Journal of Mathematical Extension
Vol. 8, No. 1, (2014), 11-27

Solving the Second Order Fuzzy Differential Equations by Fuzzy Neural Network

M. Mosleh[*]
Islamic Azad University-Firoozkooh Branch

M. Otadi
Islamic Azad University-Firoozkooh Branch

Abstract. In this paper, we interpret a two-point initial value problem for a second order fuzzy differential equation. We investigate a problem of finding a numerical approximation of the solution by using fuzzy neural network. Here neural network is considered as a part of a larger field called neural computing or soft computing. Finally, we illustrate our approach on an applied example in engineering.

AMS Subject Classification: 34K28; 97N40; 62M45; 82C32
Keywords and Phrases: Second order fuzzy differential equation, fuzzy neural networks, learning algorithm

1. Introduction

The study of fuzzy differential equations (FDEs) forms a suitable setting for mathematical modeling of real-world problems in which uncertainties or vagueness pervade. There are several approaches to the study of FDEs [4, 5, 23]. The concept of fuzzy derivative was first introduced by Chang and Zadeh [7], it was followed up by Dubois and Prade [8] who used the extension principle in their approach. Other methods have been discussed by Puri and Ralescu [26] and by Goetschel and Voxman [9]. FDEs were first formulated by Kaleva [12] and Seikkala [28] in time

Received: April 2013; Accepted: October 2013
[*]Corresponding author

dependent form. Kaleva had formulated FDEs, in terms of Hukuhara derivative [9].
In 1990 Lee and Kang [14] used parallel processor computers to solve a first order differential equation with Hopfield neural network models. Meade, Fernandez and Malek [15, 16] solved linear and nonlinear ordinary differential equations using feed forward neural networks architecture and B_1-splines. Recently, fuzzy neural networks have been successfully used for solving fuzzy polynomial equations and systems of fuzzy polynomial equations [1, 2], approximate fuzzy coefficients of fuzzy regression models [19, 20, 21], approximate solution of fuzzy linear systems and fully fuzzy linear systems [24, 25]. In year 2012 Mosleh and Otadi [22] used fuzzy neural network to solve a first order FDE, system of FDEs [17] and fuzzy linear Fredholm integro-differential equation [18].
In this paper we propose a method for approximate solution of a second order FDE using innovative mathematical tools and neural-like systems of computation. This hybrid method can result in improved numerical methods for solving fuzzy initial value problems. In this proposed method, fuzzy neural network model (FNNM) is applied as a universal approximator. The main aim of this paper is to illustrate how fuzzy connection weights are adjusted in the learning of fuzzy neural networks by the back-propagation-type learning algorithms [11] for the approximate solution of a second FDE.

2. Preliminaries

In this section the most basic notations used in fuzzy calculus are introduced. We start by defining the fuzzy number.

Definition 2.1. *A fuzzy number u is a fuzzy subset of the real line with a normal, convex and upper semicontinuous membership function of bounded support.*
The set of all the fuzzy numbers (as given by Definition. 1) is denoted by E^1.

Definition 2.2 (See [9]) *Let $u, v \in E^1$. If there exists $w \in E^1$ such that*

$u = v + w$ then w is called the H-difference of u, v and it is denoted by $u - v$.

Definition 2.3. *A function $f : (a, b) \longrightarrow E^1$ is called H-differentiable at $\hat{t} \in (a, b)$ if, for $h > 0$ sufficiently small, there exist the H-differences $f(\hat{t} + h) - f(\hat{t}), f(\hat{t}) - f(\hat{t} - h)$, and an element $f'(\hat{t}) \in E^1$ such that:*

$$lim_{h \longrightarrow 0^+} D(\frac{f(\hat{t} + h) - f(\hat{t})}{h}, f'(\hat{t})) = lim_{h \longrightarrow 0^+} D(\frac{f(\hat{t}) - f(\hat{t} - h)}{h}, f'(\hat{t})) = 0.$$

Then $f'(\hat{t})$ is called the fuzzy derivative of f at $\hat{t}$.

In this paper, we denote real numbers and fuzzy numbers by lowercase letters ($e.g., a, b, c, \dots$) and uppercase letters ($e.g., A, B, C, \dots$), respectively. We briefly mention fuzzy number operations defined by the extension principle [30]:

$$\mu_{A+B}(z) = max\{\mu_A(x) \wedge \mu_B(y) | z = x + y\},$$

$$\mu_{f(Net)}(z) = max\{\mu_{Net}(x) | z = f(x)\},$$

where A, B, Net are fuzzy numbers, $\mu_*(.)$ denotes the membership function of each fuzzy number, $\wedge$ is the minimum operator and $f(.)$ is a continuous activation function (like sigmoidal activation function) inside hidden neurons

$$f(x) = \frac{1}{1 + e^{-x}}.$$

The above operations of fuzzy numbers are numerically performed on level sets (i.e.,α-cuts). The h-level set of a fuzzy number A is defined as

$$[A]_h = \{x \in \mathbb{R} | \mu_A(x) \geqslant h\} \quad for \quad 0 < h \leqslant 1, \tag{1}$$

and $[A]_0 = \overline{\bigcup_{h \in (0,1]} [A]_h}$. Since level sets of fuzzy numbers are closed intervals, we denote $[A]_h$ as

$$[A]_h = [[A]_h^L, [A]_h^U], \tag{2}$$

where $[A]_h^L$ and $[A]_h^U$ are the lower limit and the upper limit of the h-level set $[A]_h$, respectively.

From interval arithmetic [3], the above operations of fuzzy numbers are written for h-level sets as follows:

$$[A]_h + [B]_h = [[A]_h^L + [B]_h^L, [A]_h^U + [B]_h^U], \tag{3}$$

$$k.[A]_h = \begin{cases} [k[A]_h^L, k[A]_h^U], & k \geqslant 0, \\ [k[A]_h^U, k[A]_h^L], & k < 0, \end{cases} \tag{4}$$

$$f([Net]_h) = f([[Net]_h^L, [Net]_h^U]) = [f([Net]_h^L), f([Net]_h^U)], \tag{5}$$

where f is a sigmoidal activation function.

3. Second Order Fuzzy Differential Equations

Now, we consider the following second order fuzzy differential equation

$$\begin{cases} Y'' = f(x, Y, Y'), & x \in [a, b], \\ Y(a) = A_1, \ Y'(a) = A_2, \end{cases} \tag{6}$$

such that the functions
$Y : [a, b] \to E$ and $f : [a, b] \times E \times E \to E$
where Y is a function with fuzzy derivative Y' [26] also A_1 and A_2 are fuzzy numbers in E with h-level sets

$$[A_1]_h = [[A_1]_h^L, [A_1]_h^U], \ [A_2]_h = [[A_2]_h^L, [A_2]_h^U], \quad 0 < h \leqslant 1.$$

Let us assume that a general approximate solution to Eq.(6) is in the form $Y_T(x, P)$ for Y_T depending to x and P, where P is an adjustable parameter involving weights and biases in the structure of the three-layered feed forward FNN (see Fig. 1). The fuzzy trial solution Y_T is an approximation solution to Y for the optimized value of unknown weights and biases. Thus the problem of finding the approximated fuzzy solution for Eq. 6) over some collocation points in $[a, b]$ by a set of discrete equally spaced grid points

$$a = x_1 < x_2 < \ldots < x_g = b,$$

is equivalent to calculating the functional $Y_T(x, P)$.

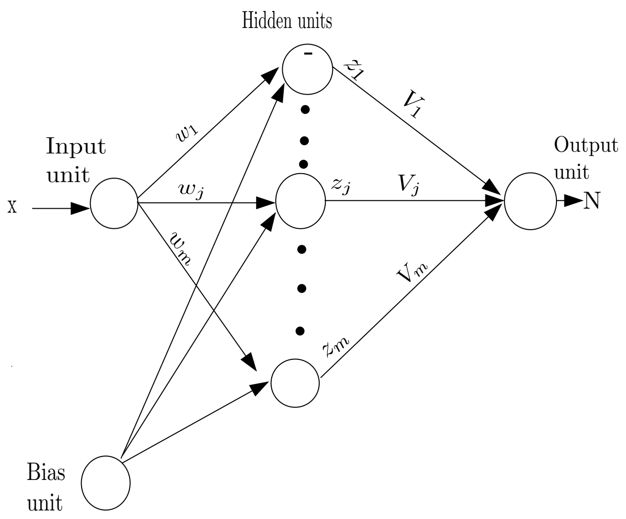

Fig. 1. Three layer fuzzy neural network

In order to obtain fuzzy approximate solution $Y_T(x, P)$, we solve unconstrained optimization problem that is simpler to deal with. So we define the fuzzy trial function to be in the following form:

$$Y_T(x, P) = \alpha(x) + \beta[x, N(x, P)], \tag{7}$$

where the first term in the right hand side does not involve the adjustable parameters and satisfies the fuzzy initial conditions while the second term is a feed forward fuzzy neural network consisting of an input x and the output $N(x, P)$.

One drawback of fully fuzzy neural networks in connection with fuzzy weights is the long computation time. Another drawback is that the learning algorithm is complicated. For reducing the complexity of the learning algorithm, we propose a partially fuzzy neural network (PFNN) with three-layer architecture where connection weights to output unit are fuzzy numbers while connection weights and biases to hidden units are real numbers [22]. Since we had good simulation results even from partially fuzzy three-layer neural networks, we do not think that the extension of our learning algorithm to neural networks with more than three layer is an attractive research direction.

Let us consider a three-layered PFNNM (see Fig. 1) with one unit entry x, one hidden layer consisting of m activation functions and one unit output $N(x,P)$. Here, the dimension of PFNNM is denoted by the number of neurons in each layer, that 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

$$net_j = x.w_j + b_j, \quad j = 1, \ldots, m, \tag{8}$$

where w_j is a weight parameter from input layer to the jth unit in the hidden layer, b_j is the jth unit in the hidden layer. The output, in the hidden neurons is

$$z_j = s(net_j), \quad j = 1, \ldots, m, \tag{9}$$

where s is the activation function which is normally a nonlinear function, the usual choices for which [10] are the sigmoid transfer function, and the output neuron make no change on its input, so the input to the output neuron is equal to output

$$N = V_1 z_1 + \ldots + V_j z_j + \ldots + V_m z_m, \tag{10}$$

where V_j is a weight parameter from jth unit in the hidden layer to the output layer.
From Eqs.(3)-(5), we can see that the h-level sets of the functions (8)-(10) can be calculated from those of the weights and biases. For our PFNN, we can derive the learning algorithm :
Input unit:

$$o = x. \tag{11}$$

Hidden units:

$$z_j = s(o.w_j + b_j), \quad j = 1, \ldots, m. \tag{12}$$

Output unit:

$$[N]_h = [[N]_h^L, [N]_h^U] = [\sum_{j=1}^{m} [V_j]_h^L . z_j, \sum_{j=1}^{m} [V_j]_h^U . z_j]. \tag{13}$$

A FNN_4 (fuzzy neural network with crisp set input signals, fuzzy number weights and fuzzy number output) solution to Eq.(6) is given in Figure 1. How is the FNN_4 going to solve the fuzzy differential equations?

The training data are $a = x_1 < x_2 < \ldots < x_g = b$ for input. We propose a learning algorithm from the cost function for adjusting weights.
Consider the following fuzzy initial value problem for a second order differential equation (6), the related trial function will be in the form

$$Y_T(x, P) = A_1 + A_2(x-a) + (x-a)^2 N(x, P), \tag{14}$$

this solution by intention satisfies the initial condition in (6). In [11], the learning of our fuzzy neural network is to minimize the difference between the fuzzy target vector $B = (B_1, \ldots, B_s)$ and the actual fuzzy output vector $O = (O_1, \ldots, O_s)$. The following cost function was used in [11, 1] for measuring the difference between B and O:

$$e = \sum_h e_h = \sum_h h.\{\sum_{k=1}^{s}([B_k]_h^L - [O_k]_h^L)^2/2 + \sum_{k=1}^{s}([B_k]_h^U - [O_k]_h^U)^2/2\}, \tag{15}$$

where e_h is the cost function for the h-level sets of B and O. The squared errors between the h-level sets of B and O are weighted by the value of h in (15).
In [11], it is shown by computer simulations that, the fitting of fuzzy outputs to fuzzy targets is not good for the h-level sets with small values of h when we use the cost function in (15). This is because the squared errors for the h-level sets are weighted by h in (15). Krishnamraju et al. [13] used the cost function without the weighting scheme:

$$e = \sum_h e_h = \sum_h \{\sum_{k=1}^{s}([B_k]_h^L - [O_k]_h^L)^2/2 + \sum_{k=1}^{s}([B_k]_h^U - [O_k]_h^U)^2/2\}. \tag{16}$$

In the computer simulations included in this paper, we mainly use the cost function in (16) without the weighting scheme.
The error function that must be minimized for problem (6) is in the form

$$e = \sum_{i=1}^{g} e_i = \sum_{i=1}^{g}\sum_h e_{ih} = \sum_{i=1}^{g}\sum_h \{e_{ih}^L + e_{ih}^U\}, \tag{17}$$

where

$$e_{ih}^L = \frac{([Y_T''(x_i, P)]_h^L - [f(x_i, Y_T(x_i, P), Y_T'(x_i, P))]_h^L)^2}{2}, \tag{18}$$

$$e_{ih}^U = \frac{([Y_T''(x_i,P)]_h^U - [f(x_i, Y_T(x_i,P), Y_T'(x_i,P))]_h^U)^2}{2}, \tag{19}$$

where $\{x_i\}_{i=1}^g$ are discrete collocation points belonging to the interval $[a,b]$ and in the cost function (17), e_{ih}^L and e_{ih}^U can be viewed as the square errors for the lower limits and the upper limits of the h-level sets, respectively.

It is easy to express the first partial derivative of $N(x,P)$ in terms of the partial derivative of the sigmoid function, i.e.

$$\frac{\partial[N]_h^L}{\partial x} = \sum_{j=1}^m [V_j]_h^L . \frac{\partial z_j}{\partial net_j} . \frac{\partial net_j}{\partial x} = \sum_{j=1}^m [V_j]_h^L . z_j . (1-z_j) . w_j, \tag{20}$$

$$\frac{\partial[N]_h^U}{\partial x} = \sum_{j=1}^m [V_j]_h^U . \frac{\partial z_j}{\partial net_j} . \frac{\partial net_j}{\partial x} = \sum_{j=1}^m [V_j]_h^U . z_j . (1-z_j) . w_j. \tag{21}$$

$$\frac{\partial^2[N]_h^L}{\partial x^2} = \sum_{j=1}^m [V_j]_h^L . (2z_j^3 - 3z_i^2 + z_j) . w_j^2, \tag{22}$$

$$\frac{\partial^2[N]_h^U}{\partial x^2} = \sum_{j=1}^m [V_j]_h^U . (2z_j^3 - 3z_i^2 + z_j) . w_j^2. \tag{23}$$

Now differentiating trial function $Y_T(x,P)$, we obtain

$$[Y_T'(x,P)]_h^L = [A_2]_h^L + 2(x-a)[N(x,P)]_h^L + (x-a)^2 . \frac{\partial[N(x,P)]_h^L}{\partial x},$$

$$[Y_T'(x,P)]_h^L = [A_2]_h^U + 2(x-a)[N(x,P)]_h^U + (x-a)^2 . \frac{\partial[N(x,P)]_h^U}{\partial x},$$

$$[Y_T''(x,P)]_h^L = 2[N(x,P)]_h^L + 4(x-a)\frac{\partial[N(x,P)]_h^L}{\partial x} + (x-a)^2 . \frac{\partial^2[N(x,P)]_h^L}{\partial x^2},$$

$$[Y_T''(x,P)]_h^U = 2[N(x,P)]_h^U + 4(x-a)\frac{\partial[N(x,P)]_h^U}{\partial x} + (x-a)^2 . \frac{\partial^2[N(x,P)]_h^U}{\partial x^2},$$

thus the expressions in equations (20)-(23) are applicable here. A learning algorithm is derived in Appendix A.

Let us consider the vibrating mass in engineering.

Example 3.1. [6]. An example in engineering application
Consider the vibrating mass in Fig. 2. The mass $m = 1$ slug, the spring constant $k = 4$ lbs/ft and there is no, or negligible, damping. The forcing function is $2cos(\beta x)$, for $0 < \beta < 2$. The differential equation of motion is

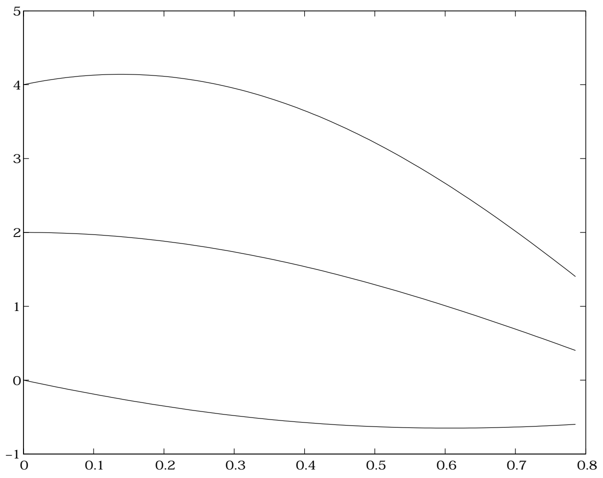

Fig. 2. Vibrating Mass in application

$$Y'' + 4Y = 2cos(\beta x),$$

subject to initial conditions $Y(0) = A_1,\ Y'(0) = A_2$. The unique solution is

$$Y = A_1 cos(2x) + \frac{A_2}{2} sin(2x) + \Phi(x),$$

where

$$\Phi(x) = \frac{2}{4 - \beta^2}[cos(\beta x) - cos(2x)].$$

The uncertain initial conditions are $[A_1]_h = [2h, 4 - 2h],\ [A_2]_h = [-2 + 2h, 2 - 2h]$. The graph of $Y(x)$, $h = 0$ and $h = 1$ cuts, for $\beta = 1.9$, is in Figure 3 (the outside curves are the $h = 0$ cut and the center curve the $h = 1$ cut).

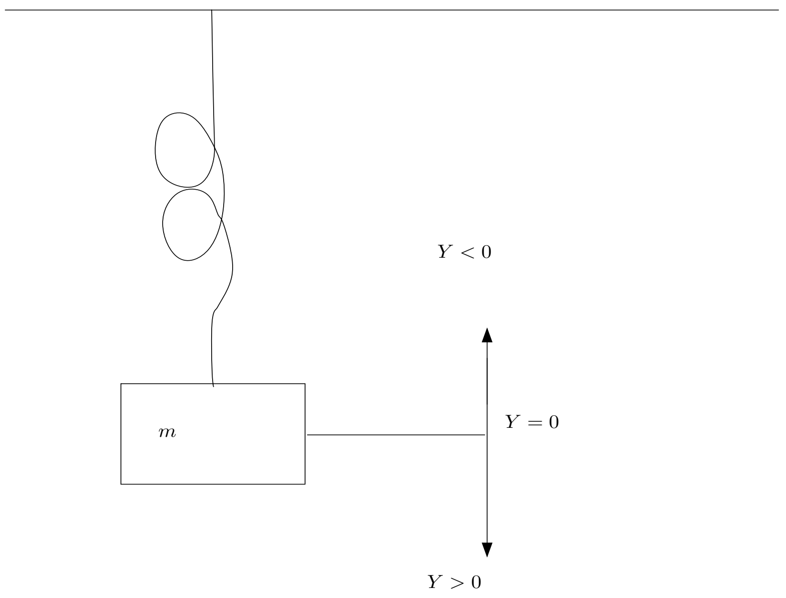

Fig. 3. Extension principle solution in the Vibrating Mass application

The fuzzy trial function for this problem is

$$[Y_T(x, P)]_h = [2h, 4 - 2h] + x[-2 + 2h, 2 - 2h] + x^2.[N(x, P)]_h.$$

Here, the dimension of PFNNM is $1 \times 5 \times 1$. The error function for the $m = 5$ sigmoid units in the hidden layer and for $g = 4$ equally spaced points inside the interval $[0, \frac{\pi}{4}]$ is trained. In the computer simulation of this section, we use the following specifications of the learning algorithm.

(1) Number of hidden units: five units.
(2) Stopping condition: 200 iterations of the learning algorithm.
(3) Learning constant: $\eta = 0.2$.
(4) Momentum constant: $\alpha = 0.1$.
(5) Initial value of the weights and biases of PFNNM are shown in Table 1, and we suppose $V_i = (v_i^{(1)}, v_i^{(2)}, v_i^{(3)})$ for $i = 1, \ldots, 5$.
We apply the proposed method to the approximate realization of the solution of example 1. Weights from the trained FNN are shown in Table 2. The exact and obtained solution of second order FDE in this example at $x = \frac{\pi}{8}$ are shown in Figure 4.

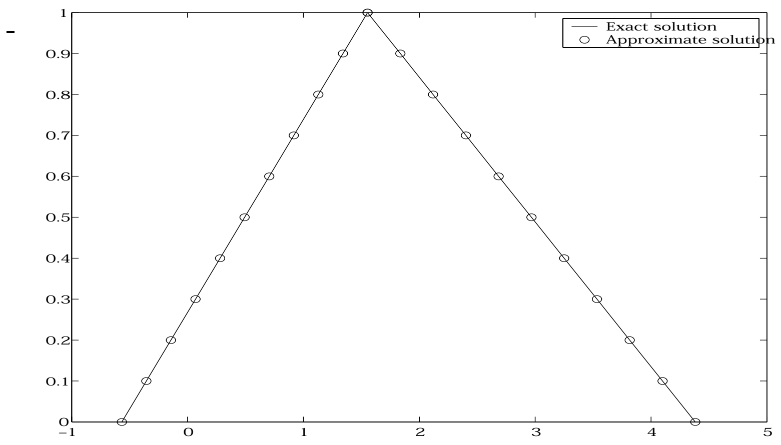

Fig. 4. Compares the exact solution and obtained solution

i	1	2	3	4	5
$v_i^{(1)}$	-0.5	-0.5	-0.5	-0.5	-0.5
$v_i^{(2)}$	0	0	0	0	0
$v_i^{(3)}$	0.5	0.5	0.5	0.5	0.5
w_i	0	0	0	0	0
b_i	0	0	0	0	0

Table 1. The initial values of weights

4. Summary and Conclusions

We investigated the second order FDEs. Solving the second order FDEs (FDEs) by using universal approximators (UA), that is, FNNM is presented in this paper. In this paper, we derived a learning algorithm of fuzzy weights of three-layer feedforward FNN whose input-output relations were defined by extension principle. The use of more general network architectures, however, makes the back-propagation-type learning algorithm much more complicated. Since we had good simulation result even from partially fuzzy three-layer neural networks, we do not think that the extension of our learning algorithm to neural networks with more than three layers is an attractive research direction. Good simulation result was obtained by this neural network in shorter computation times than fully fuzzy neural networks in our computer simulations. The effectiveness of the derived learning algorithm was demonstrated by computer simulation on numerical examples and we proposed applied examples in engineering.

Appendix

Derivation of a learning algorithm in PFNN

Let us denote the fuzzy connection weight V_j to the output unit by its parameter values as $V_j = (v_j^{(1)}, \ldots, v_j^{(q)}, \ldots, v_j^{(r)})$. The amount of modification of each parameter value is written as [**?**, 27]

$$v_j^{(q)}(t+1) = v_j^{(q)}(t) + \triangle v_j^{(q)}(t),$$

$$\triangle v_j^{q}(t) = -\eta \sum_{i=1}^{g} \frac{\partial e_{ih}}{\partial v_j^{(q)}} + \alpha . \triangle v_j^{(q)}(t-1),$$

where t indexes the number of adjustments, η is a learning rate (a positive real number) and α is a momentum constant term (positive real number).
Thus our problem is to calculate the derivatives $\frac{\partial e_{ih}}{\partial v_j^{(q)}}$. Let us rewrite $\frac{\partial e_{ih}}{\partial v_j^{(q)}}$ as follows:

$$\frac{\partial e_{ih}}{\partial v_j^{(q)}} = \frac{\partial e_{ih}}{\partial [V_j]_h^L} . \frac{\partial [V_j]_h^L}{\partial v_j^{(q)}} + \frac{\partial e_{ih}}{\partial [V_j]_h^U} . \frac{\partial [V_j]_h^U}{\partial v_j^{(q)}} .$$

In this formulation, $\frac{\partial [V_j]_h^L}{\partial v_j^{(q)}}$ and $\frac{\partial [V_j]_h^U}{\partial v_j^{(q)}}$ are easily calculated from the membership functions of the fuzzy connection weight V_j. For example, when the fuzzy connection weight V_j is trapezoidal (i.e., $V_j = (v_j^{(1)}, v_j^{(2)}, v_j^{(3)}, v_j^{(4)})$), $\frac{\partial [V_j]_h^L}{\partial v_j^{(q)}}$ and $\frac{\partial [V_j]_h^U}{\partial v_j^{(q)}}$ are calculated as follows:

$$\frac{\partial [V_j]_h^L}{\partial v_j^{(1)}} = 1 - h, \qquad \frac{\partial [V_j]_h^U}{\partial v_j^{(1)}} = 0,$$

$$\frac{\partial[V_j]_h^L}{\partial v_j^{(2)}} = h, \quad \frac{\partial[V_j]_h^U}{\partial v_j^{(2)}} = 0,$$
$$\frac{\partial[V_j]_h^L}{\partial v_j^{(3)}} = 0, \quad \frac{\partial[V_j]_h^U}{\partial v_j^{(3)}} = h,$$
$$\frac{\partial[V_j]_h^L}{\partial v_j^{(4)}} = 0, \quad \frac{\partial[V_j]_h^U}{\partial v_j^{(4)}} = 1-h.$$

These derivatives are calculated from the following relation between the h-level set of the fuzzy connection weight V_j and its parameter values:

$$[V_j]_h^L = (1-h).v_j^{(1)} + h.v_j^{(2)},$$

$$[V_j]_h^U = h.v_j^{(3)} + (1-h).v_j^{(4)}.$$

When the fuzzy connection weight V_j is a symmetric triangular fuzzy number and the following relations hold for its h-level set $[V_j]_h = [[V_j]_h^L, [V_j]_h^U]$:

$$[V_j]_h^L = (1-\frac{h}{2}).v_j^{(1)} + \frac{h}{2}.v_j^{(3)},$$

$$[V_j]_h^U = \frac{h}{2}.v_j^{(1)} + (1-\frac{h}{2}).v_j^{(3)}.$$

Therefore,

$$\frac{\partial[V_j]_h^L}{\partial v_j^{(1)}} = 1-\frac{h}{2}, \quad \frac{\partial[V_j]_h^U}{\partial v_j^{(1)}} = \frac{h}{2},$$
$$\frac{\partial[V_j]_h^L}{\partial v_j^{(3)}} = \frac{h}{2}, \quad \frac{\partial[V_j]_h^U}{\partial v_j^{(3)}} = 1-\frac{h}{2},$$

and $v_j^{(2)}(t+1)$ is updated by the following rule:

$$v_j^{(2)}(t+1) = \frac{v_j^{(1)}(t+1) + v_j^{(3)}(t+1)}{2}.$$

On the other hand, the derivatives $\frac{\partial e_{ih}}{\partial[V_j]_h^L}$ and $\frac{\partial e_{ih}}{\partial[V_j]_h^U}$ are independent of the shape of the fuzzy connection weight. They can be calculated

from the cost function e_{ih} using the input-output relations of our fuzzy neural network for the h-level sets. When we use the cost function with the weighting scheme in (17), $\frac{\partial e_{ih}}{\partial [V_j]_h^L}$ and $\frac{\partial e_{ih}}{\partial [V_j]_h^U}$ are calculated as follows:
[Calculation of $\frac{\partial e_{ih}}{\partial [V_j]_h^L}$]

$$\frac{\partial e_{ih}}{\partial [V_j]_h^L} = \delta^L.(2\frac{\partial [N(x_i,P)]_h^L}{\partial [V_j]_h^L} + 4(x_i-a).\frac{\partial z_j}{\partial x} +(x_i-a)^2.(2z_j^3-3z_j^2+z_j).w_j^2$$

$$-\frac{\partial [f(x,Y_T(x_i,P),Y'_T(x_i,P))]_h^L}{\partial [Y_T(x_i,P)]_h^L}.\frac{\partial [Y_T(x_i,P))]_h^L}{\partial [V_j]_h^L}) - \frac{\partial [f(x,Y_T(x_i,P),Y'_T(x_i,P))]_h^L}{\partial [Y'_T(x_i,P)]_h^L}.\frac{\partial [Y'_T(x_i,P))]_h^L}{\partial [V_j]_h^L}),$$

where

$$\delta^L = ([Y''_T(x_i,P)]_h^L - [f(x_i,Y_T(x_i,P),Y'_T(x_i,P))]_h^L),$$

$$\frac{\partial [Y_T(x_i,P))]_h^L}{\partial [V_j]_h^L} = (x_i-a)^2.z_j, \quad \frac{\partial N(x_i,P)]_h^L}{\partial [V_j]_h^L} = z_j.$$

$$\frac{\partial [Y'_T(x_i,P))]_h^L}{\partial [V_j]_h^L} = 2(x_i-a).z_j + (x_i-a)^2.z_j(1-z_j).w_j,$$

[Calculation of $\frac{\partial e_{ih}}{\partial [V_j]_h^U}$]

$$\frac{\partial e_{ih}}{\partial [V_j]_h^U} = \delta^U.(2\frac{\partial [N(x_i,P)]_h^U}{\partial [V_j]_h^U} + 4(x_i-a).\frac{\partial z_j}{\partial x} + (x_i-a)^2.(2z_j^3-3z_j^2+z_j).w_j^2$$

$$-\frac{\partial [f(x,Y_T(x_i,P),Y'_T(x_i,P))]_h^U}{\partial [Y_T(x_i,P)]_h^U}.\frac{\partial [Y_T(x_i,P))]_h^U}{\partial [V_j]_h^U}) - \frac{\partial [f(x,Y_T(x_i,P),Y'_T(x_i,P))]_h^U}{\partial [Y'_T(x_i,P)]_h^U}.\frac{\partial [Y'_T(x_i,P))]_h^U}{\partial [V_j]_h^U}),$$

where

$$\delta^U = ([Y''_T(x_i,P)]_h^U - [f(x_i,Y_T(x_i,P),Y'_T(x_i,P))]_h^U),$$

$$\frac{\partial [Y_T(x_i,P))]_h^U}{\partial [V_j]_h^U} = (x_i-a)^2.z_j, \quad \frac{\partial N(x_i,P)]_h^U}{\partial [V_j]_h^U} = z_j.$$

$$\frac{\partial [Y'_T(x_i,P))]_h^U}{\partial [V_j]_h^U} = 2(x_i-a).z_j + (x_i-a)^2.z_j(1-z_j).w_j,$$

In our PFNN, the connection weights and biases to the hidden units are real numbers. The non-fuzzy connection weight w_j to the jth hidden unit and the non-fuzzy bias b_j to the jth hidden unit is updated in the same manner as the parameter values of the fuzzy connection weight $[V_j]_1$.

References

[1] S. Abbasbandy and M. Otadi, Numerical solution of fuzzy polynomials by fuzzy neural network, *Appl. Math. Comput.*, 181 (2006), 1084-1089.

[2] S. Abbasbandy, M. Otadi, and M. Mosleh, Numerical solution of a system of fuzzy polynomials by fuzzy neural network, *Information Sciences,* 178 (2008), 1948-1960.

[3] G. Alefeld and J. Herzberger, *Introduction to Interval Computations,* Academic Press, New York, 1983.

[4] T. Allahviranloo, E. Ahmady, and N. Ahmady, Nth-order fuzzy linear differential eqations, *Information Sciences,* 178 (2008), 1309-1324.

[5] B. Bede, I. J. Rudas, and A. L. Bencsik, First order linear fuzzy differential eqations under generalized differentiability, *Information Sciences,* 177 (2007), 1648-1662.

[6] J. J. Buckley and Th. Feuring, Fuzzy initial value problem for N-th order linear differential equation, *Fuzzy Sets and Systems,* 121 (2001), 247-255.

[7] S. L. Chang and L. A. Zadeh, On fuzzy mapping and control, *IEEE Trans. Systems Man Cybemet.*, 2 (1972), 30-34.

[8] D. Dubois and H. Prade, Towards fuzzy differential calculus: Part 3, differentiation, *Fuzzy Sets and Systems,* 8 (1982), 225-233.

[9] R. Goetschel and W. Voxman, Elementary fuzzy calculus, *Fuzzy Sets and Systems,* 18 (1986), 31-43.

[10] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design,* PWS publishing company, Massachusetts, 1996.

[11] H. Ishibuchi, K. Morioka, and I. B. Turksen, Learning by fuzzified neural networks, *Int. J. Approximate Reasoning*, 13 (1995), 327-358.

[12] O. Kaleva, Fuzzy differential equations, *Fuzzy Sets and Systems,* 24 (1987), 301-317.

[13] P. V. Krishnamraju, J. J. Buckley, K. D. Relly, and Y. Hayashi, Genetic learning algorithms for fuzzy neural nets, *Proceedings of IEEE International Conference on Fuzzy Systems,* (1994), 1969-1974.

[14] H. Lee and I. S. Kang, Neural algorithms for solving differential equations, *Journal of Computational Physics,* 91 (1990), 110-131.

[15] A. Malek and R. Shekari Beidokhti, Numerical solution for high order differential equations using a hybrid neural network-Optimization method, *Appl. Math. Comput.*, 183 (2006), 260-271.

[16] A. J. Meade and A. A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, *Mathematical and Computer Modelling,* 19 (12) (1994), 1-25.

[17] M. Mosleh, Fuzzy neural network for solving a system of fuzzy differential equations,*Applied Soft Computing,* 13 (2013), 3597-3607.

[18] M. Mosleh, Numerical solution of fuzzy linear Fredholm integro-differential equation by fuzzy neural network, *Iranian journal of fuzzy systems,* 11 (2014), 91-112.

[19] M. Mosleh, M. Otadi, and S. Abbasbandy, Evaluation of fuzzy regression models by fuzzy neural network, *Journal of Computational and Applied Mathematics,* 234 (2010), 825-834.

[20] M. Mosleh, M. Otadi, and S. Abbasbandy, Fuzzy polynomial regression with fuzzy neural networks, *Applied Mathematical Modelling,* 35 (2011), 5400-5412.

[21] M. Mosleh, T. Allahviranloo, and M. Otadi, Evaluation of fully fuzzy regression models by fuzzy neural network, *Neural Comput and Applications,* 21 (2012), 105-112.

[22] M. Mosleh and M. Otadi, Simulation and evaluation of fuzzy differential equations by fuzzy neural network, *Applied Soft Computing,* 12 (2012), 2817-2827.

[23] M. Mosleh and M. Otadi, Minimal solution of fuzzy linear system of differential equations, *Neural Computing and Applications,* 21 (2012), 329-336.

[24] M. Otadi and M. Mosleh, Simulation and evaluation of dual fully fuzzy linear systems by fuzzy neural network, *Applied Mathematical Modelling,* 35 (2011), 5026-5039.

[25] M. Otadi, M. Mosleh, and S. Abbasbandy, Numerical solution of fully fuzzy linear systems by fuzzy neural network, *Soft Computing,* 15 (2011), 1513-1522.

[26] M. L. Puri and D. A. Ralescu, Differentials of fuzzy functions, *J. Math. Anal. Appl.*, 91 (1983), 552-558.

[27] D. E. Rumelhart, J. L. McClelland, The PDP Research Group, *Parallel Distributed Processing,* Vol. 1, MIT Press, Cambridge, MA, 1986.

[28] S. Seikkala, On the fuzzy initial value problem, *Fuzzy Sets and Systems,* 24 (1987), 319-330.

[29] M. Wu Congxin and A. Ming, On embedding problem of fuzzy number space, Part 1, *Fuzzy Sets and Systems,* 44 (1991), 33-38.

[30] L. A. Zadeh, The concept of a liguistic variable and its application to approximate reasoning: Parts 1-3, *Information Sciences,* 8 (1975), 199-249, 301-357; 9 (1975), 43-80.

Maryam Mosleh
Department of Mathematics
Assistant Professor of Mathematics
Firoozkooh Branch-Islamic Azad University
Firoozkooh, Iran
E-mail: mosleh@iaufb.ac.ir

Mahmood Otadi
Department of Mathematics
Assistant Professor of Mathematics
Firoozkooh Branch-Islamic Azad University
Firoozkooh, Iran
E-mail: otadi@iaufb.ac.ir