Article:
Jafarian, A, Jafari, R, Golmankhaneh, AK et al. (2015) Solving fully fuzzy polynomials using feed-back neural networks. International Journal of Computer Mathematics, 92 (4). pp. 742-755. ISSN: 0020-7160

https://doi.org/10.1080/00207160.2014.907404

Solving fully fuzzy polynomials using feed-back neural networks

Ahmad Jafarian[a], Raheleh Jafari[b], Alireza Khalili Golmankhaneh[c*] and Dumitru Baleanu[d,e,f]

[a]*Department of Mathematics, Urmia Branch, Islamic Azad University, P.O. BOX 969, Urmia, Iran;*
[b]*Department of Mathematics, Science and Research Branch, Islamic Azad University, Arak, Iran;*
[c]*Department of Physics, Urmia Branch, Islamic Azad University, P.O. BOX 969, Urmia, Iran;*
[d]*Department of Mathematics and Computer Science, Çankaya University, 06530 Ankara, Turkey;*
[e]*Institute of Space Sciences, P.O. Box, MG-23, R 76900, Magurele-Bucharest, Romania;*
[f]*Department of Chemical and Materials Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah, Saudi Arabia*

Recently, there has been a considerable amount of interest and practice in solving many problems of several applied fields by fuzzy polynomials. In this paper, we have designed an artificial fuzzified feed-back neural network. With this design, we are able to find a solution of fully fuzzy polynomial with degree n. This neural network can get a fuzzy vector as an input, and calculates its corresponding fuzzy output. It is clear that the input–output relation for each unit of fuzzy neural network is defined by the extension principle of Zadeh. In this work, a cost function is also defined for the level sets of fuzzy output and fuzzy target. Next a learning algorithm based on the gradient descent method will be defined that can adjust the fuzzy connection weights. Finally, our approach is illustrated by computer simulations on numerical examples. It is worthwhile to mention that application of this method in fluid mechanics has been shown by an example.

Keywords: fully fuzzy polynomials; fuzzy feed-forward neural networks; fuzzy feed-back neural networks; learning algorithm; cost function

1. Introduction

Several problems in various areas such as economics, finance, engineering and physics boil down to the solution of a fuzzy polynomial. When the estimation of the system coefficients is imprecise and only some vague knowledge about the actual value of the parameters is available, it may be convenient to represent some or all of them with interval values. The fuzzy polynomial $A_1x + \cdots + A_nx^n = A_0$, where the elements, A_i intervals, is called an interval fuzzy polynomial.

When using plain intervals, cautiousness leads to a severe overestimation of the interval width. Therefore, if some expert judgement is available, about the actual value of the parameters, it is possible to assign a degree of membership greater than a given threshold to some values within the interval. Moreover, if the knowledge is more precise, by increasing the threshold level, it is possible to find a nested set of subintervals, within which one or more values have membership equal to one, i.e. a fuzzy number (for a detailed discussion on the relation between nested intervals and fuzzy sets see [19]).

In the last years, various approaches for solving fuzzy polynomials have been reported. Many researchers have studied on methods for the solution of fuzzy polynomials. Abbasbandy and Asady [2] have considered Newton's method for solving fuzzy nonlinear equations. Linear and nonlinear fuzzy equations are solved by Asady *et al.* [8]. They have proposed a general model for solving a system of fuzzy linear equations with m variables and n equations. The original system is replaced by a crisp linear system which is solved by the least squares method.

Abbasbandy and Ezzati [3] have suggested numerical solving method for fuzzy nonlinear equations instead of standard analytical techniques which are not suitable everywhere. They wrote fuzzy nonlinear equation in a parametric form and then solved it by Newton's method. They have also proposed an efficient and iterative method for solving a system of fuzzy linear equations with n variables. They replaced the original $n \times n$ system with a $2n \times 2n$ crisp systems and called it parametric form of fuzzy linear system. The solution vector is symmetric solution if the right-hand side vector is symmetric.

Rouhparvar [23] has proposed a new method for solving fuzzy polynomial equation based on the ranking method. The ranking method was first introduced by Delgado *et al.* [11]. This work interested the readers in finding real roots of polynomial equation like $A_1x + \cdots + A_nx^n = A_0$, where the elements, A_i, are fuzzy numbers. They introduced three real indices called value, ambiguity and fuzziness to obtain simple fuzzy numbers that could be used to represent more arbitrary fuzzy numbers. Therefore these parameters were used for transform fuzzy polynomial equation to a system of crisp polynomial equations. By solving this system, real roots of fuzzy polynomial equation can be found. By little change in this method Noorani *et al.* [20] solved dual fuzzy polynomials.

Allahviranloo *et al.* [6] used the Newton–Raphson method for solving fuzzy polynomials and Otadi and Mosleh [22] investigated the Adomian decomposition method for solving these polynomials.

One approach to indirect solution is using fuzzy neural networks (FNNs). During the past few years, neural networks has received much attention [7,10,14,25]. First time, Buckley and Qu [9] applied a structure of FNNs for solving fuzzy equations. Ishibuchi *et al.* [16] defined a cost function for each pair of fuzzy output vector and its corresponding fuzzy target vector, and have proposed a learning algorithm of FNNs with triangular fuzzy weights. Hayashi *et al.* [15] used fuzzy delta learning rule for training FNN with fuzzy signals and weights. Some application of fuzzy polynomials have been considered by Abbasbandy and Amirfakhrian [1]. Abbasbandy *et al.* [4] have also proposed an architecture of feed-forward neural network for finding solution to fuzzy polynomials, but the neural network that has been presented by Abbasbandy was only able to find a crisp solution of fuzzy polynomials, and this neural network was not able to find a fuzzy solution. Jafarian *et al.* [18] solved fuzzy polynomials by using the neural networks with a new learning algorithm. It is noted in [18] that the presented neural network in comparison with Abbasbandy's was better in the speed of adjusting the weights, which caused the high speed of convergence. Jafarian and Jafari [17] applied fuzzy feed-back neural network method for approximation of the crisp solution of dual fuzzy polynomials. We refer the reader to [21,24] for more information on fuzzy polynomials.

As the suggested neural networks by previous authors were not able to find fuzzy solution of fuzzy polynomials, we propose a fuzzified feed-back neural network, which is able to solve a fuzzy polynomial like

$$A_1x + \cdots + A_nx^n = A_0, \tag{1}$$

where $A_0, A_1, \ldots, A_n$ and x are fuzzy numbers. In this paper, we have proposed a learning algorithm for training fuzzified feed-back neural network with the identity activation function. In computer simulations, such a fuzzified neural network is trained so that, the input–output relation of each unit is defined by the extension principle of Zadeh [26]. By using this neural network, we can

obtain an approximate fuzzy solution of FFP to any degree of accuracy. Furthermore, a feed-forward neural network is also proposed to solve these polynomials. In this paper, the feed-back neural network is compared with feed-forward neural network, and we discuss the advantages of feed-back neural network over feed-forward.

This paper is organized as follows: In Section 2, we present the foundations of our work. In this section, we propose an architecture of FNN3 (FNN with fuzzy input signals and fuzzy weights and fuzzy target) to find approximate solution to fully fuzzy polynomial (FFP). Output from the FNN (with fuzzy input signals and fuzzy weights and real target), is numerically calculated by interval arithmetic [5] for fuzzy weights and level sets (i.e. α-cuts) of fuzzy inputs. Next, we define a cost function for the level sets of fuzzy output and fuzzy target. Then, a fuzzy learning algorithm is derived from the cost function to find the fuzzy solutions of the FFP. In Section 3, we have collected some examples. It is worthwhile to mention that the examples have been solved by two methods, one with feed-forward FNN that we called them FFFNN3 and the other by feed-back FNN (FBFNN3). Our goal is solving the examples by using two methods, and discuss the advantages of FFFNN3 over FBFNN3. Finally, Section 4 presents concluding remarks.

2. Foundations

In the past years various approaches for solving fuzzy equations have been proposed. Buckley and Qu [9] in their work by using the classical method, which was based on the extension principle, of investigating solutions to linear and quadratic equations when the coefficients were real or complex fuzzy numbers have led to the fact that these equations too often do not have a solution. As the particular case of fuzzy equation is fuzzy polynomial, we describe about it in the following sections.

2.1 *Fully fuzzy polynomials*

In this part, we want to find a fuzzy solution of

$$A_1 x + \cdots + A_n x^n = A_0, \tag{2}$$

where $A_j, x \varepsilon E^1$ (for $j = 0, \ldots, n$). For getting an approximate solution, two architectures of feed-forward and feed-back neural nets equivalent to Equation (2) have been presented in Figures 1 and 2, respectively.

As the transfer function is identity, the input neurons make no change in their inputs, therefore the output neuron is $Y = A_1 x + A_2 x^2 + \cdots + A_n x^n$. For solving the fuzzy polynomials by using FFFNN3 and FBFNN3, we propose a learning algorithm from the cost function for adjusting fuzzy weights.

2.2 *Calculation of fuzzy output in feed-forward FNNs*

Let us fuzzify a two-layer feed-forward neural network with n inputs and one output unit. Input vectors, target vector and connection weights are fuzzified (i.e., extended to fuzzy numbers). In order to derive a learning rule, we restrict fuzzy weights within triangular fuzzy numbers while we can use any type of fuzzy numbers for fuzzy inputs and fuzzy target. The input–output relation of each unit can be written as follows:

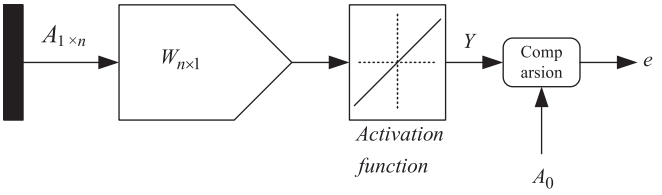

Figure 1. Feed-forward FNN to solve fuzzy polynomials.

Input unit:

$$O_j = A_j, \quad j = 1, 2, \ldots, n. \tag{3}$$

Output unit:

$$\begin{aligned} Y &= f(\text{Net}), \\ \text{Net} &= \sum_{j=1}^{n} w_j.O_j. \end{aligned} \tag{4}$$

The fuzzy output of neuron in the second layer is numerically calculated for fuzzy weights and level sets of fuzzy inputs. The input–output relations of the FNN which is shown in Figure 1 can be written for the α-level sets as follows:
Input unit:

$$[O_j]_\alpha = [A_j]_\alpha, \quad j = 1, \ldots, n. \tag{5}$$

Output unit:

$$\begin{aligned} [Y]_\alpha &= f([\text{Net}]_\alpha), \\ [\text{Net}]_\alpha &= \sum_{j=1}^{n} [w_j.O_j]_\alpha. \end{aligned} \tag{6}$$

From Equations (5) and (6), we see that the α-level sets of the fuzzy output Y are calculated from those of the fuzzy inputs and fuzzy weights. The above relations are written as follows when the α-level sets of the fuzzy input A_j are nonnegative, i.e., $0 \le [A_j]_l^\alpha \le [A_j]_u^\alpha$ for all j:
Input unit:

$$[O_j]_\alpha = [[O_j]_\alpha^l, [O_j]_\alpha^u] = [[A_j]_\alpha^l, [A_j]_\alpha^u], \quad j = 1, \ldots, n.$$

Output unit:

$$\begin{aligned} [Y]_\alpha &= [[Y]_\alpha^l, [Y]_\alpha^u] = [f([\text{Net}]_\alpha^l), f([\text{Net}]_\alpha^u)], \\ [\text{Net}]_\alpha &= [[\text{Net}]_\alpha^l, [\text{Net}]_\alpha^u], \\ [\text{Net}]_\alpha^l &= \sum_{j\epsilon M} [w_j]_\alpha^l.[O_j]_\alpha^l + \sum_{j\epsilon C} [w_j]_\alpha^l.[O_j]_\alpha^u, \\ [\text{Net}]_\alpha^u &= \sum_{j\epsilon M'} [w_j]_\alpha^u.[O_j]_\alpha^u + \sum_{j\epsilon C'} [w_j]_\alpha^u.[O_j]_\alpha^l, \end{aligned} \tag{7}$$

where $M = \{j |\ [w_j]_\alpha^l \ge 0\}$, $C = \{j |\ [w_j]_\alpha^l < 0\}$, $M' = \{j |\ [w_j]_\alpha^u \ge 0\}$, $C' = \{j |\ [w_j]_\alpha^u < 0\}$, $M \cup C = \{1, \ldots, n\}$ and $M' \cup C' = \{1, \ldots, n\}$.

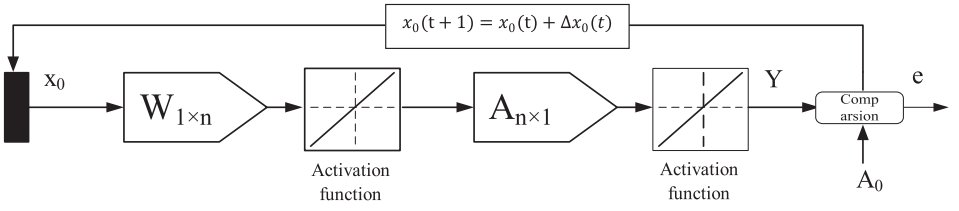

Figure 2. Feed-back FNN to solve fuzzy polynomials.

2.3 *Calculation of fuzzy output in feed-back FNNs*

In the previous section the description about the calculation of fuzzy output in feed-forward FNNs is given, now in this part, we have given a short review on learning of fuzzified feed-back neural networks. First consider a three-layer FBFNN3 with one input unit, n neuron in hidden layer and one output unit. In a given structure, we assume that the corresponding connection weights to output layer and target output are triangular fuzzy numbers. Figure 2 shows the input signal x_0, the input–output relation of each unit and the fuzzy output Y. These relations are written as follows:

Input unit:

$$o = x_0. \tag{8}$$

Hidden unit:

$$\begin{aligned} O_j* &= f(\text{net}_j), \\ \text{net}_j &= o.w_j, \quad j = 1, \ldots, n. \end{aligned}$$

Output unit:

$$\begin{aligned} Y &= f(\text{Net}), \\ \text{Net} &= \sum_{j=1}^{n} (O_j.A_j), \end{aligned} \tag{9}$$

where f is the identical activation function. The input–output relations of the FNN which is shown in Figure 2 can be written for the α-level sets as follows:

$$[o]_\alpha = [x_0]_\alpha.$$

Hidden unit:

$$\begin{aligned} [O_j]_\alpha &= f([\text{net}_j]_\alpha), \\ [\text{net}_j]_\alpha &= [o.w_j]_\alpha, \quad j = 1, \ldots, n. \end{aligned}$$

Output unit:

$$\begin{aligned} [Y]_\alpha &= f([\text{Net}]_\alpha), \\ [\text{Net}]_\alpha &= \sum_{j=1}^{n} ([O_j.A_j]_\alpha). \end{aligned}$$

The above relations are written as follows when the α-level sets of the fuzzy input A_j are nonnegative, i.e., $0 \le [A_j]_l^\alpha \le [A_j]_u^\alpha$ for all j:

Input unit:

$$[o]_\alpha = [[o]^l_\alpha, [o]^u_\alpha] = [[x_0]^l_\alpha, [x_0]^u_\alpha].$$

Hidden unit:

$$[O_j]_\alpha = [[O_j]^l_\alpha, [O_j]^u_\alpha] = [f([\text{net}_j]^l_\alpha), f([\text{net}_j]^u_\alpha)].$$

Output unit:

$$\begin{aligned} [Y]_\alpha &= [[Y]^l_\alpha, [Y]^u_\alpha] = [f([\text{Net}]^l_\alpha), f([\text{Net}]^u_\alpha)], \\ [\text{Net}]^l_\alpha &= \sum_{j\epsilon M} [O_j]^l_\alpha . [A_j]^l_\alpha + \sum_{j\epsilon C} [O_j]^l_\alpha . [A_j]^u_\alpha, \\ [\text{Net}]^u_\alpha &= \sum_{j\epsilon M'} [O_j]^u_\alpha . [A_j]^u_\alpha + \sum_{j\epsilon C'} [O_j]^u_\alpha . [A_j]^l_\alpha, \end{aligned} \tag{10}$$

where $M = \{j|\ [O_j]^l_\alpha \geq 0\}$, $C = \{j|\ [O_j]^l_\alpha < 0\}$, $M' = \{j|\ [O_j]^u_\alpha \geq 0\}$, $C' = \{j|\ [O_j]^u_\alpha < 0\}$, $M \cup C = \{1, \ldots, n\}$ and $M' \cup C' = \{1, \ldots, n\}$.

2.4 *Cost function*

Let the α-level sets of the fuzzy target output A_0 be denoted by

$$[A_0]_\alpha = [[A_0]^l_\alpha,\ [A_0]^u_\alpha], \quad \alpha \in [0, 1],$$

where $[A_0]^l_\alpha$ denotes the left-hand side and $[A_0]^u_\alpha$ denotes the right-hand side of the α-level sets of the desired output. A cost function to be minimized is defined for each α-level set as follows:

$$e_\alpha = e^l_\alpha + e^u_\alpha, \tag{11}$$

where

$$e^l_\alpha = \alpha . \frac{([A_0]^l_\alpha - [Y]^l_\alpha)^2}{2}, \tag{12}$$

$$e^u_\alpha = \alpha . \frac{([A_0]^u_\alpha - [Y]^u_\alpha)^2}{2}. \tag{13}$$

In the cost function, e^l_α and e^u_α can be viewed as the squared errors for the lower limits and the upper limits of the α-level sets of the fuzzy output Y and target output A_0, respectively. Then the total error of the given neural network is obtained as

$$e = \sum_\alpha e_\alpha .$$

2.4.1 *Learning algorithm of feed-forward FNNs*

Let us derive a learning algorithm of the FNN from the cost function e defined for the α-level sets in the last subsection, and let the triangular fuzzy weight w_j is denoted by its three parameters

as $w_j = (w_j^{\mathrm{l}}, w_j^{\mathrm{c}}, w_j^{\mathrm{u}})$ where $\mathrm{l}, \mathrm{u}, \mathrm{c}$ denote the lower, center and upper limits of a triangular fuzzy number, respectively. Let us assume that the fuzzy weight w_j is symmetric, i.e.,

$$w_j^c = \frac{w_j^l + w_j^u}{2}. \tag{14}$$

Our main aim is in adjusting w_1 by using the learning algorithm which is introduced below. The weight is updated by the following rule [5,16]:

$$\begin{aligned} w_1^l(t+1) &= w_1^l(t) + \Delta w_1^l(t), \\ w_1^u(t+1) &= w_1^u(t) + \Delta w_1^u(t), \end{aligned} \tag{15}$$

$$\Delta w_1^l(t) = -\eta . \frac{\partial e_\alpha}{\partial w_1^l} + \gamma . \Delta w_1^l(t-1), \tag{16}$$

$$\Delta w_1^u(t) = -\eta . \frac{\partial e_\alpha}{\partial w_1^u} + \gamma . \Delta w_1^u(t-1), \tag{17}$$

where t is the number of adjustments, η is the learning rate and γ is the momentum term constant. As the weights are symmetric triangular fuzzy numbers, the following relation hold for its α-level set:

$$\frac{\partial e_\alpha}{\partial w_1^l} = \frac{\partial e_\alpha}{\partial [w_1]_\alpha^l} . \left(1 - \frac{\alpha}{2}\right) + \frac{\partial e_\alpha}{\partial [w_1]_\alpha^u} . \frac{\alpha}{2}, \tag{18}$$

$$\frac{\partial e_\alpha}{\partial w_1^u} = \frac{\partial e_\alpha}{\partial [w_1]_\alpha^l} . \frac{\alpha}{2} + \frac{\partial e_\alpha}{\partial [w_1]_\alpha^u} . \left(1 - \frac{\alpha}{2}\right). \tag{19}$$

The derivatives $\partial e_\alpha / \partial w_1^l$ and $\partial e_\alpha / \partial w_1^u$ in Equations (16) and (17) can be calculated from the input–output relation of the FNN, as follows:

$$\frac{\partial e_\alpha}{\partial w_1^l} = \frac{\partial e_\alpha}{\partial [Y]_\alpha^l} . \frac{\partial [Y]_\alpha^l}{\partial [\mathrm{Net}]_\alpha^l} . \frac{\partial [\mathrm{Net}]_\alpha^l}{\partial [w_j]_\alpha^l} . \frac{\partial [w_j]_\alpha^l}{\partial [w_1]_\alpha^l} . \left(1 - \frac{\alpha}{2}\right) + \frac{\partial e_\alpha}{\partial [Y]_\alpha^u} . \frac{\partial [Y]_\alpha^u}{\partial [\mathrm{Net}]_\alpha^u} . \frac{\partial [\mathrm{Net}]_\alpha^u}{\partial [w_j]_\alpha^u} . \frac{\partial [w_j]_\alpha^u}{\partial [w_1]_\alpha^u} . \frac{\alpha}{2},$$

$$\frac{\partial e_\alpha}{\partial w_1^u} = \frac{\partial e_\alpha}{\partial [Y]_\alpha^l} . \frac{\partial [Y]_\alpha^l}{\partial [\mathrm{Net}]_\alpha^l} . \frac{\partial [\mathrm{Net}]_\alpha^l}{\partial [w_j]_\alpha^l} . \frac{\partial [w_j]_\alpha^l}{\partial [w_1]_\alpha^l} . \frac{\alpha}{2} + \frac{\partial e_\alpha}{\partial [Y]_\alpha^u} . \frac{\partial [Y]_\alpha^u}{\partial [\mathrm{Net}]_\alpha^u} . \frac{\partial [\mathrm{Net}]_\alpha^u}{\partial [w_j]_\alpha^u} . \frac{\partial [w_j]_\alpha^u}{\partial [w_1]_\alpha^u} . \left(1 - \frac{\alpha}{2}\right).$$

The other case can be calculated similarly. We can update other weights as follows:

$$w_j = w_1^j, \quad j = 2, \ldots, n. \tag{20}$$

2.4.2 *Learning algorithm of feed-back FNNs*

Consider the learning algorithm of the three-layer fuzzy feed-back neural network with one input unit, n neuron in the hidden layer and one output unit as shown in Figure 2. Let x_0 be a triangular fuzzy number that is denoted by its three parameters as $x_0 = (x_0^{\mathrm{l}}, x_0^{\mathrm{c}}, x_0^{\mathrm{u}})$ where $\mathrm{l}, \mathrm{u}, \mathrm{c}$ denote the lower, center and upper limits of a triangular fuzzy number, respectively. Let us assume that the

x_0 is symmetric, i.e.,

$$x_0^c = \frac{x_0^l + x_0^u}{2}. \tag{21}$$

Our main aim is in adjusting the parameter x_0 by using the learning algorithm which is introduced below. For fuzzy parameter x_0 adjustment rule can be written as

$$\begin{aligned} x_0^l(t+1) &= x_0^l(t) + \Delta x_0^l(t), \\ x_0^u(t+1) &= x_0^u(t) + \Delta x_0^u(t), \end{aligned} \tag{22}$$

$$\Delta x_0^l(t) = -\eta . \frac{\partial e_\alpha}{\partial x_0^l} + \gamma . \Delta x_0^l(t-1), \tag{23}$$

$$\Delta x_0^u(t) = -\eta . \frac{\partial e_\alpha}{\partial x_0^u} + \gamma . \Delta x_0^u(t-1), \tag{24}$$

where t is the number of adjustments, η is the learning rate and γ is the momentum term constant. As x_0 is symmetric triangular fuzzy number, the following relations hold:

$$\frac{\partial e_\alpha}{\partial x_0^l} = \frac{\partial e_\alpha}{\partial [x_0]_\alpha^l} . \left(1 - \frac{\alpha}{2}\right) + \frac{\partial e_\alpha}{\partial [x_0]_\alpha^u} . \frac{\alpha}{2}, \tag{25}$$

$$\frac{\partial e_\alpha}{\partial x_0^u} = \frac{\partial e_\alpha}{\partial [x_0]_\alpha^l} . \frac{\alpha}{2} + \frac{\partial e_\alpha}{\partial [x_0]_\alpha^u} . \left(1 - \frac{\alpha}{2}\right). \tag{26}$$

The derivatives $\partial e_\alpha / \partial x_0^l$ and $\partial e_\alpha / \partial x_0^u$ in Equations (23) and (24) can be calculated from the input–output relation of the FNN, as follows:

$$\frac{\partial e_\alpha}{\partial x_0^l} = \frac{\partial e_\alpha}{\partial [Y]_\alpha^l} . \frac{\partial [Y]_\alpha^l}{\partial [\mathrm{Net}]_\alpha^l} . \frac{\partial [\mathrm{Net}]_\alpha^l}{\partial [\mathrm{net}_j]_\alpha^l} . \frac{\partial [\mathrm{net}_j]_\alpha^l}{\partial [x_0]_\alpha^l} . \left(1 - \frac{\alpha}{2}\right) + \frac{\partial e_\alpha}{\partial [Y]_\alpha^u} . \frac{\partial [Y]_\alpha^u}{\partial [\mathrm{Net}]_\alpha^u} . \frac{\partial [\mathrm{Net}]_\alpha^u}{\partial [\mathrm{net}_j]_\alpha^u} . \frac{\partial [\mathrm{net}_j]_\alpha^u}{\partial [x_0]_\alpha^u} . \frac{\alpha}{2},$$

$$\frac{\partial e_\alpha}{\partial x_0^u} = \frac{\partial e_\alpha}{\partial [Y]_\alpha^l} . \frac{\partial [Y]_\alpha^l}{\partial [\mathrm{Net}]_\alpha^l} . \frac{\partial [\mathrm{Net}]_\alpha^l}{\partial [\mathrm{net}_j]_\alpha^l} . \frac{\partial [\mathrm{net}_j]_\alpha^l}{\partial [x_0]_\alpha^l} . \frac{\alpha}{2} + \frac{\partial e_\alpha}{\partial [Y]_\alpha^u} . \frac{\partial [Y]_\alpha^u}{\partial [\mathrm{Net}]_\alpha^u} . \frac{\partial [\mathrm{Net}]_\alpha^u}{\partial [\mathrm{net}_j]_\alpha^u} . \frac{\partial [\mathrm{net}_j]_\alpha^u}{\partial [x_0]_\alpha^u} . \left(1 - \frac{\alpha}{2}\right).$$

The other case can be calculated similarly. The connection weights are updated as follows:

$$w_j = x_0^{j-1}, \quad j = 1, \ldots, n. \tag{27}$$

2.4.3 *Convergence analysis*

For the convergence analysis of FNN3 (it is the general case of FNN) it is sufficient to show that the neural network is Universal approximators. Feuring [12] shows that these neural networks can approximate any fuzzy continuous monotonic function on a compact domain to any degree of accuracy. We should mention that our definition of fuzzy function is as in [13], and these functions can be described as a spatial form and it leads us to present a universal approximation theorem for these neural networks.

THEOREM 1 *Let U be a compact set in E (the set of all fuzzy number) and $F(U)$ the set of all fuzzy functions on U, then for every $f \in F(U)$ and for every $\epsilon > 0$ there exists a p in all fuzzy pseudopolynomials such that*

$$d(f(x), p(x)) < \epsilon, \quad \forall x \in U. \tag{28}$$

Proof See [12].

The corollary of this theorem is the universal approximation of FNN.

Monotony of FNNs has some further advantages. FNNs operate monotonic relative to the supports of the triangular fuzzy numbers. This means that for the output $b = (b_1, \ldots, b_n)$ of FNN with input data $a = (a_1, \ldots, a_m)$. For all input data, whose supports are subsets of the support of a, the support of the corresponding output data will be subsets of the support of b. This fact enables us to gain information about the reaction of the net on yet unknown input data. We only have to cover the input space of the net with the supports of the data of the training set This effect can be used to diminish the risk of overtraining [13]. ■

3. Applications and methodologies

This section contains three examples of FFPs. In these examples, we illustrate the use of feed-back neural network technique to approximate the solutions of the given polynomials. For each example, the computed values of the approximate solution are calculated over a number of iterations and the cost function is plotted. Also the present method is compared with feed-forward neural network. We start by the application of this method in fluid mechanics. All example programs are written in Matlab 2012 and run with computer CORE I7.

Example 1 (An example of fluid mechanics) A pump is in a pipeline from a suction reservoir to a junction to which three reservoirs are connected with pipes at a constant H, as in Figure 3. There is a check valve at the pump. Assume that the pump is operating with flow through the pump. The pump equation is given by

$$H = A_0 + A_1 Q + A_2 Q^2 + A_3 Q^3,$$

where H is the height and A_0, A_1, A_2 and A_3 are the characteristic coefficients of the pump.

Where

$$A_0 = (90, 100, 110), \quad A_1 = (0.1, 0.2, 0.3), \quad A_2 = (0.02, 0.03, 0.04),$$
$$A_3 = (0.006, 0.007, 0.008), \quad H = (99.768831, 175.232772, 378.97805),$$

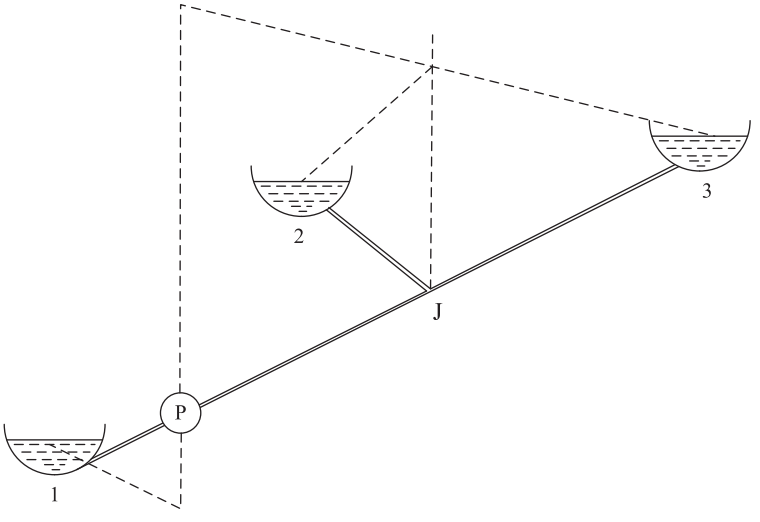

Figure 3. Pumping from one reservoir to the other two reservoirs.

Table 1. The approximate solutions with error analysis for Example 1 by FFFNN3 and FBFNN3.

t	$Q_0(t)$ by FFFNN3	Error for FFFNN3
1	(6.9724, 17.5164, 28.4604)	22461.4290
2	(7.1932, 17.9974, 28.8015)	3633.7638
3	(7.4067, 18.4572, 29.1077)	734.04810
4	(7.7929, 18.7237, 29.5544)	470.69450
5	(7.9880, 19.0756, 29.9924)	396.85044
⋮	⋮	⋮
62	(10.2886, 20.3121, 30.3190)	0.0259697
63	(10.2911, 20.3130, 30.3199)	0.0198803
64	(10.2934, 20.3138, 30.3207)	0.0135884
65	(10.2959, 20.3147, 30.3215)	0.0109384
66	(10.2979, 20.3154, 30.3220)	0.0088052
t	$Q_0(t)$ by FBFNN3	Error for FBFNN3
1	(6.9906, 17.7698, 28.6974)	22461.429
2	(7.3906, 18.2698, 29.0974)	1013.4442
3	(7.6004, 18.7926, 29.4316)	567.90418
4	(7.9267, 19.1782, 29.8704)	343.68958
5	(8.1348, 19.4102, 30.1510)	143.65824
⋮	⋮	⋮
37	(10.2864, 20.3094, 30.3234)	0.0235290
38	(10.2910, 20.3113, 30.3236)	0.0172635
39	(10.2951, 20.3129, 30.3238)	0.0111748
40	(10.2986, 20.3144, 30.3240)	0.0100014
41	(10.3018, 20.3156, 30.3241)	0.0065356

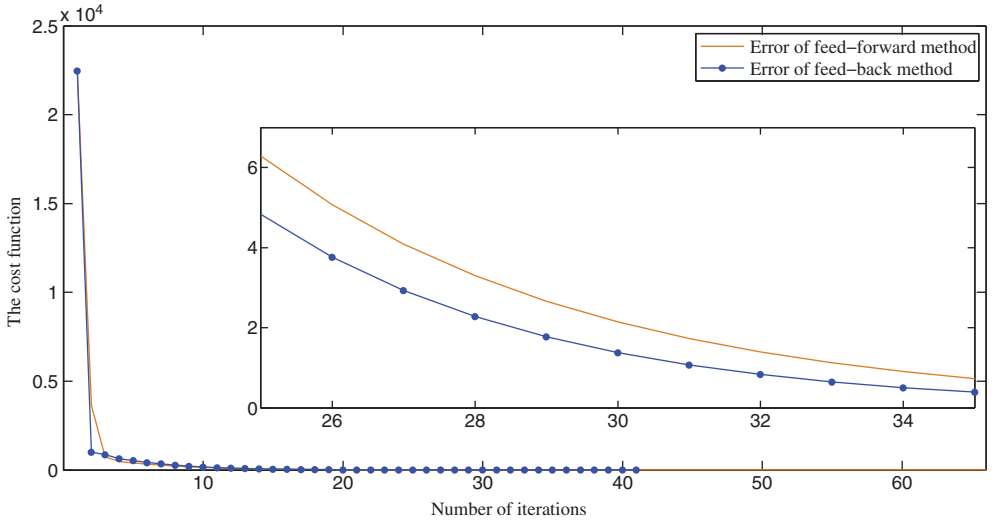

Figure 4. The cost function for Example 1 on the number of iterations with both methods (FFFNN3 and FBFNN3).

and the exact solution is $Q = (10.325, 20.325, 30.325)$. Before starting calculations, we assumed that $Q_0 = (6.325, 17.325, 28.325)$, $\eta = 2 \times 10^{-3}$ and $\gamma = 2 \times 10^{-3}$. Numerical results can be found in Table 1. The right section of the table is related to the feed-back neural network and the left section is related to the feed-forward neural network. Figure 4 shows the accuracy of the solution $Q_0(t)$ where t is the number of iterations, its noticeable that by increasing the iterations

the cost function goes to zero. In this figure the curve with line is related to feed-forward neural network and the curve with line and plot is related to feed-back neural network. The cost function of FBFNN3 goes to zero faster than FFFNN3.

Example 2 Let the fuzzy equation be as follows:

$$(3,4,5)x + (2,3,4)x^2 + (1,2,3)x^3 + (1,3,4,5)x^4 = A_0,$$

where

$$\begin{aligned}(\underline{A}_0(r), \bar{A}_0(r)) = (&2r^5 + 42r^4 + 337r^3 + 1253r^2 + 2003r + 815,\\ &-r^5 + 34r^4 - 459r^3 + 3071r^2 - 10162r + 13265), \quad 0 \le r \le 1.\end{aligned}$$

The exact solution is $x = (5,6,7)$. Before starting calculations, we assumed that $x_0 = (3,4,5)$, $\eta = 5 \times 10^{-3}$ and $\gamma = 5 \times 10^{-3}$. Similarly Table 2 shows the approximated solution over a number of iterations. The right section of the table is related to the feed-back neural network and the left section is related to the feed-forward neural network. Figure 5 shows the accuracy of the solution $x_0(t)$, where t is the number of iterations, it is noticeable that by increasing the iterations the cost function goes to zero. In this figure the curve with a line is related to feed-forward neural network and the curve with a line and a plot is related to the feed-back neural network. The cost function of FBFNN3 goes to zero faster than FFFNN3.

Table 2. The approximate solutions with error analysis for Example 2 by FFFNN3 and FBFNN3.

t	$x_0(t)$ by FFFNN3	Error for FFFNN3
1	(3.3568, 4.2366, 5.2576)	2124422.8410
2	(3.6167, 4.5395, 5.6532)	924799.2631
3	(3.9854, 4.9348, 5.9012)	325899.1532
4	(4.0049, 5.4603, 6.2565)	98695.46686
5	(4.2257, 5.6878, 6.6156)	65248.42598
⋮	⋮	⋮
54	(4.9997, 5.9994, 7.0002)	0.352426384
55	(4.9998, 5.9995, 7.0003)	0.252373257
56	(4.9998, 5.9997, 7.0004)	0.180724779
57	(4.9999, 5.9998, 7.0005)	0.129417063
58	(4.9999, 5.9999, 7.0005)	0.092675511
t	$x_0(t)$ by FBFNN3	Error for FBFNN3
1	(3.4053, 4.3280, 5.5477)	2124422.84100
2	(3.7857, 4.7280, 5.8478)	777333.86400
3	(4.0360, 5.3623, 6.0507)	165348.47820
4	(4.3165, 5.6547, 6.6515)	74618.137540
5	(4.5282, 5.8973, 6.8886)	44406.909510
⋮	⋮	⋮
33	(4.9981, 5.9993, 7.0003)	0.5460984210
34	(4.9984, 5.9995, 7.0003)	0.3542778220
35	(4.9988, 5.9996, 7.0004)	0.2298205110
36	(4.9990, 5.9997, 7.0005)	0.1490770750
37	(4.9992, 5.9997, 7.0005)	0.0966973540

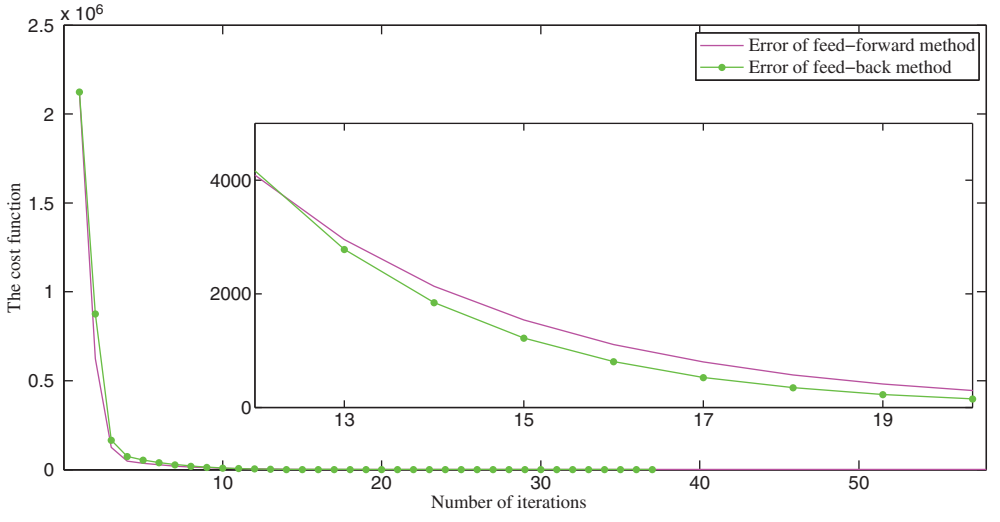

Figure 5. The cost function for Example 2 on the number of iterations with both methods (FFFNN3 and FBFNN3).

Table 3. The approximated solutions with error analysis by FFFNN3 and FBFNN3 for Example 3.

t	$x_0(t)$ by FFFNN3	Error for FFFNN3
1	$(-3.8751, -2.9129, -1.9507)$	210260.833
2	$(-3.7681, -2.8376, -1.9071)$	96524.5550
3	$(-3.6754, -2.7719, -1.8684)$	6887.66709
4	$(-3.5238, -2.6630, -1.8021)$	9065.77973
5	$(-3.4614, -2.6174, -1.7735)$	5877.84571
⋮	⋮	⋮
110	$(-2.9999, -2.0017, -1.0044)$	0.14273388
111	$(-2.9998, -2.0016, -1.0043)$	0.13286556
112	$(-2.9998, -2.0015, -1.0041)$	0.12076212
113	$(-2.9997, -2.0014, -1.0039)$	0.10976250
114	$(-2.9997, -2.0013, -1.0037)$	0.09976587
t	$x_0(t)$ by FBFNN3	Error for FBFNN3
1	$(-3.7956, -2.8495, -1.9032)$	210260.833
2	$(-3.5957, -2.7495, -1.8538)$	73796.8014
3	$(-3.4103, -2.6320, -1.7939)$	3977.03671
4	$(-3.2936, -2.5030, -1.6950)$	7605.97683
5	$(-3.1515, -2.3507, -1.4876)$	3941.37686
⋮	⋮	⋮
56	$(-2.9997, -2.0024, -1.0056)$	0.13392917
57	$(-2.9996, -2.0022, -1.0052)$	0.11566727
58	$(-2.9995, -2.0020, -1.0048)$	0.11566727
59	$(-2.9994, -2.0019, -1.0044)$	0.10368734
60	$(-2.9994, -2.0017, -1.0040)$	0.09294612

Example 3 Consider the following fuzzy equation problem:

$$(2,3,4)x + (1,2,3)x^2 + (5,6,7)x^3 = A_0,$$

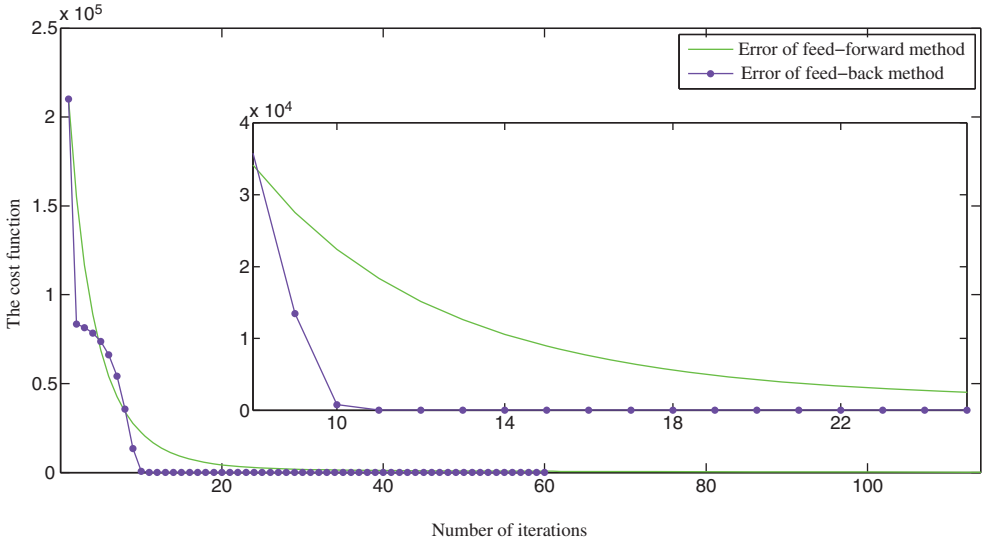

Figure 6. The cost function for Example 3 on the number of iterations with both methods (FFFNN3 and FBFNN3).

where

$$(\underline{A}_0(r), \bar{A}_0(r)) = (-r^4 + 17r^3 - 88r^2 + 226r - 200,\ -r^4 - 9r^3 - 10r^2 - 46r + 20),$$
$$0 \le r \le 1.$$

The exact solution is $x = (-3, -2, -1)$. This problem is solved by helping the feed-forward and feed-back FNNs as described in this paper. Let $x_0 = (-4, -3, -2)$, $\eta = 2 \times 10^{-3}$ and $\gamma = 2 \times 10^{-3}$. Table 3 shows the approximated solution over a number of iterations. The right section of the table is related to the feed-back neural network and the left section is related to the feed-forward neural network. We see that the FBFNN3 convergence is faster than FFFNN3 to the exact solution. Figure 6 shows the accuracy of the solution $x_0(t)$ where t is the number of iterations, its noticeable that by increasing the iterations the cost function goes to zero. In this figure the curve with a line is related to feed-forward neural network and the curve with a line and a plot is related to the feed-back neural network. The cost function of FBFNN3 goes to zero faster than FFFNN3.

4. Concluding remarks

The topics of FNNs which have been attracting growing interest for some time are being developed in recent years. In this paper, a FFNN3 model equivalent to the fuzzy polynomial is built, and a learning algorithm of feed-back FNN is introduced on the basis of the input–output relations defined by the extension principle, to find the fuzzy root of fuzzy polynomial. In this work, we assumed the signals and weights are fuzzy numbers. The model is also compared with the feed-forward model. The comparison of these two models have shown us the advantage of feed-back neural network to feed-forward in the speed of adjusting fuzzy weights. The analysed examples illustrated the ability and reliability of feed-back neural network. As the development of FNNs has suffered greatly from the lack of training method, we believe that our approach fills this void and hope that it leads to many new applications.

References

[1] S. Abbasbandy and M. Amirfakhrian, *Numerical approximation of fuzzy functions by fuzzy polynomials*, Appl. Math. Comput. 174 (2006), pp. 669–675.
[2] S. Abbasbandy and B. Asady, *Newton's method for solving fuzzy nonlinear equations*, Appl. Math. Comput. 159 (2004), pp. 379–356.
[3] S. Abbasbandy and R. Ezzati, *Newton's method for solving fuzzy nonlinear equations*, Appl. Math. Comput. 175 (2006), pp. 1189–1199.
[4] S. Abbasbandy and M. Otadi, *Numerical solution of fuzzy polynomials by fuzzy neural network*, Appl. Math. Comput. 181 (2006), pp. 1084–1089.
[5] G. Alefeld and J. Herzberger, *Introduction to Interval Computations*, Academic Press, New York, 1983.
[6] T. Allahviranloo and S. Asari, *Numerical solution of fuzzy polynomials by Newton–Raphson method*, J. Appl. Math. 27 (2011), pp. 17–24.
[7] S.I. Ao, *A hybrid neural network cybernetic system for quantifying cross-market dynamics and business forecasting*, Soft Comput. 15 (2010), pp. 1041–1053.
[8] B. Asady, S. Abbasbandy, and M. Alavi, *Fuzzy general linear systems*, Appl. Math. Comput. 169 (2005), pp. 34–40.
[9] J.J. Buckley and Y. Qu, *Solving linear and quadratic fuzzy equations*, Fuzzy Sets Syst. 35 (1990), pp. 43–59.
[10] D.K. Chaturvedi, R. Chauhan, and P.K. Kalra, *Application of generalised neural network for aircraft landing control system*, Soft Comput. 6 (2002), pp. 441–448.
[11] M. Delgado, M.A. Vila, and W. Voxman, *On a canonical representation of fuzzy Numbers*, Fuzzy Sets Syst. 93 (1998), pp. 125–135.
[12] Th. Feuring, *Fuzzy Neuronale Netze Von kooperativen uber hybride zu fusionierten vage konnektionistischen Systemen*, Ph.D Thesis Westf. Wilhelms-University-Munster Germany, 1996.
[13] Th. Feuring and W. Lippe, *Fuzzy Neural Network are Universal Approximator*, World Congress on Artificial neural network, Sãu Paulo, Brazil, 1995.
[14] X.Z. Gao, S.J. Ovaska, and A.V. Vasilakos, *A modified Elman neural network-based power controller in mobile communications systems*, Soft Comput. 9 (2005), pp. 88–93.
[15] Y. Hayashi, J.J. Buckley, and E. Czogala, *Fuzzy neural network with fuzzy signals and weights*, Int. J. Intell. Syst. 8 (1993), pp. 527–537.
[16] H. Ishibuchi, K. Kwon, and H. Tanaka, *A learning of fuzzy neural networks with triangular fuzzy weights*, Fuzzy Sets Syst. 71 (1995), pp. 277–293.
[17] A. Jafarian and R. Jafari, *Approximate solutions of dual fuzzy polynomials by feed-back neural networks*, J. Soft Comput. Appl. (2012), doi:10.5899/2012/jsca-00005.
[18] A. Jafarian and S. Measoomynia, *Solving fuzzy polynomials using neural nets with a new learning algorithm*, Appl. Math. Sci. 5 (2011), pp. 2295–2301.
[19] H.T. Nguyen and V. Kreinovich, *Nested intervals and sets: concepts, relations to fuzzy sets, and applications*, in *Applications of Interval Computations*, R.B. Kearfott and V. Kreinovich, eds., Kluwer Academic Publishers, The Netherlands, 1996, pp. 245–290.
[20] A. Noorani, J. Kavikumar, M. Mustafa, and S. Nor, *Solving dual fuzzy polynomial equation by ranking method*, Far East J. Math. Sci. 15 (2011), pp. 151–163.
[21] S.K. Oh, W. Pedrycz, and S.B. Roh, *Genetically optimized fuzzy polynomial neural networks with fuzzy set-based polynomial neurons*, Inform. Sci. 176 (2006), pp. 3490–3519.
[22] M. Otadi and M. Mosleh, *Solution of fuzzy polynomial equations by modified Adomian decomposition method*, Soft Comput. 15 (2011), pp. 187–192.
[23] H. Rouhparvar, *Solving fuzzy polynomial equation by ranking method. First Joint Congress on Fuzzy and Intelligent Systems*, Ferdowsi University of Mashhad, Iran, 2007.
[24] C.C. Wang and C.F. Tsai, *Fuzzy processing using polynomial bidirectional hetero-associative network*, Inform. Sci. 125 (2000), pp. 167–179.
[25] S.J. Yoo, J.B. Park, and Y.H. Choi, *Indirect adaptive control of nonlinear dynamic systems using self recurrent wavelet neural networks via adaptive learning rates*, Inform. Sci. 177 (2007), pp. 3074–3098.
[26] L.A. Zadeh, *The concept of a linguistic variable and its application to approximate reasoning*, Inform. Sci. 8 (1975), pp. 199–249.