solutions is measured by calculating the deviation between the MBPNN approximation solution and the analytic solution. This deviation is denoted as $\Delta y(\vec{x}) = y_{\text{trial}}(\vec{x}) - y_a(\vec{x})$. Python is used for the simulations. the activation function for each hidden unit is the sigmoid function defined as $S(x) = \frac{1}{1+e^{-x}}$.

5.1. **Problem 1.** In this problem, a first order ODE is solved by proposed method. Consider

$$\frac{\mathrm{d}y}{\mathrm{d}x} = y - x^2 + 1,$$

with initial value $y(0) = 0.5$ and $x \in [0, 2]$. The exact solution of this problem is as follows:

$$y_a(x) = (x + 1)^2 - 0.5e^x,$$

and according to Equation (4.1) the trial solution can be written in the following form:

$$y_{\text{trial}}(x, P) = 0.5 + xN(x, P).$$

Then we have:

$$\frac{\mathrm{d}y_{\text{trial}}(x, P)}{\mathrm{d}x} = N(x, P) + x\frac{\mathrm{d}N(x, P)}{\mathrm{d}x}.$$

The error quantity for this problem is as follows:

$$E(x, P) = \sum_{j=1}^{x_{j-\text{number}}} \frac{1}{2}[N(x_j, P) + x_j\frac{\mathrm{d}N(x_j, P)}{\mathrm{d}x} - x_jN(x_j, P) + x_j^2 - 1.5]^2.$$

The network is trained in $[0, 2]$ using a grid of 10 equidistant points with 6 sigmoid hidden units. The above ODE problem were also solved with the other methods (as Cosine [11] and Power series [9]). We compare the obtained solution of the proposed method with the exact solution and the solutions of cosine and power methods. It is clear that the solution of MBPNN approximation has the highest accuracy in analogy solutions of the Cosine and Power methods, although training was performed using a small number of points. these analogy is shown in the following Table 1. Also. In Figure 2(a) presents exact solution versus the MBPNN approximation solution. Figure 2(b) illustrates the error function plot. Also, the comparison of the approximate solution obtained by the proposed method with the actual answer is given in Table 2.

TABLE 1. Comparison between MBPNN approximation solution, analytic solution and the solutions of [11] and [9] methods (Problem 1).

MBPNN tra. po.	MBPNN appr. sol.	Ana. sol.	Cosine sol.	Series sol.
0.0000	0.5000	0.5000	0.5000	0.5000
0.2040	0.8365	0.8366	0.8299	0.8290
0.4081	1.2308	1.2308	1.2137	1.2136
0.6122	1.6771	1.6770	1.6466	1.6483
0.8163	2.1680	2.1679	2.1257	2.1265
1.0204	2.6948	2.6948	2.6426	2.6400
1.2244	3.2470	3.2471	3.1755	3.1788
1.4285	3.8115	3.8115	3.7289	3.7311
1.6326	4.3722	4.3721	4.2780	4.2820
1.8367	4.9090	4.9090	4.8109	4.8150
2.0000	5.3054	5.3054	5.2865	5.3125

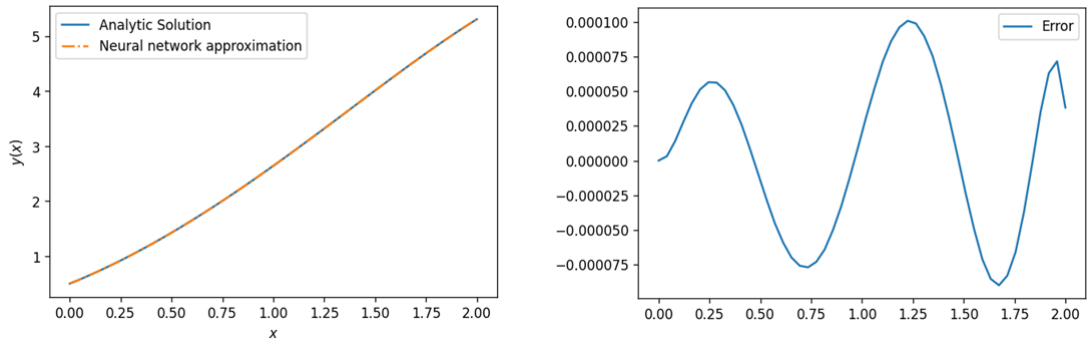
TABLE 2. Camparison of the proposed method and analytic solution at the selected test points (Problem 1).

Test points	MBPNN appr. sol.	Ana. sol.	Deviation between sol.
0.0408	0.5624	0.5624	0.0000
0.8571	2.2708	2.2707	+0.0001
1.1836	3.1351	3.1352	−0.0001
1.6734	4.4822	4.4821	+0.0001
1.8775	5.0115	5.0115	0.0000

This problem are also solved by Abdelhameed and Haweel [9] and Li-ying et al. [11]. The total deviation of the obtained solutions via power series and cosine methods at the 10 grid points is 0.0141 and 0.0432 respectively, whereas the total deviation of MBPNN approximation solution at the 10 grid points is $5.00e-9$.

It can be observed that in the Table 2. The obtained solutions in the neural method are excellent with interpolation accuracy. In order to illustrate the generalization performance (or generalization accuracy) of a selected test points we look at the approximation of its neighboring training points [18]. The extension performance of the selected testing points is poor when the deviation between solutions its neighboring training points is large; But is excellent when the deviation between solutions of its neighboring training points is close to zero. The training point 0.8163 of the network has a deviation of 0.0001 and so the deviation must be around 0.0001 in its neighboring selected testing points i.e. the selected testing point 0.8571 has a deviation of 0.0001 in network.

For example, the deviation of the selected training point 0.0408 of the network is zero because of the nearest testing point 0.0000 of the network has the zero deviation. According Figure 2(b), the maximum deviation of our method is 0.000100.



(a) (b)

FIGURE 2. (a) Plot of MBPNN solution and analytic solution. (b) Plot of error function or accuracy of the computed solution between MBPNN solution and analytic solution.

5.2. **Problem 2.** In this problem, we try to solve a second order ODE that is as follows:

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} = \frac{1}{2x^2}(y^3 - 2y^2),$$

with initial conditions $y(1) = 1$, $y'(1) = \frac{1}{2}$ and $x \in [1, 2]$. The analytic solution of this problem is as:

$$y_a(x) = \frac{2x}{x+1}.$$

According to Equation (4.1), the trial solution can be written in the following form:

$$y_{\text{trial}}(x) = 1 + \frac{1}{2}(x-1) + (x-1)^2 N(x, P).$$

Then, we have

$$\frac{\mathrm{d}^2 y_{\text{trial}}(x,P)}{\mathrm{d}x^2} = 2N(x,P) + 4(x-1)\frac{\mathrm{d}N(x,P)}{\mathrm{d}x} + (x-1)^2\frac{\mathrm{d}^2N(x,P)}{\mathrm{d}x^2}.$$

The error quantity for this problem is as:

$$\begin{aligned} E(x,P) \quad = & \sum_{j=1}^{j-\text{number}} \frac{1}{2}[2N(x_j,P) + 4(x_j-1)\frac{\mathrm{d}N(x_j,P)}{\mathrm{d}x} \\ & + (x_j-1)^2\frac{\mathrm{d}^2N(x_j,P)}{\mathrm{d}x^2} - \frac{1}{2x_j^2}((1 + \frac{1}{2}(x_j-1) + (x_j-1)^2N(x_j,P))^3 \\ & - 2(1 + \frac{1}{2}(x_j-1) + (x_j-1)^2N(x_j,P))^2)]^2, \end{aligned}$$

where, $m =$ Number of x_j. In this example ten equidistant points are used to train the network in interval $[1, 2]$ with 6 sigmoid hidden units. The comparison between the MBPNN approximation solution, the analytic solution and the LeNN [18] solution are given in Table 3. This table clearly shows the efficiency of our method.

Also, Figure 3 illustrates the comparison of the approximate solution and the exact along with the error diagram. Again, this figure shows the efficiency of our method.

TABLE 3. Comparison among MBPNN approximation solution, analytic solution and results of [18] (Problem 2).

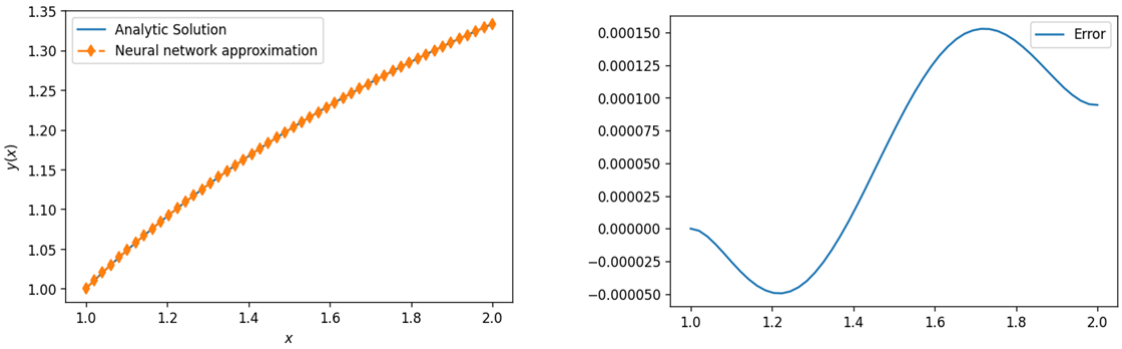
MBPNN tra. points	MBPNN appr. sol.	Ana. sol.	LeNN sol.
1.0000	1.0000	1.0000	1.0000
1.1020	1.0485	1.0485	1.0475
1.2040	1.0926	1.0925	1.0919
1.3061	1.1327	1.1327	1.1291
1.4081	1.1694	1.1694	1.1663
1.5102	1.2031	1.2032	1.2001
1.6122	1.2342	1.2343	1.2302
1.7142	1.2630	1.2631	1.2590
1.8163	1.2897	1.2898	1.2858
1.9183	1.3145	1.3146	1.3100
2.0000	1.3332	1.3333	1.3333

TABLE 4. Comparison of our results and analytic solution for selected testing points (Problem 2).

Testing points	MBPNN appr. sol.	Ana. sol.	Deviation of sol.
1.1224	1.0577	1.0576	0.0001
1.3265	1.1403	1.1403	0.0000
1.4285	1.1832	1.1833	−0.0001
1.6326	1.2401	1.2403	−0.0002
1.7346	1.2685	1.2686	−0.0001

The results for selected testing points are given in Table 4. This problem has been tested by Chackraverty solution and Mall using LeNN method [14]. The LeNN has changed the ANNs architecture that by canceling the hidden layer of the ANNs and changing it with Legendre polynomial expansion layer, he has blueuced the training parameters and tried increased the accuracy of the MBPNN solutions compablue to the traditional modified backpropagation [24]. However, It is clear that the approximate solution of LeNN has lower accuracy than our proposed method by comparing the deviation of solutions between MBPNN approximation solution and the analytic solution.The total deviation of solutions of the LeNN is 0.0042 whereas the total deviation of MBPNN approximation solution is $1.25e-7$. Mostly, in all methods, some parameters are broken and do not guarantee a good accuracy of the approximate solution of MBPNN.

The peak error of deviation of solutions in the MBPNN training points is shown in Figure 3(b) which is 0.000150. The selected testing point 1.3265 has a deviation of solution of zero because the deviation of solution of its neighboring training point in network (1.3061) is zero.



(a) (b)

FIGURE 3. (a) Plot of MBPNN approximation solution and analytic solution. (b) Error function plot between MBPNN approximation solution and analytic solution.

5.3. **Problem3.** In this problem, we would like to solve fourth order ODE that this equation is as follows:

$$\frac{\mathrm{d}^4 y}{\mathrm{d}x^4} = 120x,$$

with initial values $y(-1) = 1$, $y'(-1) = 5$, $y''(-1) = -20$, $y'''(-1) = 60$ and $x \in [-1, 1]$. The analytic solution of this problem is as:

$$y_a(x) = x^5 + 2.$$

From Equation (4.1), the trial solution can be written in the following form:

$$y_{\text{trial}}(x) = -2x^4 + 2x^3 + 4x^2 - x + (x+1)^4 N(x, P).$$

Then we have

$$\begin{aligned}\frac{\mathrm{d}^4 y_{\text{trial}}(x,P)}{\mathrm{d}x^4} &= 24N(x_j,P) + 96(x_j+1)\frac{\mathrm{d}N(x_j,P)}{\mathrm{d}x} \\ &+ 72(x_j+1)^2\frac{\mathrm{d}^2N(x_j,P)}{\mathrm{d}x^2} + 16(x_j+1)^3\frac{\mathrm{d}^3N(x_j,P)}{\mathrm{d}x^3} \\ &(x_j+1)^4\frac{\mathrm{d}^4N(x_j,P)}{\mathrm{d}x^4} - 48.\end{aligned}$$

The error quantity for this problem is as:

$$\begin{aligned}E(x,P) = \sum_{j=1}^{x_{j-\text{number}}} &\frac{1}{2}[24N(x_j,P) + 96(x_j+1)\frac{\mathrm{d}N(x_j,P)}{\mathrm{d}x} \\ &+ 72(x_j+1)^2\frac{\mathrm{d}^2N(x_j,P)}{\mathrm{d}x^2} + 16(x_j+1)^3\frac{\mathrm{d}^3N(x_j,P)}{\mathrm{d}x^3} \\ &(x_j+1)^4\frac{\mathrm{d}^4N(x_j,P)}{\mathrm{d}x^4} - 48 - 120x_j]^2.\end{aligned}$$

In this example the network is trained using a grid of ten equidistant points in $[-1, 1]$. We show the comparison between the MBPNN approximation solution and the analytic solution in the Table 5. Also, the exact solution versus our method companion of error function is given in Figure 4. This figure together with Table 2 shows the efficiency and powering of our method. The results for selected testing points are given in Table 6. We compare the deviation of the approximate solution of the MBPNN with the deviation of the exact solution, it is clear that the approximate solution of the MBPNN has higher accuracy. The maximum deviation of solutions in the training points is shown in Figure 4(b) which is 0.0008. The total deviation of solutions in the training points set is 0.000003125.

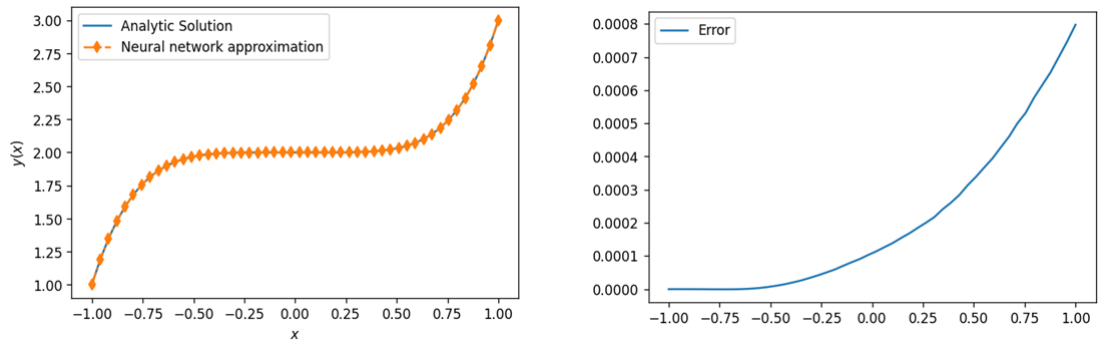
Table 5. Comparison between MBPNN approximation solution and analytic solution (Problem 3).

MBPNN training points	MBPNN appr. sol.	Ana. sol.
−1.0000	1.0000	1.0000
−0.7959	1.6805	1.6805
−0.5918	1.9273	1.9273
−0.3877	1.9912	1.9912
−0.1836	1.9997	1.9997
0.0204	1.9998	2.0000
0.2244	2.0003	2.0005
0.4285	2.0141	2.0144
0.6326	2.1009	2.1013
0.8367	2.4095	2.4101
1.0000	2.9992	3.0000

Table 6. Comparison of the our results and analytic solution for selected testing points (Problem 3).

Testing points	MBPNN appr. sol.	Ana. sol.	Deviation of sol.
−0.5510	1.9491	1.9492	−0.0001
−0.3469	1.9949	1.9949	0.0000
0.2653	2.0011	2.0013	−0.0002
0.4693	2.0224	2.0227	−0.0003
0.8775	2.5197	2.5204	−0.0007

As we seen the deviation of solutions in testing point −0.3469 is zero and is due to the zero deviation of its neighboring training point, −0.3877 in network.



(a) (b)

Figure 4. (a) Plot of MBPNN approximation solution and analytic solution. (b) error function plot between MBPNN approximation solution and analytic solution.

6. Conclusion

In this research, three ordinary differential equations were the subject of investigation using a novel approach leveraging a neural network solved by a tweaked backpropagation algorithm termed as MBPNN. Tailoring the neural

network models to the specific traits of each ODE, we meticulously constructed and experimentally scrutinized the said models. Noteworthy is the trial solution, deemed an approximate ODE solution derived from method simulations, obligingly satisfying all requisite conditions by default. Employing MBPNN facilitated enhanced learning by neural networks, compelling them to glean ODE solutions proficiently even with limited observations. Comparative analysis were performed contrasting the results of our method with other neural network methods and exact solutions. The obtained results clearly showed the effectiveness and power of our method. By applying the proposed method, any smooth ordinary differential equation with a companion to the given initial conditions can be solved with the desired accuracy. However, when the artificial neural network is used for solving differential equations that have solutions with extreme changes in some points of the solution range or have some singularity in the domain, the computational accuracy decreases. To overcome this problem, we will try to use appropriate learning methods with regression-based weights in future works.

References

[1] M. Abadi, (2015), Tensoflow.org. [Online]. Available: https://www.tensoflow.org/.

[2] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: a survey*, Journal of Marchine Learning Research, *18* (2018), 1-43.

[3] S. Chakraverty and S. Mall, (2017) *Artificial neural networks for engineers and scientists: solving ordinary differential equations*, CRC Press, doi.org/10.1201/9781315155265.

[4] M. Di Giovanni, D. Sondak, P. Protopapas, and M. Brambilla, *Finding multiple solutions of odes with neural networks*, In Combining Artificial Intelligence and Machine Learning with Physical Sciences, *2587* (2020), 1-7.

[5] R. Fletcher, (2013) *Practical methods of optimization*, John Wiley and Sons.

[6] A. H. Hadian-Rasanan, D. Rahmati, S. Gorgin, and K. Parand *A single layer fractional orthogonal neural network for solving various types of Lane-Emden equation* New Astronomy, *75* (2020), 101307.1- 101307.14.

[7] Z. Hajimohammadi, F. Baharifard, and K. Parand, *A new numerical learning approach to solve general Falkner–Skan model*, Engineering with Computers, *38* (2022), 121–137.

[8] J. Han and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Communications in mathematics and statistics, *5*(4) (2017), 349-380.

[9] T. I. Haweel, and T. N. Abdelhameed, *Power series neural network solution for ordinary differential equations with initial conditions*, In 2015 International Conference on Communications, Signal Processing, and their Applications (ICCSPA'15) IEEE., (2015), 1-5.

[10] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE transactions on neural networks, *9*(5) (1998), 987-1000.

[11] X. Li-ying, W. Hui, and Z. Zhe-zhao, *The algorithm of neural networks on the initial value problems in ordinary differential equations*, In 2007 2nd IEEE Conference on Industrial Electronics and Applications. IEEE., (2007), 813-816.

[12] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, SIAM review, *63*(1) (2021), 208-228.

[13] A. Malek and R. S. Beidokhti, *Numerical solution for high order differential equations using a hybrid neural network optimization method*, Applied Mathematics and Computation, *183*(1) (2006), 260-271.

[14] S. Mall and S. Chakraverty, *Application of Legendre neural network for solving ordinary differential equations*, Applied Soft Computing, *43* (2016), 347-356.

[15] K. Parand, A. A. Aghaei, S. Kiani, T. Ilkhas Zadeh, and Z. Khosravi *A neural network approach for solving nonlinear differential equations of Lane–Emden type*, Engineering with Computers, *40* (2024), 953-969.

[16] K. Parand, A. A. Aghaei, M. Jani, and A. Ghodsi, *Parallel LS-SVM for the numerical simulation of fractional Volterra's population model*, Alexandria Engineering Journal, *60*(6) (2021), 5637-5647.

[17] K. Parand, A. A. Aghaei, M. Jani, and A. Ghodsi, *A new approach to the numerical solution of Fredholm integral equations using least squares-support vector regression*, Math. Comput. Simul., *180* (2021), 114–128.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, Advances in neural information processing systems, *32* (2019).

[19] W. Peng, J. Zhang, W. Zhou, X. Zhao, W. Yao, and X. Chen, (2021). IDRLnet: *A physics-informed neural network library*, arXiv preprint arXiv:2107.04320.

[20] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational physics, *378* (2019), 686-707.

[21] Z. Sabir et al., *Novel design of Morlet wavelet neural network for solving second order Lane-Emden equation*, Math. Comput. Simul., *172* (2020) 1–14.

[22] A. Sacchetti, B. Bachmann, K. Löffel, U. M. Künzi, and B. Paoli, *Neural networks to solve partial differential equations: a comparison with finite elements*, IEEE Access, *10* (2022), 32271-32279.

[23] S. Susmita Mall and S. Chakraverty, *A novel Chebyshev neural network approach for solving singular arbitrary order Lane-Emden equation arising in astrophysics*, In: Network: Computation in Neural Systems, *31*(1-4) (2020), 142-165.

[24] L. S. Tan, Z. Zainuddin, and P. Ong, *Solving ordinary differential equations using neural networks*, In AIP Conference Proceedings, AIP Publishing LLC., *1974*(1) (2018), 020070.