ISLAMIC AZAD UNIVERSITY
SCIENCE AND RESEARCH BRANCH

NUMERICAL SOLUTION OF FUZZY POLYNOMIAL BY FUZZY NEURAL NETWORK

Supervisor

Dr. Saeid Abbasbandy

Advisers

Dr. Tofigh Allahviranloo

By

Mahmood Otadi

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY(Ph.D.)
IN APPLIED MATHEMATICS (NUMERICAL ANALYSIS)
JANUARY 2008

ISLAMIC AZAD UNIVERSITY
SCIENCE AND RESEARCH BRANCH
DEPARTMENT OF MATHEMATICS

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Numerical Solution Of Fuzzy Polynomial By Fuzzy Neural Network**" by **Mahmood Otadi** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: January 2008

Research Supervisor: ______________________
Dr. Saeid Abbasbandy

External Examiner: ______________________
Dr. Esmail Babolian

Examing Committee: ______________________

____
Dr. Mohamad Ali Fariborzi Aragi

ISLAMIC AZAD UNIVERSITY

SCIENCE AND RESEARCH BRANCH [1]

Date: **January 2008**

Author: **Mahmood Otadi**

Title: **Numerical Solution Of Fuzzy Polynomial By Fuzzy Neural Network**

Department: **Mathematics**

Degree: **Ph.D.** Convocation: ******** Year: **January 2008**

Signature of Author

[1] Poonak, Tehran

To My dears:

My Wife "Maryam"

My Father and My Mother

Contents

Acknowledgments

I would like to thank Dr. S. Abbasbandy, my supervisor, for his many suggestions and constant support during this research.

I am also thankful to Dr. T. Allahviranloo, my advisor, for his guidance through the early years, during the Ph.D courses and professor G. R. Jahanshahloo, head of Mathematics Department.

I am also thankful to Professor E. Babolian, Dr. F. Hosseinzadeh, Dr. M. A. Fariborzi Araghi, my examiners.

I wish to thank the following:

M. Mosleh (my wife), my family and all my friends.

Finally, I had the pleasure of Islamic Azad University, Firoozkooh branch, for their scholarship made research like this possible.

Abstract

One of the major applications by using fuzzy arithmetics is treating polynomials, which their coefficients are fuzzy numbers. In this thesis, some new approaches for solving fuzzy polynomials,

$$A_1x + A_2x^2 + \ldots + A_nx^n = A_0,$$

dual fuzzy polynomials

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

and systems of fuzzy polynomials

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

where

$$\begin{aligned} f_l(x_1, x_2, \ldots, x_n) = A_{l0} = \textstyle\sum_{i=1}^{n} C_{li}x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} C_{lij}x_ix_j + \\ \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} C_{lijk}x_ix_jx_k + \ldots, \qquad 1 \le l \le s, \end{aligned}$$

and $x_1, x_2, \ldots, x_n \in \mathbb{R}$ (if exists) and all coefficients are fuzzy numbers, based on fuzzy neural network (FNN) are presented. In this work, an architecture of fuzzy neural

networks is also proposed to find a real root of a fuzzy polynomial, dual fuzzy polynomial and a system of fuzzy polynomials (if exists) by introducing a learning algorithm.

Originality

The following chapters are proposed in this work:

Chapter 3: We define fuzzy polynomials and dual fuzzy polynomials and use fuzzy neural network for finding roots of fuzzy polynomials.

Chapter 4: In this chapter, a new approach for solving the system of fuzzy polynomials based on fuzzy neural networks is presented.

Articles

1. Numerical solution of a system of fuzzy polynomials by fuzzy neural network, *Journal of Information Sciences,* In press (ISI).

2. Numerical solution of fuzzy polynomials by fuzzy neural network, *Journal of Applied Mathematics and Computation* 181 (2006) 1084-1089 (ISI).

3. Fuzzy neural network approach to fuzzy polynomials, *Journal of Mathware soft computing*, xiii, No.2 (2006) 127-134 (AMS).

4. On the fuzzy neural networks for approximate of fuzzy integration, *Nonlinear studies*, In press (AMS).

Chapter 1

Fuzzy Mathematics

1.1 Introduction

Fuzziness is not a priori an obvious concept and demands some explanations. "Fuzziness" is what Black calls "vagueness" when he distinguishes it from "generality" and from "ambiguity". Generalizing refers to the application of a symbol to a multiplicity of objects in the field of reference, ambiguity to the association of a finite number of alternative meanings having the same phonetic form. But, the fuzziness of a symbol lies in the lack of well-defined boundaries of the set of objects to which this symbol applies.

More specifically, let X be a field of reference, also called a universe of discourse or universe for short, covering a definite range of objects. Consider a subset $\tilde{A}$ where transition between membership and nonmembership is gradual rather than abrupt. This "fuzzy subset" obviously has no well-defined boundaries. Fuzzy classes of objects are often encountered in real life. For instance, $\tilde{A}$ may be the set of tall men in a community X. Usually, there are members of X who are definitely tall, others who are definitely not tall, but there exist also borderline cases. Traditionally, the grade of membership 1 is assigned to the objects that completely belong to $\tilde{A}$-here the men who are definitely tall, conversely the objects that do not belong to $\tilde{A}$ at all are assigned a membership value 0. Quite naturally, the grades of membership of the borderline cases lie between 0 and 1. The more an element or object x belongs to $\tilde{A}$, the closer to 1 is its grade of membership $\mu_{\tilde{A}}(x)$. The use of a numerical scale such as the interval [0, 1] allows a convenient representation of the gradation in membership. Precise membership values do not exist by themselves, they are tendency indices that are subjectively assigned by an individual or a group. Moreover, they are context-dependent. The grades of membership reflect an "ordering" of the objects in the universe, induced by the predicate associated with $\tilde{A}$; this "ordering", when it exists, is more important than the membership values themselves. The membership assessment of objects can sometimes be made easier by the use of a similarity measure with respect to an ideal element. Note that a membership value $\mu_{\tilde{A}}(x)$ can be interpreted as the degree of compatibility of the predicate associated with $\tilde{A}$ and the object x. For concepts such as "tallness", related to a physical measurement scale, the assignment of membership values will often be less controversial than for more complex and subjective concepts such as "beauty".

The above approach, developed by Zadeh (1964), provides a tool for modeling human-centered systems. As a matter of fact, fuzziness seems to pervade most human perception and thinking processes. Parikh (1977) has pointed out that no nontrivial first-order-logic-like observational predicate (i.e., one pertaining to perception) can be defined on an observationally connected space; the only possible observational predicates on such a space are not classical predicates but "vague" ones. Moreover, according to Zadeh (1973), one of the most important facets of human thinking is the ability to summarize information "into labels of fuzzy sets which bear an approximate relation to the primary data". Linguistic descriptions, which are usually summary descriptions of complex situations, are fuzzy in essence.

It must be noticed that fuzziness differs from imprecision. In tolerance analysis imprecision refers to lack of knowledge about the value of a parameter and is thus expressed as a crisp tolerance interval. This interval is the set of possible values of the parameters. Fuzziness occurs when the interval has no sharp boundaries, i.e., is a fuzzy set $\tilde{A}$. Then, $\mu_{\tilde{A}}(x)$ is interpreted as the degree of possibility (Zadeh, 1978) that x is the value of the parameter fuzzily restricted by $\tilde{A}$.

The word fuzziness has also been used by Sugeno (1977) in a radically different context. Consider an arbitrary object x of the universe X; to each nonfuzzy subset A of X is assigned a value $g_x(A) \in [0,1]$ expressing the "grade of fuzziness" of the statement "x belongs to A". In fact this grade of fuzziness must be understood as a grade of certainty: according to the mathematical definition of g, $g_x(A)$ can be interpreted as the probability, the degree of subjective belief, the possibility, that x belongs to A.

Generally, g is assumed increasing in the sense of set inclusion, but not necessarily additive as in the probabilistic case. The situation modeled by Sugeno is more a matter of guessing whether $x \in A$ rather than a problem of vagueness in the sense of Zadeh. The existence of two different points of view on "fuzziness" has been pointed out by MacVicar-Whelan (1977) and Skala. The monotonicity assumption for g seems to be more consistent with human guessing than does the additivity assumption. For instance, seeing a piece of Indian pottery in a shop, we may try to guess whether it is genuine or counterfeit; obviously, genuineness is a fuzzy concept. Hence x is the Indian pottery; A is the crisp set of genuine Indian artifacts; and $g_x(A)$ expresses, for instance, a subjective belief that the pottery is indeed genuine. The situation is slightly more complicated when we try to guess whether the pottery is old: actually, the set $\tilde{A}$ of old Indian pottery is fuzzy because "old" is a vague predicate.

1.2 Fuzzy Sets

Let X be a classical set of objects, called the universe, whose generic elements are denoted x. Membership in a classical subset A of X is often viewed as a characteristic function, μ_A from X to $\{0, 1\}$ such that

$$\mu_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \in X - A. \end{cases}$$

If the valuation set, i.e., 0 and 1, is allowed to be the real interval $[0, 1]$, A is called a fuzzy set (Zadeh, 1965), and $\mu_A(x)$ is the grade of membership of x in A. Sometimes we denotes $\mu_A(x)$ by $A(x)$. The closer the value of $\mu_A(x)$ is to 1, the more

x belongs to A. Clearly, A is a subset of X that has no sharp boundary. Hence A is completely characterized by the set of pairs

$$A = \{(x, \mu_A(x)) | x \in X\}.$$

A more convenient notation was proposed by Zadeh [49]. When X is a finite set $\{x_1, ..., x_n\}$ a fuzzy set on X is expressed as

$$A = \frac{\mu_A(x_1)}{x_1} + \cdots + \frac{\mu_A(x_n)}{x_n} = \sum_{i=1}^{n} \frac{\mu_A(x_i)}{x_i}.$$

When X is not finite, we write

$$A = \int_x \mu_A(x).$$

Definition 1.2.1. (Equality of fuzzy sets) Two fuzzy sets A and B are said to be equal (denoted $A = B$) iff

$$\forall\ x \in X,\ \ \mu_A(x) = \mu_B(x).$$

Definition 1.2.2. (Support) The support of a fuzzy set A is the ordinary subset of X:

$$supp\ A = \{x \in X,\ \ \mu_A(x) > 0\}.$$

Definition 1.2.3. (Core) The core of a fuzzy set A is the set of all points with the membership degree one in A:

$$coreA = \{x \in X |\ \mu_A(x) = 1\}.$$

Definition 1.2.4. (Height of a fuzzy set) The height of A is $hgt(A) = sup_{x \in X}\ \mu_A(x)$, i.e., the least upper bound of $\mu_A(x)$.

Definition 1.2.5. (Normal fuzzy set) A is said to be normal if and only if $\exists\ x \in X,\ \mu_A(x) = 1$; this definition implies $hgt(A) = 1$.

Definition 1.2.6. (Empty fuzzy set) The empty set ϕ is defined as $\forall\ x \in X,\ \mu_\phi(x) = 0$; of course, $\forall\ X,\ \mu_X(x) = 1$.

Definition 1.2.7. $(h - Cuts)$ When we want to exhibit an element $x \in X$ that typically belongs to a fuzzy set A, we may demand its membership value to be greater than some threshold $h \in [0, 1]$. The ordinary set of such elements is the $h - cut$ of A,

$$[A]_h = \begin{cases} \{\ x \in X |\ \mu_A(x) \geq h\ \} & if \quad h > 0 \\ cl(suppA) & if \quad h = 0 \end{cases}$$

where $cl(suppA)$ denotes the closure of the support of A. One also defines the strong h-cut as $[A]_{\overline{h}} = \{\ x \in X,\ \mu_A(x) > h\ \}$.

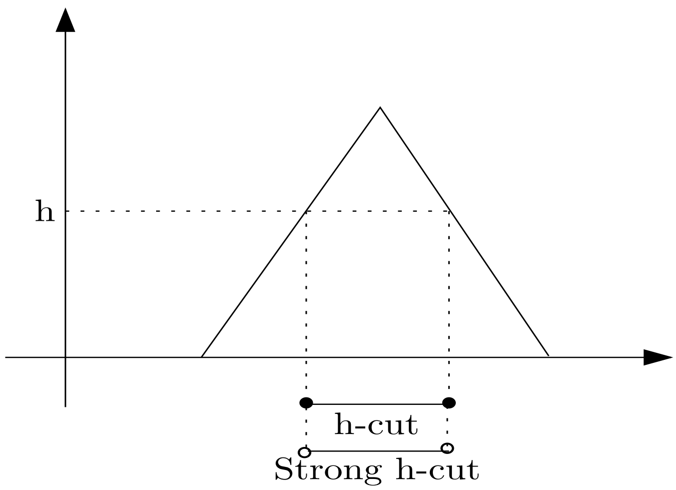

Figure 1.1: $h - Cut\ \ and\ \ Strong\ h - Cut.$

The membership function of a fuzzy set A can be expressed in terms of the characteristic functions of its $h-cuts$ according to the formula [13]

$$\mu_A(x) = \sup_{h \in]0,1]} \min(h, \mu_{A_h}(x)),$$

where

$$\mu_A(x) = \begin{cases} 1, & x \in A, \\ 0, & otherwise. \end{cases}$$

It is easily checked that the following properties hold:

$$[A \cup B]_h = [A]_h \cup [B]_h, \qquad [A \cap B]_h = [A]_h \cap [B]_h.$$

Definition 1.2.8. (Convexity) A fuzzy set A of X is called convex if $[A]_h$ is a convex subset of X for each $h \in [0,1]$. We now state a useful theorem that provides us with an alternative formulation of convexity of fuzzy sets. For the sake of simplicity, we restrict the theorem to fuzzy sets on $\mathbb{R}$, which are of primary interest in this text.

Theorem 1.2.1. *A fuzzy set A on $\mathbb{R}$ is convex if and only if*

$$A(\lambda x_1 + (1-\lambda)x_2) \geq \min\{A(x_1), A(x_2)\} \tag{1.1}$$

for all $x_1, x_2 \in \mathbb{R}$ and all $\lambda \in [0,1]$, where min *denotes the minimum operator.*

Proof. Assume that A is convex and let $h = A(x_1) \leq A(x_2)$. Hence $x_1, x_2 \in [A]_h$ and, moreover, $\lambda x_1 + (1-\lambda)x_2 \in [A]_h$ for any $\lambda \in [0,1]$ by the convexity of A. Consequently,

$$A(\lambda x_1 + (1-\lambda)x_2) \geq h = A(x_1) = \min\{A(x_1), A(x_2)\}.$$

Conversely assume that A satisfies (1.1). We need to prove that for any $h \in [0,1]$, $[A]_h$ is convex. Now for any $x_1, x_2 \in [A]_h(i.e.,\ A(x_1) \geq h,\ A(x_2) \geq h)$, and for any $\lambda \in [0,1]$, by (1.1)

$$A(\lambda x_1 + (1-\lambda)x_2) \geq \min\{A(x_1), A(x_2)\} \geq \min(h,h) = h,$$

i.e., $\lambda x_1 + (1-\lambda)x_2 \in [A]_h$. Therefore, $[A]_h$ is convex for any $h \in [0,1]$. Hence, A is convex. □

Among the various types of fuzzy sets, of special significance are fuzzy sets that are defined on the set $\mathbb{R}$ of real numbers. Membership functions of these sets, which have the form

$$A : \mathbb{R} \to [0,1],$$

clearly have a quantitative meaning and may, under certain conditions, be viewed as fuzzy number that can be defined as follows.

Definition 1.2.9. (Fuzzy number) A fuzzy number is a map $A : \mathbb{R} \to I = [0,1]$ which satisfies:

(i) A is upper semi-continuous, i.e.,

$$\{x | A(x) < t\} \ \ is \ \ open \ \ for \ all \ t \in \mathbb{R}.$$

(ii) $A(x) = 0$ outside some interval $[c,d] \subset \mathbb{R}$,

(iii) There exist real numbers a, b such that $c \leq a \leq b \leq d$ where

1. $A(x)$ is monotonic increasing on $[c,a]$,

2. $A(x)$ is monotonic decreasing on $[b, d]$,

3. $A(x) = 1, \; a \leq x \leq b$.

In this work, we show the set of fuzzy numbers by E.

Definition 1.2.10. (Quasi fuzzy number) A quasi fuzzy number A is a fuzzy set of the real line with a normal, fuzzy convex and continuous membership function satisfying the limit conditions

$$\lim_{t \to +\infty} A(t) = 0, \qquad \lim_{t \to -\infty} A(t) = 0.$$

Remark 1.2.1. Let A be a fuzzy number. Then $[A]_h$ is a closed convex (compact) subset of $\mathbb{R}$ for all $h \in [0, 1]$.

Definition 1.2.11. (Triangular fuzzy number) A fuzzy number A is called triangular fuzzy number with center a, left width $\alpha > 0$ and right width $\beta > 0$ if its membership function has the following form:

$$A(t) = \begin{cases} 1 - \frac{(a-t)}{\alpha} & if \qquad a - \alpha \leq t \leq a \\ 1 - \frac{(t-a)}{\beta} & if \qquad a \leq t \leq a + \beta \\ 0 & \qquad otherwise \end{cases}$$

and we use for it the notation $A = (a, \alpha, \beta)$. It can easily be verified that

$$[A]_h = [a - (1-h)\alpha, a + (1-h)\beta], \; \forall h \in [0, 1].$$

The support of A is $(a - \alpha, a + \beta)$. If $\alpha = \beta$ A is called symmetrical triangular fuzzy number. Let A be a symmetrical triangular fuzzy number, then we use for it the notation $A = (a, \alpha)$.

Figure 1.2 represents triangular fuzzy number $A = (6, 1, 2)$

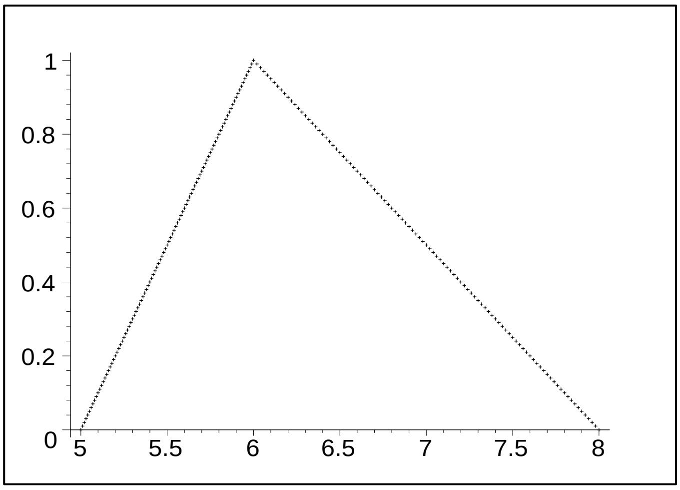

Figure 1.2: Triangular fuzzy number.

Definition 1.2.12. (Trapezoidal fuzzy number) A fuzzy number A is called trapezoidal fuzzy number with tolerance interval $[a, b]$, left width $\alpha > 0$ and right width $\beta > 0$ if its membership function has the following form:

$$A(t) = \begin{cases} 1 - \frac{(a-t)}{\alpha} & if \quad a - \alpha \le t \le a \\ 1 & if \quad a \le t \le b \\ 1 - \frac{(t-b)}{\beta} & if \quad b \le t \le b + \beta \\ 0 & otherwise \end{cases}$$

and we use for it the notation $A = (a, b, \alpha, \beta)$. It can easily be shown that

$$[A]_h = [a - (1-h)\alpha, b + (1-h)\beta], \forall h \in [0, 1].$$

The support of A is $(a-\alpha, b+\beta)$.

Figure 1.3 represent trapezoidal fuzzy number $A = (3, 5, 1, 2)$.

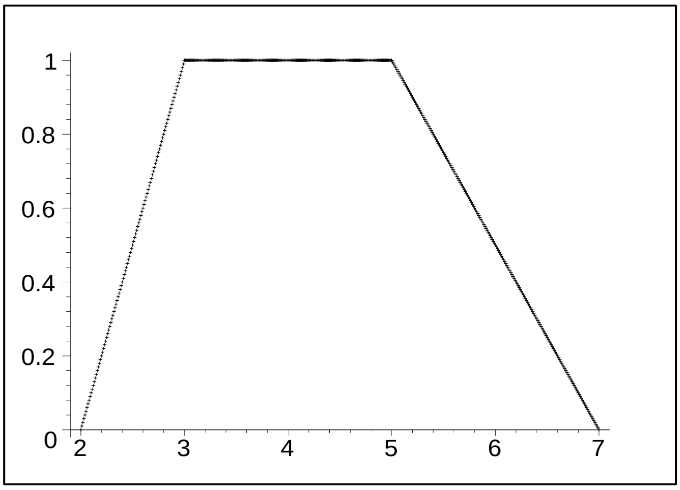

Figure 1.3: Trapezoidal fuzzy number.

Definition 1.2.13. We represent an arbitrary fuzzy number by an ordered pair of functions $(\underline{u}(r), \overline{u}(r))$, $0 \leq r \leq 1$, which satisfy the following requirements [15]:

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

A crisp number λ is simply represented by $\underline{u}(r) = \overline{u}(r) = \lambda, 0 \leq r \leq 1$. By appropriate definitions the fuzzy number space $\{\underline{u}(r), \overline{u}(r)\}$ becomes a convex cone, E^1.

Lemma 1.2.2. *Let v and w be fuzzy numbers and s be a real number. Then for*

$0 \leq r \leq 1$

$$u = v \text{ if and only if } \underline{u}(r) = \underline{v}(r) \text{ and } \overline{u}(r) = \overline{v}(r),$$

$$v + w = (\underline{v}(r) + \underline{w}(r), \overline{v}(r) + \overline{w}(r)),$$

$$v - w = (\underline{v}(r) - \overline{w}(r), \overline{v}(r) - \underline{w}(r)),$$

$$v.w = (\min\{\underline{v}(r).\underline{w}(r), \underline{v}(r).\overline{w}(r), \overline{v}(r).\underline{w}(r), \overline{v}(r).\overline{w}(r)\},$$

$$\max\{\underline{v}(r).\underline{w}(r), \underline{v}(r).\overline{w}(r), \overline{v}(r).\underline{w}(r), \overline{v}(r).\overline{w}(r)\}),$$

$$sv = s(\underline{v}(r), \overline{v}(r)).$$ *See [43].*

Definition 1.2.14. Any fuzzy number $A \in E$ can be described as

$$A(t) = \begin{cases} L(\frac{a-t}{\alpha}) & if \quad t \in [a-\alpha, a] \\ 1 & if \quad t \in [a, b] \\ R(\frac{t-b}{\beta}) & if \quad t \in [b, b+\beta] \\ 0 & otherwise \end{cases}$$

where $[a, b]$ is the core of A,

$$L : [0, 1] \to [0, 1], \quad R : [0, 1] \to [0, 1]$$

are continuous and non-increasing shape functions with $L(0) = R(0) = 1$ and $R(1) = L(1) = 0$. We call this fuzzy interval of LR-type and refer to it by

$$A = (a, b, \alpha, \beta)_{LR}.$$

The support of A is $(a - \alpha, b + \beta)$.

Definition 1.2.15. (Fuzzy point) Let A be a fuzzy number. If supp(A)$= \{x_0\}$ then A is called a fuzzy point and we use the notation $A = \overline{x_0}$.

Figure 1.4 represents fuzzy point $A = \overline{5}$.

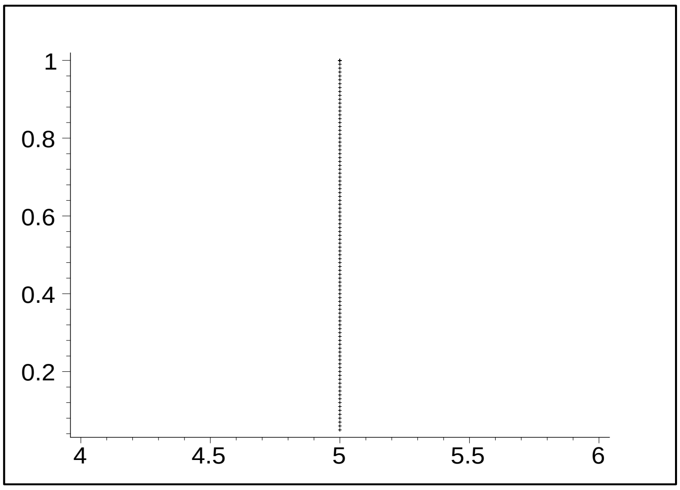

Figure 1.4: Fuzzy point $A = \overline{5}$.

Definition 1.2.16. The space E^n is all of fuzzy subsets U of $\mathbb{R}^n$ which satisfy the following conditions:

1. U is normal,
2. U is fuzzy convex,
3. U is upper semi-continuous,
4. $[U]_0$ is a bounded subset of $\mathbb{R}^n$,

when $n = 1$, elements of E^1 are Fuzzy numbers.

1.3 The extension principle

In order to use fuzzy numbers and relations in any intelligent system we must be able to perform arithmetic operations with these fuzzy quantities. In particular, we must be able to *add*, *subtract*, *multiply* and *divide* with fuzzy quantities. The process of doing these operations is called *fuzzy arithmetic*.

We shall first introduce an important concept from fuzzy set theory called the *extension principle*. We then use it to provide for these arithmetical operations on fuzzy numbers.

In general the extension principle plays a fundamental role in enabling us to extend any point operations to operations involving fuzzy sets. In the sequel, we define this principle.

Definition 1.3.1. [16](extension principle) Assume X and Y are crisp sets and let f be a mapping from X to Y,

$$f : X \to Y,$$

such that for each $x \in X$, $f(x) = y \in Y$. Assume A is a fuzzy subset of X, using extension principle, we can define $f(A)$ as a fuzzy subset of Y such that

$$f(A)(y) = \begin{cases} \sup_{x \in f^{-1}(y)} A(x) & \text{if } f^{-1}(y) \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where $f^{-1}(y) = \{x \in X | f(x) = y\}$.

Definition 1.3.2. (*sup-min extension n-place functions*) Let X_1, X_2,...,X_n and Y be a family of sets. Assume f is a mapping from the Cartesian product $X_1 \times X_2 \times \cdots \times X_n$

into Y. Let A_1, A_2,...,A_n be fuzzy subsets of X_1, X_2,...,X_n, respectively, then we use the extension principle for the evaluation of $f(A_1, A_2, ..., A_n)$. $f(A_1, A_2, ..., A_n)$ is a fuzzy set such that

$$f(A_1, A_2, ..., A_n)(y) = \begin{cases} \sup\{\min\{A_1(x_1), A_2(x_2), \ldots, A_n(x_n)\} \mid x \in f^{-1}(y)\} & \\ & \text{if} \quad f^{-1}(y) \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where $x = (x_1, x_2, ..., x_n)$.

Example 1.3.1. *Let $f : X \times X \to X$ be defined as*

$$f(x_1, x_2) = \lambda_1 x_1 + \lambda_2 x_2,\ \ \lambda_1,\ \ \lambda_2 \in \mathbb{R}.$$

Suppose A_1 and A_2 are fuzzy subsets of X. Then using the extension principle we get

$$f(A_1, A_2)(y) = \sup_{\lambda_1 x_1 + \lambda_2 x_2 = y} \min\{A_1(x_1), A_2(x_2)\}$$

and we use the notation $f(A_1, A_2) = \lambda_1 A_1 + \lambda_2 A_2$.

Definition 1.3.3. Let $X \neq \emptyset$ and $Y \neq \emptyset$ be crisp sets and let f be a function from $E^1(X)$ to $E^1(Y)$. Then f is called a fuzzy function (or mapping) and we use the notation

$$f : E^1(X) \to E^1(Y).$$

Let $A = (a_1, a_2, \alpha_1, \alpha_2)_{LR}$ and $B = (b_1, b_2, \beta_1, \beta_2)_{LR}$ be fuzzy numbers of $LR-type$. Using the (sup-min) extension principle, we can verify the following rules for addition and subtraction of fuzzy numbers of $LR - type$:

$$A + B = (a_1 + b_1, a_2 + b_2, \alpha_1 + \beta_1, \alpha_2 + \beta_2)_{LR}$$

$$A - B = (a_1 - b_1, a_2 - b_2, \alpha_1 + \beta_2, \alpha_2 + \beta_1)_{LR}$$

furthermore, if $\lambda \in \mathbb{R}$ is a real number then λA can be represented as

$$\lambda A = \begin{cases} (\lambda a_1, \lambda a_2, \lambda \alpha_1, \lambda \alpha_2)_{LR} & \text{if } \lambda \geq 0, \\ \\ (\lambda a_2, \lambda a_1, \mid \lambda \mid \alpha_2, \mid \lambda \mid \alpha_1)_{LR} & \text{if } \lambda < 0. \end{cases}$$

In particular, if $A = (a_1, a_2, \alpha_1, \alpha_2)$ and $B = (b_1, b_2, \beta_1, \beta_2)$ are fuzzy numbers of trapezoidal form, then

$$A + B = (a_1 + b_1, a_2 + b_2, \alpha_1 + \beta_1, \alpha_2 + \beta_2)$$

$$A - B = (a_1 - b_1, a_2 - b_2, \alpha_1 + \beta_2, \alpha_2 + \beta_1).$$

If $A = (a, \alpha_1, \alpha_2)$ and $B = (b, \beta_1, \beta_2)$ are fuzzy numbers of triangular form, then

$$A + B = (a + b, \alpha_1 + \beta_1, \alpha_2 + \beta_2)$$

$$A - B = (a - b, \alpha_1 + \beta_2, \alpha_2 + \beta_1),$$

and if $A = (a, \alpha)$ and $B = (b, \beta)$ are fuzzy numbers of symmetrical triangular form, then

$$A + B = (a + b, \alpha + \beta)$$

$$A - B = (a - b, \alpha + \beta),$$

$$\lambda A = (\lambda a, \mid \lambda \mid \alpha).$$

The above results can be generalized to linear combination of fuzzy numbers.

Let A and B be fuzzy numbers with $[A]_h = [a_1(h), a_2(h)]$ and $[B]_h = [b_1(h), b_2(h)]$, $0 \leq h \leq 1$. Then it can easily be shown that

$$[A + B]_h = [a_1(h) + b_1(h), a_2(h) + b_2(h)],$$

$$[-A]_h = [-a_2(h), -a_1(h)],$$

$$[A - B]_h = [a_1(h) - b_2(h), a_2(h) - b_1(h)],$$

$$[\lambda A]_h = [\lambda a_1(h), \lambda a_2(h)] \quad if \quad \lambda \geq 0,$$

$$[\lambda A]_h = [\lambda a_2(h), \lambda a_1(h)] \quad if \quad \lambda < 0,$$

for all $h \in [0, 1]$, i.e. any h-level set of the extended sum of two fuzzy numbers is equal to the sum of their h-level sets. The following two theorems show that this property is valid for any continuous function.

Theorem 1.3.2. *[37] Let $f : X \to X$ be a continuous function and let A be a fuzzy number. Then,*

$$[f(A)]_h = f([A]_h),$$

where $f(A)$ is defined by the extension principle and

$$f([A]_h) = \{f(x) \mid x \in [A]_h\}.$$

If $[A]_h = [a_1(h), a_2(h)]$ and f is continuous and monotone increasing then from the above theorem we get

$$[f(A)]_h = f([A]_h) = f([a_1(h), a_2(h)]) = [f(a_1(h)), f(a_2(h))].$$

Theorem 1.3.3. *[37] Let $f : X \times X \to X$ be a continuous function and let A and B be fuzzy numbers. Then*

$$[f(A, B)]_h = f([A]_h, [B]_h)$$

where,

$$f([A]_h, [B]_h) = \{f(x_1, x_2) \mid x_1 \in [A]_h, x_2 \in [B]_h\}.$$

Let $f(x, y) = xy$ and let $[A]_h = [a_1(h), a_2(h)]$, $[B]_h = [b_1(h), b_2(h)]$ be the $h-$level sets of two fuzzy numbers A and B. Applying above theorem we get

$$[f(A, B)]_h = f([A]_h, [B]_h) = [A]_h[B]_h$$

The equation

$$[AB]_h = [A]_h[B]_h = [a_1(h)b_1(h), a_2(h)b_2(h)]$$

holds if and only if A and B are both nonnegative, i.e. $A(x) = B(x) = 0$ for $x \leq 0$. If B is nonnegative then we have

$$[A]_h[B]_h = [\min\{a_1(h)b_1(h), a_1(h)b_2(h)\}, \max\{a_2(h)b_1(h), a_2(h)b_2(h)\}].$$

In general case, we obtain a very complicated expression for the α-level sets of the product AB

$$[A]_h[B]_h = [\min\{a_1(h)b_1(h), a_1(h)b_2(h), a_2(h)b_1(h), a_2(h)b_2(h)\},$$

$$\max\{a_1(h)b_1(h), a_1(h)b_2(h), a_2(h)b_1(h), a_2(h)b_2(h)\}].$$

1.4 Hausdorff distance for fuzzy numbers

Let A and B be fuzzy numbers with $[A]_h = [a_1(h), a_2(h)]$ and $[B]_h = [b_1(h), b_2(h)]$.

Definition 1.4.1. (Hausdorff distance) The Hausdorff distance between two (nonempty) sets $X,\ Y \subseteq \mathbb{R}$ is given as

$$d_H(X,Y) = \max\{\beta(X,Y), \beta(Y,X)\},$$

where $\beta(X,Y) = \sup_{x\in X} \rho(x,Y)$ and $\rho(x,Y) = \inf_{y\in Y} \mid x - y \mid$. The generalization

$$d_H(A,B) \quad = \quad \sup_{h\in(0,1]} d_H([A]_h, [B]_h) \ \ \forall\ A, B \in E^1,$$

defines a distance measure [20]. It is clear that

$$d_H(A,B) = \sup_{h\in[0,1]} \max\{\mid a_1(h) - b_1(h) \mid, \mid a_2(h) - b_2(h) \mid\},$$

i.e. $d_H(A,B)$ is the maximal distance between h-level sets of A and B [36].

1.5 Operation on fuzzy numbers

Some previous works related to operations on fuzzy numbers are those of Jain [25], Nahmias [35], Mizumoto and Tanaka [34],[33] Baas and Kwakernaak [9].

1.5.1 Addition and Multiplication

Addition: Addition is an increasing operation. Hence, the extended addition ($\oplus$) of fuzzy numbers gives a fuzzy number. Note that $-(M \oplus N) = (-M) \oplus (-N)$. ($\oplus$) is commutative and associative but has no group structure. The identity of ($\oplus$) is the nonfuzzy number 0. But M has no symmetrical element in the sense of a group structure. In particular, $M \oplus (-M) \neq 0, \quad \forall M \in E - \mathbb{R}$.

Multiplication: Multiplication is an increasing operation on $\mathbb{R}^+$ and a decreasing operation on $\mathbb{R}^-$. Hence, the product of fuzzy numbers $(\odot)$ that are all either positive or negative gives a positive fuzzy number. Note that $-(M) \odot N = -(M \odot N)$, so that the factors can have different signs. $(\odot)$ is commutative and associative. The set of positive fuzzy numbers is not a group for $(\odot)$: although $\forall M, \quad M \odot 1 = M$, the product $M \odot M^{-1} \neq 1$ as soon as M is not a real number. M has no inverse in the sense of group structure.

1.5.2 Subtraction

Subtraction is neither increasing nor decreasing. However, it is easy to check that $M \ominus N = M \oplus (-N), \quad \forall (M, N) \in E^2$ so that $M \ominus N$ is a fuzzy number whenever M and N are.

1.5.3 Division

Division is neither increasing nor decreasing. But, since $M \oslash N = M \odot (N^{-1}), \forall (M, N) \in E^2$, $M \oslash N$ is a fuzzy number when M and N are positive or negative fuzzy numbers. The division of ordinary fuzzy numbers can be performed similar to multiplication, by decomposition.

Definition 1.5.1. Let $X \neq \emptyset$ and $Y \neq \emptyset$ be crisp sets and let f be a function from $\mathcal{F}(X)$ to $\mathcal{F}(Y)$. Then f is said to be a fuzzy function (or mapping) and we use the notation

$$f : \mathcal{F}(X) \to \mathcal{F}(Y).$$

It should be noted, however, that a fuzzy function is not necessarily defined by Zadeh's extension principle. It can be any function which maps a fuzzy set $A \in \mathcal{F}(X)$ into a fuzzy set $B := f(A) \in \mathcal{F}(Y)$.

Definition 1.5.2. Let $X \neq \emptyset$ and $Y \neq \emptyset$ be crisp sets . A fuzzy mapping $f : \mathcal{F}(X) \to \mathcal{F}(Y)$ is said to be monotonic increasing if from $A, A' \in \mathcal{F}(X)$ and $A \subset A'$ it follow that $f(A) \subset f(A')$.

Theorem 1.5.1. *Let $X \neq \emptyset$ and $Y \neq \emptyset$ be crisp sets. Then every fuzzy mapping $f : \mathcal{F}(X) \to \mathcal{F}(Y)$ defined by the extension principle is monotonic increasing.*

Proof. Let $A, A' \in \mathcal{F}(X)$ such that $A \subset A'$. Then using the definition of sup-min extension principle we get

$$f(A)(y) = \sup_{x \in f^{-1}(y)} A(x) \leq \sup_{x \in f^{-1}(y)} A'(x) = f(A')(y)$$

for all $y \in Y$. $\square$

Example 1.5.2. *Let $f(x, y) = xy$ and let $[A]_h = [a_{1h}, a_{2h}]$ and $[B]_h = [b_{1h}, b_{2h}]$ be two fuzzy numbers. Applying above theorem we get*

$$[AB]_h = [A]_h[B]_h = [a_{1h}b_{1h}, a_{2h}b_{2h}],$$

hold if and only if A and B are both nonnegative, i.e. $A(x) = B(x) = 0$ for $x \leq 0$. In general form we obtain a very complicated expression for the α level sets of the product AB

$$[AB]_h = [m_h, M_h],$$

where

$$m_h = \min\{a_{1h}b_{1h}, a_{1h}b_{2h}, a_{2h}b_{1h}, a_{2h}b_{2h}\},$$

$$M_h = \max\{a_{1h}b_{1h}, a_{1h}b_{2h}, a_{2h}b_{1h}, a_{2h}b_{2h}\},$$

for $h \in I$.

Chapter 2

Neural network

2.1 Introduction

[16] The study of brain-style computation has its roots over 60 years ago in the work of McCulloch and Pitts (1943) and slightly later Hebb's famous Organization of Behavior (1949). The early work in artificial intelligence was torn between those who believed that intelligent systems could best be built on computer models after brains, and those like Minsky and Papert (1969) who believed that intelligence was fundamentally symbol processing of the kind readily modeled on the Von Neumann computer. For a variety of reasons, the symbol-processing approach become the dominant theme in Artificial Intelligence in the 1970s. However, the 1980s showed a rebirth in interest in neural computing:

Hopfield provided the mathematical foundation for understanding the dynamics of an important class of networks, in 1982.

Kohonen developed unsupervised learning networks for feature mapping into regular arrays of neurons, in 1984.

Rumelhart and McClelland introduced the back propagation learning algorithm for complex, multilayer network, in 1986.

Beginning in 1986-1987, many neural networks research programs were initiated. The list of applications that can be solved by neural networks has expanded from small test-size examples to large practical tasks. Very-large-scale integrated neural network chips have been fabricated. In the long term, we could expect that artificial neural systems will be used in applications involving vision, speech, decision making, and reasoning, but also as signal processors such as filters, detectors, and quality control systems.

2.2 The perceptron learning rule

Definition 2.2.1. [54] Artificial neural systems, or neural networks, are physical cellular systems which can acquire store, and utilize experimental knowledge.

The knowledge is in the form of stable states or mappings embedded in networks that can be recalled in response to the presentation of cues.

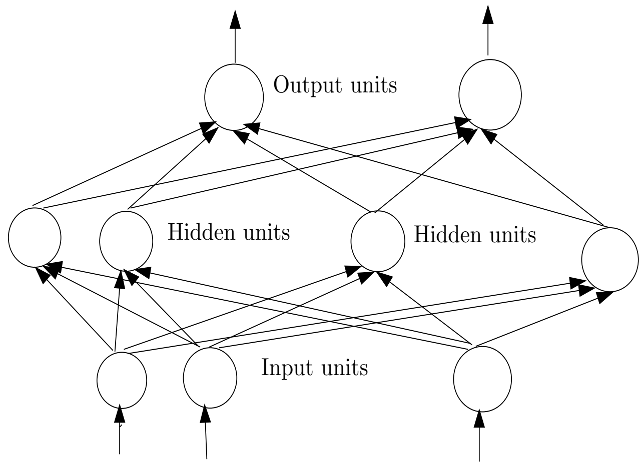

Figure 2.1: A multi-layer feedforward neural network.

The basic processing elements of neural networks are called artificial neurons, or simply neurons or nodes.

Each processing unit is characterized by an activity level (representing the state of polarization of a neuron), an output value (representing the firing rate of the neuron), a set of input connections,(representing synapses on the cell and its dendrite), a bias value (representing an internal resting level of the neuron), and a set of output connections (representing a neuron's axonal projection). Each of these aspects of the unit are represented mathematically by real numbers. Thus, each connection has an associated weight(synaptic strength)which determines the effect of the incoming input on the activation level of the unit. The weights may be positive (excitatory) or negative (inhibitory).

Definition 2.2.2. Let $w = (w_1, \ldots, w_n)^T$ and $x = (x_1, \ldots, x_n)^T$ be two vectors from $\mathbb{R}^n$. The scalar (or inner) product of w and x, denoted by $< w, x >$ or $w^T x$, is defined by

$$< w, x >= w_1x_1 + \ldots + w_nx_n = \sum_{j=1}^{n} w_jx_j$$

other definition of scalar product in two dimensional case is

$$< w, x >=\| w \|\| x \| \cos(w, x)$$

where $\| . \|$ denotes the Eucledean norm in the real plane, i.e.

$$\| w \|= \sqrt{w_1^2 + w_2^2}, \| x \|= \sqrt{x_1^2 + x_2^2}.$$

The signal flow form of neuron input, x_j, is considered to be unidirectional as indicated by arrow, as is a neuron's output signal flow. The neuron output signal is given by the following relationship

$$o = f(< w, x >) = f(w^T x) = f(\Sigma_{j=1}^{n} w_jx_j)$$

where $w = (w_1, \ldots, w_n)^T \in \mathbb{R}^n$ is the weight vectors.The function $f(w^T x)$ is often referred to as an activation (or transfer) function. Its domain is the set of activation value, net, of the neuron model, we thus often use this function as $f(net)$. The variable net is defined as a scalar product of the weight and input vectors

$$net =< w, x >= w^T x = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

and in the simplest case output value O is computed as

$$O = f(net) = \begin{cases} 1 & if \quad w^T x \geq b \\ 0 & otherwise, \end{cases}$$

where b is called threshold-level and this type of node is called a linear threshold unit.

The problem of learning in neural network is simply the problem of finding a set of connection strengths (weights) which allow the network to carry out the desired computation. The network is provided with a set of example input/output pairs (a training set) and is to modify its connections in order to approximate the function from which the input/output pairs have been drawn. The networks are then tested for ability to generalize.

The error correction learning procedure is simple enough in conception. The procedure is as follow: During training an input is put into the network and flows through the network generating a set of values on the output units. Then, the actual output is compared with the desired target, and a match is computed. If the output and target match, no change is made to the net. However, if the output differs from the target a change must be made to some of the connections.

The perceptron learning rule, introduced by Rosenblatt, is a typical error correction learning algorithm of single-layer feedforward networks with linear threshold activation function.

Usually, w_{ij} denotes the weight from the j-th input unit to the i-th output unit and w_i denotes the weight vector of the $i-$th output node.

We are given a training set of input/output pairs

No.	input values	desired output values
1.	$x^1 = (x_1^1, \dots, x_n^1)$	$y^1 = (y_1^1, \dots, y_m^1)$
$\vdots$		
$T.$	$x^T = (x_1^T, \dots, x_n^T)$	$y^T = (y_1^T, \dots, y_m^T).$

Our problem is to find weight vectors w_i such that

$$o_i(x^k) = sign(< w_i, x^k >) = y_i^k, i = 1, \dots, m,$$

for all training patterns k.

The activation function of the output nodes is linear threshold function of the form

$$o_i(x) = sign(< w_i, x >) = \begin{cases} +1 & if \quad < w_i, x > \geq 0, \\ -1 & if \quad < w_i, x > < 0, \end{cases}$$

and the weight adjustments in the perceptron learning method are performed by

$$w_i := w_i + \eta(y_i^k - sign(< w_i, x^k >))x^k, \quad i = 1, \dots, m,$$

$$w_{ij} := w_{ij} + \eta(y_i^k - sign(< w_i, x^k >))x_j^k, \quad j = 1, \dots, n,$$

where $\eta > 0$ is the learning rate.

From this equation it follows that if the desired output is equal to the computed output, $y_i^k = sign(< w_i, x^k >)$, then the weight vector of the i-th output node remains unchanged, i.e. w_i is adjusted if and only if the computed output, $o_i(x^k)$ is incorrect. The learning stops when all the weight vectors remain unchanged during a complete training cycle.

Consider now a single-layer network with one output node. Then the input components of the training patterns can be classified into two disjunct classes

$$C_1 = \{x^k | y^k = 1\}, \quad C_2 = \{x^k | y^k = -1\},$$

i.e. x belongs to class C_1 if there exists an input/output pair $(x, 1)$, and x belongs to class C_2 if there exists an input/output pair $(x, -1)$.

Taking into consideration the definition of the activation function it is easy to see that we are searching for a weight vector w such that

$$< w, x > \geq 0 \;\; for \;\; each \;\; x \in C_1, \;\; and \;\; < w, x > < 0 \;\; for \;\; each \;\; x \in C_2.$$

If such vector exists then the problem is called linearly separable.

2.3 Fuzzy neural network

In theory, neural networks, and fuzzy systems are equivalent in that they are convertible, yet in practice each has its own advantages and disadvantages. For neural networks, the knowledge is automatically acquired by the backpropagation algorithm, but the learning process is relatively slow and analysis of the trained network is difficult (black box). Neither is it possible to extract structural knowledge (rule) from the trained neural, nor can we integrate special information about the problem into the neural network in order to simplify the learning procedure.

Artificial neural systems can be considered as simplified mathematical models of brain like systems and they function as parallel distributed computing networks.

However, in contrast to conventional computers, which are programmed to perform specific task, most neural networks must be taught, or trained. They can learn new associations, new functional dependencies and new patterns. Although computers out perform both biological and artificial neural systems for tasks based on precise and fast arithmetic operations, artificial neural systems represent the promising new generation of information processing networks.

The computational process envisioned for fuzzy neural systems is as follows. It starts with the development of a "fuzzy neuron" based on the understanding of biological neuronal morphologies, followed by learning mechanisms. This leads to the following three steps in a fuzzy neural computational process:

- development of fuzzy neural models motivated by biological neurons,
- models of synaptic connections which incorporates fuzziness into neural network,
- development of learning algorithms (that is the method of adjusting the synaptic weights)

Two possible models of fuzzy neural systems are

- In response to linguistic statements, the fuzzy interface block provides an input vector to a multi-layer neural network. The neural network can be adapted (trained) to yield desired command outputs or decisions.

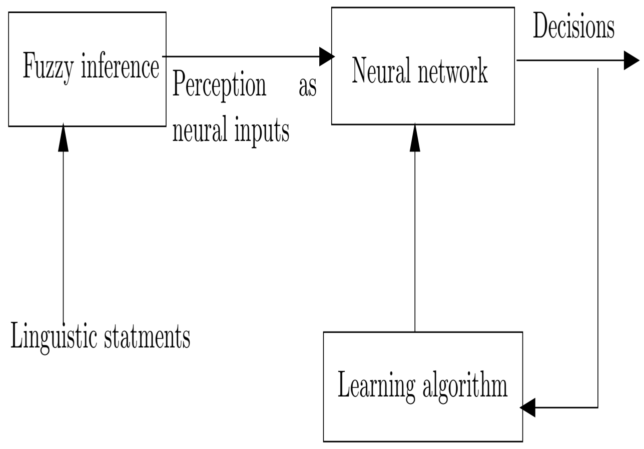

Figure 2.2: The first model of fuzzy neural system.

- A multi-layered neural network drives the fuzzy inference mechanism.

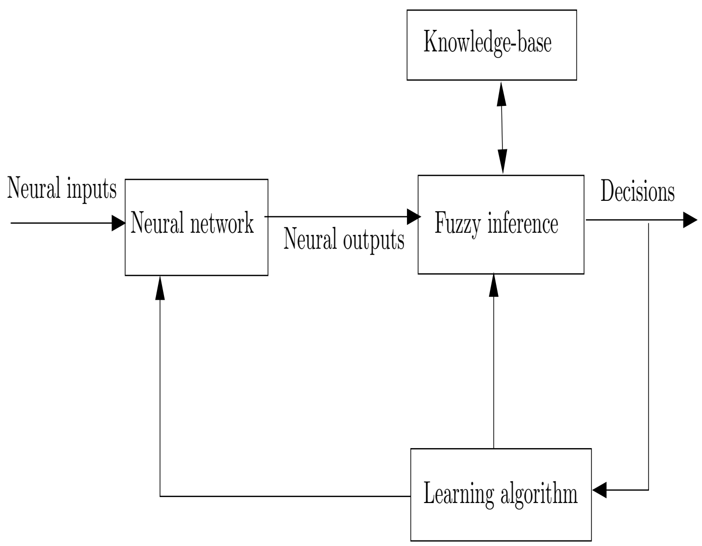

Figure 2.3: The second model of fuzzy neural system.

Neural networks are used to tune membership functions of fuzzy systems that are employed as decision-making systems for controlling equipment. Although fuzzy logic can encode expert knowledge directly using ruled with linguistic labels, it usually takes a lot of time to design and tune the membership functions which quantitatively define these linguistic labels. Neural network learning techniques can automate this process and substantially reduce development time and cost while improving performance. Consider a simple neural net in Figure 2.4. The two input neurons do not change the input signals so their output is the same as their input. The signal x_i interacts with the weight w_i to produce the product

$$p_i = w_i x_i, \qquad i = 1, 2.$$

The input information p_i is aggregated, by addition, to produce the input

$$net = p_1 + p_2 = w_1 x_1 + w_2 x_2$$

to the neuron. The neuron uses its transfer function f, to compute the output

$$y = f(net) = f(w_1 x_1 + w_2 x_2).$$

This simple neural net, which employs multiplication, addition and transfer function f, will be called as regular (or standard) neural net.

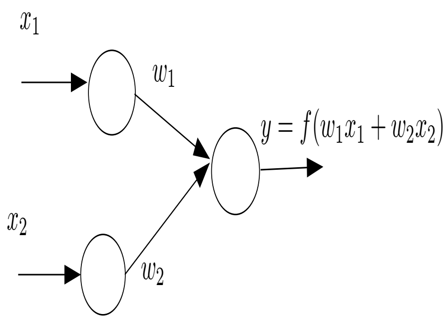

Figure 2.4: Simple neural net.

The direct fuzzification of conventional neural networks is to extend connection weights and/ or inputs and/ or fuzzy desired outputs (or targets) to fuzzy numbers. This extension is summarized in Table 2.1.

Fuzzy neural net	Weights	Inputs	Targets
Type 1	crisp	fuzzy	crisp
Type 2	crisp	fuzzy	fuzzy
Type 3	fuzzy	fuzzy	fuzzy
Type 4	fuzzy	crisp	fuzzy
Type 5	crisp	crisp	fuzzy
Type 6	fuzzy	crisp	crisp
Type 7	fuzzy	fuzzy	crisp

Table 2.1: Direct fuzzification of neural networks.

Fuzzy neural networks (FNN) of Type 1 are used in classification problem of a fuzzy input vector to a crisp class [24, 28]. The networks of Type $2, 3$ and 4 are used to implement fuzzy IF-THEN rules [23, 21].

However, the last three types in Table 2.1 are unrealistic.

- In Type 5, outputs are always real numbers because both inputs and weights are real numbers.

- In Types 6 and 7, the fuzzification of weights is not necessary because targets are real numbers.

Chapter 3

Numerical solution of fuzzy polynomials by fuzzy neural network

3.1 Introduction

Polynomials play a major role in various areas such as mathematics, engineering and social sciences. In this chapter, we are interested in finding the solution of polynomials like $A_1x + A_2x^2 + \ldots + A_nx^n = A_0$ and $A_1^{'}x + A_2^{'}x^2 + \ldots + A_n^{'}x^n = A_1^{''}x + A_2^{''}x^2 + \ldots + A_n^{''}x^n + A_0$ for $x \in \mathbb{R}$ (if exists) where $A_0, A_1, \ldots, A_n, A_1^{'}, \ldots, A_n^{''}$ are fuzzy numbers by a learning algorithm of fuzzy neural networks. The applications of fuzzy polynomials are considered by [3].

Ishibuchi *et al.* [21] proposed a learning algorithm of fuzzy neural networks with triangular fuzzy weights and Hayashi *et al.* [18] also fuzzified the delta rule. Linear and nonlinear fuzzy equations are solved by [1, 2, 5, 8, 11]. Buckley and Eslami [10] are concerned with neural net solutions to fuzzy problems.

In this chapter, 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 [48]. 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 finding the real root (if exists) of the polynomials. The proposed algorithm illustrated by solving some examples in last section.

3.2 Fuzzy neural network architecture

3.2.1 Operations of fuzzy numbers on the neural network

We briefly mention fuzzy number operations defined by the extension principle [48]. Since input vector of feedforward neural network is fuzzified in this section, the following addition, multiplication and nonlinear mapping of fuzzy numbers are necessary for defining our fuzzy neural network:

$$\mu_{A+B}(z) = max\{\mu_A(x) \wedge \mu_B(y) | z = x + y\}, \tag{3.1}$$

$$\mu_{AB}(z) = max\{\mu_A(x) \wedge \mu_B(y) | z = xy\}, \tag{3.2}$$

$$\mu_{f(Net)}(z) = max\{\mu_{Net}(x) | z = f(x)\}, \tag{3.3}$$

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. These operations are illustrated in Figure 3.1.

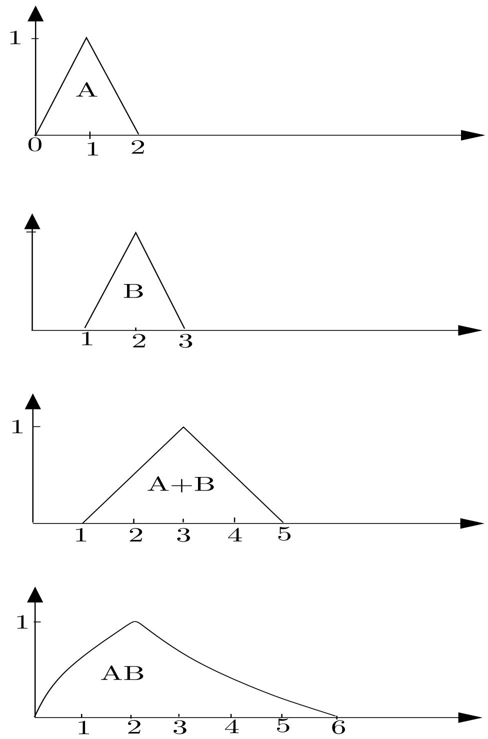

Figure 3.1: Fuzzy number operations: addition and multiplication.

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

$$[X]_h = \{x|\mu_X(x) \geq h, x \in \mathbb{R}\} \quad for \quad 0 < h \leq 1, \tag{3.4}$$

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

$$[X]_h = [[X]_h^L, [X]_h^U], \tag{3.5}$$

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:

$$[A]_h + [B]_h = [[A]_h^L + [B]_h^L, [A]_h^U + [B]_h^U], \tag{3.6}$$

$$f([Net]_h) = f([[Net]_h^L, [Net]_h^U]) = [f([Net]_h^L), f([Net]_h^U)], \tag{3.7}$$

$$\begin{aligned} w.[A]_h &= [w.[A]_h^L, w.[A]_h^U], \quad if \quad w \geq 0, \\ w.[A]_h &= [w.[A]_h^U, w.[A]_h^L], \quad if \quad w \leq 0. \end{aligned} \tag{3.8}$$

3.2.2 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:

$$O_i = A_i, \qquad i = 1, 2, \ldots, n. \tag{3.9}$$

Output unit:

$$Y = f(Net), \tag{3.10}$$

$$Net = \sum_{j=1}^{n} w_j.O_j, \tag{3.11}$$

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

The input-output relation in Eqs.(3.9)-(3.11) are defined by the extension principle [48] as in Hayashi *et al.* [18] and Ishibuchi *et al.*[22].

3.2.3 Calculation of fuzzy output

The fuzzy output from each unit in Eqs.(3.9)-(3.11) 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:

$$[O_i]_h = [A_i]_h, \qquad i = 1, 2, \ldots, n. \tag{3.12}$$

Output unit:

$$[Y]_h = f([Net]_h), \tag{3.13}$$

$$[Net]_h = \sum_{j=1}^{n} w_j.[O_j]_h. \tag{3.14}$$

From Eqs.(3.12)-(3.14), 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.(3.6)-(3.8), the above relations are written as follows:

Input units:

$$[O_i]_h = [[O_i]_h^L, [O_i]_h^U] = [[A_i]_h^L, [A_i]_h^U], \qquad i = 1, 2, \ldots, n. \tag{3.15}$$

Output unit:

$$[Y]_h = [[Y]_h^L, [Y]_h^U] = [f([Net]_h^L), f([Net]_h^U)], \tag{3.16}$$

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

$$[\sum_{j \in m} w_j.[O_j]_h^L + \sum_{j \in c} w_j.[O_j]_h^U, \sum_{j \in m} w_j.[O_j]_h^U + \sum_{j \in c} w_j.[O_j]_h^L], \tag{3.17}$$

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

3.3 Fuzzy polynomials

We are interested in finding real root

$$A_1 x + A_2 x^2 + \ldots + A_n x^n = A_0, \tag{3.18}$$

for $x \in \mathbb{R}$, when $A_i \in E$ for $i = 1, \ldots, n$. A FNN_2 (fuzzy neural network with fuzzy set input signals and real number weights) [21] solution to Eq.(3.17) is given in Figure 3.2. 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 propose a learning algorithm from the cost function for adjusting weights.

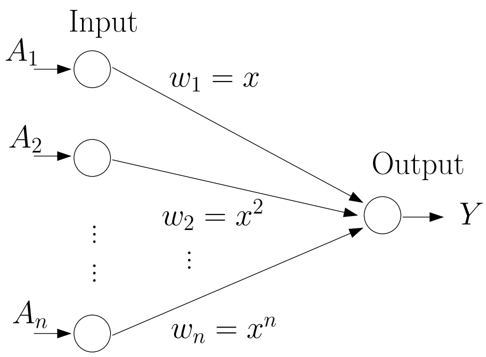

Figure 3.2: Fuzzy neural network to solve fuzzy polynomial

3.3.1 Learning of fuzzy neural network

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

$$[A_0]_h = [[A_0]_h^L, [A_0]_h^U], \qquad h \in [0,1], \tag{3.19}$$

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 (see Figure 3.3):

$$\begin{aligned} e^L(h) &= \tfrac{1}{2}(A_0^L(h) - Y^L(h))^2, \\ e^U(h) &= \tfrac{1}{2}(A_0^U(h) - Y^U(h))^2, \\ e(h) &= e^L(h) + e^U(h), \end{aligned} \tag{3.20}$$

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

$$e = \sum_h he(h). \tag{3.21}$$

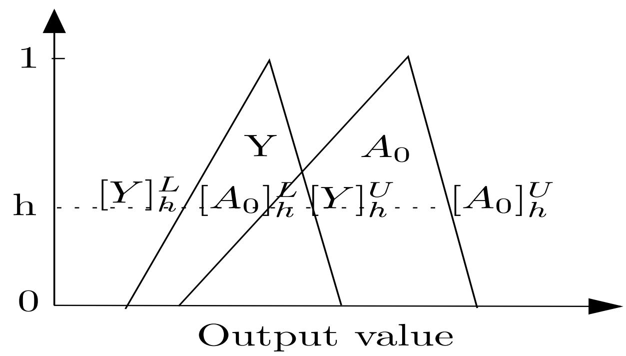

Figure 3.3: Fuzzy output Y and fuzzy target A_0.

Theoretically this cost function satisfies the following relation if we use infinite number of h-level sets in Eq.(3.21)

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

The weights are updated by the following rules [21, 41]

$$\triangle w_1(t) = -\eta \sum_h h \frac{\partial e(h)}{\partial w_1} + \alpha. \triangle w_1(t-1), \tag{3.22}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq.(3.22) can be written as follows:

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

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 $w_1 \geq 0$

$$\begin{aligned} \triangle w_1(t) = \eta \textstyle\sum_h h[(A_0^L(h) - Y^L(h))A_1^L(h) + \\ (A_0^U(h) - Y^U(h))A_1^U(h)] + \alpha . \triangle w_1(t-1), \end{aligned} \tag{3.23}$$

otherwise

$$\begin{aligned} \triangle w_1(t) = \eta \textstyle\sum_h h[(A_0^L(h) - Y^L(h))A_1^U(h) + \\ (A_0^U(h) - Y^U(h))A_1^L(h)] + \alpha . \triangle w_1(t-1). \end{aligned} \tag{3.24}$$

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.

Example 3.3.1. *Consider the following fuzzy polynomial*

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

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 3.3.2. *Consider the following fuzzy polynomial*

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

with the exact solution $x = -1$. *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* $e <= 0.01$.

3.4 Dual 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 numbers $u \in E^1$ we have

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

Therefore, the fuzzy polynomials

$$A_1^{'}x + A_2^{'}x^2 + \ldots + A_n^{'}x^n = A_1^{''}x + A_2^{''}x^2 + \ldots + A_n^{''}x^n + A_0, \tag{3.25}$$

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 to Eq.(3.25) by the neural network, with assumption $A_{i1}^{'} - A_{i1}^{''} < A_{i2}^{'} - A_{i2}^{''} < A_{i3}^{'} - A_{i3}^{''}$ for $i = 1, \ldots, n$. Therefore, we have of the Eq.(3.25)

$$A_1x + A_2x^2 + \ldots + A_nx^n = A_0, \tag{3.26}$$

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) [21] solution to Eq.(3.26) is given in Figure 3.3. The input neurons make no change in their inputs, so the input to the output neuron is

$$A_1x + A_2x^2 + \ldots + A_nx^n$$

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

$$Y = A_1x + A_2x^2 + \ldots + A_nx^n.$$

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. Learning algorithm for dual fuzzy polynomials was proposed in Section 3.3.

Example 3.4.1. *Consider the following 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 $x = 1$. 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 3.4.2. *Consider the following 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 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 3.1 for more details.

i	Example 3.3.1	Example 3.3.2	Example 3.4.1	Example 3.4.2
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
$\vdots$				$\vdots$

Table 3.1: The obtained values of $w_1(i)$, the approximation of real root

Chapter 4

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

4.1 Introduction

Systems of polynomials play a major role in various areas such as pure and applied mathematics, engineering and social sciences. In Chapter 3, we studied "Numerical solution of fuzzy polynomials by fuzzy neural network", where tried to find the numerical solution $x \in \mathbb{R}$ (if exists) of a fuzzy polynomial equation such as $A_1x + A_2x^2 + \ldots + A_nx^n = A_0$ where $A_0, A_1, \ldots, A_n$ are fuzzy numbers.

Now, consider the following system of fuzzy polynomials like

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

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

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

where $x_1, x_2, \ldots, x_n \in \mathbb{R}$ and all coefficients are fuzzy numbers. In this chapter, 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 a system of fuzzy polynomials?

In this chapter, 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 [38, 39, 46] for more information on such polynomials. Some applications of fuzzy polynomials are considered by [3, 29].

In this chapter, 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 [48, 52]. The output from the fuzzy neural network, which is also a fuzzy number, is numerically calculated by the 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 to find a real root (if exists)of a system of polynomials. The effectiveness of the proposed algorithm is proved by solving some examples in the last section.

4.2 Preliminaries of fuzzy neural network

4.2.1 Input-output relation of each unit for system of fuzzy polynomials

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

Input units:

$$O_{ij} = A_{ij}, \qquad i = 1, 2; j = 1, 2, \ldots, n. \tag{4.1}$$

Hidden units:

$$O'_{ij} = f(Net_{ij}), \tag{4.2}$$

$$Net_{ij} = w_{ij}.O_{ij}, \qquad i = 1, 2; j = 1, 2, \ldots, n. \tag{4.3}$$

Output unit:

$$Y_i = f(Net'_i), \tag{4.4}$$

$$Net'_i = \sum_{j=1}^{n} w'_{ij}.O'_{ij}, \qquad i = 1, 2, \tag{4.5}$$

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

The input-output relation in Eqs.(4.1)-(4.5) are defined by the extension principle [48, 52] as in Hayashi et al. [18] and Ishibuchi *et al.* [22].

4.2.2 Calculation of fuzzy output for system of fuzzy polynomials

The fuzzy output from each unit in Eqs.(4.1)-(4.5) 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:

$$[O_{ij}]_h = [A_{ij}]_h, \qquad i = 1,2; j = 1,2,\ldots,n. \tag{4.6}$$

Hidden units:

$$[O'_{ij}]_h = f([Net_{ij}]_h), \tag{4.7}$$

$$[Net_{ij}]_h = w_{ij}.[O_{ij}]_h, \qquad i = 1,2; j = 1,2,\ldots,n. \tag{4.8}$$

Output unit:

$$[Y_i]_h = f([Net'_i]_h), \tag{4.9}$$

$$[Net'_i]_h = \sum_{j=1}^{n} w'_{ij}.[O'_{ij}]_h, \qquad i = 1,2. \tag{4.10}$$

From Eqs.(4.6)-(4.10), 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.(4.6)-(4.10), the above relations are written as follows:

Input units:

$$[O_{ij}]_h = [[O_{ij}]_h^L, [O_{ij}]_h^U] = [[A_{ij}]_h^L, [A_{ij}]_h^U], \qquad i = 1, 2; j = 1, 2, \ldots, n. \tag{4.11}$$

Hidden units:

$$[O'_{ij}]_h = [[O'_{ij}]_h^L, [O'_{ij}]_h^U] = [f([Net_{ij}]_h^L), f([Net_{ij}]_h^U)]. \tag{4.12}$$

If $w_{ij} \geq 0$

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

otherwise

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

$$i = 1, 2; \quad j = 1, \ldots, n.$$

Output unit:

$$[Y_i]_h = [[Y_i]_h^L, [Y_i]_h^U] = [f([Net'_i]_h^L), f([Net'_i]_h^U)], \tag{4.13}$$

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

$$[\sum_{j \in m} w'_{ij}.[O'_{ij}]_h^L + \sum_{j \in c} w'_{ij}.[O'_{ij}]_h^U, \sum_{j \in m} w'_{ij}.[O'_{ij}]_h^U + \sum_{j \in c} w'_{ij}.[O'_{ij}]_h^L], \tag{4.14}$$

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

4.3 System of fuzzy polynomials

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

$$\begin{cases} A_{11}xy + A_{12}x^2y^2 + \ldots + A_{1n}x^ny^n = A_{10}, \\ A_{21}xy + A_{22}x^2y^2 + \ldots + A_{2n}x^ny^n = A_{20}, \end{cases} \tag{4.15}$$

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) [21] solution to Eq.(4.15) is given in Figure 4.1. 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

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

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

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

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 propose a learning algorithm from the cost function to adjust weights.

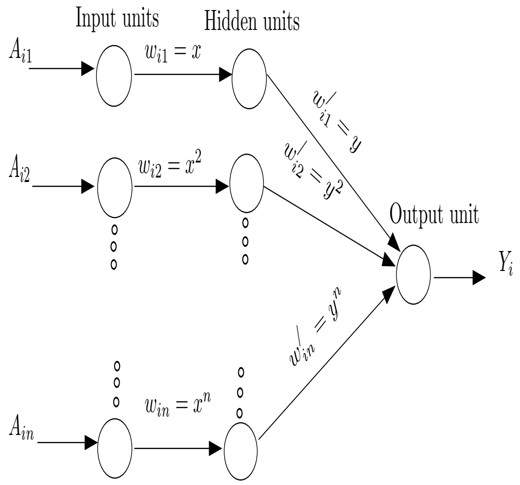

Figure 4.1: Fuzzy neural network to solve system of the fuzzy polynomials

4.3.1 Learning of fuzzy neural network

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

$$[A_{i0}]_h = [[A_{i0}]_h^L, [A_{i0}]_h^U], \qquad h \in [0, 1], \tag{4.16}$$

where $A_{i0}^L(h)$ denotes the left-hand side and $A_{i0}^U(h)$ 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 (see Figure 4.2):

$$\begin{aligned} e_i^L(h) &= \tfrac{1}{2}(A_{i0}^L(h) - Y_i^L(h))^2, \\ e_i^U(h) &= \tfrac{1}{2}(A_{i0}^U(h) - Y_i^U(h))^2, \qquad i = 1, 2, \\ e_i(h) &= e_i^L(h) + e_i^U(h), \end{aligned} \tag{4.17}$$

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

$$e = \sum_{i=1}^{2} \sum_{h} h e_i(h). \tag{4.18}$$

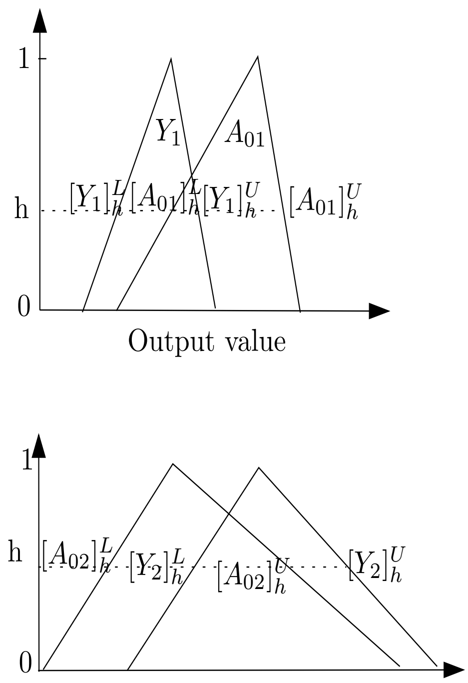

Figure 4.2: Fuzzy output Y_1, Y_2 and fuzzy target A_{01}, A_{02}.

Theoretically this cost function satisfies the following relation if we use an infinite number of h-level sets in Eq.(4.18)

$$e \longrightarrow 0 \quad \textit{if and only if} \quad Y_i \longrightarrow A_{i0}, \qquad i = 1, 2.$$

The weights are updated by the following rules [21, 41]

$$\triangle w'_{i1}(t) = -\eta \sum_{h} h \frac{\partial e_i(h)}{\partial w'_{i1}} + \alpha. \triangle w'_{i1}(t-1), \tag{4.19}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq.(4.19) can be written as follows:

$$\frac{\partial e_i(h)}{\partial w'_{i1}} = \frac{\partial e_i^L(h)}{\partial w'_{i1}} + \frac{\partial e_i^U(h)}{\partial w'_{i1}},$$

$$\frac{\partial e_i^L(h)}{\partial w'_{i1}} = \frac{\partial e_i^L(h)}{\partial Y_i^L} \times \frac{\partial Y_i^L(h)}{\partial w'_{i1}},$$

$$\frac{\partial e_i^U(h)}{\partial w'_{i1}} = \frac{\partial e_i^U(h)}{\partial Y_i^U} \times \frac{\partial Y_i^U(h)}{\partial w'_{i1}},$$

$$\frac{\partial e_i^L(h)}{\partial Y_i^L} = -(A_{i0}^L(h) - Y_i^L(h)), \qquad \frac{\partial e_i^U(h)}{\partial Y_i^U} = -(A_{i0}^U(h) - Y_i^U(h)).$$

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

$$\frac{\partial Y_i^L(h)}{\partial w'_{i1}} = O'^L_{i1}(h), \qquad \frac{\partial Y_i^U(h)}{\partial w'_{i1}} = O'^U_{i1}(h);$$

otherwise

$$\frac{\partial Y_i^L(h)}{\partial w'_{i1}} = O'^U_{i1}(h), \qquad \frac{\partial Y_i^U(h)}{\partial w'_{i1}} = O'^L_{i1}(h).$$

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

$$\begin{aligned} \triangle w'_{i1}(t) = \eta \textstyle\sum_h h[(A_{i0}^L(h) - Y_i^L(h))O'^L_{i1}(h) + \\ (A_{i0}^U(h) - Y_i^U(h))O'^U_{i1}(h)] + \alpha. \triangle w'_{i1}(t-1); \end{aligned} \tag{4.20}$$

otherwise

$$\begin{aligned} \triangle w'_{i1}(t) = \eta \textstyle\sum_h h[(A_{i0}^L(h) - Y_i^L(h))O'^U_{i1}(h) + \\ (A_{i0}^U(h) - Y_i^U(h))O'^L_{i1}(h)] + \alpha. \triangle w'_{i1}(t-1). \end{aligned} \tag{4.21}$$

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

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

and the weights are updated by the following rules [21, 41]

$$\triangle w_{i1}(t) = -\eta \sum_h h \frac{\partial e_i(h)}{\partial w_{i1}} + \alpha. \triangle w_{i1}(t-1), \tag{4.22}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq.(4.22) can be written as follows:

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

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

$$\frac{\partial e_i^L(h)}{\partial w_{i1}} = \frac{\partial e_i^L(h)}{\partial Y_i^L(h)} \times \frac{\partial Y_i^L(h)}{\partial O_{i1}'^L(h)} \times \frac{\partial O_{i1}'^L(h)}{\partial w_{i1}},$$
$$\frac{\partial e_i^U(h)}{\partial w_{i1}} = \frac{\partial e_i^U(h)}{\partial Y_i^U(h)} \times \frac{\partial Y_i^U(h)}{\partial O_{i1}'^U(h)} \times \frac{\partial O_{i1}'^U(h)}{\partial w_{i1}},$$
$$\frac{\partial e_i^L(h)}{\partial Y_i^L(h)} = -(A_{i0}^L(h) - Y_i^L(h)), \qquad \frac{\partial e_i^U(h)}{\partial Y_i^U(h)} = -(A_{i0}^U(h) - Y_i^U(h)),$$

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

Therefore, the Eq. (4.22) and the above formulas imply the following Eq:

$$\begin{aligned}\triangle w_{i1}(t) = \eta \textstyle\sum_h h[(A_{i0}^L(h) - Y_i^L(h)).w'_{i1}.O_{i1}^L(h) + \\ (A_{i0}^U(h) - Y_i^U(h)).w'_{i1}.O_{i1}^U(h)] + \alpha. \triangle w_{i1}(t-1),\end{aligned} \tag{4.23}$$

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

$$\begin{aligned}\triangle w_{i1}(t) = \eta \textstyle\sum_h h[(A_{i0}^L(h) - Y_i^L(h)).w'_{i1}.O_{i1}^U(h) + \\ (A_{i0}^U(h) - Y_i^U(h)).w'_{i1}.O_{i1}^L(h)] + \alpha. \triangle w_{i1}(t-1).\end{aligned} \tag{4.24}$$

We can adjust other weights w_{ij} by

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

We can also obtain similarly alternative cases ($w'_{i1} \geq 0$, $w_{i1} < 0$ 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 (4.15), 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 $w'_{i1} \geq 0$. For computing the complexity of our algorithm, we consider only the multiplication operator. The number of multiplications in computing Y_i where

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

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

$$[(A_{i0}^L(h) - Y_i^L(h))O_{i1}'^L(h) + (A_{i0}^U(h) - Y_i^U(h))O_{i1}'^U(h)]$$

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

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

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.

4.4 General system of fuzzy polynomials

The system (4.15) 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:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= A_{10}, \\ &\vdots \\ f_l(x_1, x_2, \ldots, x_n) &= A_{l0}, \\ &\vdots \\ f_s(x_1, x_2, \ldots, x_n) &= A_{s0}, \end{aligned} \tag{4.25}$$

where

$$\begin{aligned} f_l(x_1, x_2, \ldots, x_n) = A_{l0} = \textstyle\sum_{i=1}^{n} C_{li}x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} C_{lij}x_ix_j + \\ \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} C_{lijk}x_ix_jx_k + \ldots, \qquad 1 \le l \le s, \end{aligned} \tag{4.26}$$

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

$$f_l(x_i, x_j) = A_{l0} = A_{l1}x_i + A_{l2}x_j + A_{l3}x_ix_j + A_{l4}x_i^2 + A_{l5}x_j^2. \tag{4.27}$$

Let us build a FNN_2 equivalent to the quadratic one. A FNN_2 [21] solution to Eq.(4.27) is visualized in Figure 4.3.

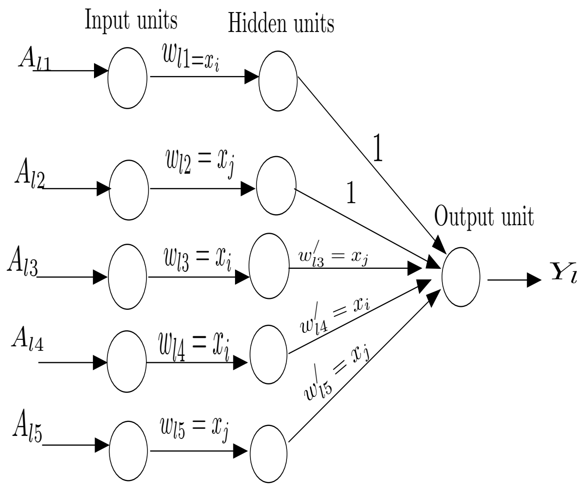

Figure 4.3: Fuzzy neural network for solving a system of the quadratic general fuzzy polynomials

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 unknowns x_i, x_j in Eq.(4.27) 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 Eq.(4.25) is complicated, we consider Eq.(4.27) and propose a learning algorithm from the cost function to adjust crisp weights.

4.4.1 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

$$[A_{l0}]_h = [[A_{l0}]_h^L, [A_{l0}]_h^U], \quad h \in [0,1], \tag{4.28}$$

where $A_{l0}^L(h)$ is the left-hand side and $A_{l0}^U(h)$ 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:

$$\begin{aligned} e_l^L(h) &= \tfrac{1}{2}(A_{l0}^L(h) - Y_l^L(h))^2, \\ e_l^U(h) &= \tfrac{1}{2}(A_{l0}^U(h) - Y_l^U(h))^2, \\ e_l(h) &= e_l^L(h) + e_l^U(h), \end{aligned} \tag{4.29}$$

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

$$e = \sum_l \sum_h h e_l(h). \tag{4.30}$$

Theoretically this cost function satisfies the following relation if we use an infinite number of h-level sets in Eq.(4.30)

$$e \longrightarrow 0 \quad if \;\; and \;\; only \;\; if \;\; Y_l \longrightarrow A_{l0}.$$

The weights are updated by the following rules [21, 41]:

$$\triangle w'_{l3}(t) = -\eta \sum_h h \frac{\partial e_l(h)}{\partial w'_{l3}} + \alpha. \triangle w'_{l3}(t-1), \tag{4.31}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq.(4.31) can be written as follows:

$$\frac{\partial e_l(h)}{\partial w'_{l3}} = \frac{\partial e_l^L(h)}{\partial w'_{l3}} + \frac{\partial e_l^U(h)}{\partial w'_{l3}},$$

$$\frac{\partial e_l^L(h)}{\partial w'_{l3}} = \frac{\partial e_l^L(h)}{\partial Y_l^L} \times \frac{\partial Y_l^L(h)}{\partial w'_{l3}},$$

$$\frac{\partial e_l^U(h)}{\partial w'_{l3}} = \frac{\partial e_l^U(h)}{\partial Y_l^U} \times \frac{\partial Y_l^U(h)}{\partial w'_{l3}},$$

$$\frac{\partial e_l^L(h)}{\partial Y_l^L} = -(A_{l0}^L(h) - Y_l^L(h)), \qquad \frac{\partial e_l^U(h)}{\partial Y_l^U} = -(A_{l0}^U(h) - Y_l^U(h)).$$

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

$$\frac{\partial Y_l^L(h)}{\partial w'_{l3}} = O'^L_{l3}(h), \qquad \frac{\partial Y_l^U(h)}{\partial w'_{l3}} = O'^U_{l3}(h);$$

otherwise

$$\frac{\partial Y_l^L(h)}{\partial w'_{l3}} = O'^U_{l3}(h), \qquad \frac{\partial Y_l^U(h)}{\partial w'_{l3}} = O'^L_{l3}(h).$$

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

$$\begin{aligned}\triangle w'_{l3}(t) = \eta \textstyle\sum_h h[(A_{l0}^L(h) - Y_l^L(h))O'^L_{l3}(h) + \\ (A_{l0}^U(h) - Y_l^U(h))O'^U_{l3}(h)] + \alpha. \triangle w'_{l3}(t-1);\end{aligned} \tag{4.32}$$

otherwise

$$\begin{aligned}\triangle w'_{l3}(t) = \eta \textstyle\sum_h h[(A_{l0}^L(h) - Y_l^L(h))O'^U_{l3}(h) + \\ (A_{l0}^U(h) - Y_l^U(h))O'^L_{l3}(h)] + \alpha. \triangle w'_{l3}(t-1).\end{aligned} \tag{4.33}$$

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 [21, 41]:

$$\triangle w_{l3}(t) = -\eta \sum_h h \frac{\partial e_l(h)}{\partial w_{l3}} + \alpha. \triangle w_{l3}(t-1), \tag{4.34}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments. The derivatives in Eq.(4.34) can be written as follows:

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

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

$$\frac{\partial e_l^L(h)}{\partial w_{l3}} = \frac{\partial e_l^L(h)}{\partial Y_l'^L(h)} \times \frac{\partial Y_l'^L(h)}{\partial O'^L_{l3}(h)} \times \frac{\partial O'^L_{l3}(h)}{\partial w_{l3}},$$

$$\frac{\partial e_l^U(h)}{\partial w_{l3}} = \frac{\partial e_l^U(h)}{\partial Y_l'^U(h)} \times \frac{\partial Y_l'^U(h)}{\partial O'^U_{l3}(h)} \times \frac{\partial O'^U_{l3}(h)}{\partial w_{l3}},$$

$$\frac{\partial e_l^L(h)}{\partial Y_l'^L(h)} = -(A^L_{l0}(h) - Y_l'^L(h)), \qquad \frac{\partial e_l^U(h)}{\partial Y_l'^U(h)} = -(A^U_{l0}(h) - Y_l'^U(h)),$$

$$\frac{\partial Y_l'^L(h)}{\partial O'^L_{l3}(h)} = w'_{l3}, \qquad \frac{\partial Y_l'^U(h)}{\partial O'^U_{l3}(h)} = w'_{l3},$$

$$\frac{\partial O'^L_{l3}(h)}{\partial w_{l3}} = O^L_{l3}(h), \qquad \frac{\partial O'^U_{l3}(h)}{\partial w_{l3}} = O^U_{l3}(h).$$

Therefore, we have

$$\begin{aligned} \triangle w_{l3}(t) = \eta \textstyle\sum_h h[(A^L_{l0}(h) - Y_l'^L(h)).w'_{l3}.O^L_{l3}(h) + \\ (A^U_{l0}(h) - Y_l'^U(h)).w'_{l3}.O^U_{l3}(h)] + \alpha. \triangle w_{l3}(t-1), \end{aligned} \tag{4.35}$$

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

$$\begin{aligned} \triangle w_{l3}(t) = \eta \textstyle\sum_h h[(A^L_{l0}(h) - Y_l'^L(h)).w'_{l3}.O^U_{l3}(h) + \\ (A^U_{l0}(h) - Y_l'^U(h)).w'_{l3}.O^L_{l3}(h)] + \alpha. \triangle w_{l3}(t-1). \end{aligned} \tag{4.36}$$

We can also obtain similar alternative cases ($w'_{l3} \geq 0$, $w_{l3} < 0$ 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 4.4.1. In the general case, to find a solution of the system (4.25) in Section 4.4, similar to (4.15) and (4.27), 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 roots of the system of polynomials.

Remark 4.4.2. The problem (4.25) with uncertainty functions f_l for $l = 1, \dots s$, where the coefficients are fuzzy numbers, is a special case of a more general problem

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= A_{10}, \\ &\vdots \\ f_l(x_1, x_2, \dots, x_n) &= A_{l0}, \\ &\vdots \\ f_s(x_1, x_2, \dots, x_n) &= A_{s0}, \end{aligned} \tag{4.37}$$

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

Remark 4.4.3. The problem (4.37) is complicated because we presume that the variables $x_1, x_2, \dots, x_n$ 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 [42].

4.5 Comparison with other methods

This study would not be completed without comparing it with the recent works. Some comparisons are as follows:

- In Chapter 3, 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 work was to extend the system of fuzzy polynomials, to a more general system F of s fuzzy polynomial equations such as

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= A_{10}, \\ &\vdots \\ f_l(x_1, x_2, \ldots, x_n) &= A_{l0}, \\ &\vdots \\ f_s(x_1, x_2, \ldots, x_n) &= A_{s0}, \end{aligned} \tag{4.38}$$

  where $x_1, x_2, \ldots, x_n \in \mathbb{R}$ and all coefficients are fuzzy numbers. In this work, 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, $Ax = b$ where A is a real matrix, the unknown vector x is a vector consisting of fuzzy numbers and the constant b is a vector whose components are fuzzy numbers. $Ax = b$ is a special case of the (4.38); 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 (4.38) by fuzzy neural network. Moreover the application of fuzzy polynomials was proposed.

- In [1] and [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 4.6.5 for more details.

4.6 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 some examples of fuzzy polynomial systems.

Example 4.6.1. *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 $e \leq 0.01$.

Example 4.6.2. *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 $e \leq 0.01$.

Example 4.6.3. *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 $e <= 0.01$. In this case there is no hope to make the measure of error close to zero. See the Table 4.1 for more details.

	Example 4.6.1	Example 4.6.2	Example 4.6.3
i	$w_{11}(i), w'_{11}(i)$	$w_{11}(i), w'_{11}(i)$	$w_{11}(i), w'_{11}(i)$
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
$\vdots$			$\vdots$

Table 4.1. The approximation of real root

Example 4.6.4. *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:*

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

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:

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

The exact solution is $(x, y) = (2, 1)$. *The training starts with* $w_{11}(0) = 1.9$, $w'_{11}(0) = 0.9$ *and it is completed with* $e <= 0.01$ *where* $\eta = 0.1$ *and* $\alpha = 0.3$.

Example 4.6.5. *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* $w_{11}(4) = 1.0067$, $w'_{11}(4) = -1.0056$, $e \leq 0.01$ *where* $\eta = 0.1$ *and* $\alpha = 0.3$.

Example 4.6.6. *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* $e <= 0.01$ *where* $\eta = 0.1$ *and* $\alpha = 0.3$*.*

4.7 Application of fuzzy neural networks in numerical integration

In this section a numerical method to identify the undetermined coefficients of Newton-Cotes integration method for fuzzy functions by neural networks is considered, so that the fuzzy linear system is solved by using neural network methods.

4.7.1 Identify the undetermined coefficients of Newton-Cotes methods

Let f be a fuzzy function. For any natural number n, the folloowing Newton-Cotes formula is:

$$\int_a^b f(x)dx = \sum_{i=0}^{n} c_i f_i + E, \quad f_i = f(a+ik), \ k = \frac{b-a}{n}. \tag{4.39}$$

Let $f(x) = 1, 1x, 1x^2$ and $a = 0, b = 2k$, where $\tilde{1}$ is a fuzzy number.

$$\begin{aligned}
\tilde{1} \times 2k &= \int_0^{2k} \tilde{1} dx = \tilde{1} \times c_0 + \tilde{1} \times c_1 + \tilde{1} \times c_2, \\
\tilde{1} \times 2k^2 &= \int_0^{2k} \tilde{1} \times x dx = \tilde{1} \times kc_1 + \tilde{1} \times 2kc_2, \\
\tilde{1} \times \tfrac{8}{3}k^3 &= \int_0^{2k} \tilde{1} \times x^2 dx = \tilde{1} \times k^2 c_1 + \tilde{1} \times 4k^2 c_2,
\end{aligned} \tag{4.40}$$

therefore,

$$\begin{pmatrix} \tilde{1} & \tilde{1} & \tilde{1} \\ 0 & \tilde{1} \times k & \tilde{1} \times 2k \\ 0 & \tilde{1} \times k^2 & \tilde{1} \times 4k^2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} \tilde{1} \times 2k \\ \tilde{1} \times 2k^2 \\ \tilde{1} \times \frac{8}{3}k^3 \end{pmatrix}. \tag{4.41}$$

and let $f(x) = \tilde{1}, \tilde{1}x, \tilde{1}x^2, \tilde{1}x^3$ and $a = 0, b = 3k$, where $\tilde{1}$ is a fuzzy number.

$$\begin{aligned}
\tilde{1} \times 3k &= \int_0^{3k} \tilde{1} dx = \tilde{1} \times \acute{c}_0 + \tilde{1} \times \acute{c}_1 + \tilde{1} \times \acute{c}_2 + \tilde{1} \times \acute{c}_3, \\
\tilde{1} \times \tfrac{9}{2}k^2 &= \int_0^{3k} \tilde{1} \times x dx = \tilde{1} \times k\acute{c}_1 + \tilde{1} \times 2k\acute{c}_2 + \tilde{1} \times 3k\acute{c}_3, \\
\tilde{1} \times 9k^3 &= \int_0^{3k} \tilde{1} \times x^2 dx = \tilde{1} \times k^2\acute{c}_1 + \tilde{1} \times 4k^2\acute{c}_2 + \tilde{1} \times 9k^2\acute{c}_3, \\
\tilde{1} \times \tfrac{81}{4}k^4 &= \int_0^{3k} 1 \times x^3 dx = \tilde{1} \times k^3\acute{c}_1 + \tilde{1} \times 8k^3\acute{c}_2 + \tilde{1} \times 27k^3\acute{c}_3,
\end{aligned} \tag{4.42}$$

therefore,

$$\begin{pmatrix} \tilde{1} & \tilde{1} & \tilde{1} & \tilde{1} \\ 0 & \tilde{1} \times k & \tilde{1} \times 2k & \tilde{1} \times 3k \\ 0 & \tilde{1} \times k^2 & \tilde{1} \times 4k^2 & \tilde{1} \times 9k^2 \\ 0 & \tilde{1} \times k^3 & \tilde{1} \times 8k^3 & \tilde{1} \times 27k^3 \end{pmatrix} \begin{pmatrix} \acute{c}_0 \\ \acute{c}_1 \\ \acute{c}_2 \\ \acute{c}_3 \end{pmatrix} = \begin{pmatrix} 1 \times 3k \\ \tilde{1} \times \frac{9}{2}k^2 \\ \tilde{1} \times 9k^3 \\ \tilde{1} \times \frac{81}{4}k^4 \end{pmatrix}. \tag{4.43}$$

Hence, we wish to solve

$$Ax = b \tag{4.44}$$

for a given $m \times m$ fuzzy matrix $A = [a_{ij}]$ where a_{ij} are triangular fuzzy numbers and given $b^t = (b_1, \ldots, b_m)$ an $m \times 1$ vector made up of triangular shaped fuzzy numbers, and $x^t = (x_1, \ldots, x_m)$ is the unknown $m \times 1$ vector of real numbers.

A FNN_2 solution to Eq. (4.44) is given in Figure 4.4. The input to the net is the i-th row of A. The input neurons make no change in their inputs, so the input to the output neuron is

$$a_{i1}x_1 + \ldots + a_{im}x_m, \tag{4.45}$$

which is the i-th component in the product Ax. The output, in the output neuron, equals its input, so

$$Y_i = a_{i1}x_1 + \ldots + a_{im}x_m, \qquad for \quad 1 \leq i \leq m. \tag{4.46}$$

How is the FNN_2 going to solve the fuzzy matrix equation? The training data for input is $(a_{i1}, \ldots, a_{im})$, also for target output is $b_i, 1 \leq i \leq m$.

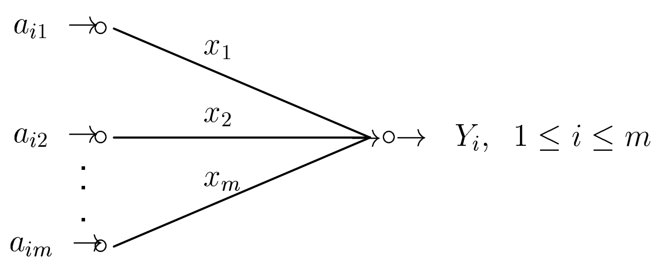

Figure 4.4 : Fuzzy neural net to solve fuzzy equation.

4.7.2 Learning of fuzzy neural network

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

$$[b_i]_h = [[b_i]_h^L, [b_i]_h^U], \qquad h \in [0, 1], \tag{4.47}$$

where $b_i^L(h)$ denotes the left-hand side and $b_i^U(h)$ denotes the right-hand side of the h-level sets of the desired output and let the h-level sets of the fuzzy inputs $\tilde{a}_{ij}$ are nonnegative, i.e,

$$0 \leq [a_{ij}]_h^L \leq [a_{ij}]_h^U, \qquad for \;\; all \;\; i, j.$$

A cost function to be minimized is defined for each h-level sets as follows:

$$\begin{aligned} e_i^L(h) &= \tfrac{1}{2}(b_i^L(h) - Y_i^L(h))^2, \\ e_i^U(h) &= \tfrac{1}{2}(b_i^U(h) - Y_i^U(h))^2, \\ e_i(h) &= e_i^L(h) + e_i^U(h), \end{aligned} \tag{4.48}$$

i.e. $e_i^L(h)$ denotes the error between the left-hand sides of the h-level sets of the desired and the computed outputs, and $e_i^U(h)$ denotes the error between the right-hand sides of the h-level sets of the desired and the computed outputs.

The cost function is:

$$e_i = \sum_h h e_i(h),$$

then error function for the training pattern is

$$e = \sum_{i=1}^{m} e_i. \tag{4.49}$$

Theoretically this cost function satisfies the following relation if we use infinite number of h-level sets in Eq. (4.49)

$$e \longrightarrow 0 \quad if \;\; and \;\; only \;\; if \;\; Y_i \longrightarrow b_i \;\; for \;\; i = 1, \ldots, m.$$

From the cost function $e(h)$ the following learning rules can the derived

$$\triangle x_i(t) = -\eta \sum h \frac{\partial e(h)}{\partial x_i} + \alpha . \triangle x_i(t-1) \qquad for \;\; i = 1, \ldots, m.$$

$$\frac{\partial e(h)}{\partial x_i} = \frac{\partial e^L(h)}{\partial x_i} + \frac{\partial e^U(h)}{\partial x_i}$$

$$\frac{\partial e^L(h)}{\partial x_i} = \frac{\partial e^L(h)}{\partial Y_i^L} \times \frac{\partial Y_i^L(h)}{\partial x_i}$$

$$\frac{\partial e^U(h)}{\partial x_i} = \frac{\partial e^U(h)}{\partial Y_i^U} \times \frac{\partial Y_i^U(h)}{\partial x_i}$$

$$\frac{\partial e^L(h)}{\partial \tilde{Y}_i^L} = -(b_i^L(h) - Y_i^L(h)), \qquad \frac{\partial Y_i^L(h)}{\partial x_i} = a_{ii}^L(h) \quad if \quad x_i \geq 0$$

otherwise

$$\frac{\partial Y_i^L(h)}{\partial x_i} = a_{ii}^U(h),$$

also

$$\frac{\partial e^U(h)}{\partial Y_i^U} = -(b_i^U(h) - Y_i^U(h)), \qquad \frac{\partial Y_i^U(h)}{\partial x_i} = a_{ii}^U(h) \quad if \quad x_i \geq 0$$

otherwise

$$\frac{\partial Y_i^U(h)}{\partial x_i} = a_{ii}^L(h)$$

for all i's.

Thus if $x_i \geq 0$ then

$$\triangle x_i(t) = \eta \sum h[(b_i^L(h) - Y_i^L(h))a_{ii}^L(h) + (b_i^U(h) - Y_i^U(h))a_{ii}^U(h)] + \alpha. \triangle x_i(t-1), \tag{4.50}$$

otherwise

$$\triangle x_i(t) = \eta \sum h[(b_i^L(h) - Y_i^L(h))a_{ii}^U(h) + (b_i^U(h) - Y_i^U(h))a_{ii}^L(h)] + \alpha. \triangle x_i(t-1), \tag{4.51}$$

for all i's.

The fuzzy matrix equation may have no solution for crisp $x_i, 1 \leq i \leq m$. In this case, there is no hope in making the error measure close to zero.

The solution of system (4.41) is:

$$c_0 = \frac{k}{3}, c_1 = \frac{4k}{3}, c_2 = \frac{k}{3}.$$

The training starts by $c_0(0) = k, c_1(0) = k, c_2(0) = k$ and training is completed in step 2 with the solution $c_0(2) = 0.3333k, c_1(2) = 1.3332k, c_2(2) = 0.3332k$. See Table 4.2 for more details.

i	c_0(i)	c_1(i)	c_2(i)
0	k	k	k
1	0.567k	1.215k	0.562k
2	0.3333k	1.3332k	0.3332k

Table 4.2. The obtained values of system (4.41).

The solution of system (4.43) is:

$$\acute{c}_0 = \frac{3k}{8}, \acute{c}_1 = \frac{9k}{8}, \acute{c}_2 = \frac{9k}{8}, \acute{c}_3 = \frac{3k}{8}.$$

The training starts by $\acute{c}_0(0) = \frac{3k}{2}, \acute{c}_1(0) = \frac{7k}{2}, \acute{c}_2(0) = \frac{7k}{2}, \acute{c}_3(0) = \frac{3k}{2}$ and training is completed in step 3 with the solution $\acute{c}_0(3) = 0.3752k, \acute{c}_1(3) = 1.12506k, \acute{c}_2(3) = 1.12507k, \acute{c}_3(3) = 0.37501k$, with $e \leq 0.01$. See Table 4.3 for more details.

i	$\acute{c}_0$(i)	$\acute{c}_1$(i)	$\acute{c}_2$(i)	$\acute{c}_3$(i)
0	$\frac{3k}{2}$	$\frac{7k}{2}$	$\frac{7k}{2}$	$\frac{3k}{2}$
1	1.232k	2.874k	2.876k	1.245k
2	0.734k	1.732k	1.735k	0.736k
3	0.3752k	1.12506k	1.12507k	0.37501k

Table 4.3. The obtained values of system (4.43).

Learning algorithm

Step 0: Initialize the weights.

Step 1: Repeat Step 2 for $h = h_1, h_2, \ldots, h_n$.

Step 2: Repeat the following procedures for $p = 1, 2, \ldots, m$.

(1) Forward calculation: Calculate the h-level set of the fuzzy output vector and corresponding to the fuzzy input vector.

(2) Back-propagation: Adjust the weights using the cost function e.

Step 3: If a prespecified stopping condition is not satisfied, go to Step 1.

Conclusions

In this work, we built a FNN_2 equivalent to the fuzzy polynomials, dual fuzzy polynomials and system of fuzzy polynomials. We introduced a learning algorithm of crisp weights of two and 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 fuzzy polynomials, dual fuzzy polynomials and a system of fuzzy polynomials. The unknowns were 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 degree 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 work, 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 the tools needed to handle more general fuzzy sets.

Bibliography

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

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

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

[4] S. Abbasbandy and B. Asady, Ranking of fuzzy numbers by sign distance, Inform. Sci. 176 (2006) 2405-2416.

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

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

[7] T. Allahviranloo, Newton-Cotes methods for integration of fuzzy functions, Appl. Math. Comput. 166 (2005) 339-348.

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

[9] S. M. Baas and H. Kwakernaak, Rating and ranking of multiple-aspect alternatives using fuzzy sets. Automatica 13 (1978) 47-58.

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

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

[12] P.A. Castillo, J.J. Merelo, M.G. Arenas and G. Romero, Comparing evolutionary hybrid systems for design and optimization of multilayer perceptron structure along training parameters, Inform. Sci. 177 (2007) 2884-2905.

[13] D.Dubois and H. Prade, Fuzzy sets and systems theory and applications, (Mathematics in science and engineering a series of monographs and textbooks, Academic press, INC. 1980).

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

[15] M. Friedman, M. Ming and A. Kandel, Fuzzy linear systems, Fuzzy Sets and Systems, 96 (1998) 201-209.

[16] R. Fuller, Neural fuzzy systems, (1995) ISBN 951-650-624-0, ISSN 0358-5654 .

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

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

[19] Y.C. Hu, Fuzzy integral-based perceptron for two-class pattern classification problems, Inform. Sci. 177 (2007) 1673-1686.

[20] E. Hüllermeier, Numerical methods for fuzzy initial value problems, *International Journal of Uncertainty, Fuzziness and Knowledge-Base Systems* 7 (1999) 439-461.

[21] 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.

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

[23] H. Ishibuchi, H.Okada and H. Tanaka, Interpolation of fuzzy IF-THEN rules by neural networks, International Journal of Approximate Reasoning 10 (1994) 3-27.

[24] H. Ishibuchi and H. Tanaka, Approximate pattern classification using neural networks, in: R. Lowen and M. Roubens eds., Fuzzy Logic: State of the Art (Kluwer, Dordrecht, 1993) 225-236.

[25] R. Jain, Tolerance analysis using fuzzy sets. J. Syst. Sci. 7 (1976) 1393-1401.

[26] O. Kaleva, Fuzzy deifferential equations, Fuzzy Sets and Systems 24 (1987) 301-317.

[27] D. Kincaid, W. Cheny, Numerical analysis, mathematics of scientific computation.

[28] H. M. Lee and W. T. Wang, A neural network architecture for classification of fuzzy inputs, Fuzzy Sets and Systems 63 (1994) 159-173.

[29] P. Liu, Analysis of approximation of continuous fuzzy functions by multivariate fuzzy polynomials, Fuzzy Sets and Systems 127 (2002) 299-313.

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

[31] M. Matloka, On fuzzy integrals, Proc. 2nd polish symp. On Interval and Fuzzy mathematics, Polite Chnika Poznansk (1987) 167-170.

[32] 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.

[33] M. Mizumoto and K. Tanaka, Algebraic properties of fuzzy numbers. Int. Conf. Cybern. Soc., Washington, D.C. (1976).

[34] M. Mizumoto and K. Tanaka, The four operations of arithmetic on fuzzy numbers(II) System Computatin Controls 8 (1977)123-133.

[35] S. Nahmias, Fuzzy variables, Fuzzy Sets and Systems 12 (1978) 97-111.

[36] Juan J. Nieto and R. Rodriguez-Lopez, Existence of extremal solutions for quadratic fuzzy equations, *Fixed Point Theory and Applications*, 72 (2005) 117-128.

[37] H. T. Nguyen, A note on the extension principle for fuzzy sets, *Journal of Mathematical Analysis and Applications* 64 (1978) 369-380.

[38] S.K. Oh and W. Pedrycz, The design of self-organizing polynomial neural networks, Inform. Sci. 141 (2002) 237-258.

[39] 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) 3490-3519.

[40] M. L. Puri, D. Ralescu, Fuzzy random variables, J. Math. Anal. Appl. 114 (1986) 409-422.

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

[42] L.M. San José-Revuelta, A new adaptive genetic algorithm for fixed channel assignment, Inform. Sci. 177 (2007) 2655-2678.

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

[44] J. R. Sims, Z. Y. Wang, Fuzzy measures and fuzzy integrals: an overview, Internat. J. Gen. Sestems 17 (1990) 157-189.

[45] M. Sugeno, Theory of fuzzy integrals and its applications, Ph.D Tesis, Tokyo Institute of Technology, 1974.

[46] C. C. Wang and C. F. Tsai, Fuzzy processing using polynomial bidirdectional hetero-associative network, Inform. Sci. 125 (2000) 167-179.

[47] 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) 3074-3098.

[48] 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.

[49] L. A. Zadeh, A fuzzy set theoretic interpretation of linguistic hedges, Inform. Sci. 3 (1972) 4-34.

[50] L. A. Zadeh, Fuzzy sets, Informatiuon Control, 8 (1965) 338-353.

[51] L. A. Zadeh, A Theory of Approximate Reasoning (AR), Memo UCB/ERL, M77/58. Univ. of California, Berkeley, 9 (1977) 149-194.

[52] L.A. Zadeh, Toward a generalized theory of uncertainty (GTU)an outline, Inform. Sci. 172 (2005) 1-40.

[53] H. J. Zimmerman, Fuzzy Set Theory and its applications, Kluwer Academic, New York, 1996.

[54] J. M. Zurada, Introduction to artificial neural systems (West publishing company, New York, 1992).