An International Journal of Optimization and Control: Theories & Applications
ISSN:2146-0957 eISSN:2146-5703
Vol.14, No.3, pp.249-260 (2024)
http://doi.org/10.11121/ijocta.1519

RESEARCH ARTICLE

Design optimal neural network based on new LM training algorithm for solving 3D - PDEs

Farah F. Ghazi [a], Luma N. M. Tawfiq [a*]

[a] *Department of Mathematics, College of Education for Pure Science, Ibn Al-Haitham, University of Baghdad, Baghdad, Iraq*
farah.f.g@ihcoedu.uobaghdad.edu.iq, luma.n.m@ihcoedu.uobaghdad.edu.iq

ARTICLE INFO

Article History:
Received 4 January 2024
Accepted 28 April 2024
Available Online 19 July 2024

Keywords:
PDEs
Neural networks
BP-training algorithm
Unconstrained optimization
LM training algorithm
Convergence analysis

AMS Classification 2010:
78M32; 78M50; 68U99

ABSTRACT

In this article, we design an optimal neural network based on new LM training algorithm. The traditional algorithm of LM required high memory, storage and computational overhead because of it required the updated of Hessian approximations in each iteration. The suggested design implemented to converts the original problem into a minimization problem using feed forward type to solve non-linear 3D - PDEs. Also, optimal design is obtained by computing the parameters of learning with highly precise. Examples are provided to portray the efficiency and applicability of this technique. Comparisons with other designs are also conducted to demonstrate the accuracy of the proposed design.

1. Introduction

Partial differential equations (PDEs) based mathematical models can be used to describe a wide variety of physical issues. The PDEs govern a wide range of physical, chemical, and biological events [1, 2]. A mathematical model is a condensed, mathematically stated depiction of physical reality. Nonlinear PDEs are also crucial for study in a wide range of domains, including hydrodynamics, engineering, quantum field theory, optics, plasma physics, etc [3–5]. Since they frequently do not have exact solutions, numerical techniques are used to approximate them.

In addition, many researchers have been solve nonlinear PDEs by using homotopy analysis method (HAM) [6], Homotopy perturbation method (HPM) [7, 8], Variational Iteration method (VIM) [9], and Adomain decomposition methods (ADM) [10–17]. Moreover, a number of methods, including numerical approach used to solve different type of PDEs for more details see [18–22], iterations, differential, and Laplace transformation approaches, have been utilized to numerically and analytically solve comparable types of the wave-like and also heat-like problems. It is important to use a suitable method for solving any equation or problem. In recent years some authors used neural networks as an important method to solve many of real-world problems because of their specification. Some authors used ANNs for solving different types of differential equations such that Oraibi et. al. [23] first gave the concept of solving ordinary differential equations using a neural network by formulating a trial solution of the differential equation. The authors tested the applicability and accuracy of their developed method not only for ordinary differential equations but also for systems of coupled differential equations. Further, the authors compared their results with the results obtained by using other numerical methods, and reported

*Corresponding Author

that developed ANN method is superior in terms of memory requirements and accuracy.

Several attempts have been made to solve different types of differential equations using feed-forward neural networks. Hussein and Mohammed [24] reported a hybrid method by combining optimization techniques with neural networks to solve high-order ordinary differential equations. In a related work, Tawfiq and Hussein [25] introduced a novel method for solving boundary value problems using artificial neural networks. They also implemented the method for irregular domain boundaries with Dirichlet as well as Neumann boundary conditions and used for processing face recognition. Tawfiq [26] solved initial and boundary value problems using a single-layer finite element neural network and investigated the accuracy of the method for nonlinear forward and inverse problem, and also for a system of ordinary differential equations. Salih and Tawfiq [27] presented a functionally weighted neural network (FWNN) a new class of artificial neural networks incorporating an infinite number of nodes and showed that their new network has superior extrapolation capability over other networks then used to solve Troesch's problem. Hussien et. al. [28] proposed an artificial neural networks-based deep neural network and dropout to solve time dependent differential equations. The authors showed that artificial neural network-based deep is very well approximating dynamic systems represented by time-dependent differential equations. Ali and Tawfiq [29] in their paper used artificial neural networks to approximate the solution of unsteady state confined aquifer problem. The authors used linear and non-linear terms in different types of unsteady sate differential equations to illustrate the accuracy of the method. Ali et. al. [30] proposed feed forward neural network design for solving nonlinear second order, eigenvalue problem for partial differential equation. They presented example to show speed, accuracy and effectiveness of applying neural network technique and found their results more precise than other numerical methods. The proposed neural network based on new modification of BFGS update algorithm. Gupta and Batra [31] developed a vectorized algorithm and impleme-nted it in Python code using a deep artificial neural network to solve the system of ordinary differential equations. Further, to show the effectiveness of the proposed method he compared his results with the fourth-order Runge-Kutta method and showed the high accuracy of his proposed method. Hussien and Dhannoon [28] presented a meshless parameter estimation method for solving a system of partial differential equations using an artificial neural network. The authors demonstrated that the deep learning ANN-based approach is very effective in solving differential equations in reasonable computing times. They illustrated their method for linear and non-linear partial differential equations with Dirichlet and Neumann boundary conditions for both regular and irregular boundaries. Khamas et. al. [32] design suitable neural network to solve singular initial and boundary value problems. The proposed design used to determine the effect hookah smoking on health with different types of tobacco. Tawfiq et. al. [33] in their paper discussed pitfalls for solving differential equations with neural networks. They considered examples and counter-examples for numerical tests to substantiate their findings. ANNs have a lot of advantages including high learning ability, adaptiveness, parallel processing, fault-tolerance, error computation, and machine training making this method the preferred choice to solve ordinary, partial and singular differential equations with initial or/and boundary conditions [34]. The researchers used different design of ANNs depending on type of problems; number of given data or samples. While, the ANN reliability has been assessed in this research. The new approach of training based on the LM training algorithm has been proposed. The objective function for this research include the minimizing.

This article has been consisting as follows: In next section, define and gives a background of the ANNs. In section 3, LM training algorithm is presented. In section 4, modification for LM training algorithm will be given. In section 5, 3D equation Linear & non PDE presented then we design optimal ANN for solving this equation with implementation and discussions for the result will be given. Finally, the conclusions are given in section 6.

2. Neural networks

A neural network is a structure of parallel processing for distributing information in the form of connected layers consist of a set of nodes called neurons (also are called processing elements) is the basic processor in ANNs, along with directed line segments between them called links (also are called connections). All nodes can be taken any number of arrival connections and can have any number of coming out connections, but the signs must be the same [31]. In effect, all nodes have a one coming out connection that can branch out to form multiple output connections, each of which carries the same sign. Each node possesses a transfer (activation) function which can use input signs, and which produces the node's output sign. Generally, ANNs have been generalizations of mathematical models of human brain, based on the processing of information occurs at many connections nodes; signs are passed between nodes over connection links which has an associated weight; each node applies an transfer function to its weighted input net to determine its sign of output.

Thus for a given input vector x, the input to this neuron is $W_j^T x$. We assume that each of the hidden neurons has identical transfer function σ, but that bias bj. So the output from the j-th hidden neuron is $\sigma(W_j^T x + b_j)$.

Now we denote the weight connecting the j^{th} hidden node to the output by $\mho_j$. The output function $g(x)$ of the ANN is therefore [35]:

$$g\left(x\right) = \sum_{j=1}^{k} \mho j \sigma(W_j^T x + bj) \tag{1}$$

Note that σ must be sigmoidal functions, so we choice suitable σ herein defined as [32]:

$$\sigma\left(n_i\right) = \frac{2}{e^{-2ni} + 1} - 1 \tag{2}$$

Then, the ANN input-output equation is:

$$\hat{Y} = \Phi(x^T W^T + b^T)\mho^T$$

where $W \epsilon R^{n\times r}; \mho \epsilon R^{1\times n}$ and $b \epsilon R^{n\times 1}$ are the adjustable input weights, output weights and bias respectively.

The structure of interconnections ANN can be classified to different classes of ANNs architecture such feed forward neural network (FFNN): organized of nodes are in the form of layers and arrival input from the previous layer then feed their output to the next layer, in a strictly the data goes from the input node to the output node as feed-forward way i.e., forward loops. Feedback neural network (FBNN): all possible connections are allowed between layers and neurons. The data transfer in the network as back loops. Herein we choose FFNN.

3. LM Training algorithm

Here's a simplified mathematical breakdown of the "trainlm" algorithm:

(1) Initialization:
  - Initialize weights (W) and biases (b) randomly.
  - Set the learning rate ($\eta = 0.001$) for the Levenberg-Marquardt algorithm.

(2) Forward Propagation:
  - For each input sample xi:
  - Calculate the weighted sum and apply the activation function for each neuron in the hidden layer:

$$a_{ij} = \sum_{k=1}^{n} w_{ijk} x_{ik} + b_{ij} \tag{3}$$

$$u_{ij} = \sigma(a_{ij})$$

  - Propagate the activations to the output layer using a similar process:

$$a_{ik} = \sum_{j=1}^{m} w_{ijk} a_{ij} + b_{ik} \tag{4}$$

$$u_{ik} = \sigma(a_{ik})$$

(3) Calculate Error:
Compute the error (E_i) between predicted (neural) output (u_{ik}) and target (exact) output $(\dot{u}_{ik})$

$$E_i = \frac{1}{2} \sum_{k=1}^{K} \left(\dot{u}_{ik} - u_{ik}\right)^2 \tag{5}$$

(4) Backpropagation:
  - Compute the gradient of the error with respect to weights and biases in the output layer:

$$g_{ik} = -(\dot{u}_{ik} - u_{ik})\ \acute{\sigma}(a_{ik}) \tag{6}$$

$$\frac{\partial E_i}{\partial w_{ijk}} = g_{ik} a_{ij}$$

$$\frac{\partial E_i}{\partial b_{ik}} = g_{ik}$$

  - Propagate the error gradient back to the hidden layer and compute gradients there

$$g_{ij} = \acute{\sigma}(a_{ij}) \sum_{k=1}^{K} w_{ijk} g_{ik} \tag{7}$$

$$\frac{\partial E_i}{\partial w_{ijk}} = g_{ij} x_{ik}$$

$$\frac{\partial E_i}{\partial b_{ij}} = g_{ij}$$

(5) Update Weights and Biases Using Levenberg-Marquardt:
  - The Update the weights and biases using the Levenberg-Marquardt update rule:

$$w_{ijk}^{(t+1)} = w_{ijk}^{(t)} - \eta\ \rho$$

$$w_{ijk}^{(t+1)} = w_{ijk}^{(t)} - \left(J^T J + \lambda I\right)^{-1} J_k^T \; e \qquad (8)$$

$$b_{ijk}^{(t+1)} = b_{ijk}^{(t)} - \eta \; \rho \qquad (9)$$

Where, ρ is search direction.

(6) Repeat:
- Iterate through the dataset multiple times, adjusting weights and biases after each iteration.
- Stop when the error converges or a predefined number of iterations is reached.

Based on its speed, the algorithm seems to be the most efficient way to train feedforward neural networks of moderate size (with up to several hundred weights). Additionally, it has a streamlined implementation in MATLAB software, as the matrix equation solution is built-in. These attributes make it particularly effective in a MATLAB environment [28].

4. Suggested modification for LM training algorithm

In this section we will present suggested modified for LM training algorithm denoted by MLM as follow:

Algorithm 1.

Step 1: *Given point $x_0 \in R^n$ and constants d_0, d_1, d_2, μ_0 and m such that $\mu_0 > m > 0$; $0 < d_0 < d_1 < d_2 < 1.\sigma \in (0, 2], \theta \in [0, 1]$ Let $k = 0$.*

Step 2: *If $\left\|J_k^T \; E_k\right\| < \epsilon$, then stop. otherwise Solve*

$$\lambda_k = \mu_k \left(\frac{\theta \left\|E_k\right\|^\sigma}{1 + \left\|E_k\right\|^\sigma}\right) + \frac{(1-\theta)\left\|J_k^T E_k\right\|^\sigma}{1 + \left\|J_k^T E_k\right\|^\sigma} \quad (10)$$

Step 2. Compute the search direction p_k

$$p_k = \left(J_k^T \; J_k + \lambda_k I\right)^{-1} J_k^T \; E_k. \qquad (11)$$

Step 3: *Calculate $r_k = {}^{Ared_k}/{}_{Pred_k}$, where $Ared_k$ is an actual reduction which equal to:*

$$Ared_k = \left\|E_k\right\|^2 - \left\|E\left(x_k + p_k\right)\right\|^2 \qquad (12)$$

and $Pred_k$ is a predicted reduction which equal to:

$$Pred_k = \left\|E_k\right\|^2 - \left\|E_k + J_k p_k\right\|^2 \qquad (13)$$

set

$$x_{k+1} = \begin{cases} x_k + p_k & if \; r_k \geq d_0 \\ x_k & otherwise \end{cases}$$

Step 4: *Choose μ_{k+1} as*

$$\mu_{k+1} = \begin{cases} 4\mu_k & if \; r_k < d_1 \\ \mu_k & if \; r_k \in [d_1, d_2] \\ \max\left\{\frac{\mu_k}{4}, m\right\} & if r_k > d_2 \end{cases}$$

Step 5: *Take $k := k + 1$ and go to Step 2.*

5. Design optimal ANN to solve 3D-differential equations

In this section we suggest optimal design ANN to solve 3D- PDEs. The optimum based on suitable choice of number of neurons in the hidden layer depending on trial and error. That is design ANN requires fully interconnection three layers; 1st layer is input layer consist 4 neurons in the input layer $(x, y, z\&t)$; 3rd layer is output layer consist one neuron with linsig. transfer function which represents the solution of the network and 2nd layer is hidden layer with tanhsig. transfer function consist 9 neurons in 1st trial then 10 neurons in 2nd trial then 13 neurons in 3rd trial and 15 neurons in 4th trial. So, we comparing between the number of neurons in hidden layer in the training ANN, for solving non- linear PDE we see that in case solving the linear equation when the number of neurans large (15 neurons) that make a good design for ANN to solve it according to time 00:00:08 with performance 4.7370e-07 and best epoch 726, see Figures 2, 3, 4 and 5. But Figure 1, illustrat the implementation and accuracy of suggested design in different values of time t. Whereas, in nonlinear equation the lower number of neurons (9 nodes) in the hidden layer give the better value according to time 00:00:00 with performance of the network solution $u_{net}\left(x, y, z, t; \theta\right)$ is **8.7805e-30** and best epoch 8, see Figures 7, 8, 9 and 10. The preformance of the network solution $u_{net}\left(x, y, z, t; \theta\right)$ is **8.7470e-10** which is best from archticher of ANN with one hidden layer. But Figure 6, illustrat the implementation and accuracy of suggested design in different values of time t. While in the case solving nonlinear equation take long time 00:02:21 in ANN consist 9 nodes in 1st hidden layers and 3 nodes in 2nd hidden layers in 1000 epoch and the value of preformance is 1.8910e-11. However, this value is not good when comparing with one hidden layer network. In other words the best archticher is one hidden layer ANN with 9 nodes in hidden layer since it is sufficient to give good result for solving nonlinear equation.

Training suggested ANN by back propagation rule and using unconstrain optimazation methods new LM algorithm. For every input data x, y, z and t, the process from input layer to the hidden layer described as follows:

$$n_i = \sum_{i=1}^{9} \left(\; W_{x_i} x + W_{y_i} y + W_{z_i} z + W_{t_i} t\right) + b_1$$

where W_{x_i}, Wy_i, W_{z_i} and Wt_i are the weights interrelate of the inputs x, y, z and t to the hidden layer respectively, and b1 is the biases of hidden layer. Hence, it is activated by the log. sig. function as Eq.(2). The next step is the process of the interrelate of the hidden layer to the output layer which is based on the following formula:

$$h_i = \sum_{j=1}^{9} \mho_{ij}\sigma(n_i) + b_2 \tag{14}$$

where $\mho_{ij}$ are the weights of the hidden layer with output, and b2 is the biases. When Eq.(6) became to output layer, it turned into the form

$$u_{net}(x, y, z, t; \theta) = \sum_{j=1}^{9} \mho_i \sigma(h_i)$$

where $\mho_j$ are the weights of the hidden layers to the output layers.

Then, it is also easy to express the k-th derivatives of $u_{net}(x, y, z, t; \theta)$ in terms:

$$\frac{\partial^k u_{net}(x, y, z, t; \theta)}{\partial x^k} = \sum_{j=1}^{n} \frac{\partial^k \mho_j f(h_2)}{\partial x^k} \tag{15}$$

$$\frac{\partial^k u_{net}(x, y, z, t; \theta)}{\partial y^k} = \sum_{j=1}^{n} \frac{\partial^k \mho_j f(h_2)}{\partial y^k} \tag{16}$$

$$\frac{\partial^k u_{net}(x, y, z, t; \theta)}{\partial z^k} = \sum_{j=1}^{n} \frac{\partial^k \mho_j f(h_2)}{\partial z^k}$$

$$\frac{\partial^k u_{net}(x, y, z, t; \theta)}{\partial t^k} = \sum_{j=1}^{n} \frac{\partial^k \mho_j f(h_2)}{\partial t^k} \tag{17}$$

For $k = 1, \ldots, n$.

The mean square error (mse) will be computed to check the accuracy of the approximate solutions that obtained in these cases for different values of the epochs. Moreover, illustrates the target of output in each case and the behavior of gradient in the validation case at epoch 1000. Target values of training is 70, validation 15 and testing 15. The learning rate $(\eta) = 0.001$.

Example 1. *Consider the 2^{nd} order, 3D linear homogeneous hyperbolic PDE :*

$$u(x, y, z, t) = u_{xx} + u_{yy} + u_{zz} + u_t \ \textit{for}\ 0 < x, y \ \textit{and}\ z < 1$$

IC: $u(x, y, z, 0) = \sin(\pi x)\sin(\pi y)\sin(\pi z)$
BCs:$u(0, y, z, t) = 0, u(1, y, z, t) = 0, u(x, 0, z, t) = 0, u(x, 1, z, t) = 0,\ u(x, y, 0, t) = 0, u(x, y, 1, t) = 0,\ u(x, y, 0, t) = 0,\ u(x, y, 1, t) = 0$
The exact solution [19] is $u(x, y, z, t) = \sin(\pi x)\sin(\pi y)\sin(\pi z)e^{xyzt}$.

We solve this equation by suggested design of ANN and implemented in MATLAB vol. 2023a, after training suggested ANN we see below the result of the equation at different time in Figures 1-7 and the value of neural network Table 1 with using sigmoidal functions as in eq.2 between the first and the hidden layer while between the hidden and last layer purlin function. Figure 8 show the performances of ANN, Figures 9-12 explain the performances of test, validation & training, Figure 13 show the valued of gradient, Mu & validation, finally in Figure 14 explain the errors between exact & suggested solution.

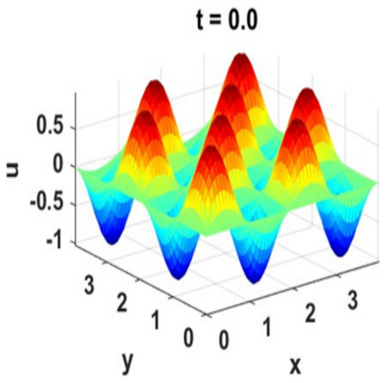

Figure 1. Results of suggested design for zero time of Example 1.

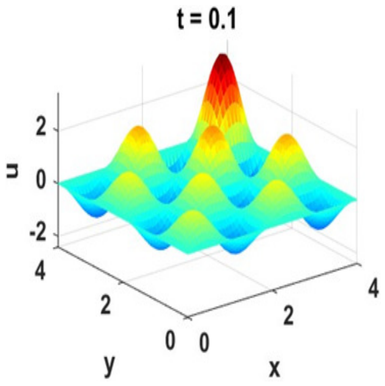

Figure 2. Results of suggested design for time 0.1 of Example 1.

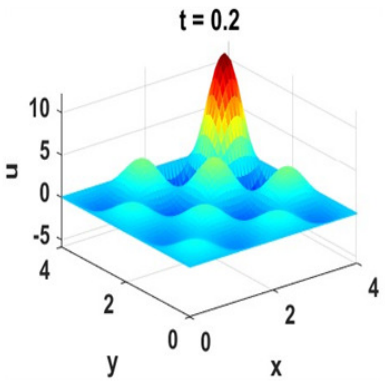

Figure 3. Results of suggested design for Example 1when time t= 0.2.

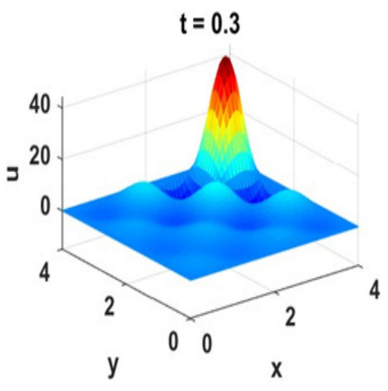

Figure 4. Results of suggested design when time t= 0.3 for Example 1.

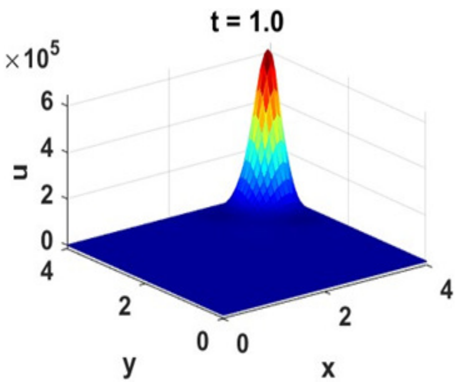

Figure 7. Results of suggested design when time t= 1 for Example 1.

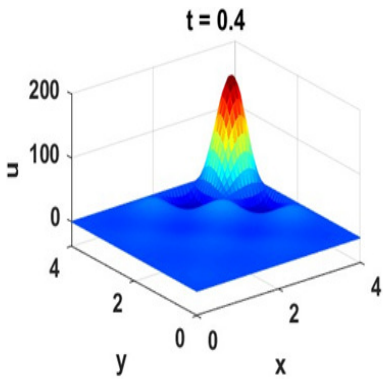

Figure 5. Results of suggested design when time t= 0.4 for Example 1.

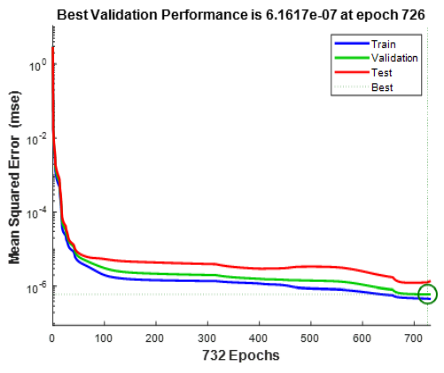

Figure 8. Comparison of Performances of ANN for Example 1, between train, test & validation in case 15 neurons in hidden layer.

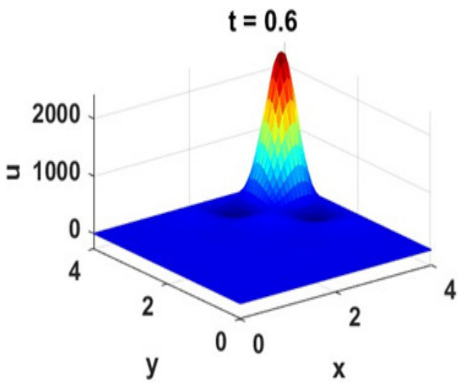

Figure 6. Results of suggested design when time t= 0.6 for Example 1.

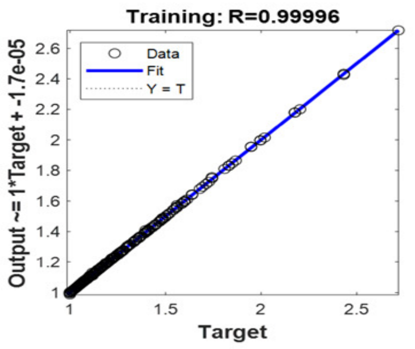

Figure 9. Performances of training for Example 1, in case 15 neurons in hidden layer.

Table 1. Results of suggested design in different cases for Example 1

No.	Layer & Nodes	Best epoch	Time	Best-perf.	Best-Vperf.	Best-tperf.	Gradient	lr.
1	9	1000	00:00:08	7.5071e-06	7.4696e-06	8.5628e-06	0.000104	0.001
2	10	538	00:00:06	6.6823e-06	6.7689e-06	6.7437e-06	0.000121	0.001
3	13	314	00:00:05	1.5217e-06	1.5500e-06	1.2775e-06	0.000241	0.001
4	15	726	00:00:08	4.7370e-07	6.1617e-07	1.2674e-06	0.00012	0.001
5	[9 3]	1000	00:00:11	4.7538e-08	5.2877e-08	4.8976e-08	0.00031	0.001
6	[9 9]	1000	00:00:17	6.4486e-09	6.4733e-09	6.3662e-09	1.98e-06	0.001
7	[9 19]	1000	00:00:33	8.7470e-10	9.0352e-10	1.2115e-09	9.35e-06	0.001

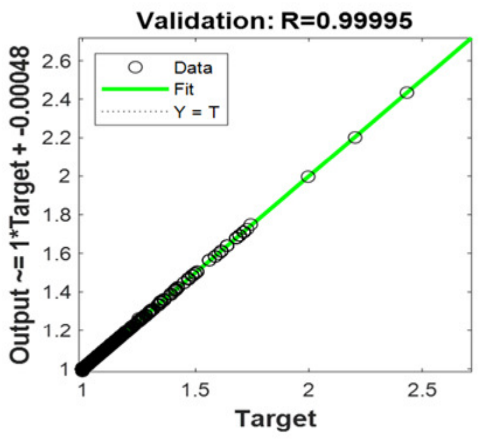

Figure 10. Performances of validation for Example 1, in case 15 neurons in hidden layer.

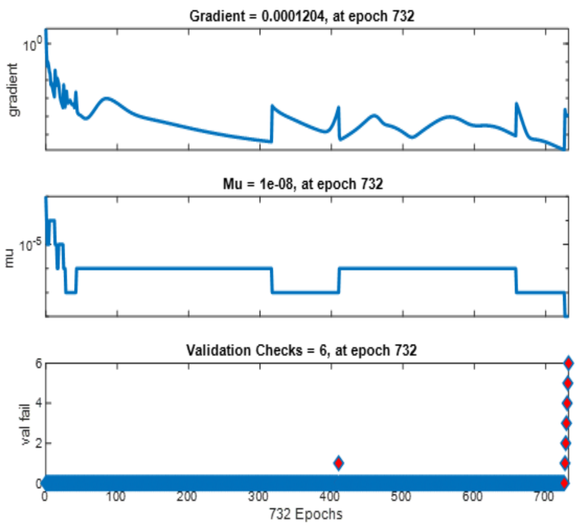

Figure 13. Gradient, Mu & validation for Example 1, in case 15 neurons in hidden layer.

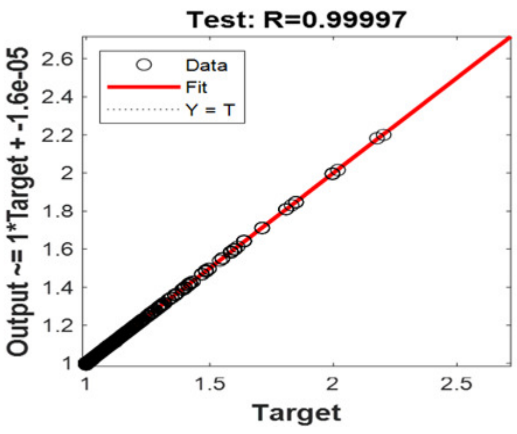

Figure 11. Performances of test for Example 1, in case 15 neurons in hidden layer.

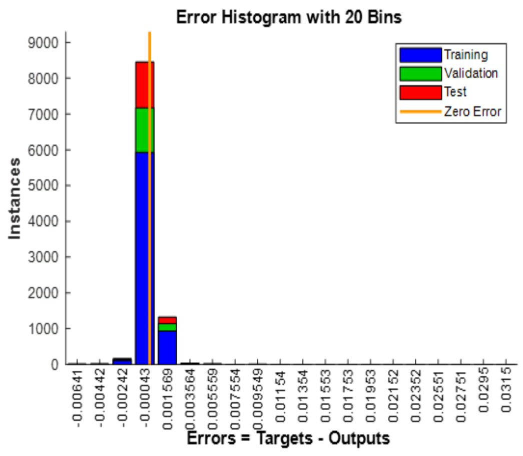

Figure 14. Errors between exact & suggested solution for Example 1, in case 15 neurons in hidden layer .

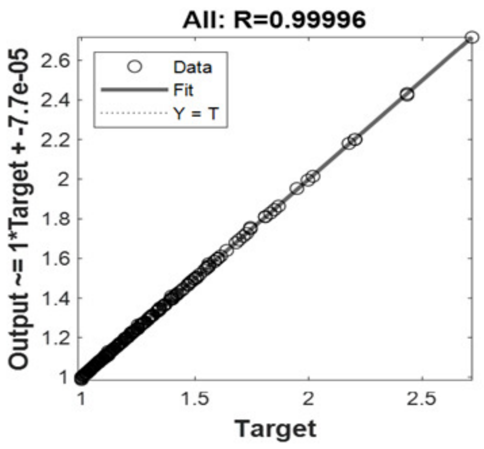

Figure 12. Comparison between exact & ANN results for Example 1, in case 15 neurons in hidden layer .

Example 2. *Consider the following 4th order 3D nonlinear Jimbo-miwa equation*

$$u_{xxxy} + 3u_{xy}u_x + 3u_yu_{xx} + 2u_{yt} - 3u_{xz} = 0$$

With ICs: $u_y(x, y, z, 0) = \frac{9}{2}\ sech^2\left(\frac{3}{2}(x + y + z)\right)$
Exact solution in [33, 34]: $u(x, y, z, t) = 3\tanh\left(\frac{3}{2}(x + y + z - 3t)\right)$

We solve that equation by suggested ANN and implemented in MATLAB vol. 2023a suggested design consist three layers: 1st layer (input layer) consist of 4 nodes represent $\{x, y, z\ \&\ t\}$. In the hidden layer, we take the different case depending on number of neurons and 3rd layer (output layer) gives the solution of the network. Other design illustrate the results for different values of

Table 2. The value of parameters for suggested ANN in different cases for Example 2.

No.	Layer & Nodes	Best epoch	Time	Best-perf.	Best-Vperf.	Best-tperf.	Gradient	lr.
1	9	8	00:00:00	8.7805e-30	8.6926e-30	8.8637e-30	5.24e-15	0.001
2	10	11	00:00:09	3.2411e-31	3.2205e-31	3.2382e-31	1.41e-15	0.001
3	13	8	00:00:14	7.4300e-28	7.4141e-28	7.3858e-28	5.59e-14	0.001
4	15	8	00:00:22	1.0665e-23	1.0648e-23	1.0641e-23	3.06e-11	0.001
5	[9 3]	1000	00:02:21	1.8910e-11	1.8696e-11	1.8787e-11	7.22e-07	0.001
6	[9 9]	1000	00:03:35	3.863 6e-09	3.8756e-09	3.8627e-09	2.98e-06	0.001
7	[9 19]	1000	00:06:03	3.3635e-09	3.3775e-09	3.3723e-09	1.84e-06	0.001

time see Figures 15-21 and the value of parameters given in Table 2. Figure 22 shows the performances of ANN. Figures 23-26 illustrate the performances of training, test & validation case. Figure 27 illustrate the value of gradient, Mu & validation, finally in Figure 28, the errors between exact and neural solution in each case are presented.

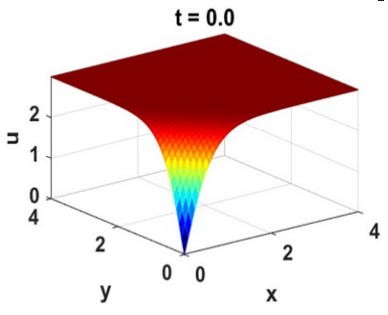

Figure 15. Results of suggested design for zero time of Example 2.

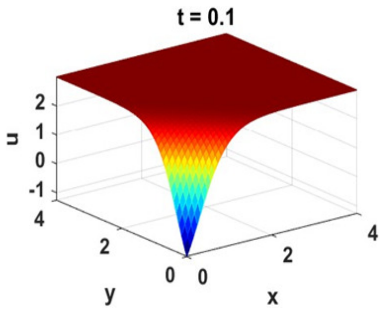

Figure 16. Results of suggested design when time t = 0.1 for Example 2.

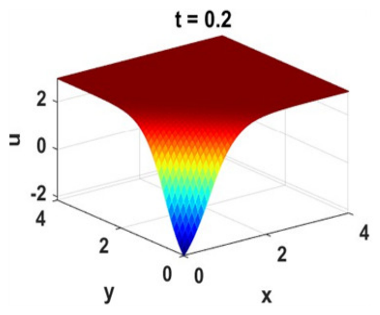

Figure 17. Results of suggested design when time t= 0.2 of Example 2.

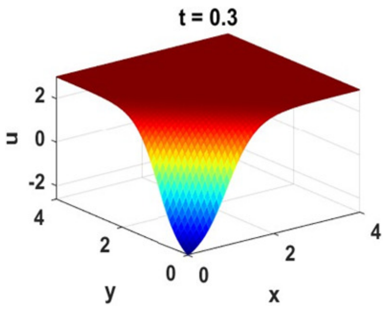

Figure 18. Results of suggested design when time t= 0.3 for Example 2.

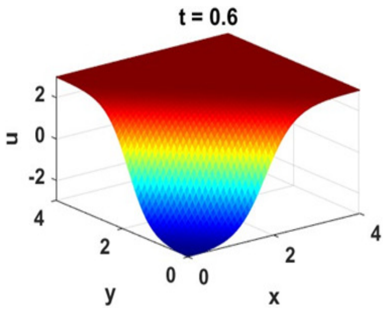

Figure 19. Results of suggested design when time t= 0.6 for Example 2.

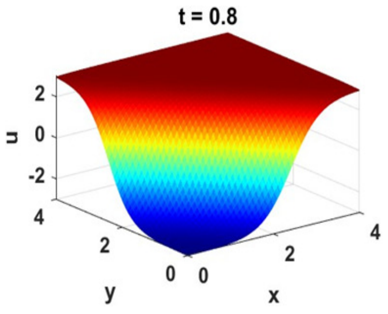

Figure 20. Results of suggested design when time t= 0.8 for Example 2.

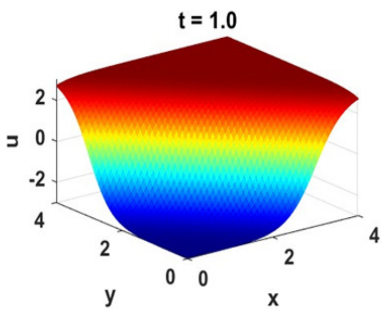

Figure 21. Results of suggested design when time t= 1 for Example 2.

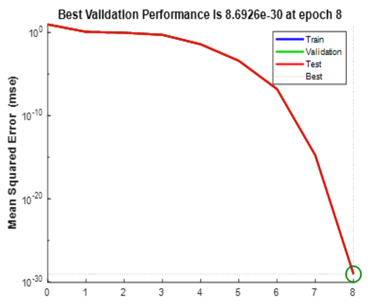

Figure 22. Performances of ANN for Example 2, in the case of 9 neurons in hidden layer.

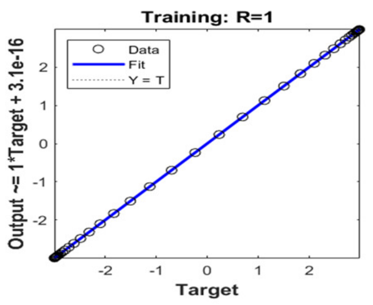

Figure 23. Performances of training ANN for Example 2, in the case 9 neurons in hidden layer.

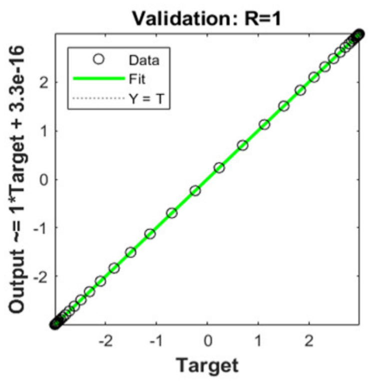

Figure 24. Performances of validation for Example 2, in the case 9 neurons in hidden layer.

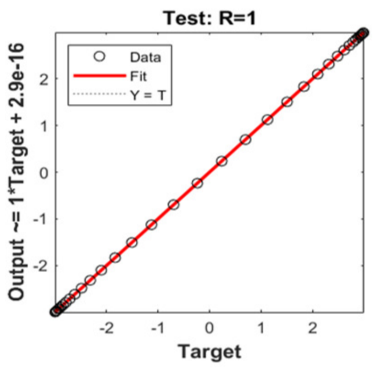

Figure 25. Performances of test for Example 2, in the case 9 neurons in hidden layer.

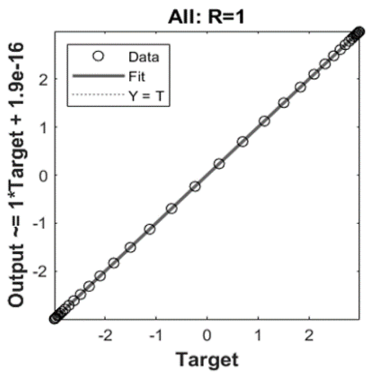

Figure 26. Comparison between exact & ANN result for 2, in the case 9 neurons in hidden layer.

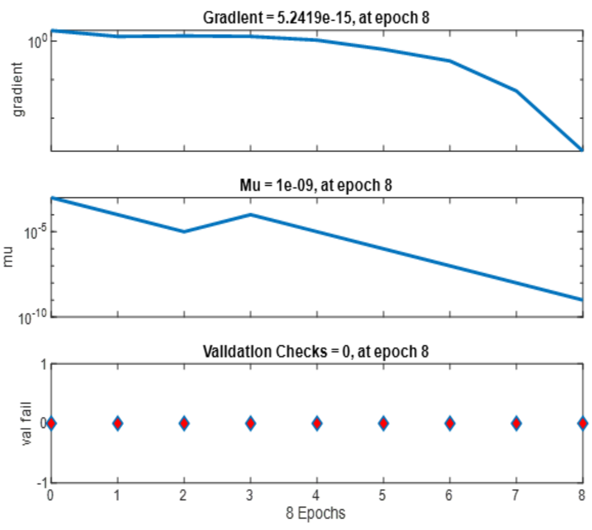

Figure 27. Gradient, Mu & validation for Example 2, in case 9 neurons in hidden layer.

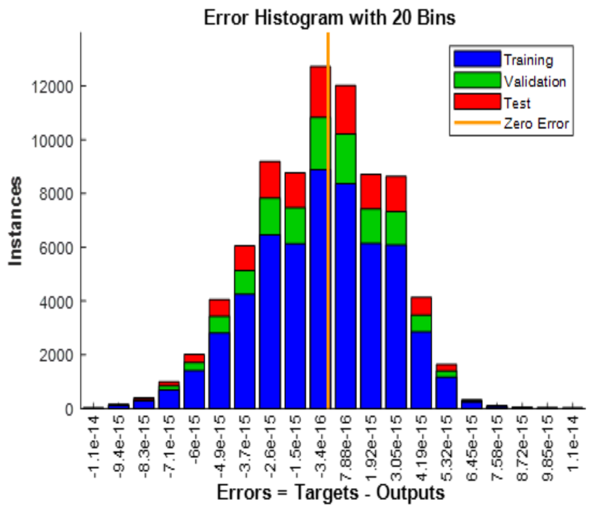

Figure 28. Errors between exact & neural solution for Example 2, in the case 9 neurons in hidden layer.

6. Conclusion

In this article, we suggest ANNs with different architecture based on number of layers and number of nodes in each layers. Suggested design trained by unconstrained optimization especially new LM training algorithm then used to solve 3D linear and nonlinear differential equations. The comparison between different design depending on the number of nodes in hidden layer has been presented. We see that in Example 1 (linear case) when the number of nodes large (15 neurons) we get good results and represent optimal design for ANN to solve this type of equations according to the time 00:00:08 with performance 4.7370e-07 and best epoch 726, whereas in Example 2, (nonlinear case) we see that the lower number of neurons (9 nodes) in the hidden layer gives the better results according to the time 00:00:00 with performance of the network solution $u_{net}(x, y, z, t; \theta)$ is 8.7805e-30 and best epoch 8. Also in the case of two hidden layers, the best archticher of ANN that gives good result in linear case is as [9 19] nodes, the best epoch is 1000 with long time 00:00:33 when comparing with one hidden layer but the preformance of the network solution $u_{net}(x, y, z, t; \theta)$ is 8.7470e-10 is best comparing with one hidden layer this means that the two hidden layer with the [9 19] nodes gives better result for linear case. While in the nonlinear case take long time 00:02:21 in [9 3] nodes with the best epoch is 1000 and the value of preformance is 1.8910e-11. However, that results is not good when we comparing with one hidden layer. This means that the one hidden layer ANN with 9 nodes in hidden layer is sufficient to get good result for solving nonlinear problems. Also, we conclude that many important article used original LM for training ANN such [38–41] can be resolve by training with new LM training algorithm to get best results

References

[1] Salih, H., Tawfiq, L. N. M., Yahya, Z. R., & Zin, S. M. (2018). Solving modified regularized long wave equation using collocation method. *Journal of Physics: Conference Series*, 1003(1), 012062. `https://doi.org/10.1088/1742-6596/1003/1/012062`

[2] Hussein, N.A., & Tawfiq L. N.M. (2023). Exact soliton solution for systems of non-linear (2+1)D-DEs. *AIP Conference Proceedings*, 2834(1), 1-7.

[3] Jabber, A. K., & Tawfiq, L. N. M. (2018). New transform fundamental properties and its applications. *Ibn Alhaitham Journal for Pure and Applied Science*, 31(1), 151-163. `https://doi.org/10.30526/31.2.1954`

[4] Ali, S., Khan, A., Shah, K., Alqudah, M. A., & Abdeljawad, T. (2022). On computational analysis of highly nonlinear model addressing real world applications. *Results in Physics*, 36, 105431. `https://doi.org/10.1016/j.rinp.2022.105431`

[5] Gul, H., Alrabaiah, H., Ali, S., Shah, K., & Muhammad, S. (2020). Computation of solution to fractional order partial reaction diffusion equations. *Journal of Advanced Research*, 25, 31-38. `https://doi.org/10.1016/j.jare.2020.04.021`

[6] Tawfiq, L. N., & Hussein, N. A. (2023). Efficient approach for solving (2+ 1) D-differential equations. *Baghdad Science Journal*, 20(1), 0166-0166. `https://doi.org/10.21123/bsj.2022.6541`

[7] Enadi, M. O., & Tawfiq, L. N. M. (2019). New approach for solving three dimensional space partial differential equation. *Baghdad Science Journal*, 16(3), 786-792. `https://doi.org/10.21123/bsj.2019.16.3(Suppl.).0786`

[8] Tawfiq, L. N. M., & Altaie, H. (2020). Recent modification of homotopy perturbation method for solving system of third order PDEs. *Journal of Physics: Conference Series*, 1530(1), 1-7. `https://doi.org/10.1088/1742-6596/1530/1/012073`

[9] Ghazi, F. F. (2020). Modeling the contamination of soil adjacent to Mohammed AL-Qassim highway in Baghdad. *Iraqi Journal of Science*, 61(10), 2663-2670. `https://doi.org/10.24996/ijs.2020.61.10.23`

[10] Tawfiq, L. N. M., & Kareem, Z. H. (2021). Efficient modification of the decomposition method for solving a system of PDEs. *Iraqi Journal of Science*, 62(9), 3061-3070. `https://doi.org/10.24996/ijs.2021.62.9.21`

[11] Kareem, Z. H., & Tawfiq, L. N. M. (2020). Recent modification of decomposition method for solving nonlinear partial differential equations.

Journal of Advances in mathematics, 18, 154-161. https://doi.org/10.24297/jam.v18i.8744

[12] Kareem, Z. H., & Tawfiq, L. N. M. (2023). Recent modification of decomposition method for solving wave-like Equation. *Journal of Interdisciplinary Mathematics*, 26(5), 809-820. https://doi.org/10.47974/JIM-1235

[13] Tawfiq, L. N., & Hussein, N. A. (2022). Exact solution for systems of nonlinear (2+ 1) D-differential equations. *Iraqi Journal of Science*, 63(10), 4388-4396. https://doi.org/10.24996/ijs.2022.63.10.25

[14] Tawfiq, L. N. M., & Abed, A. I. (2021). Efficient method for solving fourth order PDEs. *Journal of Physics: Conference Series*, 1818(1), 1-10. https://doi.org/10.1088/1742-6596/1818/1/012166

[15] Kareem, Z.H., & Tawfiq, L. N.M. (2022). New modification of decomposition method for solving high order strongly nonlinear partial differential equations. *AIP Conference Proceedings*, 2398(1), 1-9.

[16] Hussein, N. A., & Tawfiq, L. N. M. (2020, May). New approach for solving (1+ 1)-dimensional differential equation. *Journal of Physics: Conference Series*, 1530(1), 1-11. https://doi.org/10.1088/1742-6596/1530/1/012098

[17] Tawfiq, L. N., & Yassien, S. M. (2013). Solution of high order ordinary boundary value problems using semi-analytic technique. *Ibn Al-Haitham Journal for Pure & Applied Sciences*, 26(1), 281-291.

[18] Hussein, N.A., & Tawfiq, L.N.M. (2022). Efficient approach for solving high order (2+1) D-differential equation. *AIP Conference Proceedings*, 2398(1), pp. 1-11. https://doi.org/10.1063/5.0093671

[19] Salih, H., & Tawfiq, L. (2020, November). Solution of modified equal width equation using quartic trigonometric-spline method. *Journal of Physics: Conference Series*, 1664(1), 1-10. https://doi.org/10.1088/1742-6596/1664/1/012033

[20] Tawfiq, L. N. M., & Khamas, A. H. (2020, May). New coupled method for solving Burger's equation. *Journal of Physics: Conference Series*, 1530(1), 1-11. https://doi.org/10.1088/1742-6596/1530/1/012069

[21] Tawfiq, L. N. M., & Khamas, A. H. (2023). New approach for calculate exponential integral function. *Iraqi Journal of Science*, 64(8), 4034-4042. https://doi.org/10.24996/ijs.2023.64.8.27

[22] Tawfiq, L. N. M., Al-Noor, N. H., & Al-Noor, T. H. (2019, September). Estimate the rate of contamination in baghdad soils by using numerical method. *Journal of Physics: Conference Series*, 1294(3), 1-11. https://doi.org/10.1088/1742-6596/1294/3/032020

[23] Tawfiq, L. N., & Oraibi, Y. A. (2017). Fast training algorithms for feed forward neural networks. *Ibn Al-Haitham Journal for Pure and Applied Science*, 26(1), 275-280.

[24] Tawfiq, L. N., & Hussein, A. A. (2013). Design feed forward neural network to solve singular boundary value problems. *International Scholarly Research Notices*, 2013, 1-7. https://doi.org/10.1155/2013/650467

[25] Tawfiq, L. N. M., & Hussein, W. R. (2016). Design suitable neural network for processing face recognition. *Global Journal of Engineering Science and Researches*, 3(3), 58-64.

[26] Tawfiq, L. N. M. (2017). The finite element neural network and its applications to forward and inverse problems. *Ibn AL-Haitham Journal For Pure and Applied Science*, 19(4), 109-124.

[27] Tawfiq, L. N. M., & Salih, O. M. (2019). Design suitable feed forward neural network to solve Troesch's problem. *Sci. Int.(Lahore)*, 31(1), 41-48.

[28] Hussien, Z. (2020). Anomaly detection approach based on deep neural network and dropout. *Baghdad Science Journal*, 17(2 (SI)), 0701-0701. https://doi.org/10.21123/bsj.2020.17.2(SI).0701

[29] Ali, M. H., & Tawfiq, L. N. (2023). Design optimal neural network for solving unsteady state confined aquifer problem. *Mathematical Modelling of Engineering Problem*s, 10(2), 565-571. https://doi.org/10.18280/mmep.100225

[30] Alia, M. H., & Tawfiqa, L. N. (2023). Novel neural network based on New modification of BFGS update algorithm for solving partial differential equations. *Advances in the Theory of Nonlinear Analysis and its Applications*, 7(4), 76-88.

[31] Gupta, R., & Batra, C. M. (2022). Performance assessment of solar-transformer-consumption system using neural network approach. *Baghdad Science Journal*, 19(4), 0865-0865. https://doi.org/10.21123/bsj.2022.19.4.0865

[32] Tawfiq, L. N. M., & Khamas, A. H. (2021). Determine the effect hookah smoking on health with different types of tobacco by using parallel processing technique. *Journal of Physics: Conference Series*, 1818(1), 1-10. https://doi.org/10.1088/1742-6596/1818/1/01217

[33] Tawfiq, L. N. M., & Tawfiq, M. N. M. (2017). The effect of number of training samples for artificial neural network. *Ibn AL-Haitham Journal For Pure and Applied Science*, 23(3), 1-7.

[34] Ghazi, F. F., & Tawfiq, L. N. M. (2020). New approach for solving two dimensional spaces PDE. *Journal of Physics: Conference Series*, 1530(1), 012066. https://doi.org/10.1088/1742-6596/1530/1/012066

[35] Jamil, H.J., Albahri, M.R.A., Al-Noor, N.H., Al-Noor, T.H., Heydari, A.R., Rajan, A.K., Arnetz, J., Arnetz, B. & Tawfiq, L.N.M. (2020). Hookah smoking with health risk perception of different types of tobacco. *Journal of Physics: Conference Series*, 1664(1), 012127. https://doi.org/10.1088/1742-6596/1664/1/012127

[36] Kareema, Z. H., & Tawfiqa, L. N. (2023). Solving (3+ 1) D-New Hirota bilinear equation using tanh method and new modification of extended tanh method. *Advances in the Theory of Nonlinear Analysis and its Applications*, 7(4), 114-122.

[37] Hussein, N. A., Helal, M. M., & Tawfiq, L. N. M. (2023). Double LA-transform and their properties for solving partial differential equations. *AIP Conference Proceedings*, 2834(1), 1-10

[38] Kumar, A., Kumar, M., & Goswami, P. (2024). Numerical solution of coupled system of Emden-Fowler equations using artificial neural network technique. *An International Journal of Optimization and Control: Theories & Applications*, 14(1), 62-73. https://doi.org/10.11121/ijocta.1424

[39] Okkan, U. (2011). Application of Levenberg-Marquardt optimization algorithm based multilayer neural networks for hydrological time series modeling. *An International Journal of Optimization and Control: Theories & Applications*, 1(1), 53-63. https://doi.org/10.11121/ijocta.01.2011.0038

[40] Kumar, K., Parida, M., & Katiyar, V. K. (2011). Road traffic noise prediction with neural networks - A review. *An International Journal of Optimization and Control: Theories & Applications*, 2(1), 29-37. https://doi.org/10.11121/ijocta.01.2012.0059

[41] Demirtas, M., & Alci, M. (2011). A comparative study of neural networks and fuzzy systems in modeling of a nonlinear dynamic system. *An International Journal of Optimization and Control: Theories & Applications*, 1(1), 65-73. https://doi.org/10.11121/ijocta.01.2011.0055

Farah F. Ghazi *is PhD student in mathematics. She holds (Bachelor's in 2010 and Master's in 2016) degrees from the University of Baghdad, College of Education for Pure Sciences Ibn Al-Haytham, Department of Mathematics. Also, she teaches in the Mathematics Department. The number of published and accepted papers is 16. She is interested in ODE, PDE, Integral equations, Numerical methods for solving problem , neural networks, artificial intelligence, and machine learning.*
https://orcid.org/0000-0002-8444-7779

Luma N. M. Tawfiq *Professor in Applied Mathematics in the Department of Mathematics, University of Baghdad, Iraq has more than 230 research publications in international and Arab magazines, supervised more than 53 Msc. & PhD Theses in different branches of Applied Mathematics, and discussed dozens of master's and doctoral dissertations. Also, published more than 20 books.*
https://orcid.org/0000-0001-5778-4983