Mathware & Soft Computing 13 (2006) 127-134

Fuzzy Neural Network Approach to Fuzzy Polynomials

S. Abbasbandy[a,b] M. Otadi[a]
[a] Department of Mathematics, Science and Research Branch, Islamic Azad University, Tehran, 14778, Iran
[b]Department of Mathematics, Faculty of Science, Imam Khomeini International University, Ghazvin, 34194, Iran

Abstract

In this paper, an architecture of fuzzy neural networks is proposed to find a real root of a dual fuzzy polynomial (if exists) by introducing a learning algorithm. We proposed a learning algorithm from the cost function for adjusting of crisp weights. According to fuzzy arithmetic, dual fuzzy polynomials can not be replaced by a fuzzy polynomials, directly. Finally, we illustrate our approach by numerical examples.

Keywords: Fuzzy neural networks; Fuzzy polynomials

1 Introduction

Polynomials play a major role in various areas such as mathematics, engineering and social sciences [2]. The numerical solution of a fuzzy polynomial by fuzzy neural network is investigated by Abbasbandy and Otadi [5], by finding the solution of polynomial $A_1x+A_2x^2+\ldots+A_nx^n = A_0$ for $x \in \mathbb{R}$ (if exists) where $A_0, A_1, \ldots, A_n$ are fuzzy numbers. In this paper we are interested in finding solution to a dual fuzzy polynomials like $A_1^{'}x+A_2^{'}x^2+\ldots+A_n^{'}x^n = A_1^{''}x+A_2^{''}x^2+\ldots+A_n^{''}x^n+A_0$ for $x \in \mathbb{R}$ (if exists) where $A_0, A_1^{'}, \ldots, A_n^{''}$ are fuzzy numbers by a learning algorithm of fuzzy neural networks.

Ishibuchi *et al.* [11] proposed a learning algorithm of fuzzy neural networks with triangular fuzzy weights and Hayashi *et al.* [10] also fuzzified the delta rule. Linear and nonlinear fuzzy equations are solved by [1, 3, 4, 7, 9]. Buckley and Eslami [8] is concerned with neural net solutions to fuzzy problems.
In this paper, first we 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 [15]. Output from the fuzzy neural network, which is also fuzzy number, is numerically calculated by interval arithmetic [6] for crisp 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 for find the real root (if exists) of the polynomials. The proposed algorithm illustrated by solving some examples in last section.

2 Preliminaries

We represent an arbitrary fuzzy number by an ordered pair of functions $(\underline{u}(r), \overline{u}(r))$, for $0 \leq r \leq 1$, which satisfies the following requirements [13]:

1. $\underline{u}(r)$ is a bounded left continuous non decreasing function over $[0, 1]$.
2. $\overline{u}(r)$ is a bounded left continuous non increasing function over $[0, 1]$.
3. $\underline{u}(r) \leq \overline{u}(r)$, for $0 \leq r \leq 1$.

A crisp number α is simply represented by $\underline{u}(r) = \overline{u}(r) = \alpha$, for $0 \leq r \leq 1$. The set of all the fuzzy numbers is denoted by E^1. A popular fuzzy number is the triangular fuzzy number $u = (m - \alpha, m, m + \beta) = (u_1, u_2, u_3)$ with membership function

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

for $\alpha, \beta > 0$ where $u_1 = m - \alpha$, $u_2 = m$ and $u_3 = m + \beta$. Its parametric form is

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

2.1 Operations of fuzzy numbers

We briefly mention fuzzy number operations defined by the extension principle [15]. Since input vector of feedforward neural network is fuzzified in this paper, the following addition, multiplication and nonlinear mapping of fuzzy numbers are necessary for defining our fuzzy neural network:

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

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

where A, B, Net are fuzzy numbers, $\mu_*(.)$ denotes the membership function of each fuzzy number, $\wedge$ is the minimum operator, and $f(x) = x$ is the activation function of output unit of our fuzzy neural network.
The above operations of fuzzy numbers are numerically performed on level sets (i.e., α-cuts). The h-level set of a fuzzy number X is defined as

$$[X]_h = \{x | \mu_X(x) \geq h, x \in \mathbb{R}\} \quad for \quad 0 < h \leq 1, \tag{3}$$

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

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

where $[X]_h^L$ and $[X]_h^U$ are the lower limit and the upper limit of the h-level set $[X]_h$, respectively.
From interval arithmetic [6], the above operations of fuzzy numbers are written for h-level sets as follows:

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

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

$$\begin{aligned} w.[A]_h &= [w.[A]_h^L, w.[A]_h^U], \quad if \quad w \geq 0, \\ w.[A]_h &= w.[A]_h^U, w.[A]_h^L], \quad if \quad w \leq 0. \end{aligned} \tag{7}$$

Theorem 1. Let a and c are fuzzy numbers. The equation $a+x = c$ has a solution x if and only if $c_1 - a_1 < c_2 - a_2 < c_3 - a_3$.
Proof. Taking α-cuts we obtain $[a]_\alpha^U + [x]_\alpha^U = [c]_\alpha^U$ and $[a]_\alpha^L + [x]_\alpha^L = [c]_\alpha^L$. Then

$$x_1 < x_2 < x_3,$$

and $[\dot{x}]_\alpha^L > 0$, $[\dot{x}]_\alpha^U < 0$ if and only if $c_1 - a_1 < c_2 - a_2 < c_3 - a_3$, where $[\dot{x}]_\alpha^L$, $[\dot{x}]_\alpha^U$ are the derivative of $[x]_\alpha^L$ and $[x]_\alpha^U$, respectively, with respect to α.

□

2.2 Input-output relation of each unit

Let us fuzzify a two layer feedforward neural network with n input units and one output unit. Input vector, target vector are fuzzified and weights are crisp. In order to derive a crisp learning rule, we restrict fuzzy inputs and fuzzy target within triangular fuzzy numbers. The input-output relation of each unit of the fuzzified neural network can be written as follows:
Input units:

$$O_i = A_i, \qquad i = 1, 2, \ldots, n. \tag{8}$$

Output unit:

$$Y = f(Net), \tag{9}$$

$$Net = \sum_{j=1}^{n} w_j.O_j, \tag{10}$$

where A_i is a fuzzy input and w_j is crisp weight.

The input-output relation in Eqs.(8)-(10) are defined by the extension principle [15] as in Hayashi et al.[10] and Ishibuchi *et al.*[12].

2.3 Calculation of fuzzy output

The fuzzy output from each unit in Eqs.(8)-(10) 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:
Input units:

$$[O_i]_h = [A_i]_h, \qquad i = 1, 2, \ldots, n. \tag{11}$$

Output unit:

$$[Y]_h = f([Net]_h), \tag{12}$$

$$[Net]_h = \sum_{j=1}^{n} w_j.[O_j]_h. \tag{13}$$

From Eqs.(11)-(13), we can see that the h-level sets of the fuzzy output Y is calculated from those of the fuzzy inputs and crisp weights. From Eqs.(5)-(7), the above relations are written as follows:
Input units:

$$[O_i]_h = [[O_i]_h^L, [O_i]_h^U] = [[A_i]_h^L, [A_i]_h^U], \qquad i = 1, 2, \ldots, n. \tag{14}$$

Output unit:

$$[Y]_h = [[Y]_h^L, [Y]_h^U] = [f([Net]_h^L), f([Net]_h^U)], \tag{15}$$

$$[Net]_h = [[Net]_h^L, [Net]_h^U] =$$

$$[\sum_{j\in m} w_j.[O_j]_h^L + \sum_{j\in c} w_j.[O_j]_h^U, \sum_{j\in m} w_j.[O_j]_h^U + \sum_{j\in c} w_j.[O_j]_h^L], \tag{16}$$

where $m = \{j \mid w_j \geq 0\}$, $c = \{j \mid w_j < 0\}$ and $m \cup c = \{1, \ldots, n\}$.

3 Fuzzy polynomials

Usually, there is no inverse element for an arbitrary fuzzy number $u \in E^1$, i.e., there exists no element $v \in E^1$ such that

$$u + v = 0.$$

Actually, for all non-crisp fuzzy number $u \in E^1$ we have

$$u + (-u) \neq 0.$$

Therefore, the dual fuzzy polynomials

$$A_1^{'}x + A_2^{'}x^2 + \ldots + A_n^{'}x^n = A_1^{''}x + A_2^{''}x^2 + \ldots + A_n^{''}x^n + A_0, \tag{17}$$

cannot be equivalently replaced by the fuzzy polynomials $(A_1^{'} - A_1^{''})x + (A_2^{'} - A_2^{''})x^2 + \ldots + (A_n^{'} - A_n^{''})x^n = A_0$ which had been investigated. Therefore, we find solution

Eq.(17) by the neural network, with suppose $A_{i1}^{'} - A_{i1}^{''} < A_{i2}^{'} - A_{i2}^{''} < A_{i3}^{'} - A_{i3}^{''}$ for $i = 1, \ldots, n$. Therefore, we have of the Eq.(17)

$$A_1 x + A_2 x^2 + \ldots + A_n x^n = A_0, \tag{18}$$

for $x \in \mathbb{R}$, when $A_i \in E$, for $i = 1, \ldots, n$ that $A_i = (A_{i1}, A_{i2}, A_{i3}) = (A_{i1}^{'} - A_{i1}^{''}, A_{i2}^{'} - A_{i2}^{''}, A_{i3}^{'} - A_{i3}^{''})$ for $i = 1, \ldots, n$.

A FNN_2 (fuzzy neural network with fuzzy set input signals and real number weights) [11] solution to Eq.(18) is given in Figure 1. The input neurons make no change in their inputs, so the input to the output neuron is

$$A_1 x + A_2 x^2 + \ldots + A_n x^n$$

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

$$Y = A_1 x + A_2 x^2 + \ldots + A_n x^n.$$

How is the FNN_2 going to solve the fuzzy polynomials? The training data is $(A_1, \ldots, A_n)$ for input and target (desired) output is A_0. We proposed a learning algorithm from the cost function for adjusting weights.

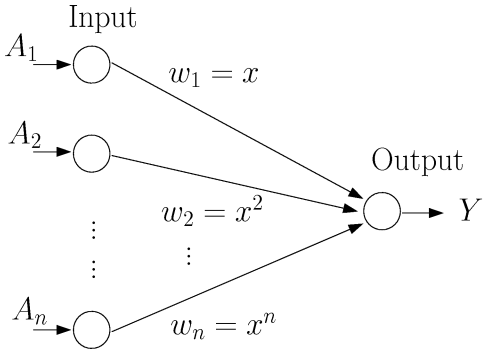

Figure 1: Fuzzy neural network to solve fuzzy polynomial

3.1 Learning of fuzzy neural network

Let the h-level sets of the target output A_0 are denoted by

$$[A_0]_h = [[A_0]_h^L, [A_0]_h^U], \qquad h \in [0, 1], \tag{19}$$

where $A_0^L(h)$ denotes the left-hand side and $A_0^U(h)$ 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:

$$\begin{array}{l} e(h) = e^L(h) + e^U(h), \\ e^L(h) = \frac{1}{2}(A_0^L(h) - Y^L(h))^2, \\ e^U(h) = \frac{1}{2}(A_0^U(h) - Y^U(h))^2, \end{array} \tag{20}$$

hence $e^L(h)$ denotes the error between the left-hand sides of the h-level sets of the desired and the computed output, and $e^U(h)$ denotes the error between the right-hand sides of the h-level sets of the desired and the computed output. Then the error function for the training pattern is

$$e = \sum_h he(h). \tag{21}$$

Theoretically this cost function satisfies the following equation if we use infinite number of h-level sets in Eq.(21)

$$e \longrightarrow 0 \quad if \ \ and \ \ only \ \ if \ \ Y \longrightarrow A_0$$

The weights are updated by the following rules [11, 14]

$$\triangle w_1(t) = -\eta \sum_h h \frac{\partial e(h)}{\partial w_1} + \alpha. \triangle w_1(t-1), \tag{22}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq.(22) can be written as follows:

$$\frac{\partial e(h)}{\partial w_1} = \frac{\partial e^L(h)}{\partial w_1} + \frac{\partial e^U(h)}{\partial w_1},$$
$$\frac{\partial e^L(h)}{\partial w_1} = \frac{\partial e^L(h)}{\partial Y^L} \times \frac{\partial Y^L(h)}{\partial w_1},$$
$$\frac{\partial e^U(h)}{\partial w_1} = \frac{\partial e^U(h)}{\partial Y^U} \times \frac{\partial Y^U(h)}{\partial w_1},$$
$$\frac{\partial e^L(h)}{\partial Y^L} = -(A_0^L(h) - Y^L(h)), \qquad \frac{\partial e^U(h)}{\partial Y^U} = -(A_0^U(h) - Y^U(h)).$$

If $w_1 \geq 0$

$$\frac{\partial Y^L(h)}{\partial w_1} = A_1^L(h), \qquad \frac{\partial Y^U(h)}{\partial w_1} = A_1^U(h),$$

otherwise

$$\frac{\partial Y^L(h)}{\partial w_1} = A_1^U(h), \qquad \frac{\partial Y^U(h)}{\partial w_1} = A_1^L(h).$$

Therefore, if $w_1 \geq 0$

$$\begin{aligned} \triangle w_1(t) = \eta \textstyle\sum_h h[(A_0^L(h) - Y^L(h))A_1^L(h) + \\ (A_0^U(h) - Y^U(h))A_1^U(h)] + \alpha. \triangle w_1(t-1), \end{aligned} \tag{23}$$

otherwise

$$\begin{aligned} \triangle w_1(t) = \eta \textstyle\sum_h h[(A_0^L(h) - Y^L(h))A_1^U(h) + \\ (A_0^U(h) - Y^U(h))A_1^L(h)] + \alpha. \triangle w_1(t-1). \end{aligned} \tag{24}$$

We can adjust other weights by

$$w_i = w_1^i \quad for \ \ i = 2, \ldots, n.$$

The fuzzy polynomials may have no real root for crisp $w_i, 1 \leq i \leq n$. In this case there is no hope in making the error measure close to zero.

4 Numerical examples

Example 4.1. Consider the following dual fuzzy polynomial

$$(-2,0,3)x + (-2,0,3)x^2 = (-1,0,2)x + (-1,0,2)x^2 + (-2,0,2),$$

with the exact solution $x = 1$. The training starts by $w_1(0) = 0.25$ and training is completed in step 2 with the solution $w_1(2) = 1.00984$, where $\eta = 0.1$, $\alpha = 0.3$ and $e <= 0.01$.

Example 4.2. Consider the following dual fuzzy polynomial

$$(-2,0,3)x + (-3,0,4)x^2 + (0,2,4)x^3 = (-1,0,2)x + (-1,0,2)x^2 + (0,1,2)x^3 + (-5,-1,3),$$

with the exact solution is $x = -1$. The training starts by $w_1(0) = -1.5$ and training is completed in step 3 with the solution $w_1(3) = -1.00780$, where $\eta = 0.1$, $\alpha = 0.3$ and $e <= 0.01$.

Example 4.3. Consider the following dual fuzzy polynomial

$$(2,4,6)x + (0,2,4)x^2 + (3,6,8)x^3 + (-3,0,2)x^4 = (1,2,3)x + (0,1,2)x^2 + (1,2,3)x^3 + (-2,0,1)x^4 + (2,7,11),$$

with the exact solution is $x = 1$. The training starts by $w_1(0) = 0.5$ and training is completed in step 3 with the solution $w_1(3) = 1.00801$, where $\eta = 0.1$, $\alpha = 0.3$ and $e <= 0.01$.

Example 4.4. Consider the following dual fuzzy polynomial

$$(-2,0,3)x + (-2,0,2)x^2 = (-1,0,2)x + (-1,0,1)x^2 + (12,14,19),$$

which has no any real root. The training start by $w_1(0) = 7$, $\eta = 0.1$, $\alpha = 0.3$, and $e <= 0.01$. In this case there is no hope in making the measure of error close to zero, see Table 1 for more details.

i	Example 1	Example 2	Example 3	Example 4
0	0.25	-1.5	0.5	7.0
1	0.87982	-1.06313	0.8317	-0.26250
2	1.00984	-1.04608	0.98421	6.9475
3		-1.00780	1.00801	-0.1435
4				6.97130
5				-0.1973
6				6.96053
⋮				⋮

Table 1. The obtained values of $w_1(i)$, the approximation of real root

5 Conclusions

In this paper, we introduced a learning algorithm of crisp weights of two-layer feedforward fuzzy neural network, that input-output relations were defined by the extension principle, for finding the real root of a dual fuzzy polynomials.

References

[1] S. Abbasbandy and M. Alavi. A method for solving fuzzy linear systems, Iranian J. Fuzzy Systems, 2 (2005) 37-43.

[2] S. Abbasbandy and M. Amirfakhrian, Numerical approximation of fuzzy functions by fuzzy polynomials, Appl. Math. Comput. 174 (2006) 1001-1006.

[3] S. Abbasbandy and B. Asady, Newton's method for solving fuzzy nonlinear equations, Appl. Math. Comput. 159 (2004) 349-356.

[4] S. Abbasbandy and R. Ezzati, Newton's method for solving a system of fuzzy nonlinear equations, Appl. Math. Comput. 175 (2006) 1189-1199.

[5] S. Abbasbandy and M. Otadi, Numerical solution of fuzzy polynomials by fuzzy neural network, Appl. Math. Comput. In Press.

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

[7] B. Asady, S. Abbasbandy and M. Alavi, Fuzzy general linear systems, Appl. Math. Comput. 169 (2005) 34-40.

[8] J.J. Buckley and E. Eslami, Neural net solutions to fuzzy problems: The quadratic equation, Fuzzy Sets and Sestems 86 (1997) 289-298.

[9] J.J. Buckley and Y. QU, Solving linear and quadratic fuzzy equations, Fuzzy Sets and Systems 35 (1990) 43-59.

[10] Y. Hayashi, J.J. Buckley and E. Czogala, Fuzzy neural network with fuzzy signals and weights, Internat. J. Intelligent Systems 8 (1993) 527-537.

[11] 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.

[12] H. Ishibuchi, H. Okada and H. Tanaka, Fuzzy neural networks with fuzzy weights and fuzzy biases, Proc. ICNN 93(San Francisco) (1993) 1650-1655.

[13] M. Ma, M. Friedman and A. Kandel, A new fuzzy arithmetic, Fuzzy Sets and Systems, 108 (1999) 83-90.

[14] D.E. Rumelhart, J.L. McClelland and the PDP Research Group, Parallel Distributed Processing, Vol. 1, MIT Press, Cambridge, MA, 1986.

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