

Numerical Solution of Fuzzy Partial Differential Equations by Using Modified Fuzzy Neural Networks

Abstract: In this paper a new method based on learning algorithm of modified Fuzzy neural networks has been developed for obtaining numerical solution of fuzzy partial differential equations. Using modified fuzzy neural network makes that training points should be selected over an open interval without training the network in the range of first and end points. Therefore, the calculating volume involving computational error is reduced. In fact, the training points depending on the distance selected for training fuzzy neural network are converted to similar points in the open interval by using a new approach, then the fuzzy network is trained in these similar areas. In comparison with existing similar fuzzy neural networks, the proposed method provides solutions with high accuracy. Finally, we illustrate our approach by two numerical examples

Keywords: Fuzzy partial differential equation, Fuzzy neural network, Feed-forward neural network, BFGS method, Hyperbolic tangent function.

1. Introduction

Nowadays, fuzzy differential equations (FDEs) is a popular topic studied by many researchers since it is utilized widely for the purpose of modeling problems in science and engineering. Most of the practical problems require the solution of a fuzzy differential equation which satisfies fuzzy initial or fuzzy boundary conditions.

The theory of fuzzy differential equations was treated by Kaleva [20], Ouyang and Wu [36], Khanna[22], Nieto[32], Buckley and Feuring [12], Seikkala also recently there appeared the papers of Bede, Bede and Gal [9], Diamond [14,15], Georgiou and Nieto and et al. [17], Nieto and Lopez [33].

In the following, we have mentioned some numerical solution which have proposed by other scientists. Abbasbandy and Allahviranloo have

solved fuzzy differential equations by Runge-Kuta and Taylor methods [1,2]. Also , Allahviranloo and et al. solved differential equations by predictor- corrector and transformation methods [5,6,7]. Ghazanfari and Shakerami developed Runge-Kuta like formula of order 4 for solving fuzzy differential equations [18]. Nystrom method has been introduced for solving fuzzy differential equations [21]. Allahviranloo and Kermani solved Fuzzy Linear Partial Differential Equations Under New Definition of fuzzy Derivative[8]. Dahalan and Muthuvalu and et al. developed the Performance of HSAGE Method with Seikkala Derivative For 2-D Fuzzy Poisson Equation[13].

In 1990 Lee and Kang [23] used parallel processor computers to solve a first order differential equations with Hopfield neural network models. Meade and Fernandes[24,25] solved linear and nonlinear ordinary differential equations using feed-forward neural network architecture and B_1 -splines. Recently, fuzzy neural networks have been successfully used for solving fuzzy polynomial equations and systems of fuzzy polynomial equations [3,4], approximate fuzzy coefficients of fuzzy regression models [26,27,28], approximate solution of fuzzy linear system and fully fuzzy linear systems[35]. In Year 2012 Mosleh and Otadi [29] used fuzzy neural network to solve a first order fuzzy neural network , system of fuzzy differential equations [30]and second order fuzzy differential equation[31].

In this work we proposed a new numerical method to find the approximate solution of fuzzy partial differential equations, this method can result in improved numerical methods for solving fuzzy partial differential equations (FPDEs). In this proposed method, fuzzy neural network model (FNNM) is applied as universal approximator. We use fuzzy trial function, this fuzzy trial function is a combination of two terms. A first term is responsible for the fuzzy conditions while the second term contains the fuzzy neural network adjustable parameters to be calculated. The main aim of this paper is to illustrate how fuzzy connection weights are adjusted in the learning of fuzzy neural networks. Our fuzzy neural network in this work is a three-Layer feed-

forward neural network where connection weights and biases are fuzzy numbers .

This modified method is called modified fuzzy neural network(MFNN) for solving fuzzy partial differential equations. This new method based on replacing each x in the training set (where $x \in [a, b]$) by the polynomial $Q(x) = \epsilon(x + 1)$ such that $Q(x) \in (a, b)$ by choosing a suitable $\epsilon \in (0, 1)$. In this paper, we will illustrate this modified method by solving two numerical examples . In general, this modified method is effective for solving fuzzy partial differential equations .

2. preliminaries

In this section the basic notations used in fuzzy calculus are introduced.

Definition 2.1 [32]: A fuzzy number u is completely determined by any pair $u = (\underline{u}, \bar{u})$ of functions $\underline{u}(r), \bar{u}(r) : R \rightarrow [0, 1]$ satisfying the conditions:

(1) $\underline{u}(r)$ is a bounded, monotonic, increasing (non – decreasing) left continuous function for all $r \in (0, 1]$ and right continuous for $r=0$.

(2) $\bar{u}(r)$ is a bounded, monotonic, decreasing (non – increasing) left continuous function for all $r \in (0, 1]$ and right continuous for $r=0$.

(3) For all $r \in (0, 1]$ we have $\underline{u}(r) \leq \bar{u}(r)$.

For every $u = (\underline{u}, \bar{u}), v = (\underline{v}, \bar{v})$ and $k > 0$ we define addition and multiplication as follows:

$$(1) \underline{(u + v)}(r) = \underline{u}(r) + \underline{v}(r) \quad (1)$$

$$(2) \overline{(u + v)}(r) = \bar{u}(r) + \bar{v}(r) \quad (2)$$

$$(3) \underline{(ku)}(r) = K \underline{u}(r), \overline{(ku)}(r) = K \bar{u}(r) \quad (3)$$

The collection of all fuzzy numbers with addition and multiplication as defined by Eqs. (1) $\rightarrow$ (3) is denoted by E^1 . For $r \in (0, 1]$, we define

the r - cuts of fuzzy number u with $[u]_r = \{x \in R | u(x) \geq r\}$ and for $r=0$, the support of u is defined as $[u]_0 = \overline{\{x \in R | u(x) > 0\}}$

Definition 2.2 [32]: The function $f : \mathbb{R} \rightarrow E^1$ is called a fuzzy function. Now if, for an arbitrary fixed $t_1 \in \mathbb{R}$ and $\epsilon > 0$ there exist a $\delta > 0$ such that: $|t - t_1| < \delta \Rightarrow d[f(t), f(t_1)] < \epsilon$

Then f is said to be continuous function.

Definition 2.3 [17]: let $u, v \in E^1$. If there exist $w \in E^1$ such that $u = v + w$ then w is called the H-difference (Hukuhara-difference) of u, v and it is denoted by $w = u \ominus v$. In this work the $\ominus$ sign stands always for H-difference, and let us remark that $u \ominus v \neq u + (-1)v$.

Definition 2.4 [17]: Let $f : [a, b] \rightarrow E^1$ and $t_0 \in [a, b]$. We say that f is H-differential (Hukuhara-differential) at t_0 , if there exists an element $f'(t_0) \in E^1$ such that for all $h > 0$ sufficiently small, $\exists f(t_0 + h) \ominus f(t_0)$, $f(t_0) \ominus f(t_0 - h)$ and the limits

$$\lim_{h \rightarrow 0} \frac{f(t_0 + h) \ominus f(t_0)}{h} = \lim_{h \rightarrow 0} \frac{f(t_0) \ominus f(t_0 - h)}{h} = f'(t_0). \quad (4)$$

3. Fuzzy Neural Network

A fuzzy neural network or neuro-fuzzy system is a learning machine that finds the parameters of a fuzzy system (i.e. fuzzy sets, fuzzy rules) by exploiting approximation from the neural networks. Combining fuzzy system with neural network. Both neural network and fuzzy system have some things in common [4].

Artificial neural networks are an exciting form of the artificial intelligence which mimic the learning process of the human brain in order to extract patterns from historical data. Simple perceptrons need a teacher to tell the network what the desired output should be. These are supervised networks. In an unsupervised net, the network adapts purely in response to its input [19].

4. Operations of Fuzzy Numbers

we briefly mention fuzzy numbers operation defined by the extension principle. Since output vector of feed-forward neural network is fuzzy in this paper, the following addition, multiplication and nonlinear

mapping of fuzzy number are necessary for defining our fuzzy neural network [28]:

$$(1) \mu_{A+B}(z) = \text{Max} \{ \mu_A(x) \wedge \mu_B(y) \mid z = x + y \} \quad (5)$$

$$(2) \mu_{AB}(z) = \text{Max} \{ \mu_A(x) \wedge \mu_B(y) \mid z = x y \} \quad (6)$$

$$(3) \mu_{f(\text{net})}(z) = \text{Max} \{ \mu_{\text{net}}(x) \mid z = f(x) \} \quad (7)$$

Where A,B and net are fuzzy number , μ (*) denotes the membership function of each fuzzy number, $\wedge$ is the Minimum operator and $f(\cdot)$ is a continuous activation function (such as Hyperbolic tangent function) inside the hidden neurons. The above operations of fuzzy numbers are numerically performed on level sets (i.e. r-cuts) .

The r-level set of a fuzzy number A is defined as:

$$[A]_r = \{ x \in R \mid \mu_A(x) \geq r \} , 0 < r \leq 1 \quad (8)$$

Since level sets of fuzzy numbers become closed intervals we denote

$$[A]_r \text{ as } : [A]_r = \left[[A]_r^L, [A]_r^U \right]$$

Where $[A]_r^L$ and $[A]_r^U$ are the lower limit and the upper limit of the r-level set $[A]_r$ respectively . From interval arithmetic , the above operations of fuzzy number are written for r-level set as follows [35] :

$$[A]_r + [B]_r = \left[[A]_r^L + [B]_r^L, [A]_r^U + [B]_r^U \right] \quad (9)$$

$$[A]_r [B]_r = \left[\begin{array}{l} \text{Min} \{ [A]_r^L \cdot [B]_r^L, [A]_r^L \cdot [B]_r^U, [A]_r^U \cdot [B]_r^L, [A]_r^U \cdot [B]_r^U \} \\ \text{Max} \{ [A]_r^L \cdot [B]_r^L, [A]_r^L \cdot [B]_r^U, [A]_r^U \cdot [B]_r^L, [A]_r^U \cdot [B]_r^U \} \end{array} \right] \quad (10)$$

$$f([net]_r) = f \left(\left[[net]_r^L, [net]_r^U \right] \right) = \left[f \left([net]_r^L \right), f \left([net]_r^U \right) \right] \quad (11)$$

5. Input – Output Relation of Each Unit

Let us consider a fuzzy three – layer feed – forward neural network with n input units , m hidden units and s output units . Target vector , connection weights and biases are fuzzy numbers and input vector is

real number. For convenience in this discussion , FNNM with an input layer, a single hidden layer, and an output layer in Fig. (1) is represented as a basic structural architecture. Here, the dimension of FNNM is denoted by the number of neurons in each layer , that is $n \times m \times s$, where n , m and s are the number of the neurons in the input layer, the hidden layer and the output layer , respectively [29] .

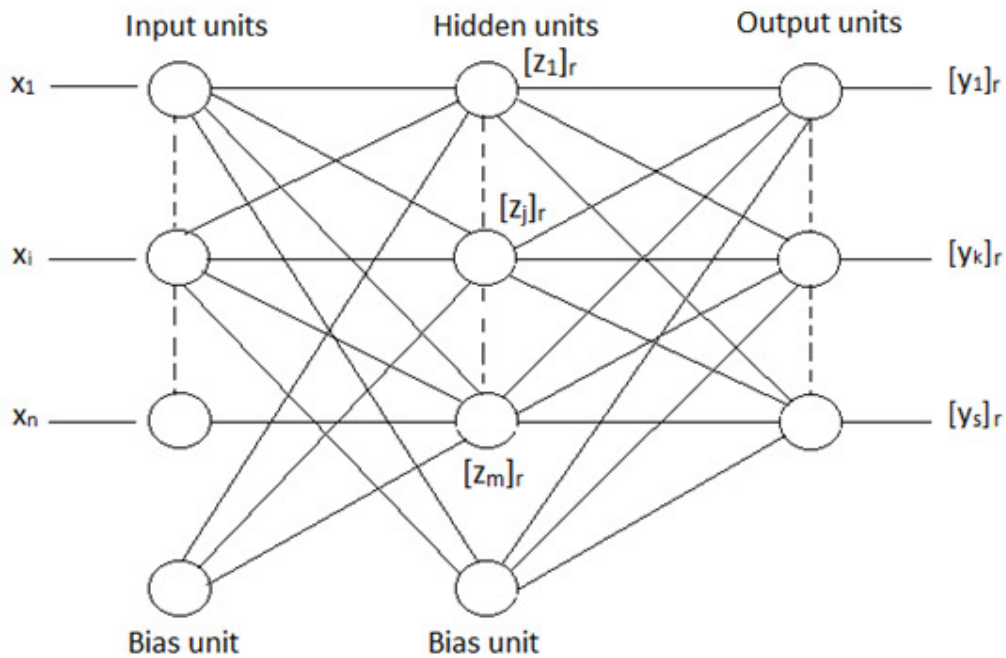


Fig.(1):Three-layer feed-forward Fuzzy neural network .

The architecture of the model shows how FNNM transforms the n inputs $(x_1, x_2, \dots, x_i, \dots, x_n)$ into the s fuzzy outputs $([y_1]_r, [y_2]_r, \dots, [y_k]_r, \dots, [y_s]_r)$ throughout the m hidden fuzzy neurons $([z_1]_r, [z_2]_r, \dots, [z_j]_r, \dots, [z_m]_r)$, where the cycles represent the neurons in each layer. Let $[b_j]_r$ be the fuzzy bias for the fuzzy neuron $[z_j]_r$, $[c_k]_r$ be the fuzzy bias for the fuzzy neuron $[y_k]_r$, $[w_{ji}]_r$ be the fuzzy weight connecting crisp neuron x_i to fuzzy neuron $[z_j]_r$, and $[w_{kj}]_r$ be the fuzzy weight connecting fuzzy neuron $[z_j]_r$ to fuzzy neuron $[y_k]_r$.

When an n – dimensional input vector $(x_1, x_2, \dots, x_i, \dots, x_n)$ is presented to our fuzzy neural network , its input – output relation can be written as follows , where $F : R^n \rightarrow E^s$:

Input units :

$$o_i = x_i, \quad i = 1, 2, 3, \dots, n \quad (12)$$

Hidden units :

$$z_j = F(\text{net}_j), \quad j = 1, 2, 3, \dots, m, \quad (13)$$

$$\text{net}_j = \sum_{i=1}^n o_i w_{ji} + b_j \quad (14)$$

Output units :

$$y_k = F(\text{net}_k), \quad k = 1, 2, 3, \dots, s, \quad (15)$$

$$\text{net}_k = \sum_{j=1}^m w_{kj} z_j + c_k \quad (16)$$

The architecture of our fuzzy neural network is shown in Fig.(1) , where connection weights , biases , and targets are fuzzy numbers and inputs are real numbers. The input – output relation in Eqs. (12 – 16) is defined by the extension principle .

6. Calculation of Fuzzy Output

The fuzzy output from each unit in Eqs. (12 – 16) is numerically calculated for real inputs and level sets of fuzzy weights and fuzzy biases. The input – output relations of our fuzzy neural network can be written for the r – level sets [30] .

Input units :

$$o_i = x_i, \quad i = 1, 2, 3, \dots, n \quad (17)$$

Hidden units :

$$[z_j]_r = F([\text{net}_j]_r), \quad j = 1, 2, 3, \dots, m, \quad (18)$$

$$[\text{net}_j]_r = \sum_{i=1}^n o_i [w_{ji}]_r + [b_j]_r \quad (19)$$

Output units :

$$[y_k]_r = F([\text{net}_k]_r), \quad k = 1, 2, 3, \dots, s, \quad (20)$$

$$[\text{net}_k]_r = \sum_{j=1}^m [w_{kj}]_r [z_j]_r + [c_k]_r. \quad (21)$$

From Eqs. (17 – 21) , we can see that the r – level sets of the fuzzy outputs y_k ’s are calculated from those of the fuzzy weights, fuzzy biases and the crisp inputs.

From the operations of fuzzy numbers , the above relations are rewritten as follows when the inputs x_i ’s are non – negative , i.e., $x_i \geq 0$

Input units :

$$o_i = x_i \quad (22)$$

Hidden units :

$$[z_j]_r = F([net_j]_r) = \left[[z_j]_r^L, [z_j]_r^U \right] = \left[F([net_j]_r^L), F([net_j]_r^U) \right] \quad (23)$$

where

$$[net_j]_r^L = \sum_{i=1}^n o_i [w_{ji}]_r^L + [b_j]_r^L \quad (24)$$

$$[net_j]_r^U = \sum_{i=1}^n o_i [w_{ji}]_r^U + [b_j]_r^U \quad (25)$$

Output units :

$$[y_k]_r = F([net_k]_r) = \left[[y_k]_r^L, [y_k]_r^U \right] = \left[F([net_k]_r^L), F([net_k]_r^U) \right] \quad (26)$$

$$[net_k]_r^L = \sum_{j \in a} [w_{kj}]_r^L [z_j]_r^L + \sum_{j \in b} [w_{kj}]_r^L [z_j]_r^U + [c_k]_r^L \quad (27)$$

$$[net_k]_r^U = \sum_{j \in c} [w_{kj}]_r^U [z_j]_r^U + \sum_{j \in d} [w_{kj}]_r^U [z_j]_r^L + [c_k]_r^U \quad (28)$$

For $[z_j]_r^U \geq [z_j]_r^L \geq 0$, where

$$a = \left\{ j : [w_{kj}]_r^L \geq 0 \right\}, b = \left\{ j : [w_{kj}]_r^L < 0 \right\}$$

$$c = \left\{ j : [w_{kj}]_r^U \geq 0 \right\}, d = \left\{ j : [w_{kj}]_r^U < 0 \right\},$$

$$a \cup b = \{1, 2, 3, \dots, m\} \text{ and } c \cup d = \{1, 2, 3, \dots, m\} .$$

7.Fuzzy Neural Network Approach For Solving FDEs

To solve any fuzzy ordinary differential equation (i.e., first order FDE , second order FDE ,etc.) we consider a three – layered fuzzy neural

network model (FNNM) with one unit entry x , one hidden layer consisting of m activation functions and one unit output $N(x, p)$. The activation function for the hidden units of our fuzzy neural network is hyperbolic tangent function. Here, the dimension of FNNM is $(1 \times m \times 1)$.

For every entry x the input neuron makes no changes in its input, so the input to the hidden neurons is:

$$\text{net}_j = x w_j + b_j, j = 1, 2, 3, \dots, m, \quad (29)$$

Where w_j is a weight parameter from input layer to the j th unit in the hidden layer, b_j is an j th bias for the j th unit in the hidden layer.

The output , in the hidden neurons is :

$$z_j = s(\text{net}_j), j = 1, 2, 3, \dots, m, \quad (30)$$

Where s is the hyperbolic tangent activation function . The output neuron make no change in its input , so the input to the output neuron is equal to output :

$$N = v_1 z_1 + v_2 z_2 + v_3 z_3 + \dots + v_j z_j + \dots + v_m z_m = \sum_{j=1}^m v_j z_j \quad (31)$$

Where v_j is a weight parameter from j th unit in the hidden layer to the output layer.

From Eqs. (22 – 28) , we can see that the r – level sets of the Eqs. (29 – 31) are calculated from those of the fuzzy weights , fuzzy biases and crisp inputs. For our fuzzy neural network , we can derive the learning algorithm without assuming that the input x is non – negative. For reducing the complexity of the learning algorithm , input x usually assumed as non-negative in the fuzzy neural network, i.e., $x \geq 0$ [29,30]:

Input unit :

$$o = x, \quad (32)$$

Hidden units :

$$[z_j]_r = \left[[z_j]_r^L, [z_j]_r^U \right] = \left[s\left([net_j]_r^L\right), s\left([net_j]_r^U\right) \right] \quad (33)$$

$$\text{where } [net_j]_r^L = o [w_j]_r^L + [b_j]_r^L, \quad [net_j]_r^U = o [w_j]_r^U + [b_j]_r^U$$

Output unit :

$$[N]_r = [[N]_r^L, [N]_r^U] \quad , \quad \text{where}$$

$$[N]_r^L = \sum_{j \in a} [v_j]_r^L [z_j]_r^L + \sum_{j \in b} [v_j]_r^L [z_j]_r^U \quad (34)$$

$$[N]_r^U = \sum_{j \in c} [v_j]_r^U [z_j]_r^U + \sum_{j \in d} [v_j]_r^U [z_j]_r^L \quad (35)$$

For $[z_j]_r^U \geq [z_j]_r^L \geq 0$, where : $a = \{j : [v_j]_r^L \geq 0\}$, $b = \{j : [v_j]_r^L < 0\}$

$$c = \{j : [v_j]_r^U \geq 0\} \quad , \quad d = \{j : [v_j]_r^U < 0\} \quad ,$$

$a \cup b = \{1, 2, \dots, m\}$ and $c \cup d = \{1, 2, \dots, m\}$.

For illustration the solution steps, we will consider the first order fuzzy differential equation [29] :

$$\frac{dy(x)}{dx} = F(x, y) \quad , \quad x \in [a, b] \quad , \quad y(a) = A \quad (36)$$

Where A is a fuzzy number in E^1 with r – level sets :

$$[A]_r = [[A]_r^L, [A]_r^U] \quad , \quad r \in [0, 1] \quad .$$

The fuzzy trial solution for this problem is :

$$[y_t(x, p)]_r = [A]_r + (x - a) [N(x, p)]_r \quad (37)$$

This fuzzy solution by intention satisfies the fuzzy initial condition in (36)

The error function that must be minimized for the problem (36) is in the form : $E = \sum_{i=1}^g (E_{ir}^L + E_{ir}^U)$ (38)

$$E_{ir}^L = \left[\left[\frac{dy_t(x_i, p)}{dx} \right]_r^L - [F(x_i, y_t(x_i, p))]_r^L \right]^2 \quad (39)$$

$$E_{ir}^U = \left[\left[\frac{dy_t(x_i, p)}{dx} \right]_r^U - [F(x_i, y_t(x_i, p))]_r^U \right]^2 \quad (40)$$

Where $\{x_i\}_{i=1}^g$ are discrete points belonging to the interval $[a, b]$ (training set) and in the cost function (38) , E_r^L and E_r^U can be viewed as the squared errors for the lower and upper limits of the r – level sets .

It is easy to express the first derivative of $[N(x, p)]_r$ in terms of the derivative of the hyperbolic tangent , i.e.,

$$\frac{\partial [N]_r^L}{\partial x} = \sum_a [v_j]_r^L \frac{\partial [z_j]_r^L}{\partial [net_j]_r^L} \frac{\partial [net_j]_r^L}{\partial x} + \sum_b [v_j]_r^L \frac{\partial [z_j]_r^U}{\partial [net_j]_r^U} \frac{\partial [net_j]_r^U}{\partial x} \quad (41)$$

$$\frac{\partial [N]_r^U}{\partial x} = \sum_c [v_j]_r^U \frac{\partial [z_j]_r^U}{\partial [net_j]_r^U} \frac{\partial [net_j]_r^U}{\partial x} + \sum_d [v_j]_r^U \frac{\partial [z_j]_r^L}{\partial [net_j]_r^L} \frac{\partial [net_j]_r^L}{\partial x} \quad (42)$$

Where $a = \{j : [v_j]_r^L \geq 0\}$, $b = \{j : [v_j]_r^L < 0\}$,

$c = \{j : [v_j]_r^U \geq 0\}$, $d = \{j : [v_j]_r^U < 0\}$,

$a \cup b = \{1, 2, 3, \dots, m\}$ and $c \cup d = \{1, 2, 3, \dots, m\}$. Also we have

$$\frac{\partial [net_j]_r^L}{\partial x} = [w_j]_r^L \quad , \quad \frac{\partial [net_j]_r^U}{\partial x} = [w_j]_r^U$$

$$\frac{\partial [z_j]_r^L}{\partial [net_j]_r^L} = 1 - ([z_j]_r^L)^2 \quad , \quad \frac{\partial [z_j]_r^U}{\partial [net_j]_r^U} = 1 - ([z_j]_r^U)^2$$

Now differentiating from fuzzy trial function $[y_t(x, p)]_r$ in (39) and (40) we obtain :

$$\frac{[y_t(x, p)]_r^L}{\partial x} = [N(x, p)]_r^L + (x - a) \frac{\partial [N(x, p)]_r^L}{\partial x} \quad (43)$$

$$\frac{[y_t(x, p)]_r^U}{\partial x} = [N(x, p)]_r^U + (x - a) \frac{\partial [N(x, p)]_r^U}{\partial x} \quad (44)$$

Therefore , we get

$$E_{ir}^L = \left[\begin{array}{c} \sum_a [v_j]_r^L [z_j]_r^L + \sum_b [v_j]_r^L [z_j]_r^U + (x_i - a) \\ \left(\sum_a [v_j]_r^L [w_j]_r^L \left(1 - ([z_j]_r^L)^2 \right) + \sum_b [v_j]_r^L [w_j]_r^U \left(1 - ([z_j]_r^U)^2 \right) \right) \\ -F \left(x_i, [A]_r^L + (x_i - a) \left(\sum_a [v_j]_r^L [z_j]_r^L + \sum_b [v_j]_r^L [z_j]_r^U \right) \right) \end{array} \right]^2 \quad (45)$$

$$E_{ir}^U = \left[\begin{array}{c} \sum_c [v_j]_r^U [z_j]_r^U + \sum_d [v_j]_r^U [z_j]_r^L + (x_i - a) \\ \left(\sum_c [v_j]_r^U [w_j]_r^U \left(1 - ([z_j]_r^U) \right) + \sum_d [v_j]_r^U [w_j]_r^L \left(1 - ([z_j]_r^L) \right) \right) \\ -F \left(x_i, [A]_r^U + (x_i - a) \left(\sum_c [v_j]_r^U [z_j]_r^U + \sum_d [v_j]_r^U [z_j]_r^L \right) \right) \end{array} \right]^2 \quad (46)$$

Now we substitute (45) and (46) in (38) to find the error function that must be minimized for problem (36).

For the higher order fuzzy ordinary differential equations and fuzzy partial differential equations eq. (45) and eq. (46) will be very complex and the computations are very difficult.

Therefore, for reducing the complexity of the learning algorithm, we will propose a partially fuzzy neural network in the next section.

8. Partially Fuzzy Neural Networks

One drawback of the fully fuzzy neural networks with fuzzy connection weights is long computation time. Another drawback is that the learning algorithm is complicated. Therefore, for reducing the complexity of the learning algorithm, a partially fuzzy neural network (PFNN) architecture has been proposed where connection weights to the output unit are fuzzy numbers while connection weights and biases to the hidden units are real numbers. [29,31] .

The input – output relation of each unit of our partially fuzzy neural network in Eqs. (32-35) can be rewritten for r – level sets as follows :

Input unit : $o = x$

Hidden units : $z_j = s(\text{net}_j)$, $j = 1, 2, 3, \dots m$

Where $\text{net}_j = o w_j + b_j$

Output unit : $[N]_r = [[N]_r^L, [N]_r^U] = [\sum_{j=1}^m [v_j]_r^L z_j, \sum_{j=1}^m [v_j]_r^U z_j]$

Now to find the minimized error function(under PFNN) for problem (36):

$$\frac{\partial [N]_r^L}{\partial x} = \sum_{j=1}^m [v_j]_r^L \frac{\partial z_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial x} = \sum_{j=1}^m [v_j]_r^L w_j (1 - z_j^2) \quad (47)$$

$$\frac{\partial [N]_r^U}{\partial x} = \sum_{j=1}^m [v_j]_r^U \frac{\partial z_j}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial x} = \sum_{j=1}^m [v_j]_r^U w_j (1 - z_j^2) \quad (48)$$

By substituting Eqs (47 and 48) in Eqs (39 and 40) , we obtain :

$$E_{ir}^L = \left[\sum_{j=1}^m z_j [v_j]_r^L + (x_i - a) \sum_{j=1}^m w_j (1 - z_j^2) [v_j]_r^L \right]^2 - F \left(x_i, [A]_r^L + (x_i - a) \sum_{j=1}^m z_j [v_j]_r^L \right) \quad (49)$$

$$E_{ir}^U = \left[\sum_{j=1}^m z_j [v_j]_r^U + (x_i - a) \sum_{j=1}^m w_j (1 - z_j^2) [v_j]_r^U \right]^2 - F \left(x_i, [A]_r^U + (x_i - a) \sum_{j=1}^m z_j [v_j]_r^U \right) \quad (50)$$

And then we substitute (49) and (50) in (38) to find the error function that must be minimized for problem (36) (with respect to PFNN).

9. Fuzzy Neural Network Approach For Solving FPDEs

To solve any fuzzy partial differential equation , we consider a three – layered fuzzy neural network model (FNNM) with two unit entries x and y , one hidden layer consisting of m activation functions , and one unit output $N(x, y, p)$. Here , the dimension of FNNM is $(2 \times m \times 1)$.

For every entries x and y , the input neurons makes no changes in its inputs , so the input to the hidden neurons is :

$$\text{net}_j = x w_{j1} + y w_{j2} + b_j , j = 1, 2, 3, \dots, m \quad (51)$$

Where w_{j1} and w_{j2} are a weights from the input layer to the j th unit in the hidden layer, b_j is an j th bias for the j th unit in the hidden layer.

The output in the hidden neurons is :

$$z_j = s(\text{net}_j) , j = 1, 2, 3, \dots, m \quad (52)$$

The output neuron make no changes in its input , so the input to the . output neuron is equal to output :

$$N = \sum_{j=1}^m v_j z_j \quad (53)$$

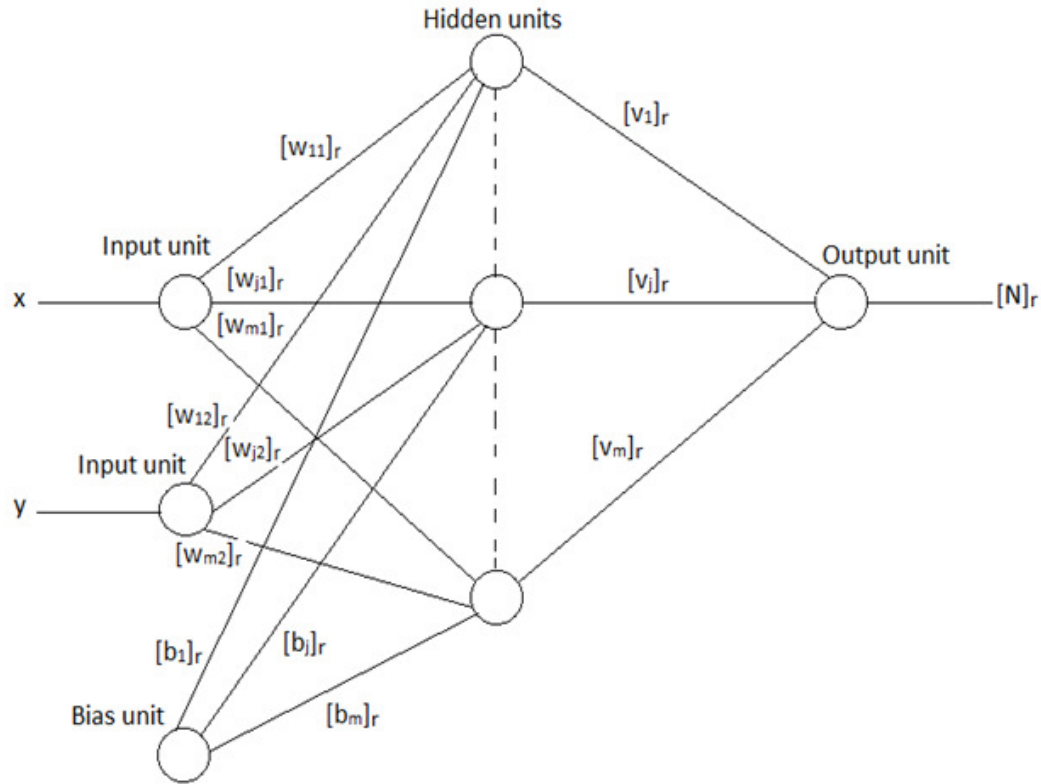


Fig.(2): (2 x m x 1)Feed-forward fuzzy neural network.

From Eqs.(22-28) , we can see that the r – level sets of the Eqs. (51 – 53) are calculated from those of the fuzzy weights , fuzzy biases and crisp inputs (Fig.2) . For our fuzzy neural network , we can derive the learning algorithm without assuming that the inputs x and y are non – negative . For reducing the complexity of the learning algorithm , the inputs x and y usually assumed as non – negative in the fuzzy neural network , i.e., $x \geq 0$ and $y \geq 0$:

$$\text{Input units : } x = x , y = y \quad (54)$$

$$\text{Hidden units : } [z_j]_r = \left[[z_j]_r^L , [z_j]_r^U \right] = \left[s \left([\text{net}_j]_r^L \right) , s \left([\text{net}_j]_r^U \right) \right] \quad (55)$$

$$[\text{net}_j]_r^L = x [w_{j1}]_r^L + y [w_{j2}]_r^L + [b_j]_r^L \quad (56)$$

$$[\text{net}_j]_r^U = x [w_{j1}]_r^U + y [w_{j2}]_r^U + [b_j]_r^U \quad (57)$$

$$\text{Output unit : } [N]_r = \left[[N]_r^L , [N]_r^U \right] \quad (58)$$

$$[N]_r^L = \sum_{j \in a} [v_j]_r^L [z_j]_r^L + \sum_{j \in b} [v_j]_r^L [z_j]_r^U \quad (59)$$

$$[N]_r^U = \sum_{j \in c} [v_j]_r^U [z_j]_r^U + \sum_{j \in d} [v_j]_r^U [z_j]_r^L \quad (60)$$

For $[z_j]_r^U \geq [z_j]_r^L \geq 0$, where : $a = \{j : [v_j]_r^L \geq 0\}$, $b = \{j : [v_j]_r^L < 0\}$

$$c = \{j : [v_j]_r^U \geq 0\}, \quad d = \{j : [v_j]_r^U < 0\},$$

$a \cup b = \{1, 2, 3, \dots, m\}$ and $c \cup d = \{1, 2, 3, \dots, m\}$.

Also, the input – output of each unit of our partially fuzzy neural network in Eqs. (54-60) can be rewritten for r – level sets as follows :

$$\text{Input unit : } x = x, \quad y = y \quad (61)$$

$$\text{Hidden units : } z_j = s(\text{net}_j), \quad j = 1, 2, 3, \dots, m \quad (62)$$

$$\text{where } \text{net}_j = x w_{j1} + y w_{j2} + b_j \quad (63)$$

Output unit :

$$[N]_r = [[N]_r^L, [N]_r^U] = \left[\sum_{j=1}^m [v_j]_r^L z_j, \sum_{j=1}^m [v_j]_r^U z_j \right] \quad (64)$$

10. Description of The Method

We treat here one and two-dimensional problems only. However, it is straightforward to extend the method to more dimensions. For example, if we consider the two-dimensional fuzzy Poisson equation:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x, y), \quad x, y \in [a, b] \quad (65)$$

With the Dirichlet fuzzy boundary conditions (for $x, y \in [0, 1]$):

$$U(0, y) = f_0(y), \quad U(1, y) = f_1(y), \quad U(x, 0) = g_0(x) \text{ and } U(x, 1) = g_1(x).$$

Where: $f(x, y)$, $f_0(y)$, $f_1(y)$, $g_0(x)$ and $g_1(x)$ are fuzzy numbers or fuzzy functions with r -level sets (parametric form) :

$$[f_0(y)]_r = [\underline{f}_0(y), \bar{f}_0(y)], \quad [f_1(y)]_r = [\underline{f}_1(y), \bar{f}_1(y)]$$

$$[g_0(x)]_r = [\underline{g}_0(x), \bar{g}_0(x)], \quad [g_1(x)]_r = [\underline{g}_1(x), \bar{g}_1(x)]$$

The fuzzy trial solution $[U_T(x, y)]_r = [\underline{U}_T(x, y, r, \underline{p}), \bar{U}_T(x, y, r, \bar{p})]$ can be chosen as follows:

$$\begin{aligned}\underline{U}_T(x, y, r, \underline{p}) &= \underline{A}(x, y) + x y (1 - x) (1 - y) \underline{N}(x, y, r, \underline{p}) \\ \bar{U}_T(x, y, r, \bar{p}) &= \bar{A}(x, y) + x y (1 - x) (1 - y) \bar{N}(x, y, r, \bar{p})\end{aligned}\quad (66)$$

Where $\underline{A}(x, y)$ and $\bar{A}(x, y)$ are chosen so as to satisfy the fuzzy boundary conditions, namely:

$$\begin{aligned}\underline{A}(x, y) &= (1 - x) \underline{f}_0(y) + x \underline{f}_1(y) + (1 - y) \\ &\left[\underline{g}_0(x) - \left[(1 - x) \underline{g}_0(0) + x \underline{g}_0(1) \right] \right] + \\ &y \left[\underline{g}_1(x) - \left[(1 - x) \underline{g}_1(0) + x \underline{g}_1(1) \right] \right] \\ \bar{A}(x, y) &= (1 - x) \bar{f}_0(y) + x \bar{f}_1(y) + (1 - y) \\ &\left[\bar{g}_0(x) - \left[(1 - x) \bar{g}_0(0) + x \bar{g}_0(1) \right] \right] + \\ &y \left[\bar{g}_1(x) - \left[(1 - x) \bar{g}_1(0) + x \bar{g}_1(1) \right] \right]\end{aligned}\quad (67)$$

The minimized error function will be: $E = \sum_{i=1}^g (E_{ir}^L + E_{ir}^U)$, where

$$\begin{aligned}E_{ir}^L &= \left[\left[\frac{\partial^2 U_T}{\partial x^2} + \frac{\partial^2 U_T}{\partial y^2} \right]_r^L - [f(x_i, y_i)]_r^L \right]^2 \\ E_{ir}^U &= \left[\left[\frac{\partial^2 U_T}{\partial x^2} + \frac{\partial^2 U_T}{\partial y^2} \right]_r^U - [f(x_i, y_i)]_r^U \right]^2\end{aligned}\quad (68)$$

Where (x_i, y_i) are points in the domain $[0, 1] \times [0, 1]$.

11. Proposed Method

In this section we will introduce a novel method to modify the fuzzy neural network. This new method based on replacing each x in the input vector (training set) $\vec{x} = (x_1, x_2, \dots, x_n)$, $x_j \in [a, b]$ by a polynomial of first degree.

In [16] Ezadi and parandin used the function: $Q(x) = \epsilon (x + 1)$, $\epsilon \in (0, 1)$

then the input vector will be: $(Q(x_1), Q(x_2), \dots, Q(x_n)), Q(x_j) \in (a, b)$. In this paper, we named this proposed method modified fuzzy neural network (MFNN). Using modified fuzzy neural network makes that training points should be selected over the open interval (a, b) without training the neural network in the range of first and end points. Therefore, the calculating volume involving computational error is reduced. In fact, the training points depending on the distance $[a, b]$ selected for training fuzzy neural network are converted to similar points in the open interval (a, b) by using a new approach, then the fuzzy network is trained in these similar areas.

12. Numerical Examples

To show the behavior and properties of the new method, two problems will be solved in this section. For each example, the accuracy of the method is illustrated by computing the deviations $[\text{error}]_r^L, [\text{error}]_r^U$ $[\text{error}]_r^L = | [U_a(x, y)]_r^L - [U_T(x, y)]_r^L |, [\text{error}]_r^U = | [U_a(x, y)]_r^U - [U_T(x, y)]_r^U |$ where

$$\begin{aligned} [U_a(x, y)]_r &= [[U_a(x, y)]_r^L, [U_a(x, y)]_r^U] && \text{the exact solution} \\ [U_T(x, y)]_r &= [[U_T(x, y)]_r^L, [U_T(x, y)]_r^U] && \text{the trial solution} \end{aligned}$$

For all examples, multilayer perceptron consisting of one hidden layer with 10 units and one linear output unit is used. To minimize the error function, we used BFGS quasi-Newton method (For more details, see [10, 11]).

Example (1) : Consider the fuzzy Poisson problem :

$$\frac{\partial^2 \tilde{U}}{\partial x^2}(x, y) + \frac{\partial^2 \tilde{U}}{\partial y^2}(x, y) = \tilde{K} x e^y, x, y \in [0, 1],$$

Where $\tilde{K}(r) = [0.75 + 0.25 r, 1.25 - 0.25 r]$ with the fuzzy conditions:

$$\tilde{U}(0, y) = 0, \tilde{U}(1, y) = \tilde{K} e^y, 0 \leq y \leq 1 \text{ and}$$

$$\tilde{U}(x, 0) = \tilde{K} x, \tilde{U}(x, 1) = \tilde{K} e x, 0 \leq x \leq 1.$$

The fuzzy analytical solution for this problem is :

$$[U_a(x, y)]_r = [(0.75 + 0.25r) x e^y, (1.25 - 0.25r) x e^y].$$

The fuzzy trial solution for this problem is :

$$[U_T(x, y)]_r = [(0.75 + 0.25r) x e^y, (1.25 - 0.25r) x e^y] \\ + xy (1 - x) (1 - y) [N(x, y, p)]_r .$$

The error function for $m = 10$ units in the hidden layer and for $g = 11$ equally spaced points inside the interval $[0, 1]$ for each variable x and y is trained. For $\epsilon = 0.4$, the training set will be :

$$0.4 (x + 1), \forall x \in [0, 1] \text{ and } 0.4 (y + 1), \forall y \in [0, 1] .$$

For this problem, the minimized error function : $E = \sum_{i=1}^{11} (E_{ir}^L + E_{ir}^U)$

$$E_{ir}^L = \left[\begin{aligned} & (y_i - y_i^2) \left(\begin{aligned} & (x_i - x_i^2) \sum_{j=1}^{10} [v_j]_r^L w_{j1}^2 s''(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & + (2 - 4x_i) \sum_{j=1}^{10} [v_j]_r^L w_{j1} s'(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & - 2 \sum_{j=1}^{10} [v_j]_r^L s(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \end{aligned} \right) \\ & + \\ & (x_i - x_i^2) \left(\begin{aligned} & (y_i - y_i^2) \sum_{j=1}^{10} [v_j]_r^L w_{j2}^2 s''(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & + (2 - 4y_i) \sum_{j=1}^{10} [v_j]_r^L w_{j2} s'(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & - 2 \sum_{j=1}^{10} [v_j]_r^L s(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \end{aligned} \right) \end{aligned} \right]^2$$

$$E_{ir}^U = \left[\begin{aligned} & (y_i - y_i^2) \left(\begin{aligned} & (x_i - x_i^2) \sum_{j=1}^{10} [v_j]_r^U w_{j1}^2 s''(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & + (2 - 4x_i) \sum_{j=1}^{10} [v_j]_r^U w_{j1} s'(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & - 2 \sum_{j=1}^{10} [v_j]_r^U s(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \end{aligned} \right) \\ & + \\ & (x_i - x_i^2) \left(\begin{aligned} & (y_i - y_i^2) \sum_{j=1}^{10} [v_j]_r^U w_{j2}^2 s''(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & + (2 - 4y_i) \sum_{j=1}^{10} [v_j]_r^U w_{j2} s'(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \\ & - 2 \sum_{j=1}^{10} [v_j]_r^U s(Q(x_i) w_{j1} + Q(y_i) w_{j2} + b_j) \end{aligned} \right) \end{aligned} \right]^2$$

Analytical and trial solutions for this problem can be found in table (1) and table (2).

Table (1) : Numerical results for example (1) , for $r = 0.5$.

x	y	$[U_a(x, y)]_r^L$	$[U_a(x, y)]_r^U$	$[U_T(x, y)]_r^L$	$[U_T(x, y)]_r^U$	$[\text{error}]_r^L$	$[\text{error}]_r^U$
0	0	0	0	0	0	0	0
0.1	0.1	0.096702455	0.124331728	0.096702461	0.124331737	6.6854 e-9	9.3281 e-9
0.2	0.2	0.213745482	0.274815620	0.213745474	0.274815626	8.5672 e-9	6.5423 e-9
0.3	0.3	0.354337937	0.455577347	0.354337929	0.455577342	8.3301 e-9	5.7664 e-9
0.4	0.4	0.522138644	0.671321113	0.522138646	0.671321109	2.4376 e-9	4.9806 e-9
0.5	0.5	0.721315555	0.927405714	0.721315505	0.927405718	5.0158 e-8	4.5307 e-9
0.6	0.6	0.956612370	1.229930190	0.956612401	1.229930266	3.1330 e-8	7.6771 e-8
0.7	0.7	1.233423533	1.585830257	1.233423539	1.585830346	6.7012 e-9	8.9137 e-8
0.8	0.8	1.557878650	2.002986836	1.557878654	2.002986840	4.8833 e-9	4.0264 e-9
0.9	0.9	1.936937450	2.490348150	1.936937457	2.490348154	7.6347 e-9	4.3421 e-9
1	1	2.378496600	3.058067057	2.378496600	3.058067057	0	0

Table (2) : Numerical results for example (1) , for $x = y = 0.7$.

r	$[U_a(x, y)]_r^L$	$[U_a(x, y)]_r^U$	$[U_T(x, y)]_r^L$	$[U_T(x, y)]_r^U$	$[\text{error}]_r^L$	$[\text{error}]_r^U$
0	1.057220171	1.762033619	1.057220173	1.762033614	2.1229 e-9	5.0107 e-9
0.1	1.092460844	1.726792947	1.092460840	1.726792941	4.7724 e-9	6.7036 e-9
0.2	1.127701516	1.691552274	1.127701508	1.691552268	8.3444 e-9	6.7939 e-9
0.3	1.162942189	1.656311602	1.162942196	1.656311611	7.0643 e-9	9.4339 e-9
0.4	1.198182861	1.621070930	1.198182810	1.621070921	5.1151 e-8	9.0052 e-9
0.5	1.233423533	1.585830257	1.233423539	1.585830346	6.7012 e-9	8.9137 e-8
0.6	1.268664206	1.550589585	1.268664248	1.550589588	4.2159 e-8	3.9211 e-9
0.7	1.303904878	1.515348912	1.303904925	1.515348894	4.7441 e-8	1.8195 e-8
0.8	1.339145550	1.480108240	1.339145556	1.480108230	6.2102 e-9	1.0122 e-8
0.9	1.374386223	1.444867568	1.374386230	1.444867571	7.7177 e-9	3.9898 e-9
1	1.409626895	1.409626895	1.409626903	1.409626898	8.0020 e-9	3.7220 e-9

Example (2) : Consider the fuzzy Wave problem :

$$\frac{\partial^2 \tilde{U}}{\partial t^2} (x, t) - 4 \frac{\partial^2 \tilde{U}}{\partial x^2} (x, t) = 0, \quad x, t \in [0, 1],$$

with the following fuzzy boundary and initial conditions:

$$\tilde{U}(0, t) = 0, \quad \tilde{U}(1, t) = 0, \quad 0, \quad \tilde{U}(x, 0) = \tilde{K} \sin(\pi x) \quad \text{and} \quad \frac{\partial}{\partial t} \tilde{U}(x, 0) = 0.$$

Where $\tilde{K}(r) = [0.75 + 0.25 r, 1.25 - 0.25 r]$.

The fuzzy analytical solution for this problem is :

$$[U_a(x, t)]_r = [(0.75 + 0.25r) \sin(\pi x) \cos(2\pi t), (1.25 - 0.25r) \sin(\pi x) \cos(2\pi t)].$$

The fuzzy trial solution for this problem is :

$$[U_T(x, t)]_r = [(0.75 + 0.25r)(1 - t^2) \sin(\pi x), (1.25 - 0.25r)(1 - t^2) \sin(\pi x)] + x(1 - x)t^2 [N(x, t, p)]_r.$$

We used the same procedure in the previous example to find the minimized error function. In [8], Allahviranloo and Kermani solved this problem by using Finite Difference method for $h=0.1$ and $k=0.001$. The max absolute error at the point $(0.1, 0.001), \forall r \in [0, 1]$ is 7.6247×10^{-6} .

For this problem, we will minimize the error function :

$$E = \sum_{i=1}^{11} (E_{ir}^L + E_{ir}^U)$$

$$E_{ir}^L = \left[\begin{aligned} & (-4t_i^2) \left((x_i - x_i^2) \sum_{j=1}^{10} [v_j]_r^L w_{j1}^2 s''(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \\ & \quad \left. + (2 - 4x_i) \sum_{j=1}^{10} [v_j]_r^L w_{j1} s'(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \\ & \quad \left. - 2 \sum_{j=1}^{10} [v_j]_r^L s(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right) \\ & \quad + \\ & (x_i - x_i^2) \left((t_i^2) \sum_{j=1}^{10} [v_j]_r^L w_{j2}^2 s''(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \\ & \quad \left. + (4t_i) \sum_{j=1}^{10} [v_j]_r^L w_{j2} s'(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \\ & \quad \left. + 2 \sum_{j=1}^{10} [v_j]_r^L s(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right) \\ & \quad + (4\pi^2(1 - t_i^2) - 2)(0.75 + 0.25r) \sin \pi x_i \end{aligned} \right]^2$$

$$E_{ir}^U = \left[\begin{aligned} & \left((x_i - x_i^2) \sum_{j=1}^{10} [v_j]_r^U w_{j1}^2 s''(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \\ & \left. (-4t_i^2) \left(+ (2 - 4x_i) \sum_{j=1}^{10} [v_j]_r^U w_{j1} s'(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \right. \\ & \quad \left. \left. - 2 \sum_{j=1}^{10} [v_j]_r^U s(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right) \right. \\ & \quad + \\ & \left. (x_i - x_i^2) \left((t_i^2) \sum_{j=1}^{10} [v_j]_r^U w_{j2}^2 s''(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \right. \\ & \quad \left. \left. + (4t_i) \sum_{j=1}^{10} [v_j]_r^U w_{j2} s'(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right. \right. \\ & \quad \left. \left. + 2 \sum_{j=1}^{10} [v_j]_r^U s(Q(x_i) w_{j1} + Q(t_i) w_{j2} + b_j) \right) \right. \\ & \quad \left. + (4\pi^2(1 - t_i^2) - 2)(1.25 - 0.25r) \sin \pi x_i \right] \end{aligned} \right]^2$$

Analytical and trial solutions for this problem can be found in tables (3) and (4).

Table (3) : Numerical results for example (2) , for x =0.1,t=0.001.

r	$[U_a(x, t)]_r^L$	$[U_a(x, t)]_r^U$	$[U_T(x, t)]_r^L$	$[U_T(x, t)]_r^U$	$[\text{error}]_r^L$	$[\text{error}]_r^U$
0	0.231758171	0.386263618	0.231758196	0.386263624	2.5029e-8	0.6369e-8
0.1	0.239483443	0.378538345	0.239483480	0.378538333	3.7617e-8	1.2021e-8
0.2	0.247208715	0.370813073	0.247208676	0.370813168	3.9641e-8	9.5407e-8
0.3	0.254933988	0.363087801	0.254933954	0.363087888	3.4237e-8	8.7311e-8
0.4	0.262659260	0.355362528	0.262659284	0.355361709	2.4999e-8	8.1995e-7
0.5	0.270384532	0.347637256	0.270384547	0.347638049	1.5216e-8	7.9307e-7
0.6	0.278109805	0.339911984	0.278110262	0.339911194	4.5775e-7	7.9054e-7
0.7	0.285835077	0.332186711	0.285835487	0.332186630	4.1010e-7	8.1042e-8
0.8	0.293560349	0.324461439	0.293560292	0.324461524	5.7922e-8	8.5142e-8
0.9	0.301285622	0.316736167	0.301285692	0.316736258	7.0355e-8	9.1373e-8
1	0.309010894	0.309010894	0.309010824	0.309010990	7.0922e-8	9.6334e-8

Table (4) : Numerical results for example (2) , for x =0.001,t=0.1.

r	$[U_a(x, t)]_r^L$	$[U_a(x, t)]_r^U$	$[U_T(x, t)]_r^L$	$[U_T(x, t)]_r^U$	$[\text{error}]_r^L$	$[\text{error}]_r^U$
---	-------------------	-------------------	-------------------	-------------------	----------------------	----------------------

0	0.001906198	0.003176997	0.001906962	0.003177081	7.6442e-7	8.4491e-8
0.1	0.001969738	0.003113457	0.001969130	0.003113543	6.0894e-7	8.6409e-8
0.2	0.002033278	0.003049917	0.002033340	0.003049851	6.2334e-8	6.6226e-8
0.3	0.002096818	0.002986377	0.002096748	0.002987041	7.0810e-8	6.6492e-7
0.4	0.002160358	0.002922837	0.002160426	0.002923537	6.8840e-8	7.0076e-7
0.5	0.002223897	0.002859297	0.002223827	0.002859920	7.0098e-8	6.2343e-7
0.6	0.002287437	0.002795757	0.002287382	0.002795806	5.5665e-8	4.9970e-8
0.7	0.002350977	0.002732217	0.002351027	0.002732264	5.0226e-8	4.7977e-8
0.8	0.002414517	0.002668677	0.002414570	0.002668647	5.3307e-8	3.0222e-8
0.9	0.002478057	0.002605137	0.002478014	0.002605161	4.3366e-8	2.4430e-8
1	0.002541597	0.002541597	0.002541638	0.002541570	4.1121e-8	2.7786e-8

13. Conclusions

In this paper, we presented a novel approach based on fuzzy neural networks for solving fuzzy partial differential equations. We demonstrate the ability of fuzzy neural networks to approximate the solutions of FPDEs. By comparing our results with the results obtained by other numerical methods, it can be observed that the proposed method yields more accurate approximations. Even better results may be possible if one uses more neurons or more training points. Moreover, after solving a FPDE the solution is obtainable at any arbitrary point in the training interval (even between training points). The main reason for using fuzzy neural networks was their applicability in function approximation. Further research is in progress to apply and extend this method to solve three-dimensional fuzzy partial differential equations.

References

- [1] S. **Abbasbandy** and T. Allahviranloo, Numerical solution of fuzzy Differential equations by Runge-Kutta method, J. Sci. Teacher Training University, 1(3), **2002**.
- [2] S. **Abbasbandy** and T. Allahviranloo, Numerical solution of fuzzy Differential equations by Taylor method, Journal of Computational Methods in Applied Mathematics, 2, 113-124, **2002**.
- [3] S. **Abbasbandy** and M. Otadi, Numerical solution of fuzzy polynomials by fuzzy neural network, Appl. Math. Comput., 181, 1084-1089, **2006**.

- [4] S. **Abbasbandy** and M. Otadi and et al., Numerical solution of a system of fuzzy polynomials by fuzzy neural network, Information sciences, 178 , 1948 – 1960 , **2008** .
- [5] T. **Allahviranloo** and N. Ahmady and et al., Numerical solution of fuzzy Differential equations by predictor- corrector method, Information Sciences, 177 , 1633-1647, **2007** .
- [6] T. **Allahviranloo** and E. Ahmady and et al., Nth- order fuzzy linear Differential equations , Information Sciences, 178 , 1309-1324, **2008** .
- [7] T. **Allahviranloo** and N. A. Kiani and et al., Solving fuzzy differential equations by differential transformation method , Information Sciences, 179 , 956-966, **2009** .
- [8] T. **Allahviranloo** and M. A. Kermani , "Numerical Methods For Fuzzy Linear Partial Differential Equations Under New Definition For Derivative", Iranian Journal of Fuzzy Systems, 7(3), 33-50 , **2010** .
- [9] B. **Bede** and S. G. Gal, Generalizations of the differentiability of fuzzy number- valued functions with applications to fuzzy differential equations, Fuzzy Sets and Systems, 151 , 581 – 599 , **2005** .
- [10] P. **Blomgren** , Numerical Optimization . Quasi-Newton Methods ; Convergence Analysis , Computational Sciences Research Center, San Diego State University , San Diego , **2014** .
- [11] P. **Blomgren** , Numerical Optimization . Quasi-Newton Methods ; Convergence Analysis , Computational Sciences Research Center, San Diego State University , San Diego , **2014** .
- [12] J.J. **Buckley** and T. Feuring , Fuzzy differential equations, Fuzzy Sets and Systems, 110 , 43 – 54 , **2000** .
- [13] A. A. **Dahalan** and M. S. Muthuvalu and et al. , Performance of HSAGE Method with Seikkala Derivative For 2-D Fuzzy Poisson Equation, Applied Mathematical Sciences , 8(18) , 885-899 , **2014** .
- [14] P. **Diamond** , Stability and periodicity in fuzzy differential equations, IEEE Trans , Fuzzy Systems , 8 , 583 – 590 , **2000** .

- [15] P. **Diamond** , Brief note on the variation of constants formula for fuzzy differential equations,Fuzzy Sets and Systems, 129 ,65 – 71 , **2002** .
- [16] S. **Ezadi** and N. Parandin , An Application of Neural Networks to Solve Ordinary Differential Equations , International Journal of Mathematical Modelling and Computations , 3(3) , 245-252 , **2013** .
- [17] D. N. **Georgiou** and J.J.Nieto, and et al . , Initial value problems for higher-order fuzzy differential equations , Nonlinear Anal. ,63,587-600,**2005** .
- [18] B. **Ghazanfari** and A. Shakerami, Numerical solution of fuzzy differential equations extended Runge – Kutta- like formulae of order 4, Fuzzy Sets and Systems, 189 , 74 – 91, **2011** .
- [19] K. **Hornick** and M. Stinchcombe, Multi-layer feed forward networks are universal approximators, Neural Networks 2 , 359-366, **1989** .
- [20] O. **Kaleva** ,Fuzzy differential equations, Fuzzy Sets and Sydtems, 24 , 301 – 317, **1987** .
- [21] A. **Kastan** and K. Ivaz , Numerical solution of fuzzy differential equations by Nystrom method,Chaos,Solitons and Fractals,41,859 – 868, **2009** .
- [22] T. **Khanna** , Foundations of neural networks, Addison-Wesly, Reading, MA, **1990** .
- [23] H. **Lee** and I. S. Kang , Neural algorithms for solving differential equations , Journal of Computational Physics,91 ,110-131 , **1990** .
- [24] A. J. **Meade** and A. A. Fernandes , Solution of Nonlinear Ordinary Differential Equations by Feed-Forward Neural Networks , Mathematical and Computer Modelling , 20(9), 19-44 , **1994** .
- [25] A. J. **Meade** and A. A. Fernandez, The numerical solution of linear ordinary differential equations by feed-forward neural network, Mathematical and Computer Modelling, 19(12) , 1 – 25 , **1994** .

- [26] M. **Mosleh** and M.Otadi and et al. , Evaluation of fuzzy regression models by fuzzy neural network,Journal of Computational and Applied Mathematics,234 , 825 – 834 , **2010** .
- [27] M. **Mosleh** and M.Otadi and et al. , Fuzzy polynomial regression with fuzzy neural network, Applied Mathematical Modeling,35, 5400 – 5412 , **2011** .
- [28] M. **Mosleh** and T. Allahviranloo, and et al. ,Evaluation of fully fuzzy regression models by fuzzy neural network, Neural Comput and Applications,21 , 105 – 112, **2012** .
- [29] M. **Mosleh** and M. Otadi, Simulation and evaluation of fuzzy differential equation by fuzzy neural network, Applied soft computing ,12 , 2817-2827 , **2012** .
- [30] M. **Mosleh** ,Fuzzy neural network for solving a system of fuzzy differential equations,Applied Soft Computing,13,3597-3607 , **2013** .
- [31] M. **Mosleh** and M.Otadi,Solving the second order fuzzy differential equations by fuzzy neural network, Journal of Mathematical Extension, 81 , 11 – 27 , **2014** .
- [32] J .J. **Nieto**, The Cauchy problem for continuous fuzzy differential equations , Fuzzy Sets and Systems,102 , 259 – 262 , **1999** .
- [33] J.J. **Nieto** and R. Lopez, Bounded solutions for fuzzy differential and integral equations,Chaos, Solitons and Fractals, 27, 1376 – 1386, **2006** .
- [34] M. **Otadi** and M. Mosleh and et al., solving fuzzy linear system by neural network and applications in Economics, Journal of mathematics extension, 47-66 , **2011** .
- [35] M. **Otadi** and M.Mosleh and et al ., Numerical solution of fully fuzzy linear systems by fuzzy neural network,Soft Computing, 15 , 1513 – 1522, **2011** .
- [36] H . **Ouyang** and Y . Wu, On fuzzy differential equations, Fuzzy Sets and Systems, 32 , 321 – 325 , **1989** .