NUMERICAL SOLUTION OF FUZZY LINEAR FREDHOLM INTEGRO-DIFFERENTIAL EQUATION BY FUZZY NEURAL NETWORK

M. MOSLEH

Abstract. In this paper, a novel hybrid method based on learning algorithm of fuzzy neural network and Newton-Cotes methods with positive coefficient for the solution of linear Fredholm integro-differential equation of the second kind with fuzzy initial value is presented. Here neural network is considered as a part of large field called neural computing or soft computing. We propose a learning algorithm from the cost function for adjusting fuzzy weights. This paper is one of the first attempts to derive learning algorithms from fuzzy neural networks with real input, fuzzy output, and fuzzy weights. Finally, we illustrate our approach by numerical examples.

1. Introduction

Proper design for engineering applications requires detailed information of system-property distributions such as temperature, velocity, density, etc. in space and time domain. This information can be obtained by either experimental measurement or computational simulation [23,39,41,42,50]. Although experimental measurement is reliable, it needs a lot of labor efforts and time. Therefore, the computational simulation has become a more and more popular method as a design tool since it needs only a fast computer with a large memory. Frequently, those engineering design problems deal with a set of linear Fredholm integro-differential equations (LFIDEs), which have to numerically solved such as heat transfer, solid and fluid mechanics.

When a physical problem is transformed into a deterministic linear Fredholm integro-differential equation with initial value

$$\begin{cases} \frac{dy(x)}{dx} = g(x) + \lambda \int_a^b k(x,t)y(t)dt, \\ y(a) = A, \end{cases} \tag{1}$$

we usually cannot be sure that this modelling is perfect. The initial value may not be known exactly and the function g may contain unknown parameters. If the nature of errors is random, then instead of a deterministic problem (1) we get a random integro-differential equation with random initial value. But if the underlying structure is not probabilistic, e.g. because of subjective choice, then it may be appropriate to use fuzzy numbers instead of real random variables.

Received: October 2011; Revised: January 2012 and June 2012; Accepted: May 2013

Key words and phrases: Fuzzy neural networks, Fuzzy linear Fredholm integro-differential, Feedforward neural network, Learning algorithm.

The topic of fuzzy differential equations (FDEs) and fuzzy integral equations (FIEs) have been rapidly growing in recent years. The fuzzy initial value problem have been studied by several authors [5,6,10,12,51,53,54,63]. The concept of fuzzy derivative was first introduced by Chang and Zadeh [14], it was followed up by Dubois and Prade [18] who used the extension principle in their approach. Other methods have been discussed by Goetschel and Voxman [22]. Fuzzy differential equations were first formulated by Kaleva [34] and Seikkala [66] in time dependent form. Kaleva had formulated fuzzy differential equations, in terms of Hukuhara derivative [34]. In [60, 20], investigated the existence and uniqueness of solution for fuzzy random differential equations with non-lipschitz coefficients and fuzzy differential equations with piecewise constant argument. The fuzzy mapping function was introduced by Chang and Zadeh [14]. Later, Dubois and Prade [17] presented an elementary fuzzy calculus based on the extension principle also the concept of integration of fuzzy functions was first introduced by Dubois and Prade [18]. Babolian et al. and Abbasbandy et al. in [1, 8] obtained a numerical solution of linear Fredholm fuzzy integral equations of the second kind.

Fuzzy neural network have been extensively studied [11, 15]. Fuzzy neural network (FNN) systems are hybrid systems that combine the theories of fuzzy logic and neural networks. Designing the FNN system based on the input-output data is a very important problem [43, 69]. Recently, FNN successfully used for solving fuzzy polynomial equation and systems of fuzzy polynomials [2, 3], approximate fuzzy coefficients of fuzzy regression models [55, 56, 52, 59], approximate solution of fuzzy linear systems and fully fuzzy linear systems [57, 58]. One of the major applications of fuzzy neural networks is treating control and synchronization of chaos [44, 45, 46, 70, 71, 74, 75]. Lagaris et al. in [38] used multilayer perceptron to estimate the solution of differential equation. Their neural network model was trained over an interval (over which the differential equation must be solved), so the inputs of the neural network model were the training points. Recently Effati *et al.* [19] used three layers neural network to estimate the fuzzy solution of differential equation. They used parametric form of fuzzy numbers. The comparison of their methods with the existing numerical method shows that their method was more accurate and the solution had also more generalizations. In this work we propose a new solution method for the approximated solution of fuzzy linear Fredholm integro-differential equations using innovative mathematical tools and neural-like systems of computation which is our main objective. 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 initial while the second term contains the fuzzy neural network adjustable parameters to be calculated. Moreover, 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 [30, 33]. Our fuzzy neural network in this paper is a three layers feedforward neural network where connection weights and biases are fuzzy numbers.

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 is a fuzzy set $u : \mathbb{R}^1 \longrightarrow I = [0, 1]$ such that (see [36])

i. u is upper semi-continuous;

ii. $u(x) = 0$ outside some interval $[a, d]$;

iii. There are real numbers b and c, $a \leq b \leq c \leq d$, for which

1. $u(x)$ is monotonically increasing on $[a, b]$,
2. $u(x)$ is monotonically decreasing on $[c, d]$,
3. $u(x) = 1, b \leq x \leq c$.

The set of all fuzzy numbers (as given in Definition 2.1) is denoted by E^1. An alternative definition which yields the same E^1 is given by Kaleva [34].

Definition 2.2. A fuzzy number u is a pair $(\underline{u}, \overline{u})$ of functions $\underline{u}(r)$ and $\overline{u}(r)$, $0 \leq r \leq 1$, which satisfy the following requirements:

i. $\underline{u}(r)$ is a bounded monotonically increasing, left continuous function on $(0, 1]$ and right continuous at 0;

ii. $\overline{u}(r)$ is a bounded monotonically decreasing, left continuous function on $(0, 1]$ and right continuous at 0;

iii. $\underline{u}(r) \leq \overline{u}(r), 0 \leq r \leq 1$.

Definition 2.3. For arbitrary fuzzy numbers u, v, we use the distance (see [22])

$$D(u, v) = sup_{0 \leq r \leq 1} max\{|\overline{u}(r) - \overline{v}(r)|, |\underline{u}(r) - \underline{v}(r)|\}$$

and it is shown that (E^1, D) is a complete metric space(see [62]).

Definition 2.4. Let $f : [a, b] \longrightarrow E^1$, for each partition $P = \{t_0, t_1, \ldots, t_n\}$ of $[a, b]$ and arbitrary $\xi_i \in [t_{i-1}, t_i], 1 \leq i \leq n$, suppose

$$R_p = \sum_{i=1}^{n} f(\xi_i)(t_i - t_{i-1}),$$

$$\Delta := max\{|t_i - t_{i-1}|, i = 1, 2, \ldots, n\}.$$

The definite integral of $f(t)$ over $[a, b]$ is

$$\int_a^b f(t)dt = lim_{\Delta \longrightarrow 0} R_p$$

provided that this limit exists in the metric D (see [21, 22]).

If the fuzzy function $f(t)$ is continuous in the metric D, its definite integral exists [22] and also,

$$\underline{(\int_a^b f(t; r)dt)} = \int_a^b \underline{f}(t; r)dt,$$

$$\overline{(\int_a^b f(t; r)dt)} = \int_a^b \overline{f}(t; r)dt.$$

Definition 2.5. 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 $u - v$ (see [62]).

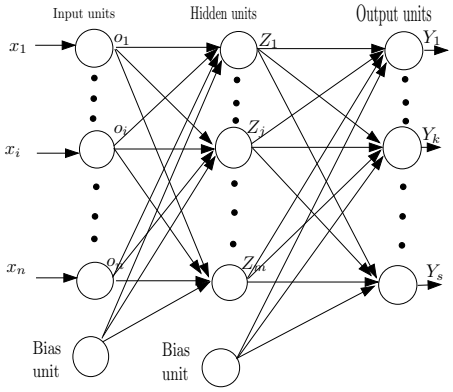

FIGURE 1. Three Layers Feed-forward FNNM

Definition 2.6. 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 $\hat{t}$.

Artificial neural networks are exciting forms of artificial intelligence, which mimic the learning process of the human brain in order to extract patterns from historical data [65]. For many years this technology has been successfully applied to a wide variety of real-word applications [61]. 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 inputs. Such networks can learn to pick out structure in their input. Figure 1 shows typical three layers perceptron. Multi layered perceptrons with more than three layers, use more hidden layers [26, 35]. Multi layered perceptrons correspond the input units to the output units by a specific nonlinear mapping [67]. The most important application of multi layered perceptrons is their ability in function approximation [13]. From Kolmogorov existence theorem we know that a three layered perceptron with $n(2n + 1)$ nodes can compute any continuous function of n variables [47, 28]. The accuracy of the approximation depends on the number of neurons in the hidden layer and does not depend on the number of the hidden layers [40].

Before describing a fuzzy neural network architecture, we denote real numbers and fuzzy numbers by lowercase letters $(e.g., a, b, c, \ldots)$ and uppercase letters $(e.g., A, B, C, \ldots)$, respectively.

Our fuzzy neural network is a three layers feedforward neural network where connection weights, biases and targets are given as fuzzy numbers and inputs are given as real numbers. In this work, we employ the modified sigmoid function $s(x)$ given as

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

For convenience in this discussion, FNNM with an input layer, a single hidden layer, and an output layer in Figure 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 m, n and s are the number of neurons in the input layer, the hidden layer and the output layer, respectively. The architecture of the model shows how FNNM transforms the n inputs $(x_1, \ldots, x_i, \ldots, x_n)$ into the s outputs $(Y_1, \ldots, Y_k, \ldots, Y_s)$ throughout the m hidden neurons $(Z_1, \ldots, Z_j, \ldots, Z_m)$, where the cycles represent the neurons in each layer. Let B_j be the bias for neuron Z_j, C_k be the bias for neuron Y_k, W_{ji} be the weight connecting neuron x_i to neuron Z_j, and W_{kj} be the weight connecting neuron Z_j to neuron Y_k.

2.1. **Operations on Fuzzy Numbers.** We briefly mention fuzzy number operations defined by the extension principle [16,72,73]. Since input vector of feedforward neural network is fuzzy in this paper, the following addition, multiplication and nonlinear mapping of fuzzy numbers are necessary for defining our fuzzy neural network:

$$\mu_{A+B}(z) = max\{\mu_A(x) \wedge \mu_B(y) | z = x + y\}, \tag{2}$$

$$\mu_{AB}(z) = max\{\mu_A(x) \wedge \mu_B(y) | z = xy\}, \tag{3}$$

$$\mu_{f(Net)}(z) = max\{\mu_{Net}(x) | z = f(x)\}, \tag{4}$$

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.

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

$$[A]_h = \{x \in \mathbb{R} | \mu_A(x) \geq h\} \quad for \quad 0 < h \leq 1, \tag{5}$$

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

$$[A]_h = [[A]_h^L, [A]_h^U], \tag{6}$$

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 [4], the above operations of fuzzy numbers are written for h-level sets as follows:

$$[A]_h + [B]_h = [[A]_h^L + [B]_h^L, [A]_h^U + [B]_h^U], \tag{7}$$

$$\begin{aligned}[A]_h.[B]_h = [&min\{[A]_h^L.[B]_h^L, [A]_h^L.[B]_h^U, [A]_h^U.[B]_h^L, [A]_h^U.[B]_h^U\}, \\ &max\{[A]_h^L.[B]_h^L, [A]_h^L.[B]_h^U, [A]_h^U.[B]_h^L, [A]_h^U.[B]_h^U\}],\end{aligned} \tag{8}$$

$$f([Net]_h) = f([[Net]_h^L, [Net]_h^U]) = [f([Net]_h^L), f([Net]_h^U)], \tag{9}$$

where f is increasing function. In the case of $0 \leq [B]_h^L \leq [B]_h^U$, equation (8) can be simplified as

$$[A]_h.[B]_h = [min\{[A]_h^L.[B]_h^L, [A]_h^L.[B]_h^U\}, max\{[A]_h^U.[B]_h^L, [A]_h^U.[B]_h^U\}]. \tag{10}$$

2.2. **Input-output Relation of each Unit.** Let us consider a fuzzy three layers feedforward neural network with n input units, m hidden units and s output units. Target vector, connection weights and biases are fuzzy and input vector is real number. In order to derive a crisp learning rule, we restrict connection weights and biases by four types of (real numbers, symmetric triangular fuzzy numbers, asymmetric triangular fuzzy numbers and asymmetric trapezoidal fuzzy numbers) while we can use any type of fuzzy numbers for fuzzy targets. For example, an asymmetric triangular fuzzy connection weight is specified by its three parameters as $W_{kj} = (w_{kj}^L, w_{kj}^C, w_{kj}^U)$.

When an n-dimensional input vector $(x_1, \ldots, x_i, \ldots, x_n)$ is presented to our fuzzy neural network, its input-output relation can be written as follows, where $f : \mathbb{R}^n \longrightarrow E^s$:

Input units:

$$o_i = x_i, \ i = 1, 2, \ldots, n. \tag{11}$$

Hidden units:

$$Z_j = s(Net_j), \ j = 1, 2, \ldots, m, \tag{12}$$

$$Net_j = \sum_{i=1}^{n} o_i.W_{ji} + B_j. \tag{13}$$

Output units:

$$Y_k = Net_k, \ k = 1, 2, \ldots, s, \tag{14}$$

$$Net_k = \sum_{j=1}^{m} W_{kj}.Z_j + C_k. \tag{15}$$

The architecture of our fuzzy neural network is shown in Figure 1 where connection weights, biases, and targets are fuzzy and inputs are real numbers. The input-output relation in equations (11)-(15) are defined by the extension principle (see [25, 32, 72]).

2.3. **Calculation of Fuzzy Output.** The fuzzy output from each unit in equations (11)-(15) 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 h-level sets:

Input units:

$$o_i = x_i, \ i = 1, 2, \ldots, n. \tag{16}$$

Hidden units:

$$[Z_j]_h = s([Net_j]_h), \ j = 1, 2, \ldots, m, \tag{17}$$

$$[Net_j]_h = \sum_{i=1}^{n} o_i.[W_{ji}]_h + [B_j]_h. \tag{18}$$

Output unit:

$$[Y_k]_h = [Net_k]_h, \ k = 1, 2, \ldots, s, \tag{19}$$

$$[Net_k]_h = \sum_{j=1}^{m} [W_{kj}]_h.[Z_j]_h + [C_k]_h. \tag{20}$$

From equations (16)-(20), we can see that the h-level sets of the fuzzy outputs Y_k's are calculated from those of the fuzzy weights, fuzzy biases and crisp inputs. From equations (7)-(10), the above relations are rewritten as follows when the inputs x_i's are nonnegative, i.e., $0 \leq x_i$:

Input unit:

$$o = x. \tag{21}$$

Hidden units:

$$[Z_j]_h = [[Z_j]_h^L, [Z_j]_h^U] = [s([Net_j]_h^L), s([Net_j]_h^U)], \tag{22}$$

where $s(.)$ is an increasing function.

$$[Net_j]_h^L = \sum_{i=1}^{n} o_i.[W_{ji}]_h^L + [B_j]_h^L, \tag{23}$$

$$[Net_j]_h^U = \sum_{i=1}^{n} o_i.[W_{ji}]_h^U + [B_j]_h^U. \tag{24}$$

Output units:

$$[Y_k]_h = [[Y_k]_h^L, [Y_k]_h^U] = [[Net_k]_h^L, [Net_k]_h^U]. \tag{25}$$

$$[Net_k]_h^L = \sum_{j\in a} [W_{kj}]_h^L.[Z_j]_h^L + \sum_{j\in b} [W_{kj}]_h^L.[Z_j]_h^U + [C_k]_h^L,$$

$$[Net_k]_h^U = \sum_{j\in c} [W_{kj}]_h^U.[Z_j]_h^U + \sum_{j\in d} [W_{kj}]_h^U.[Z_j]_h^L + [C_k]_h^U, \tag{26}$$

for $[Z_j]_h^U \geq [Z_j]_h^L \geq 0$, where

$$a = \{j \mid [W_{kj}]_h^L \geq 0\},\ b = \{j \mid [W_{kj}]_h^L < 0\},\ a \cup b = \{1, \ldots, m\},$$

$$c = \{j \mid [W_{kj}]_h^U \geq 0\},\ d = \{j \mid [W_{kj}]_h^U < 0\},\ c \cup d = \{1, \ldots, m\}.$$

3. **Fuzzy Integro-differential Equations**

The linear Fredholm integro-differential equations of the second kind (see [27,68])

$$\begin{cases} \frac{dy(x)}{dx} = g(x) + \lambda \int_a^b k(x,t)y(t)dt, \\ y(a) = A, \end{cases} \tag{27}$$

where $\lambda > 0$, k is an arbitrary given kernel function over the square $0 \leq a \leq x, t \leq b$ and $g(x)$ is a given function of $x \in [a, b]$. If y is a fuzzy function, $y(x)$ is a given fuzzy function of $x \in [a, b]$ and y' is the fuzzy derivative (according to Definition 2.6) of y, this equation may only possess fuzzy solution. Sufficiency for the existence of equation of the second kind are given in [9].

The Newton-Cotes method is given by (see [7])

$$\int_a^b Z(t)dt = \sum_{i=0}^{r} c_i Z(t_i) + O(h^\nu),$$

where ν depends upon the used method of Newton-Cotes with positive coefficient for estimating of the integral in equation (27). suppose that

$$f(x, y) = g(x) + \lambda \int_a^b k(x, t)y(t)dt,$$

and let, $k(x, t)$ be nonnegative over $[a, b]$, therefore we have

$$[f(x, y)]_h^L = [g(x)]_h^L + \lambda \sum_{i=0}^{r} c_i k(x, t_i)[y(t_i)]_h^L,$$

$$[f(x, y)]_h^U = [g(x)]_h^U + \lambda \sum_{i=0}^{r} c_i k(x, t_i)[y(t_i)]_h^U.$$

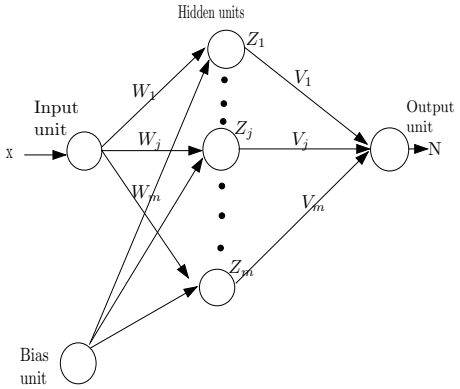

FIGURE 2. Three Layers FNN with One Input and One Output

suppose that a general approximation solution of equation (27) is of the form $y_T(x, P)$, for y_T as a dependent variable of x and P, where P is an adjustable parameter involving weights and biases in the structure of the three-layered feed forward fuzzy neural network (see Figure 2). The fuzzy trial solution y_T is an approximation solution of y for the optimized value of unknown weights and biases. Thus the problem of finding the approximated fuzzy solution for equation (27) over some collocation points in $[a, b]$ by a set of discrete equally spaced grid points

$$a = x_0 < x_1 < \ldots < x_r = b,$$

is equivalent to calculate the functional $y_T(x, P)$.

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

$$y_T(x, P) = \alpha(x) + \beta[x, N(x, P)], \tag{28}$$

where the first term in the right hand side does not involve with adjustable parameters and satisfies the fuzzy initial condition. The second term in the right hand side is a feed forward three layered fuzzy neural network consisting of an input x and the output $N(x, P)$. The crisp trial function was used in [48]. In the next subsection, this FNNM with some weights and biases is considered and we train in order to compute the approximate solution of problem (27).

Let us consider a three layered FNNM (see Figure 2) with one unit entry x, one hidden layer consisting of m activation functions and one unit output $N(x, P)$. In this paper, we use the sigmoidal activation function $s(.)$ for the hidden units of our fuzzy neural network.

Here, the dimension of FNNM is $1 \times m \times 1$. For every entry x the input neuron does not change its input, so the input of the hidden neurons is

$$Net_j = x.W_j + B_j, \ j = 1, \ldots, m, \tag{29}$$

where W_j is a weight parameter from input layer to the jth unit in the hidden layer, B_j is the jth bias for the jth unit in the hidden layer. The output, in the

hidden neurons is

$$Z_j = s(Net_j),\ j = 1, \ldots, m, \tag{30}$$

where s is the activation function which is normally a nonlinear function, the usual choices of the activation function [24] are the sigmoid transfer functions, and the output neuron does not change its input, so the input to the output neuron is equal to output

$$N = V_1 Z_1 + \ldots + V_j Z_j + \ldots + V_m Z_m, \tag{31}$$

where V_j is a weight parameter from jth unit in the hidden layer to the output layer.

From equations (21)-(26), we can see that the h-level sets of the equations (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 fully fuzzy neural networks, i.e., $0 \leq x$ [30]:
Input unit:

$$o = x. \tag{32}$$

Hidden units:

$$[Z_j]_h = [[Z_j]_h^L, [Z_j]_h^U] = [s([Net_j]_h^L), s([Net_j]_h^U)], \tag{33}$$

$$[Net_j]_h^L = o.[W_j]_h^L + [B_j]_h^L, \tag{34}$$

$$[Net_j]_h^U = o.[W_j]_h^U + [B_j]_h^U. \tag{35}$$

$$[N]_h = [[N]_h^L, [N]_h^U], \tag{36}$$

$$\begin{aligned} [N]_h^L &= \sum_{j \in a} [V_j]_h^L . [Z_j]_h^L + \sum_{j \in b} [V_j]_h^L . [Z_j]_h^U, \\ [N]_h^U &= \sum_{j \in c} [V_j]_h^U . [Z_j]_h^U + \sum_{j \in d} [V_j]_h^U . [Z_j]_h^L, \end{aligned} \tag{37}$$

A FNN_4 (fuzzy neural network with crisp set input signals, fuzzy number weights and fuzzy number output) [29] solution to equation (27) is given in Figure 2. How is the FNN_4 going to solve the fuzzy differential equations? The training data are $a = x_1 < x_2 < \ldots < x_r = b$ for input. We propose a learning algorithm from the cost function for adjusting weights.

Consider the linear Fredholm integro-differential equations of the second kind (27), the related trial function will be in the form

$$y_T(x, P) = A + (x - a)N(x, P), \tag{38}$$

this solution by intention satisfies the initial condition in (27). In [30], 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 [2, 30] for measuring the difference between B and O:

$$e = \sum_h e_h = \sum_h h.\{\sum_{k=1}^{s} ([B_k]_h^L - [O_k]_h^L)^2/2 + \sum_{k=1}^{s} ([B_k]_h^U - [O_k]_h^U)^2/2\}, \tag{39}$$

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 (39).

In [31], it is shown by computer simulations that their paper, 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 (39). This is because the squared errors for the h-level sets are weighted by h in equation (39). Krishnamraju et al. [37] used the cost function without the weighting scheme:

$$e = \sum_h e_h = \sum_h \{\sum_{k=1}^{s} ([B_k]_h^L - [O_k]_h^L)^2/2 + \sum_{k=1}^{s} ([B_k]_h^U - [O_k]_h^U)^2/2\}. \tag{40}$$

In the computer simulations included in this paper, we mainly use the cost function in (40) without the weighting scheme.

The error function that must be minimized for problem (27) is of the form

$$e = \sum_{i=1}^{r} e_i = \sum_{i=1}^{r} \sum_h e_{ih} = \sum_{i=1}^{g} \sum_h \{e_{ih}^L + e_{ih}^U\}, \tag{41}$$

where

$$e_{ih}^L = \frac{([\frac{dy_T(x_i,P)}{dx}]_h^L - [f(x_i, y_T(x_i,P))]_h^L)^2}{2}, \tag{42}$$

$$e_{ih}^U = \frac{([\frac{dy_T(x_i,P)}{dx}]_h^U - [f(x_i, y_T(x_i,P))]_h^U)^2}{2}, \tag{43}$$

where $\{x_i\}_{i=1}^g$ are discrete points belonging to the interval $[a, b]$ and in the cost function (41), e_{ih}^L and e_{ih}^U can be viewed as the squared errors for the lower limits and the upper limits of the h-level sets, respectively.

It is easy to express the first derivative (by means of H-difference) of $N(x, P)$ in terms of the derivative of the sigmoid function, i.e.

$$\frac{\partial [N]_h^L}{\partial x} = \sum_a [V_j]_h^L \cdot \frac{\partial [Z_j]_h^L}{\partial [Net_j]_h^L} \cdot \frac{\partial [Net_j]_h^L}{\partial x} + \sum_b [V_j]_h^L \cdot \frac{\partial [Z_j]_h^U}{\partial [Net_j]_h^U} \cdot \frac{\partial [Net_j]_h^U}{\partial x} \tag{44}$$

$$\frac{\partial [N]_h^U}{\partial x} = \sum_c [V_j]_h^U \cdot \frac{\partial [Z_j]_h^U}{\partial [Net_j]_h^U} \cdot \frac{\partial [Net_j]_h^U}{\partial x} + \sum_d [V_j]_h^U \cdot \frac{\partial [Z_j]_h^L}{\partial [Net_j]_h^L} \cdot \frac{\partial [Net_j]_h^L}{\partial x} \tag{45}$$

where $a = \{j \mid [V_j]_h^L \geq 0\}$, $b = \{j \mid [V_j]_h^L < 0\}$, $c = \{j \mid [V_j]_h^U \geq 0\}$,$d = \{j \mid [V_j]_h^U < 0\}$, $a \cup b = \{1, \ldots, m\}$ and $c \cup d = \{1, \ldots, m\}$ and

$$\frac{\partial [Net_j]_h^L}{\partial x} = [W_j]_h^L, \ \frac{\partial [Z_j]_h^L}{\partial [Net_j]_h^L} = [Z_j]_h^L.(1 - [Z_j]_h^L),$$

$$\frac{\partial [Net_j]_h^U}{\partial x} = [W_j]_h^U, \ \frac{\partial [Z_j]_h^U}{\partial [Net_j]_h^U} = [Z_j]_h^U.(1 - [Z_j]_h^U).$$

Now differentiating from trial function $y_T(x, P)$ in (42) and (43), we obtain

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

thus the expression in (44) and (45) are applicable here. A learning algorithm is derived in Appendix A.

3.1. **Partially Fuzzy Neural Networks.** One drawback of fully fuzzy neural networks with fuzzy connection weights is 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) architecture where connection weights to output unit are fuzzy numbers while connection weights and biases to hidden units are real numbers [31]. Since we had good simulation results even from partially fuzzy three layers 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.

The input-output relation of each unit of our partially fuzzy neural network in (32)-(37) can be rewritten for h-level sets as follows:

Input unit:
$$o = x. \tag{46}$$

Hidden units:
$$z_j = s(net_j),\ j = 1, \ldots, m, \tag{47}$$
$$net_j = o.w_j + b_j. \tag{48}$$

Output unit:
$$[N]_h = [[N]_h^L, [N]_h^U] = [\sum_{j=1}^{m} [V_j]_h^L . z_j, \sum_{j=1}^{m} [V_j]_h^U . z_j]. \tag{49}$$

The error function that must be minimized for problem (27) is of the form
$$e = \sum_{i=1}^{r} e_i = \sum_{i=1}^{r} \sum_{h} e_{ih} = \sum_{i=1}^{r} \sum_{h} \{e_{ih}^L + e_{ih}^U\}, \tag{50}$$

where
$$e_{ih}^L = \frac{([\frac{dy_T(x_i,P)}{dx}]_h^L - [f(x_i, y_T(x_i,P))]_h^L)^2}{2}, \tag{51}$$
$$e_{ih}^U = \frac{([\frac{dy_T(x_i,P)}{dx}]_h^U - [f(x_i, y_T(x_i,P))]_h^U)^2}{2}, \tag{52}$$

where $\{x_i\}_{i=1}^{g}$ are discrete points belonging to the interval $[a, b]$ and in the cost function (50), e_{ih}^L and e_{ih}^U can be viewed as the squared errors for the lower limits and the upper limits of the h-level sets, respectively.

It is easy to express the first derivative of $N(x, P)$ in terms of the derivative of the sigmoid function, i.e.
$$\frac{\partial [N]_h^L}{\partial x} = \sum_{j=1}^{m} [V_j]_h^L . \frac{\partial z_j}{\partial net_j} . \frac{\partial net_j}{\partial x} = \sum_{j=1}^{m} [V_j]_h^L . z_j . (1 - z_j) . w_j, \tag{53}$$
$$\frac{\partial [N]_h^U}{\partial x} = \sum_{j=1}^{m} [V_j]_h^U . \frac{\partial z_j}{\partial net_j} . \frac{\partial net_j}{\partial x} = \sum_{j=1}^{m} [V_j]_h^U . z_j . (1 - z_j) . w_j. \tag{54}$$

Now differentiating trial function $y_T(x, P)$ in (51) and (52), we obtain
$$\frac{\partial [y_T(x,P)]_h^L}{\partial x} = [N(x,P)]_h^L + (x - a) . \frac{\partial [N(x,P)]_h^L}{\partial x},$$
$$\frac{\partial [y_T(x,P)]_h^U}{\partial x} = [N(x,P)]_h^U + (x - a) . \frac{\partial [N(x,P)]_h^U}{\partial x},$$

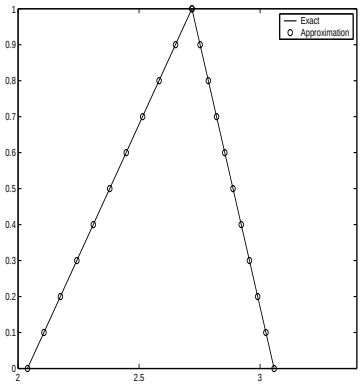

FIGURE 3. Analytical Solution and Approximate Solution by PFNNM

thus the expressions in (53) and (54) are applicable here. A learning algorithm is derived in Appendix B.

4. **Numerical Results**

To illustrate the technique proposed in this paper, consider the following examples. For each fuzzy numbers, we use $h = 0, 0.1, \ldots, 1$. In the computer simulation of this section, all of the programs are written by MATLAB 6.5.1.

Example 4.1. Consider the following fuzzy linear Fredholm integro-differential equation

$$\frac{dy(x)}{dx} = g(x) + \int_0^1 txy(t)dt,$$

where $[g(x)]_h = [0.75 + 0.25h, 1.125 - 0.125h](e^x - x)$ and

$$[y(0)]_h = [0.75 + 0.25h, 1.125 - 0.125h]; \quad 0 \leq h \leq 1, \quad 0 \leq x, t \leq 1.$$

The trial function for this problem is

$$y_T(x, P) = y(0) + xN(x, P).$$

Here, the dimension of PFNNM is $1 \times 5 \times 1$. The error function for $m = 5$ sigmoid units in the hidden layer and for $g = 6$ equally spaced points inside the interval $[0, 1]$ 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: 100 iterations of the learning algorithm.
(3) Learning constant: $\eta = 0.3$.
(4) Momentum constant: $\alpha = 0.2$.
(5) Initial value of the weights and biases of PFNNM are shown in Table 1, that 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 solution of problem (27). A comparison between the exact and approximate solutions at $x = 1$ is shown in Table 2 and Figure 3.

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

h	$[y]_h^L$	$[y_T]_h^L$	Error	$[y]_h^U$	$[y_T]_h^U$	Error
0	2.038711	2.038703	0.08e-4	3.058067	3.058043	0.24e-4
0.1	2.106668	2.106647	0.21e-4	3.024088	3.024066	0.22e-4
0.2	2.174625	2.174612	0.13e-4	2.990110	2.990107	0.03e-4
0.3	2.242582	2.242561	0.21e-4	2.956131	2.956121	0.10e-4
0.4	2.310539	2.310518	0.21e-4	2.922152	2.922123	0.29e-4
0.5	2.378496	2.378469	0.27e-4	2.888174	2.888153	0.21e-4
0.6	2.446453	2.446427	0.26e-4	2.854195	2.854178	0.17e-4
0.7	2.514410	2.514401	0.09e-4	2.820217	2.820201	0.16e-4
0.8	2.582367	2.582359	0.08e-4	2.786238	2.786212	0.26e-4
0.9	2.650324	2.650301	0.23e-4	2.752260	2.752249	0.21e-4
1	2.718281	2.718266	0.15e-4	2.718281	2.718263	0.18e-4

TABLE 2. Exact and Approximate Solution

Example 4.2. Consider the fuzzy linear Fredholm integro-differential equation with

$$[g(x)]_h^L = cos(\frac{x}{2})[0.462288(h^2+h) + 0.091046(4-h^3-h)],$$
$$[g(x)]_h^U = cos(\frac{x}{2})[0.091046(h^2+h) + 0.462288(4-h^3-h)],$$

and kernel

$$k(x,t) = 0.1cos(\frac{x}{2})sin(2t),$$

and $a = 0,\ b = \Pi$, with $y(0) = (0,0)$. The fuzzy trial function for this problem is

$$y_T(x,P) = x.N(x,P).$$

Here, the dimension of PFNNM is $1 \times 5 \times 1$. The error function for $m = 5$ sigmoid

i	1	2	3	4	5
$v_i^{(1)}$	-1.006	10.0789	10.4149	11.3400	3.9762
$v_i^{(2)}$	-0.9504	10.4519	10.6509	12.4322	4.0697
$v_i^{(3)}$	-0.1002	11.1001	10.8400	12.9713	5.2107
w_i	7.7483	-0.1238	0.0826	2.5070	-7.7486
b_i	0.3710	-1.8912	-15.0217	-9.5468	-8.6859

TABLE 3. The Values of Weights

h	$[y]_h^L$	$[y_T]_h^L$	Error	$[y]_h^U$	$[y_T]_h^U$	Error
0	0	0	0.08e-4	2.828427	2.828411	0.16e-4
0.1	0.077781	0.077772	0.09e-4	2.757009	2.757001	0.08e-4
0.2	0.169705	0.169701	0.04e-4	2.681348	2.681332	0.16e-4
0.3	0.275771	0.275758	0.13e-4	2.597203	2.597200	0.03e-4
0.4	0.395979	0.395959	0.2e-4	2.500329	2.500311	0.18e-4
0.5	0.530330	0.530311	0.19e-4	2.386485	2.386467	0.18e-4
0.6	0.678822	0.678815	0.07e-4	2.251427	2.251413	0.14e-4
0.7	0.841457	0.841443	0.14e-4	2.090914	2.090902	0.12e-4
0.8	1.018233	1.018212	0.21e-4	1.900703	1.900700	0.03e-4
0.9	1.209152	1.209137	0.15e-4	1.676550	1.676538	0.12e-4
1	1.414213	1.414201	0.12e-4	1.414213	1.414202	0.11e-4

TABLE 4. Exact and Approximate Solution

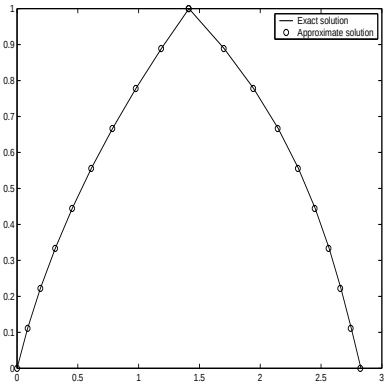

FIGURE 4. Analytical Solution and Approximate Solution by PFNNM

units in the hidden layer and for $g = 6$ equally spaced points inside the interval $[0, \Pi]$ 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: 100 iterations of the learning algorithm.

(3) Learning constant: $\eta = 0.3$.

(4) Momentum constant: $\alpha = 0.2$.

(5) Initial value of the weights and biases of PFNNM are shown in Table 1, that 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 solution of problem (27). Fuzzy weights from the trained fuzzy neural network are shown in Table 3. A comparison between the exact and approximate solutions at $x = \frac{\Pi}{2}$ is shown in Table 4 and Figure 4.

5. Conclusions

Solving linear Fredholm integro-differential equation of the second kind by using universal approximators (UA), that is, FNNM is presented in this paper. To obtain the "Best-approximated" solution of linear Fredholm integro-differential equation of the second kind, the adjustable parameters of FNNM is systematically adjusted by using the learning algorithm.

In this paper, we derived a learning algorithm of fuzzy weights of tree layers feedforward FNN whose input-output relations were defined by extension principle. The effectiveness of the derived learning algorithm was demonstrated by computer simulation on numerical example. Our computer simulation in this paper were performed for three layers feedforward neural networks using the back-propagation-type learning algorithm. 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 layers 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 FFNN in our computer simulations.

6. Appendix

6.1. Appendix A. Derivation of a Learning Algorithm in Fuzzy Neural Networks.

Let us denote the fuzzy connection weight V_j by its parameter values as $V_j = (v_j^{(1)}, \ldots, v_j^{(q)}, \ldots, v_j^{(r)})$ where V_j is a weight parameter from jth unit in the hidden layer to the output layer. The amount of modification of each parameter value is written as [29, 31, 64]

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

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

where t indexes the number of adjustments, η is a learning rate (positive real number) and α is a momentum term constant (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 function of the fuzzy connection weight V_j. For example, when the fuzzy connection weight V_j is trapezoidal (i.e., $V_j = (v_j^{(1)}, v_j^{(2)}, v_j^{(3)}, v_j^{(4)})$, as shown in Figure 3), $\frac{\partial [V_j]_h^L}{\partial v_j^{(q)}}$ and $\frac{\partial [V_j]_h^U}{\partial v_j^{(q)}}$ are calculated as follows:

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

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

These derivatives are calculated from the following relation between the h-level set of the fuzzy connection weight V_j and its parameter values (see Figure 2):

$$[V_j]_h^L = (1-h).v_j^{(1)} + h.v_j^{(2)},$$
$$[V_j]_h^U = h.v_j^{(3)} + (1-h).v_j^{(4)}.$$

When the fuzzy connection weight V_j is a symmetric triangular fuzzy number, 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,

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

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 relation of our fuzzy neural network for the h-level sets. When we use the cost function with the weighting scheme in equation (41), $\frac{\partial e_{ih}}{\partial [V_j]_h^L}$ and $\frac{\partial e_{ih}}{\partial [V_j]_h^U}$ are calculated as follows:

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

(i) If $[V_j]_h^L \geq 0$ then

$$\frac{\partial e_{ih}}{\partial [V_j]_h^L} = \delta^L.([Z_j]_h^L + (x_i - a).\frac{\partial [Z_j]_h^L}{\partial x} - \frac{\partial [f(x, y_T(x_i,P))]_h^L}{\partial [y_T(x_i,P)]_h^L}.(x_i-a).[Z_j]_h^L),$$

where

$$\delta^L = ([\frac{dy_T(x_i,P)}{dx}]_h^L - [f(x_i, y_T(x_i,P))]_h^L).$$

(ii) If $[V_j]_h^L < 0$ then

$$\frac{\partial e_{ih}}{\partial [V_j]_h^L} = \delta^L.([Z_j]_h^U + (x_i - a).\frac{\partial [Z_j]_h^U}{\partial x} - \frac{\partial [f(x, y_T(x_i,P))]_h^L}{\partial [y_T(x_i,P)]_h^L}.(x_i-a).[Z_j]_h^U).$$

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

(i) If $[V_j]_h^U \geq 0$ then

$$\frac{\partial e_{ih}}{\partial [V_j]_h^U} = \delta^U.([Z_j]_h^U + (x_i - a).\frac{\partial [Z_j]_h^U}{\partial x} - \frac{\partial [f(x, y_T(x_i, P))]_h^U}{\partial [y_T(x_i, P)]_h^U}.(x_i - a).[Z_j]_h^U),$$

where

$$\delta^U = ([\frac{dy_T(x_i, P)}{dx}]_h^U - [f(x_i, y_T(x_i, P))]_h^U).$$

(ii) If $[V_j]_h^U < 0$ then

$$\frac{\partial e_{ih}}{\partial [V_j]_h^U} = \delta^U.([Z_j]_h^L + (x_i - a).\frac{\partial [Z_j]_h^L}{\partial x} - \frac{\partial [f(x, y_T(x_i, P))]_h^U}{\partial [y_T(x_i, P)]_h^U}.(x_i - a).[Z_j]_h^L).$$

In our fuzzy neural network, the connection weights and biases to the hidden units are updated in the same manner as the parameter values of the fuzzy connection weights V_j as follows:

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

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

Thus our problem is to calculate the derivatives $\frac{\partial e_{ih}}{\partial w_j^{(q)}}$. Let us rewrite $\frac{\partial e_{ih}}{\partial w_j^{(q)}}$ as follows:

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

In this formulation, $\frac{\partial [W_j]_h^L}{\partial w_j^{(q)}}$ and $\frac{\partial [W_j]_h^U}{\partial w_j^{(q)}}$ are easily calculated from the membership function of the fuzzy connection weight W_j. Derivatives $\frac{\partial e_{ih}}{\partial [W_j]_h^L}$ and $\frac{\partial e_{ih}}{\partial [W_j]_h^U}$ can be calculated from the cost function e_{ih} using the input-output relation of our fuzzy neural network for the h-level sets. When we use the cost function with the weighting scheme in equation (41), $\frac{\partial e_{ih}}{\partial [W_j]_h^L}$ is calculated as follows:

[Calculation of $\frac{\partial e_{ih}}{\partial [W_j]_h^L}$]

(i) If $[V_j]_h^L \geq 0$ then

$$\begin{aligned}\frac{\partial e_{ih}}{\partial [W_j]_h^L} = \delta^L.[[V_j]_h^L.[Z_j]_h^L.(1 - [Z_j]_h^L).x_i + (x_i - a).[V_j]_h^L.[Z_j]_h^L \\ +(x_i - a).x_i.[V_j]_h^L.[Z_j]_h^L.(1 - [Z_j]_h^L)w_j - (x_i - a).[V_j]_h^L.([Z_j]_h^L)^2 \\ -2(x_i - a).x_i.[V_j]_h^L.([Z_j]_h^L)^2(1 - [Z_j]_h^L)w_j \\ -(\frac{\partial [f(x, y_T(x_i, P))]_h^L}{\partial [y_T(x_i, P)]_h^L}.(x_i - a).[V_j]_h^L.[Z_j]_h^L.(1 - [Z_j]_h^L).x_i)].\end{aligned}$$

(ii) If $[V_j]_h^U < 0$ then

$$\begin{aligned}\frac{\partial e_{ih}}{\partial [W_j]_h^L} = \delta^U.[[V_j]_h^U.[Z_j]_h^L.(1 - [Z_j]_h^L).x_i + (x_i - a).[V_j]_h^U.[Z_j]_h^L \\ +(x_i - a).x_i.[V_j]_h^U.[Z_j]_h^L.(1 - [Z_j]_h^L)w_j - (x_i - a).[V_j]_h^U.([Z_j]_h^L)^2\end{aligned}$$

$$-2(x_i - a).x_i.[V_j]_h^U.([Z_j]_h^L)^2(1 - [Z_j]_h^L)w_j$$
$$-(\frac{\partial[f(x, y_T(x_i, P))]_h^U}{\partial[y_T(x_i, P)]_h^U}.(x_i - a).[V_j]_h^U.[Z_j]_h^L.(1 - [Z_j]_h^L).x_i)].$$

In the other cases, $\frac{\partial e_{ih}}{\partial[W_j]_h^U}$ and the fuzzy biases to the hidden units are updated in the same manner as the fuzzy connection weights to the hidden units and fuzzy connection to the output unit.

6.2. **Appendix B. Derivation of a Learning Algorithm in Partially Fuzzy Neural Networks.** 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 [29, 64]

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

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

where t indexes the number of adjustments, η is a learning rate (positive real number) and α is a momentum term constant (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 function of the fuzzy connection weight V_j.

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 relation of our fuzzy neural network for the h-level sets. When we use the cost function with the weighting scheme in equation (50), $\frac{\partial e_{ih}}{\partial [V_j]_h^L}$ and $\frac{\partial e_{ih}}{\partial [V_j]_h^U}$ are calculated as follows:

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

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

where

$$\delta^L = ([\frac{dy_T(x_i, P)}{dx}]_h^L - [f(x_i, y_T(x_i, P))]_h^L),$$
$$\frac{\partial [y_T(x_i, P))]_h^L}{\partial [V_j]_h^L}] = (x_i - a).z_j, \ \frac{\partial N(x_i, P)]_h^L}{\partial [V_j]_h^L} = z_j.$$

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

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

where

$$\delta^U = ([\frac{dy_T(x_i,P)}{dx}]_h^U - [f(x_i,y_T(x_i,P))]_h^U),$$

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

In our partially fuzzy neural network, the connection weights and biases to the hidden units are real numbers. The non-fuzzy connection weight w_j to the jth hidden unit is updated in the same manner as the parameter values of the fuzzy connection weight V_j as follows:

$$w_j(t+1) = w_j(t) + \triangle w_j(t),$$

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

The derivative $\frac{\partial e_{ih}}{\partial w_j}$ can be calculated from the cost function e_{ih} using the input-output relation of our partially fuzzy neural network for the h-level sets. When we use the cost function with the weighting scheme in equaion (41), $\frac{\partial e_{ih}}{\partial w_j}$ is calculated as follows:

$$\frac{\partial e_{ih}}{\partial w_j} = \delta^L.[\frac{\partial[N(x_i,P)]_h^L}{\partial w_j} + (x_i-a).[V_j]_h^L.z_j + (x_i-a).x_i.[V_j]_h^L.z_j(1-z_j)w_j$$

$$-(x_i-a).[V_j]_h^L.z_j^2 - 2(x_i-a).x_i.[V_j]_h^L.z_j^2(1-z_j)w_j - (\frac{\partial[f(x_i,y_T(x_i,P))]_h^L}{\partial[y_T(x_i,P))]_h^L}$$

$$.\frac{\partial[y_T(x_i,P)]_h^L}{\partial w_j} + \frac{\partial[f(x_i,y_T(x_i,P))]_h^L}{\partial[y_T(x,P))]_h^U}.\frac{\partial[y_T(x_i,P)]_h^U}{\partial w_j})]+$$

$$\delta^U.[\frac{\partial N(x_i,P)]_h^U}{\partial w_j} + (x_i-a).[V_j]_h^U.z_j + (x_i-a).x_i.[V_j]_h^U.z_j(1-z_j)w_j$$

$$-(x_i-a).[V_j]_h^U.z_j^2 - 2(x_i-a).x_i.[V_j]_h^U.z_j^2(1-z_j)w_j - (\frac{\partial[f(x_i,y_T(x_i,P))]_h^U}{\partial[y_T(x_i,P))]_h^L}$$

$$.\frac{\partial[y_T(x_i,P)]_h^L}{\partial w_j} + \frac{\partial[f(x_i,y_T(x_i,P))]_h^U}{\partial[y_T(x_i,P))]_h^U}.\frac{\partial[y_T(x_i,P)]_h^U}{\partial w_j})],$$

where

$$\frac{\partial[N(x_i,P)]_h^L}{\partial w_j} = \frac{\partial[N(x_i,P)]_h^L}{\partial z_j}.\frac{\partial z_j}{\partial net_j}.\frac{\partial net_j}{\partial w_j} = [V_j]_h^L.z_j.(1-z_j).x_i,$$

$$\frac{\partial[N(x_i,P)]_h^U}{\partial w_j} = \frac{\partial[N(x_i,P)]_h^U}{\partial z_j}.\frac{\partial z_j}{\partial net_j}.\frac{\partial net_j}{\partial w_j} = [V_j]_h^U.z_j.(1-z_j).x_i,$$

$$\frac{\partial[y_T(x_i,P))]_h^L}{\partial w_j} = (x_i-a).\frac{\partial[N(x_i,P)]_h^L}{\partial w_j},$$

$$\frac{\partial[y_T(x_i,P))]_h^U}{\partial w_j} = (x_i-a).\frac{\partial[N(x_i,P)]_h^U}{\partial w_j}.$$

The non-fuzzy biases to the hidden units are updated in the same manner as the non-fuzzy connection weights to the hidden units.

References

[1] S. Abbasbandy, E. Babolian and M. Alavi, *Numerical method for solving linear fredholm fuzzy integral equations of the second kind*, Chaos Solitons & Fractals, **31** (2007), 138-146.

[2] S. Abbasbandy and M. Otadi, *Numerical solution of fuzzy polynomials by fuzzy neural network*, Applied Mathematics and Computation, **181** (2006), 1084-1089.

[3] 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.

[4] G. Alefeld and J. Herzberger, *Introduction to interval computations*, Academic Press, New York, 1983.

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

[6] T. Allahviranloo, N. Ahmady and E. Ahmady, *Numerical solution of fuzzy differential eqations by predictor-corrector method*, Information Sciences, **177** (2007), 1633-1647.

[7] K. E. Atkinson, *An introduction to numerical analysis*, New York, Wiley, 1987.

[8] E. Babolian, H. S. Goghary and S. Abbasbandy, *Numerical solution of linear fredholm fuzzy integral equations of the second kind by adomian method*, Applied Mathematics and Computation, **161** (2005), 733-744.

[9] P. Balasubramaniam and S. Muralisankar, *Existence and uniqueness of fuzzy solution for the nonlinear fuzzy integro-differential equations*, Applied mathematics letters, **14** (2001), 455-462.

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

[11] J. F. Bernard, *Use of rule-based system for process control*, IEEE Control System Management, **8** (1988), 3-13.

[12] J. J. Buckley and T. Feuring, *Fuzzy differential equations*, Fuzzy Sets and Systems, **110** (2000), 69-77.

[13] J. J. Buckley and Y. Hayashi, *Can fuzzy neural nets approximate continuous fuzzy functions?*, Fuzzy Sets and Systems, **61** (1994), 43-51.

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

[15] Y. C. Chen and C. C. Teng, *A model reference control structure using a fuzzy neural network*, Fuzzy Sets and Systems, **73** (1995), 291-312.

[16] W. Congxin and M. Ming, *On embedding problem of fuzzy number space*, Fuzzy Sets and Systems, **44** (1991), 33-38.

[17] D. Dubois and H. Prade, *Operations on fuzzy numbers*, International Journal of Systems Science, **9** (1978), 613-626.

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

[19] S. Effati and M. Pakdaman, *Artificial neural network approach for solving fuzzy differential equations*, Information Sciences, **180** (2010), 1434-1457.

[20] W. Fei, *Existence and uniqueness of solution for fuzzy random differential equations with non-lipschitz coefficients*, Information Sciences, **177** (2007), 4329-4337.

[21] M. Friedman, M. Ma and A. Kandel, *Numerical solutions of fuzzy differential and integral equations*, Fuzzy Sets and Systems, **106** (1999), 35-48.

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

[23] D. Gottlieb and S.A. Orszag, *Numerical analysis of spectral methods*, Theory and applications, CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, **26** (1977).

[24] M. T. Hagan, H. B. Demuth and M. Beale, *Neural network design*, PWS publishing company, Massachusetts, 1996.

[25] Y. Hayashi, J. J. Buckley and E. Czogala, *Fuzzy neural network with fuzzy signals and weights*, International Journal of Intelligent Systems, **8** (1993), 527-537.

[26] S. Haykin,*Neural networks: a comprehensive foundation*, Prentice Hall, New Jersey, 1999.
[27] H. Hochstadt, *Integral equations*, New York: Wiley, 1973.
[28] K. Hornick, M. Stinchcombe and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks, **2** (1989), 359-366.
[29] H. Ishibuchi, K. Kwon and H. Tanaka, *A learning algorithm of fuzzy neural networks with triangular fuzzy weights*, Fuzzy Sets and Systems, **71** (1995), 277-293.
[30] H. Ishibuchi, K. Morioka and I.B. Turksen, *Learning by fuzzified neural networks*, International Journal of Approximate Reasoning, **13** (1995), 327-358.
[31] H. Ishibuchi and M. Nii, *Numerical analysis of the learning of fuzzified neural networks from fuzzy if-then rules*, Fuzzy Sets and Systems, **120** (2001), 281-307.
[32] H. Ishibuchi, H. Okada and H. Tanaka, *Fuzzy neural networks with fuzzy weights and fuzzy biases*, Proceedings ICNN, **93** (1993), 1650-1655.
[33] H. Ishibuchi, H. Tanaka and H. Okada, *Fuzzy neural networks with fuzzy weights and fuzzy biases*, IEEE International Conferences on Neural Networks, (1993), 1650-1655.
[34] O. Kaleva, *Fuzzy differential equations*, Fuzzy Sets and Systems, **24** (1987), 301-317.
[35] T. Khanna, *Foundations of neural networks*, Addison-Wesly, Reading, MA, 1990.
[36] G. J. Klir, U. S. Clair, B. Yuan, *Fuzzy set theory: foundations and applications*, Prentice-Hall, 1997.
[37] P. V. Krishnamraju, J. J. Buckley, K. D. Relly and Y. Hayashi, *Genetic learning algorithms for fuzzy neural nets*, IEEE International Conference on Fuzzy Systems, (1994), 1969-1974.
[38] I. E. Lagaris and A. Likas, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Transactions on Neural Networks, September, **9(5)** (1998).
[39] J. D. Lamber, *Computational methods in ordinary differential equations*, John Wiley & Sons, New York, 1983.
[40] A. Lapedes and R. Farber, *How neural nets work?*, Neural Information Processing Systems, AIP, 1988, 442-456.
[41] H. Lee and I. S. Kang, *Neural algorithms for solving differential equations*, Journal of Computational Physics, **91** (1990), 110-131.
[42] T. Leephakpreeda, *Novel determination of differential-equation solutions: universal approximation method*, Computational and Applied Mathematics, **146** (2002), 443-457.
[43] G. Leng, G. Prasad and T. M. McGinnity, *An on-line algorithm for creating self-organizing fuzzy neural networks*, Neural Networks, **17** (2004), 1477-1493.
[44] D. Lin and X. Wang, *Observer-based decentralized fuzzy neural sliding mode control for interconnected unknown chaotic systems via network structure adaptation*, Fuzzy Sets and Systems, **161** (2010), 2066-2080.
[45] D. Lin and X. Wang, *Self-organizing adaptive fuzzy neural control for the synchronization of uncertain chaotic systems with random-varying parameters*, Neurocomputing, **74** (2011), 2241-2249.
[46] D. Lin, X. Wang, F. Nian and Y. Zhang, *Dynamic fuzzy neural networks modeling and adaptive backstepping tracking control of uncertain chaotic systems*, Neurocomputing, **73** (2010), 2873-2881.
[47] R. P. Lippmann, *An introduction to computing with neural nets*, IEEE ASSP Magazine, (1987), 4-22.
[48] A. Malek and R. Shekari Beidokhti, *Numerical solution for high order differential equations using a hybrid neural network-Optimization method*, Applied Mathematics and Computation, **183** (2006), 260-271.
[49] A. J. Meade Jr 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.
[50] A. J. Meade Jr and A. A. Fernandez, *Solution of nonlinear ordinary differential equations by feedforward neural networks*, Mathematical and Computer Modelling, **20(9)** (1994), 19-44.
[51] M. T. Mizukoshi, L. C. Barros, Y. Chalco-Cano, H. Romn-Flores and R. C. Bassanezi, *Fuzzy differential equations and the extention principle*, Information Sciences, **177** (2007), 3627-3635.

[52] 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.
[53] M. Mosleh and M. Otadi, *Minimal solution of fuzzy linear system of differential equations*, Neural Comput and Applications, **21** (2012), 329-336.
[54] M. Mosleh and M. Otadi, *Simulation and evaluation of fuzzy differential equations by fuzzy neural network*, Applied Soft Computing, **12** (2012), 28172827.
[55] 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.
[56] M. Mosleh, M. Otadi and S. Abbasbandy, *Fuzzy polynomial regression with fuzzy neural networks*, Applied Mathematical Modelling, **35** (2011), 5400-5412.
[57] 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.
[58] M. Otadi, M. Mosleh and S. Abbasbandy, *Numerical solution of fully fuzzy linear systems by fuzzy neural network*, Soft Computing, **15** (2011), 1513-1522.
[59] M. Otadi, M. Mosleh, S. Saidanlu and N. A. Aris, *Fuzzy hyperbolic regression with fuzzy neural networks*, Australian Journal of Basic and Applied Sciences, **5(10)** (2011), 838-847.
[60] G. Papaschinopoulos, G. Stefanidou and P. Efraimidis, *Existence, uniquencess and asymptotic behavior of the solutions of a fuzzy differential equation with piecewise constant argument*, Information Sciences, **177** (2007), 3855-3870.
[61] P. Picton, *Neural Networks*, Second edition, Palgrave, Great Britain, 2000.
[62] M. L. Puri and D. Ralescu, *Fuzzy random variables*, Journal of Mathematical Analysis and Applications, **114** (1986), 409-422.
[63] R. Rodriguez-Lopez, *Comparison results for fuzzy differential eqations*, Information Sciences, **178** (2008), 1756-1779.
[64] D. E. Rumelhart and J. L. McClelland, *Parallel distributed processing*, MIT Press, Cambridge, MA, 1986.
[65] R. J. Schalkoff, *Artificial neural networks*, McGraw-Hill, New York, 1997.
[66] S. Seikkala, *On the fuzzy initial value problem*, Fuzzy Sets and Systems, **24** (1987), 319-330.
[67] J. Stanley, *Introduction to neural networks*, Sierra Mardre, 1990.
[68] J. Store and R. Bulirsch, *Introduction to numerical analysis*, Springer-Verlag, New York, 1993.
[69] W. L. Tung and C. Quek, *A generic self-organizing fuzzy neural network*, IEEE Transactions on Neural networks, **13** (2002), 1075-1086.
[70] X. Wang and J. Zhao, *Cryptanalysis on a parallel keyed hash function based on chaotic neural network*, Neurocomputing, **73** (2010,) 3224-3228.
[71] W. Xingyuan, X. Bing and Z. Huaguang, *A multi-ary number communication system based on hyperchaotic system of 6th-order cellular neural network*, Communications in Nonlinear Science and Numerical Simulation, **15** (2010), 124-133.
[72] L. A. Zadeh, *The concept of a liguistic variable and its application to approximate reasoning* Information Sciences, **8** (1975), 199-249, 301-357; **9** (1975), 43-80.
[73] L. A. Zadeh, *Is there a need for fuzzy logic?*, Information Sciences, **178** (2008), 2751-2779.
[74] H. G. Zhang and D. R. Liu, *Fuzzy modeling and fuzzy control*, Boston, 2006.
[75] H. G. Zhang and Y. B. Quan, *Modeling, identification and control of a class of nonlinear systems*, IEEE Transactions on Fuzzy Systems, **9** (2001), 349-354.

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