Available online at www.sciencedirect.com

ScienceDirect

Information Sciences 178 (2008) 1948–1960

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Numerical solution of a system of fuzzy polynomials by fuzzy neural network

S. Abbasbandy[a,*], M. Otadi[b], M. Mosleh[b]

[a] Department of Mathematics, Faculty of Science, Imam Khomeini International University, Ghazvin 34149-16818, Iran
[b] Iran Department of Mathematics, Islamic Azad University, Firuozkooh Branch, Firuozkooh, Iran

Received 2 September 2006; received in revised form 24 November 2007; accepted 30 November 2007

Abstract

In this paper, a new approach for solving systems of fuzzy polynomials based on fuzzy neural network (FNN) is presented. This method can also lead to improve numerical methods. In this work, an architecture of fuzzy neural networks is also proposed to find a real root of a system of fuzzy polynomials (if exists) by introducing a learning algorithm. Finally, we illustrate our approach by numerical examples.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Fuzzy numbers; Fuzzy neural networks; System of fuzzy polynomials

1. Introduction

Systems of polynomials play a major role in various areas such as pure and applied mathematics, engineering and social sciences. A previous paper "Numerical solution of fuzzy polynomials by fuzzy neural network" presented by Abbasbandy and Otadi [6], was aimed at solving a fuzzy polynomial equation such as $A_1x + A_2x^2 + \cdots + A_nx^n = A_0$, where $A_0, A_1, \ldots, A_n$ are fuzzy numbers.

Here, we consider the following system of fuzzy polynomials of the form:

$$\begin{cases} A_{11}xy + A_{12}x^2y^2 + \cdots + A_{1n}x^ny^n = A_{10}, \\ A_{21}xy + A_{22}x^2y^2 + \cdots + A_{2n}x^ny^n = A_{20}, \end{cases}$$

where $x, y \in \mathbb{R}$ and $A_{10}, A_{11}, \ldots, A_{1n}, A_{20}, A_{21}, \ldots, A_{2n}$ are fuzzy numbers. We can also consider a more general system F, where F denotes a system of s fuzzy polynomial equations such as

* Corresponding author. Tel.: +98 9112305326; fax: +98 21 2442052.
E-mail address: abbasbandy@yahoo.com (S. Abbasbandy).

0020-0255/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2007.11.026

$$\begin{aligned} f_1(x_1,x_2,\ldots,x_n) &= A_{10},\\ &\vdots\\ f_l(x_1,x_2,\ldots,x_n) &= A_{l0},\\ &\vdots\\ f_s(x_1,x_2,\ldots,x_n) &= A_{s0}, \end{aligned}$$

where $x_1,x_2,\ldots,x_n \in \mathbb{R}$ and all coefficients are fuzzy numbers. In this paper, we are interested in finding solution of such a system of fuzzy polynomials.

We wish to find the answer to this question: How is the fuzzy neural network going to solve the system of the fuzzy polynomials? In this paper, an architecture of fuzzy neural network is proposed to find a real root of a system of fuzzy polynomials by introducing a learning algorithm. We refer the reader to [19,20,23] for more information on such polynomials. Some applications of fuzzy polynomials are considered by Abbasbandy, Amirfakhrian, and Li [3,16].

During the past few years, neural networks have received much attention [11,13,18,22,24]. Ishibuchi et al. [14] proposed a learning algorithm of fuzzy neural networks with triangular fuzzy weights and Hayashi et al. [12] also used the fuzzy delta rule. Linear and nonlinear fuzzy equations are solved by [1,2,5,8,10], also Buckley and Eslami [9] considered neural net solutions to fuzzy problems.

In this paper, we first propose an architecture of fuzzy neural networks with crisp weights for fuzzy input vector and fuzzy target. The input–output relation of each unit is defined by the extension principle of Zadeh [25,26]. The output from the fuzzy neural network, which is also a fuzzy number, is numerically calculated by the interval arithmetic [7] for numeric 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 crisp learning algorithm is derived from the cost function to find a real root (if exists) of a system of polynomials. The effectiveness of the proposed algorithm is illustrated by solving some examples.

2. Preliminaries

We represent an arbitrary fuzzy number by an ordered pair of functions $(\underline{u}(r),\bar{u}(r)), 0 \leqslant r \leqslant 1$, which satisfy the following requirements [4,17]:

1. $\underline{u}(r)$ is a bounded left continuous non decreasing function on [0, 1].
2. $\bar{u}(r)$ is a bounded left continuous non increasing function on [0, 1].
3. $\underline{u}(r) \leqslant \bar{u}(r),\ 0 \leqslant r \leqslant 1$.

The crisp number λ is simply represented by $\underline{u}(r) = \bar{u}(r) = \lambda,\ 0 \leqslant r \leqslant 1$. The set of all the fuzzy numbers is denoted by E^1. A popular fuzzy number is the triangular fuzzy number $u = (m,\alpha,\beta)$ with membership function,

$$\mu_u(x) = \begin{cases} \frac{x-m}{\alpha}+1, & m-\alpha \leqslant x \leqslant m,\\ \frac{m-x}{\beta}+1, & m \leqslant x \leqslant m+\beta,\\ 0, & \text{otherwise,} \end{cases}$$

where $\alpha > 0$ and $\beta > 0$. Its parametric form is

$$\underline{u}(r) = m + \alpha(r-1), \quad \bar{u}(r) = m + \beta(1-r).$$

2.1. Operations on fuzzy numbers

We briefly mention fuzzy number operations defined by the extension principle [25,26]. Since input vector of feedforward neural network is fuzzy in this paper, the following addition, multiplication and nonlinear mapping of fuzzy numbers are necessary to define when dealing with fuzzy neural network:

$$\mu_{A+B}(z) = \max\{\mu_A(x) \wedge \mu_B(y) | z = x + y\}, \tag{1}$$

$$\mu_{f(Net)}(z) = \max\{\mu_{Net}(x) | z = f(x)\}, \tag{2}$$

where A, B, Net are fuzzy numbers, $\mu_*(\cdot)$ denotes the membership function of each fuzzy number, $\wedge$ is the minimum operator, and $f(x) = x$ is the activation function of hidden units and output unit of our fuzzy neural network.

The above operations on fuzzy numbers are numerically performed on level sets (i.e., α-cuts). For $0 < h \leqslant 1$, a h-level set of a fuzzy number X is defined as

$$[X]_h = \{x | \mu_X(x) \geqslant h, x \in \mathbb{R}\}, \tag{3}$$

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

$$[X]_h = [[X]_h^{\mathrm{L}}, [X]_h^{\mathrm{U}}], \tag{4}$$

where $[X]_h^{\mathrm{L}}$ and $[X]_h^{\mathrm{U}}$ are the lower and the upper limits of the h-level set $[X]_h$, respectively.

From interval arithmetic [7], the above operations on fuzzy numbers are written for the h-level sets as follows:

$$[A]_h + [B]_h = [[A]_h^{\mathrm{L}} + [B]_h^{\mathrm{L}}, [A]_h^{\mathrm{U}} + [B]_h^{\mathrm{U}}], \tag{5}$$

$$f([Net]_h) = f([[Net]_h^{\mathrm{L}}, [Net]_h^{\mathrm{U}}]) = [f([Net]_h^{\mathrm{L}}), f([Net]_h^{\mathrm{U}})], \tag{6}$$

$$\begin{aligned} w \cdot [A]_h &= [w \cdot [A]_h^{\mathrm{L}}, w \cdot [A]_h^{\mathrm{U}}], \quad \text{if } w \geqslant 0, \\ w \cdot [A]_h &= [w \cdot [A]_h^{\mathrm{U}}, w \cdot [A]_h^{\mathrm{L}}], \quad \text{if } w < 0. \end{aligned} \tag{7}$$

2.2. Input–output relation of each unit

Let's consider a fuzzy three-layer feedforward neural network with n input units and one-output unit. Input vector and target vector are fuzzy and weights are crisp. The input–output relation of each unit of the fuzzy neural network can be written as follows:

Input units:

$$O_{ij} = A_{ij}, \quad i = 1, 2; \ j = 1, 2, \ldots, n. \tag{8}$$

Hidden units:

$$O'_{ij} = f(Net_{ij}), \tag{9}$$

$$Net_{ij} = w_{ij} \cdot O_{ij}, \quad i = 1, 2; \ j = 1, 2, \ldots, n. \tag{10}$$

Output unit:

$$Y_i = f(Net'_i), \tag{11}$$

$$Net'_i = \sum_{j=1}^{n} w'_{ij} \cdot O'_{ij}, \quad i = 1, 2, \tag{12}$$

where A_{ij} is a fuzzy input and w_{ij}, w'_{ij} are crisp weights.

The input–output relation in Eqs. (8)–(12) are defined by the extension principle [25,26] as in Hayashi et al. [12] and Ishibuchi et al. [15].

2.3. Calculation of fuzzy output

The fuzzy output from each unit in Eqs. (8)–(12) is numerically calculated for crisp weights and level sets of fuzzy inputs. The input–output relations of our fuzzy neural network can be written for the h-level sets as follows:

Input units:

$$[O_{ij}]_h = [A_{ij}]_h, \quad i = 1,2;\ \ j = 1,2,\ldots,n. \tag{13}$$

Hidden units:

$$[O'_{ij}]_h = f([Net_{ij}]_h), \tag{14}$$

$$[Net_{ij}]_h = w_{ij} \cdot [O_{ij}]_h, \quad i = 1,2;\ \ j = 1,2,\ldots,n. \tag{15}$$

Output unit:

$$[Y_i]_h = f([Net'_i]_h), \tag{16}$$

$$[Net'_i]_h = \sum_{j=1}^{n} w'_{ij} \cdot [O'_{ij}]_h, \quad i = 1,2. \tag{17}$$

From Eqs. (13)–(17), we can see that the h-level sets of the fuzzy output Y_i is calculated from those of the fuzzy inputs and crisp weights. From Eqs. (5)–(7), the above relations are written as follows:

Input units:

$$[O_{ij}]_h = [[O_{ij}]_h^{\mathrm{L}}, [O_{ij}]_h^{\mathrm{U}}] = [[A_{ij}]_h^{\mathrm{L}}, [A_{ij}]_h^{\mathrm{U}}], \quad i = 1,2; \quad j = 1,2,\ldots,n. \tag{18}$$

Hidden units:

$$[O'_{ij}]_h = [[O'_{ij}]_h^{\mathrm{L}}, \quad [O'_{ij}]_h^{\mathrm{U}}] = [f([Net_{ij}]_h^{\mathrm{L}}), f([Net_{ij}]_h^{\mathrm{U}})]. \tag{19}$$

If $w_{ij} \geqslant 0$

$$[Net_{ij}]_h = [[Net_{ij}]_h^{\mathrm{L}}, \quad [Net_{ij}]_h^{\mathrm{U}}] = [w_{ij}[O_{ij}]_h^{\mathrm{L}}, w_{ij}[O_{ij}]_h^{\mathrm{U}}],$$

otherwise

$$[Net_{ij}]_h = [[Net_{ij}]_h^{\mathrm{L}}, [Net_{ij}]_h^{\mathrm{U}}] = [w_{ij}[O_{ij}]_h^{\mathrm{U}}, w_{ij}[O_{ij}]_h^{\mathrm{L}}], \quad i = 1,2;\ \ j = 1,\ldots,n.$$

Output unit:

$$[Y_i]_h = [[Y_i]_h^{\mathrm{L}}, \quad [Y_i]_h^{\mathrm{U}}] = [f([Net'_i]_h^{\mathrm{L}}), f([Net'_i]_h^{\mathrm{U}})], \tag{20}$$

$$\begin{aligned} [Net'_i]_h &= [[Net'_i]_h^{\mathrm{L}}, [Net'_i]_h^{\mathrm{U}}] \\ &= \left[\sum_{j\in m} w'_{ij} \cdot [O'_{ij}]_h^{\mathrm{L}} + \sum_{j\in c} w'_{ij} \cdot [O'_{ij}]_h^{\mathrm{U}}, \sum_{j\in m} w'_{ij} \cdot [O'_{ij}]_h^{\mathrm{U}} + \sum_{j\in c} w'_{ij} \cdot [O'_{ij}]_h^{\mathrm{L}}\right], \end{aligned} \tag{21}$$

where $m = \{j | w'_{ij} \geqslant 0\}$, $c = \{j | w'_{ij} < 0\}$ and $m \cup c = \{1,\ldots,n\}$.

3. Fuzzy polynomials

We are interested in finding the solution to the following system:

$$\begin{cases} A_{11}xy + A_{12}x^2y^2 + \cdots + A_{1n}x^ny^n = A_{10}, \\ A_{21}xy + A_{22}x^2y^2 + \cdots + A_{2n}x^ny^n = A_{20}, \end{cases} \tag{22}$$

where $x, y \in \mathbb{R}$, and $A_{ij} \in E$ for $i = 1,2;\ j = 0,\ldots,n$. A FNN_2 (fuzzy neural network with fuzzy set input signals and real number weights) [14] solution to Eq. (22) is given in Fig. 1. The input neurons make no changes in their inputs, so the inputs to the hidden neurons are $A_{ij}x^j$ for $i = 1,2,\ j = 1,\ldots n$, and the hidden neurons make no changes in their inputs, so the input to the output neuron is

$$A_{i1}xy + A_{i2}x^2y^2 + \cdots + A_{in}x^ny^n, \quad i = 1,2$$

and the output, in the output neuron, equals its input, so

$$Y_i = A_{i1}xy + A_{i2}x^2y^2 + \cdots + A_{in}x^ny^n, \quad i = 1,2.$$

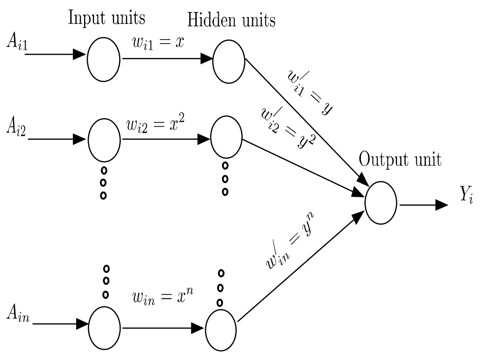

Fig. 1. Fuzzy neural network to solve system of the fuzzy polynomials.

How is the FNN_2 going to solve system of the fuzzy polynomials? The training data is $(A_{i1}, \ldots, A_{in})$ for input and desired target output is A_{i0}. We proposed a learning algorithm from the cost function to adjust weights.

3.1. Learning of fuzzy neural network

Denote the h-level sets of the target outputs $A_{i0},\ i = 1, 2$, by

$$[A_{i0}]_h = [[A_{i0}]_h^{\mathrm{L}}, [A_{i0}]_h^{\mathrm{U}}], \quad h \in [0, 1], \tag{23}$$

where $A_{i0}^{\mathrm{L}}(h)$ denotes the left-hand side and $A_{i0}^{\mathrm{U}}(h)$ denotes the right-hand side of the h-level sets of the desired output. A cost function to be minimized is defined for each h-level sets as follows:

$$\begin{aligned} e_i(h) &= e_i^{\mathrm{L}}(h) + e_i^{\mathrm{U}}(h), \\ e_i^{\mathrm{L}}(h) &= \frac{1}{2}(A_{i0}^{\mathrm{L}}(h) - Y_i^{\mathrm{L}}(h))^2, \\ e_i^{\mathrm{U}}(h) &= \frac{1}{2}(A_{i0}^{\mathrm{U}}(h) - Y_i^{\mathrm{U}}(h))^2, \quad i = 1, 2, \end{aligned} \tag{24}$$

where $e_i^{\mathrm{L}}(h)$ is the error between the left-hand side of each of the h-level sets of the desired and the computed output i-th equation, and $e_i^{\mathrm{U}}(h)$ is the error between the right-hand side of each of the h-level sets of the desired and the computed output ith equation. Then the error function for the training pattern is

$$e = \sum_{i=1}^{2} \sum_h h e_i(h). \tag{25}$$

Theoretically, this cost function satisfies the following equation if we use an infinite number of h-level sets in Eq. (25)

$$e \to 0 \text{ if and only if } Y_i \to A_{i0}, \quad i = 1, 2.$$

The weights are updated by the following rules [14,21]:

$$\Delta w'_{i1}(t) = -\eta \sum_h h \frac{\partial e_i(h)}{\partial w'_{i1}} + \alpha \cdot \Delta w'_{i1}(t-1), \tag{26}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq. (26) can be written as follows:

$$\begin{aligned} \frac{\partial e_i(h)}{\partial w'_{i1}} &= \frac{\partial e_i^{\mathrm{L}}(h)}{\partial w'_{i1}} + \frac{\partial e_i^{\mathrm{U}}(h)}{\partial w'_{i1}}, \\ \frac{\partial e_i^{\mathrm{L}}(h)}{\partial w'_{i1}} &= \frac{\partial e_i^{\mathrm{L}}(h)}{\partial Y_i^{\mathrm{L}}} \times \frac{\partial Y_i^{\mathrm{L}}(h)}{\partial w'_{i1}}, \end{aligned}$$

$$\frac{\partial e_i^{\mathrm{U}}(h)}{\partial w'_{i1}} = \frac{\partial e_i^{\mathrm{U}}(h)}{\partial Y_i^{\mathrm{U}}} \times \frac{\partial Y_i^{\mathrm{U}}(h)}{\partial w'_{i1}},$$
$$\frac{\partial e_i^{\mathrm{L}}(h)}{\partial Y_i^{\mathrm{L}}} = -(A_{i0}^{\mathrm{L}}(h) - Y_i^{\mathrm{L}}(h)), \quad \frac{\partial e_i^{\mathrm{U}}(h)}{\partial Y_i^{\mathrm{U}}} = -(A_{i0}^{\mathrm{U}}(h) - Y_i^{\mathrm{U}}(h)).$$

If $w'_{i1} \geqslant 0$, then

$$\frac{\partial Y_i^{\mathrm{L}}(h)}{\partial w'_{i1}} = O_{i1}^{\prime L}(h), \quad \frac{\partial Y_i^{\mathrm{U}}(h)}{\partial w\prime_{i1}} = O_{i1}^{\prime U}(h);$$

otherwise

$$\frac{\partial Y_i^{\mathrm{L}}(h)}{\partial w'_{i1}} = O_{i1}^{\prime U}(h), \quad \frac{\partial Y_i^{\mathrm{U}}(h)}{\partial w'_{i1}} = O_{i1}^{\prime L}(h).$$

Therefore, if $w'_{i1} \geqslant 0$, then

$$\Delta w'_{i1}(t) = \eta \sum_h h[(A_{i0}^{\mathrm{L}}(h) - Y_i^{\mathrm{L}}(h))O_{i1}^{\prime L}(h) + (A_{i0}^{\mathrm{U}}(h) - Y_i^{\mathrm{U}}(h))O_{i1}^{\prime U}(h)] + \alpha \cdot \Delta w'_{i1}(t-1); \tag{27}$$

otherwise

$$\Delta w'_{i1}(t) = \eta \sum_h h[(A_{i0}^{\mathrm{L}}(h) - Y_i^{\mathrm{L}}(h))O_{i1}^{\prime U}(h) + (A_{i0}^{\mathrm{U}}(h) - Y_i^{\mathrm{U}}(h))O_{i1}^{\prime L}(h)] + \alpha \cdot \Delta w'_{i1}(t-1). \tag{28}$$

We can adjust other weights w'_{ij} by

$$w'_{ij} = w_{i1}^{\prime j} \quad \text{for } i = 1,2; \;\; j = 2,\ldots,n,$$

and the weights are updated by the following rules [14,21]:

$$\Delta w_{i1}(t) = -\eta \sum_h h \frac{\partial e_i(h)}{\partial w_{i1}} + \alpha \cdot \Delta w_{i1}(t-1), \tag{29}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq. (29) can be written as follows:

$$\frac{\partial e_i(h)}{\partial w_{i1}} = \frac{\partial e_i^{\mathrm{L}}(h)}{\partial w_{i1}} + \frac{\partial e_i^{\mathrm{U}}(h)}{\partial w_{i1}}.$$

Suppose that $w'_{i1} \geqslant 0$ and $w_{i1} \geqslant 0$ for $i = 1,2$. Then

$$\frac{\partial e_i^{\mathrm{L}}(h)}{\partial w_{i1}} = \frac{\partial e_i^{\mathrm{L}}(h)}{\partial Y_i^{\mathrm{L}}(h)} \times \frac{\partial Y_i^{\mathrm{L}}(h)}{\partial O'_{i1}L(h)} \times \frac{\partial O_{i1}^{\prime L}(h)}{\partial w_{i1}},$$
$$\frac{\partial e_i^{\mathrm{U}}(h)}{\partial w_{i1}} = \frac{\partial e_i^{\mathrm{U}}(h)}{\partial Y_i^{\mathrm{U}}(h)} \times \frac{\partial Y_i^{\mathrm{U}}(h)}{\partial O_{i1}^{\prime U}(h)} \times \frac{\partial O_{i1}^{\prime U}(h)}{\partial w_{i1}},$$
$$\frac{\partial e_i^{\mathrm{L}}(h)}{\partial Y_i^{\mathrm{L}}(h)} = -(A_{i0}^{\mathrm{L}}(h) - Y_i^{\mathrm{L}}(h)), \quad \frac{\partial e_i^{\mathrm{U}}(h)}{\partial Y_i^{\mathrm{U}}(h)} = -(A_{i0}^{\mathrm{U}}(h) - Y_i^{\mathrm{U}}(h)),$$
$$\frac{\partial Y_i^{\mathrm{L}}(h)}{\partial O_{i1}^{\prime L}(h)} = w'_{i1}, \quad \frac{\partial Y_i^{\mathrm{U}}(h)}{\partial O_{i1}^{\prime U}(h)} = w'_{i1},$$
$$\frac{\partial O_{i1}^{\prime L}(h)}{\partial w_{i1}} = O_{i1}^{\mathrm{L}}(h), \quad \frac{\partial O_{i1}^{\prime U}(h)}{\partial w_{i1}} = O_{i1}^{\mathrm{U}}(h).$$

Therefore, Eq. (29) and the above formulas imply the following relation:

$$\Delta w_{i1}(t) = \eta \sum_h h[(A_{i0}^{\mathrm{L}}(h) - Y_i^{\mathrm{L}}(h)) \cdot w'_{i1} \cdot O_{i1}^{\mathrm{L}}(h) + (A_{i0}^{\mathrm{U}}(h) - Y_i^{\mathrm{U}}(h)) \cdot w'_{i1} \cdot O_{i1}^{\mathrm{U}}(h)] + \alpha \cdot \Delta w_{i1}(t-1) \tag{30}$$

and if $w'_{i1} < 0$ and $w_{i1} \geqslant 0$ for $i = 1, 2$, then we obtain

$$\Delta w_{i1}(t) = \eta \sum_h h[(A^{\mathrm{L}}_{i0}(h) - Y^{\mathrm{L}}_i(h)) \cdot w'_{i1} \cdot O^{\mathrm{U}}_{i1}(h) + (A^{\mathrm{U}}_{i0}(h) - Y^{\mathrm{U}}_i(h)) \cdot w'_{i1} \cdot O^{\mathrm{L}}_{i1}(h)] + \alpha \cdot \Delta w_{i1}(t-1). \tag{31}$$

We can adjust other weights w_{ij} by

$$w_{ij} = w^j_{i1} \quad \text{for } i = 1, 2; \; j = 2, \ldots, n.$$

We can also obtain similarly alternative cases ($w'_{i1} \geqslant 0$, $w_{i1} < 0$ and $w'_{i1} < 0$, $w_{i1} < 0$ for $i = 1, 2$).

The fuzzy polynomials may have no real root for crisp w_{ij} and w'_{ij} for $i = 1, 2; \; j = 1 \ldots n$. In this case there is no hope to make the error measure close to zero.

Theorem 1. *The complexity of the above algorithm, in updating of* w'_{i1} *in* (22), *is* $2\{(2n+1)k+1\}$, *where k is the number of the h-level sets. And also, the complexity of computing* w'_{ij} *for* $i = 1, 2; \; j = 2, \ldots, n$ *is* $2(n-1)$.

Proof. Let $w'_{i1} \geqslant 0$. Computing the complexity of our algorithm, we consider only the multiplication operator. The number of the multiplications in computing of Y_i, where

$$Y_i = A_{i1}xy + A_{i2}x^2y^2 + \cdots + A_{in}x^ny^n,$$

is $2n$. Hence the number of the multiplications in

$$[(A^{\mathrm{L}}_{i0}(h) - Y^{\mathrm{L}}_i(h))O'^{L}_{i1}(h) + (A^{\mathrm{U}}_{i0}(h) - Y^{\mathrm{U}}_i(h))O'^{U}_{i1}(h)]$$

is $2(2n+1)$. Therefore the number of the multiplications in the computation of $w'_{i1}(t)$ is $2\{(2n+1)k+1\}$. Also with respect to

$$w'_{ij} = w'^{j}_{i1} \quad \text{for } i = 1, 2; \; j = 2, \ldots, n,$$

the number of the multiplications in the computation of all w'_{ij} for $i = 1, 2; j = 2, \ldots, n$ is $2(n-1)$. If $w'_{i1} < 0$, the proof is similar and it has the same number of the multiplications. $\square$

4. General fuzzy polynomials

The system (22) with fuzzy coefficients and deterministic unknowns is a special case of a more general system F, where F denotes a system of s polynomial equations in E:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= A_{10}, \\ &\vdots \\ f_l(x_1, x_2, \ldots, x_n) &= A_{l0}, \\ &\vdots \\ f_s(x_1, x_2, \ldots, x_n) &= A_{s0}, \end{aligned} \tag{32}$$

where

$$f_l(x_1, x_2, \ldots, x_n) = A_{l0} = \sum_{i=1}^n C_{li}x_i + \sum_{i=1}^n \sum_{j=1}^n C_{lij}x_ix_j + \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n C_{lijk}x_ix_jx_k + \cdots, \quad 1 \leqslant l \leqslant s, \tag{33}$$

and $x_1, x_2, \ldots, x_n \in \mathbb{R}$ (if exists) and all coefficients are fuzzy numbers.

This full form of mathematical description can be represented by a system of partial quadratic fuzzy polynomials consisting of only two variables in the form of

$$f_l(x_i, x_j) = A_{l0} = A_{l1}x_i + A_{l2}x_j + A_{l3}x_ix_j + A_{l4}x_i^2 + A_{l5}x_j^2. \tag{34}$$

Let us build a FNN_2 equivalent to the quadratic one. A FNN_2 [14] solution to Eq. (34) is visualized in Fig. 2.

In the above figure, the architecture neural network of the fuzzy polynomials, multi-input single-output (MISO) is shown. All signals are fuzzy numbers, arcs are real numbers and transfer functions are identity maps.

The unknown x_i, x_j in Eq. (34) are calculated using fuzzy neural network, so that the difference between the target output, A_{l0}, and the calculated, Y_l, for each input variable is minimized.

Because the general form equation (32) is complicated, we consider Eq. (34) and propose a learning algorithm from the cost function to adjust crisp weights. This algorithm is simply generalized to more variables and upper degrees as well.

4.1. Learning of fuzzy neural network for system of general quadratic fuzzy polynomials

Denote the h-level sets of the target outputs A_{l0} are denoted by

$$[A_{l0}]_h = [[A_{l0}]_h^{\mathrm{L}}, [A_{l0}]_h^{\mathrm{U}}], \quad h \in [0,1], \tag{35}$$

where $A_{l0}^{\mathrm{L}}(h)$ is the left-hand side and $A_{l0}^{\mathrm{U}}(h)$ is the right-hand side of the h-level sets of the desired output. A cost function to be minimized is defined for each h-level set as follows:

$$\begin{aligned} e_l(h) &= e_l^{\mathrm{L}}(h) + e_l^{\mathrm{U}}(h), \\ e_l^{\mathrm{L}}(h) &= \frac{1}{2}(A_{l0}^{\mathrm{L}}(h) - Y_l^{\mathrm{L}}(h))^2, \\ e_l^{\mathrm{U}}(h) &= \frac{1}{2}(A_{l0}^{\mathrm{U}}(h) - Y_l^{\mathrm{U}}(h))^2, \end{aligned} \tag{36}$$

where $e_l^{\mathrm{L}}(h)$ is the error between the left-hand side of the h-level set of the desired and the computed output ith equation, and $e_l^{\mathrm{U}}(h)$ is the error between the right-hand side of the h-level set of the desired and the computed output ith equation. Therefore, the error function for the training pattern is

$$e = \sum_l \sum_h h e_l(h). \tag{37}$$

Theoretically, this cost function satisfies the following equation if we use an infinite number of h-level sets in Eq. (37)

$$e \to 0 \text{ if and only if } Y_l \to A_{l0}.$$

The weights are updated by the following rules [14,21]:

$$\Delta w'_{l3}(t) = -\eta \sum_h h \frac{\partial e_l(h)}{\partial w'_{l3}} + \alpha \cdot \Delta w'_{l3}(t-1), \tag{38}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq. (38) can be written as follows:

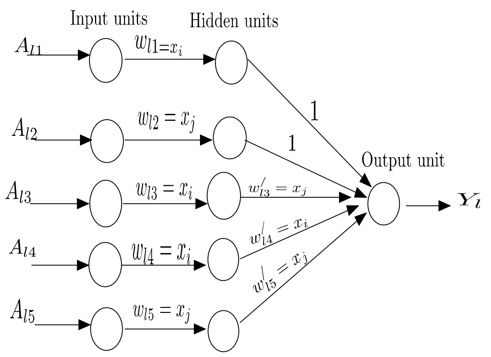

Fig. 2. Fuzzy neural network for solving a system of the quadratic general fuzzy polynomials.

$$\frac{\partial e_l(h)}{\partial w'_{l3}} = \frac{\partial e_l^{\mathrm{L}}(h)}{\partial w'_{l3}} + \frac{\partial e_l^{\mathrm{U}}(h)}{\partial w'_{l3}},$$
$$\frac{\partial e_i^{\mathrm{L}}(h)}{\partial w'_{l3}} = \frac{\partial e_l^{\mathrm{L}}(h)}{\partial Y_l^{\mathrm{L}}} \times \frac{\partial Y_l^{\mathrm{L}}(h)}{\partial w'_{l3}},$$
$$\frac{\partial e_l^{\mathrm{U}}(h)}{\partial w'_{l3}} = \frac{\partial e_l^{\mathrm{U}}(h)}{\partial Y_l^{\mathrm{U}}} \times \frac{\partial Y_l^{\mathrm{U}}(h)}{\partial w'_{l3}},$$
$$\frac{\partial e_l^{\mathrm{L}}(h)}{\partial Y_l^{\mathrm{L}}} = -(A_{l0}^{\mathrm{L}}(h) - Y_l^{\mathrm{L}}(h)), \quad \frac{\partial e_l^{\mathrm{U}}(h)}{\partial Y_l^{\mathrm{U}}} = -(A_{l0}^{\mathrm{U}}(h) - Y_l^{\mathrm{U}}(h)).$$

If $w'_{l3} \geqslant 0$,

$$\frac{\partial Y_l^{\mathrm{L}}(h)}{\partial w'_{l3}} = O_{l3}'^{L}(h), \quad \frac{\partial Y_l^{\mathrm{U}}(h)}{\partial w'_{l3}} = O_{l3}'^{U}(h);$$

otherwise

$$\frac{\partial Y_l^{\mathrm{L}}(h)}{\partial w'_{l3}} = O_{l3}'^{U}(h), \quad \frac{\partial Y_l^{\mathrm{U}}(h)}{\partial w'_{l3}} = O_{l3}'^{L}(h).$$

Therefore, if $w'_{l3} \geqslant 0$

$$\Delta w'_{l3}(t) = \eta \sum_h h[(A_{l0}^{\mathrm{L}}(h) - Y_l^{\mathrm{L}}(h))O_{l3}'^{L}(h) + (A_{l0}^{\mathrm{U}}(h) - Y_l^{\mathrm{U}}(h))O_{l3}'^{U}(h)] + \alpha \cdot \Delta w'_{l3}(t-1); \tag{39}$$

otherwise

$$\Delta w'_{l3}(t) = \eta \sum_h h[(A_{l0}^{\mathrm{L}}(h) - Y_l^{\mathrm{L}}(h))O_{l3}'^{U}(h) + (A_{l0}^{\mathrm{U}}(h) - Y_l^{\mathrm{U}}(h))O_{l3}'^{L}(h)] + \alpha \cdot \Delta w'_{l3}(t-1). \tag{40}$$

We can adjust weights w'_{lk} like w'_{l3}.

The weights of the arcs between input and hidden neurons are updated by the following rules [14,21]:

$$\Delta w_{l3}(t) = -\eta \sum_h h \frac{\partial e_l(h)}{\partial w_{l3}} + \alpha \cdot \Delta w_{l3}(t-1), \tag{41}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq. (41) can be written as follows:

$$\frac{\partial e_l(h)}{\partial w_{l3}} = \frac{\partial e_l^{\mathrm{L}}(h)}{\partial w_{l3}} + \frac{\partial e_l^{\mathrm{U}}(h)}{\partial w_{l3}}.$$

Now, suppose that $w'_{l3} \geqslant 0$ and $w_{l3} \geqslant 0$

$$\frac{\partial e_l^{\mathrm{L}}(h)}{\partial w_{l3}} = \frac{\partial e_l^{\mathrm{L}}(h)}{\partial Y_l^{\mathrm{L}}(h)} \times \frac{\partial Y_l^{\mathrm{L}}(h)}{\partial O_{l3}'^{L}(h)} \times \frac{\partial O_{l3}'^{L}(h)}{\partial w_{l3}},$$
$$\frac{\partial e_l^{\mathrm{U}}(h)}{\partial w_{l3}} = \frac{\partial e_l^{\mathrm{U}}(h)}{\partial Y_l^{\mathrm{U}}(h)} \times \frac{\partial Y_l^{\mathrm{U}}(h)}{\partial O_{l3}'^{U}(h)} \times \frac{\partial O_{l3}'^{U}(h)}{\partial w_{l3}},$$
$$\frac{\partial e_l^{\mathrm{L}}(h)}{\partial Y_l^{\mathrm{L}}(h)} = -(A_{l0}^{\mathrm{L}}(h) - Y_l^{\mathrm{L}}(h)), \quad \frac{\partial e_l^{\mathrm{U}}(h)}{\partial Y_l^{\mathrm{U}}(h)} = -(A_{l0}^{\mathrm{U}}(h) - Y_l^{\mathrm{U}}(h)),$$
$$\frac{\partial Y_l^{\mathrm{L}}(h)}{\partial O_{l3}'^{L}(h)} = w'_{l3}, \quad \frac{\partial Y_l^{\mathrm{U}}(h)}{\partial O_{l3}'^{U}(h)} = w'_{l3},$$
$$\frac{\partial O_{l3}'^{L}(h)}{\partial w_{l3}} = O_{l3}^{\mathrm{L}}(h), \quad \frac{\partial O_{l3}'^{U}(h)}{\partial w_{l3}} = O_{l3}^{\mathrm{U}}(h).$$

Therefore, we have

$$\Delta w_{l3}(t) = \eta \sum_h h[(A^{\mathrm{L}}_{l0}(h) - Y^{\mathrm{L}}_l(h)) \cdot w'_{l3} \cdot O^{\mathrm{L}}_{l3}(h) + (A^{\mathrm{U}}_{l0}(h) - Y^{\mathrm{U}}_l(h)) \cdot w'_{l3} \cdot O^{\mathrm{U}}_{l3}(h)] + \alpha \cdot \Delta w_{l3}(t-1), \tag{42}$$

and if $w'_{l3} < 0$ and $w_{l3} \geqslant 0$, then we obtain

$$\Delta w_{l3}(t) = \eta \sum_h h[(A^{\mathrm{L}}_{l0}(h) - Y^{\mathrm{L}}_l(h)) \cdot w'_{l3} \cdot O^{\mathrm{U}}_{l3}(h) + (A^{\mathrm{U}}_{l0}(h) - Y^{\mathrm{U}}_l(h)) \cdot w'_{l3} \cdot O^{\mathrm{L}}_{l3}(h)] + \alpha \cdot \Delta w_{l3}(t-1). \tag{43}$$

We can also obtain similar alternative cases ($w'_{l3} \geqslant 0$, $w_{l3} < 0$ and $w'_{l3} < 0$, $w_{i1} < 0$). The fuzzy polynomials may have no real root for crisp weights; in this case there is no hope to make the error measure close to zero.

Remark 1. In the general case, to find a solution of the system (32) in Section 4, similar to (22) and (34), we put the coefficients of the system of polynomials as input and the unknowns as weights; then using the learning algorithm proposed, we can minimize the distance between the target output and the computed output. It is obvious that weights are the root of the system of polynomials.

Remark 2. The problem (32) with uncertainty functions f_l for $l = 1, \ldots s$, where the coefficients are fuzzy numbers, is a special case of a more general problem

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= A_{10}, \\ &\vdots \\ f_l(x_1, x_2, \ldots, x_n) &= A_{l0}, \\ &\vdots \\ f_s(x_1, x_2, \ldots, x_n) &= A_{s0}, \end{aligned} \tag{44}$$

where f_l is a mapping from $E^n \to E$.

Remark 3. The problem (44) is complicated because we presume that the variables $x_1, x_2, \ldots, x_n$ are fuzzy numbers. The main topic for future research would be to develop a learning algorithm for fuzzy neural nets that will have more generality (i.e. LR fuzzy weights). We now need to assume that the signals/weights are fuzzy numbers (of any type) or general fuzzy sets. It appears that (fuzzy) genetic algorithms might be useful tools to handle the more general fuzzy sets [22].

5. Comparison with other methods

This study would not be completed without comparing it with the recent papers [1–6]. Some comparisons are as follows:

- In [6] a FNN_2 equivalent to the fuzzy polynomial equation was built and a learning algorithm of crisp weights of three-layer feedforward fuzzy neural network was introduced, so that input–output relations were defined by the extension principle to find the root of a fuzzy polynomial equation. The purpose of this paper was to extend the system of fuzzy polynomials, to a more general system F of s fuzzy polynomial equations such as

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= A_{10}, \\ &\vdots \\ f_l(x_1, x_2, \ldots, x_n) &= A_{l0}, \\ &\vdots \\ f_s(x_1, x_2, \ldots, x_n) &= A_{s0}, \end{aligned} \tag{45}$$

where $x_1, x_2, \ldots, x_n \in \mathbb{R}$ and all coefficients are fuzzy numbers. In this paper, an architecture of fuzzy neural networks was proposed to find a real root of a system of fuzzy polynomials by introducing a learning algorithm.

- In [2] a method for solving fuzzy linear systems was proposed, $Ax = b$ were A is a real matrix, the unknown vector x is a vector consisting of fuzzy numbers and the constant y is a vector whose components are fuzzy numbers. $Ax = b$ is a special case of the (45); where $f_i, i = 1, \ldots, n$ are linear functions, however, we have proposed a new method using fuzzy neural networks for solving systems of fuzzy polynomials.
- In [3] the problem of best approximation is applied to the fuzzy functions, to obtain a fuzzy polynomial using optimization; but we have obtained the solution of a system of fuzzy polynomials such as (45) by fuzzy neural network. Moreover, the application of fuzzy polynomials was proposed.
- In [1,5] researchers used the Newton's method for solving nonlinear equations and systems of nonlinear equations and from the beginning they assumed that the solutions are negative or positive, otherwise we could not use it. See Example 5 for more details.

6. Applications and numerical examples

The general formula of quadratic fuzzy surfaces is

$$f_l(x_i, x_j, x_k) = A_{l0} = A_{l1}x_i + A_{l2}x_j + A_{l3}x_k + A_{l4}x_ix_j + A_{l5}x_ix_k + A_{l6}x_jx_k + A_{l7}x_i^2 + A_{l8}x_j^2 + A_{l9}x_k^2,$$

which is the same as a three variables quadratic fuzzy polynomial. One of the applications of fuzzy polynomial systems is to determine the cross location of three quadratic surfaces.

In the following, we study the some examples of fuzzy polynomial systems.

Example 1. Consider the following system of fuzzy polynomials:

$$\begin{cases} (2,1,1)xy + (3,2,1)x^2y^2 = (1,3,2), \\ (1,1,1)xy + (1,1,1)x^2y^2 = (0,2,2), \end{cases}$$

with the exact solution $x = 1$ and $y = -1$. The training starts with $w_{11}(0) = 0.75, w'_{11}(0) = -0.75$ and it is completed in step 2 with the solution $w_{21}(2) = 1.00973$ and $w'_{21}(2) = -1.00876$, where $\eta = 0.1$, $\alpha = 0.3$ and $e \leqslant 0.01$.

Example 2. Consider the following system of fuzzy polynomials:

$$\begin{cases} (2,1,1)xy + (3,1,1)x^2y^2 + (2,1,1)x^3y^3 = (7,3,3), \\ (5,1,1)xy + (2,3,1)x^2y^2 + (2,2,1)x^3y^3 = (9,6,3), \end{cases}$$

with the exact solution $x = y = 1$. The training starts with $w_{11}(0) = 0.75, w'_{11}(0) = 0.75$ and it is completed in step 2 with the solution $w_{21}(2) = 1.00943$ and $w'_{21}(2) = 1.00956$, where $\eta = 0.1$, $\alpha = 0.3$ and $e \leqslant 0.01$.

Example 3. Consider the following system of fuzzy polynomials:

$$\begin{cases} (1,1,1)xy + (-2,1,1)x^2y^2 = (14,2,5), \\ (5,1,1)xy + (2,3,1)x^2y^2 = (-2,1,1), \end{cases}$$

which has no real root. The training start with $w_{11}(0) = w'_{11}(0) = 7$, $\eta = 0.1$, $\alpha = 0.3$, and $e \leqslant 0.01$. In this case there is no hope to make the measure of error close to zero. See the Table 1 for more details.

Example 4. Now, we consider an application of approximate solution of fuzzy nonlinear systems by fuzzy neural networks in economics. The market price of a good and the quantity produced are determined by the balance between supply and demand. Suppose that demand and supply are nonlinear functions of the price:

$$\begin{cases} q_d + a = b \cdot p^2, \\ q_s + c = d \cdot p^2, \end{cases}$$

Table 1
The approximation of real root

i	Example 1 $w_{11}(i),\ w'_{11}(i)$	Example 2 $w_{11}(i),\ w'_{11}(i)$	Example 3 $w_{11}(i),\ w'_{11}(i)$
0	0.75, −0.75	0.75, 0.75	7,7
1	0.87672, −0.89683	0.89764, 0.89953	−0.26203, −0.43054
2	1.00873, −1.00976	1.00943, 1.00956	6.9475, 5.7657
3			−0.1435, −0.2234
⋮			⋮

where q_s is the quantity supplied, which is required to be equal to q_d, the quantity requested, p is the price and a, b, c and d are coefficients to be estimated, where the coefficients a, b, c and d are represented by fuzzy triangular numbers and q_s, q_d and p are deterministic. By imposing the equality between quantity supplied and requested, the following fuzzy nonlinear system should be solved:

$$\begin{cases} x + (-1,1,1) = (-2,1,1)y^2, \\ x + (3,1,1) = (2,1,1)y^2. \end{cases}$$

The exact solution is $(x, y) = (-1, 1)$. The training starts with $w_{11}(0) = -0.75,\ w'_{11}(0) = 0.75$ and it is completed with $e \leqslant 0.01$ where $\eta = 0.1$ and $\alpha = 0.3$.

Example 5. Consider the following system of fuzzy polynomials:

$$\begin{cases} (2,1,1)x^2 + (3,2,2)y^2 = (5,3,3), \\ (2,3,1)x + (2,1,2)y = (0,5,2) \end{cases}$$

with the exact solution $x = 1,\ y = -1$.

Using the Newton's method in [5], suppose that x and y are positive; then the parametric form of this system is as follows:

$$\begin{cases} (r+1)x^2 + (2r+1)y^2 = 3r + 2, \\ (-r+3)x^2 + (-2r+5)y^2 = -3r + 8, \\ (3r-1)x + (r+1)y = 5r - 5, \\ (-r+3)x + (-2r+4)y = -2r + 2. \end{cases}$$

For $r = 0$, the above system has no real solution and we can not use the Newton's method of [5] to obtain the approximate solution.

Using fuzzy neural networks, the training starts with $w_{11}(0) = 0.1, w'_{11}(0) = 0.75$ and it is completed with $e \leqslant 0.01$, where $\eta = 0.1$ and $\alpha = 0.3$.

Example 6. Consider the following system of fuzzy polynomials:

$$\begin{cases} (0,0.5,0.5)x^5 + (0.5,0.5,0.25)y = (0.25,0.75,0.625), \\ (0.5,1,0.5)x^4 + (1.5,0.5,0.5)xy = (1.25,1.25,0.75), \end{cases}$$

with the exact solution $(x, y) = (1, 0.5)$. Using fuzzy neural networks, the training starts with $w_{11}(0) = 0.9,\ w'_{11}(0) = 0.75$ and it is completed with $e \leqslant 0.01$, where $\eta = 0.1$ and $\alpha = 0.3$.

7. Conclusions

In this paper, we built a FNN_2 equivalent to the system of fuzzy polynomials, and we introduced a learning algorithm of crisp weights of three-layer feedforward fuzzy neural networks on the basis of which the input–output relations were defined by the extension principle, to find the real roots of a system of fuzzy polynomials. We extended fuzzy polynomials which have already been studied by Abbasbandy and Otadi [6], to a system of fuzzy polynomials. The unknown calculated by using fuzzy neural network through minimizing the differences between the target output, A_{l0}, and the calculated, Y_l. Also, the complexity of learning algorithm is $O(n)$ and hence this algorithm is effective with respect to the order of the polynomial.

The topic for further research is to develop learning algorithms for fuzzy neural networks that have more generality (i.e. LR fuzzy weights). In this paper, we assumed the signals are fuzzy numbers and weights are crisp. We now need to assume the signals/weights are fuzzy numbers (of any type) or general fuzzy sets. It appears that (fuzzy) genetic algorithms might be a suitable tool to handle more general fuzzy sets.

Acknowledgements

We would like to offer particular thanks to Dr. M.A. Rezvani for the editing of this paper. We would also like to thank the referees for valuable suggestions.

References

[1] S. Abbasbandy, B. Asady, Newton's method for solving fuzzy nonlinear equations, Appl. Math. Comput. 159 (2004) 349–356.
[2] S. Abbasbandy, M. Alavi, A method for solving fuzzy linear systems, Iran. J. Fuzzy Syst. 2 (2005) 37–43.
[3] S. Abbasbandy, M. Amirfakhrian, Numerical approximation of fuzzy functions by fuzzy polynomials, Appl. Math. Comput. 174 (2006) 1001–1006.
[4] S. Abbasbandy, B. Asady, Ranking of fuzzy numbers by sign distance, Inform. Sci. 176 (2006) 2405–2416.
[5] S. Abbasbandy, R. Ezzati, Newton's method for solving a system of fuzzy nonlinear equations, Appl. Math. Comput. 175 (2006) 1189–1199.
[6] S. Abbasbandy, M. Otadi, Numerical solution of fuzzy polynomials by fuzzy neural network, Appl. Math. Comput. 181 (2006) 1084–1089.
[7] G. Alefeld, J. Herzberger, Introduction to Interval Computations, Academic Press, New York, 1983.
[8] B. Asady, S. Abbasbandy, M. Alavi, Fuzzy general linear systems, Appl. Math. Comput. 169 (2005) 34–40.
[9] J.J. Buckley, E. Eslami, Neural net solutions to fuzzy problems: the quadratic equation, Fuzzy Sets Syst. 86 (1997) 289–298.
[10] J.J. Buckley, Y. QU, Solving linear and quadratic fuzzy equations, Fuzzy Sets Syst. 35 (1990) 43–59.
[11] P.A. Castillo, J.J. Merelo, M.G. Arenas, G. Romero, Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters, Inform. Sci. 177 (2007) 2884–2905.
[12] Y. Hayashi, J.J. Buckley, E. Czogala, Fuzzy neural network with fuzzy signals and weights, Int. J. Intell. Syst. 8 (1993) 527–537.
[13] Y.C. Hu, Fuzzy integral-based perceptron for two-class pattern classification problems, Inform. Sci. 177 (2007) 1673–1686.
[14] H. Ishibuchi, K. Kwon, H. Tanaka, A learning algorithm of fuzzy neural networks with triangular fuzzy weights, Fuzzy Sets Syst. 71 (1995) 277–293.
[15] H. Ishibuchi, H. Okada, H. Tanaka, Fuzzy neural networks with fuzzy weights and fuzzy biases, in: Proceedings of the ICNN 93, San Francisco, 1993, pp. 1650–1655.
[16] P. Liu, Analysis of approximation of continuous fuzzy functions by multivariate fuzzy polynomials, Fuzzy Sets Syst. 127 (2002) 299–313.
[17] M. Ma, M. Friedman, A. Kandel, A new fuzzy arithmetic, Fuzzy Sets Syst. 108 (1999) 83–90.
[18] P. Melin, O. Castillo, An intelligent hybrid approach for industrial quality control combining neural networks, fuzzy logic and fractal theory, Inform. Sci. 177 (2007) 1543–1557.
[19] S.K. Oh, W. Pedrycz, The design of self-organizing polynomial neural networks, Inform. Sci. 141 (2002) 237–258.
[20] S.K. Oh, W. Pedrycz, S.B. Roh, Genetically optimized fuzzy polynomial neural networks with fuzzy set-based polynomial neurons, Inform. Sci. 176 (2006) 3490–3519.
[21] D.E. Rumelhart, J.L. McClellandPDP Research Group, Parallel Distributed Processing, vol. 1, MIT Press, Cambridge, MA, 1986.
[22] L.M. San José-Revuelta, A new adaptive genetic algorithm for fixed channel assignment, Inform. Sci. 177 (2007) 2655–2678.
[23] C.C. Wang, C.F. Tsai, Fuzzy processing using polynomial bidirectional hetero-associative network, Inform. Sci. 125 (2000) 167–179.
[24] S.J. Yoo, J.B. Park, Y.H. Choi, Indirect adaptive control of nonlinear dynamic systems using self recurrent wavelet neural networks via adaptive learning rates, Inform. Sci. 177 (2007) 3074–3098.
[25] L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning: Parts 1–3, Inform. Sci. 8 (1975), 199–249, 301–357;
L.A. Zadeh, The concept of a linguistic variable and its application to approximate reasoning: Parts 1–3, Inform. Sci. 9 (1975) 43–80.
[26] L.A. Zadeh, Toward a generalized theory of uncertainty (GTU) an outline, Inform. Sci. 172 (2005) 1–40.