Solving fuzzy polynomials using neural nets with a new learning algorithm

Jafarian, M.S.A. Measoomy Nia
Department of Mathematics, Ourmia Branch, Islamic Azad University, Ourmia, Iran

Abstract: This paper mainly intends to offer a new method for finding a real solution of fuzzy polynomials (if exist) of the form
\[ A_n x^n + A_{n-1} x^{n-1} + \cdots + A_1 x + A_0 = 0 \]
and fuzzy target output \( A_0 \), that based on the use of fuzzified feed-forward neural networks. The input-output relation of each unit of the fuzzy neural network is defined by the extension principle of Zadeh. In this paper we define a cost function for the level sets of fuzzy output and fuzzy target and derivation a learning algorithm from the cost function for adjusting of crisp weights.

Key words: Fuzzy polynomials; Fuzzy feed-forward neural networks (FFNN); Cost function

INTRODUCTION

Recently various approaches for solving fuzzy polynomials have been proposed. One approach to indirect solution is using fuzzy neural networks (FNNs). (Ishibuchi et al., 1995) defined a cost function for every pair of fuzzy output vector and its corresponding fuzzy target vector and proposition of a learning algorithm of fuzzy neural networks with triangular and trapezoidal fuzzy weights. (Hayashi et al., 1993) also fuzzified the delta rule and summarized much of the work in fuzzy neural networks. Linear and nonlinear fuzzy equations have been solved in (Abbaspard and Asady, 2004; Abbaspard and Alavi, 2005; Abbaspard and Ezzati, 2006; Asady et al., 2005; Buckley and Qu, 1990). (Buckley and Eslami, 1997) employed neural nets to solve fuzzy problems with both real and complex fuzzy numbers. Also (Abbaspard and Otadi, 2006) has proposed an architecture of feed-forward fuzzy neural network for finding solution to fuzzy polynomials. The input-output relation of each unit is defined by the extension principle of (Zadeh, 1975). In this paper, we proposed a learning algorithm for training fuzzified feed-forward neural network with the identity activation function. This algorithm enables the fuzzified neural network to approximate a crisp solution of fuzzy polynomial to any desired degree of accuracy. In the following, in section 2, we briefly presented the necessary preliminaries for defining the fuzzy number. Section 3 describes how to find a real solution of the fuzzy polynomials by using FFNs. Finally, some examples are collected in Section 4.

Preliminaries:
In this section the most basic used notations in fuzzy calculus are briefly introduced. In addition, the input-output relation of the fuzzified neural nets is presented in this paper. We started by defining the fuzzy number.

Definition 1:
A fuzzy number is a fuzzy set \( u: \mathbb{R} \to [0, 1] \) which satisfies

i. \( u \) is upper semicontinuous,
ii. \( u(x) = 0 \) outside some interval \([a, d]\),
iii. There is a real numbers \( b, c: a \leq b \leq c \leq d \) for which:

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

The set of all fuzzy numbers (as given by definition 1 ) is denoted by \( E^1 \) (Goetschel and Voxman, 1986; Nguyen, 1978).

Definition 2:
A fuzzy number \( u \) is a pair \((\underline{u}, \overline{u})\) with functions \( \underline{u}(r), \overline{u}(r): 0 \leq r \leq 1 \), which satisfy the following requirements:

i. \( \underline{u}(r) \) is a bounded monotonic increasing left continuous function,
ii. \( \overline{u}(r) \) is a bounded monotonic decreasing left continuous function,
iii. \( \underline{u}(r) \leq \overline{u}(r), 0 \leq r \leq 1 \).

A popular fuzzy number is the triangular fuzzy number \( u = (a, b, c) \) with membership function,
where \( a \leq b \leq c \). Its parametric form is:
\[
\mu_a(x) = \begin{cases} 
  x - a & a \leq x \leq b \\
  b - a & b \leq x \leq c \\
  0 & \text{otherwise,}
\end{cases}
\]

**Operation on fuzzy numbers:**

We briefly mentioned fuzzy number operations that had been defined by the extension principle (Zadeh, 2005; Zadeh, 1975).

\[
\mu_{A+B}(z) = \max \left\{ \mu_A(x) \land \mu_B(y) \mid z = x + y \right\},
\]

\[
\mu_{f(\text{Net})}(z) = \max \left\{ \mu_A(x) \land \mu_B(y) \mid z = xy \right\},
\]

where \( A \) and \( B \) are fuzzy numbers, \( \mu(\cdot) \) denotes the membership function of each fuzzy number, \( \land \) is the minimum operator and \( f(x) = x \) is the activation function of output unit of our fuzzy neural network.

The above operations on fuzzy numbers are numerically performed on level sets (i.e. \( \alpha \)-cuts). For \( 0 \leq \alpha \leq 1 \), a \( \alpha \)-level set of a fuzzy number \( A \) is defined as:

\[
[A]^\alpha = \{ x \mid \mu_A(x) \geq \alpha, \ x \in \mathbb{R} \},
\]

\[
[A]^{\alpha_L}[A]^{\alpha_U} = \left[ [A]^{\alpha_L}, [A]^{\alpha_U} \right],
\]

where \( [A]^{\alpha_L} \) and \( [A]^{\alpha_U} \) are the lower and the upper limits of the \( \alpha \)-level set \( [A]^\alpha \), respectively.

From interval arithmetic (Alefeld and Herzberger, 1983), the above operations on fuzzy numbers are written for the \( \alpha \)-level sets as follows:

\[
\begin{align*}
f([\text{Net}]) &= f([\text{Net}]) = f([\text{Net}]) = \left[ f([\text{Net}]) \right] = \left[ f([\text{Net}]) \right], \\
k[A]^\alpha &= k[[A]^\alpha, [A]^\alpha] = [k[A]^\alpha, k[A]^\alpha], \text{ if } k \geq 0, \\
k[A]^\alpha &= k[[A]^\alpha, [A]^\alpha] = [k[A]^\alpha, k[A]^\alpha], \text{ if } k < 0.
\end{align*}
\]

For arbitrary \( u = (\underline{u}, \overline{u}) \) and \( v = (\underline{v}, \overline{v}) \) we define addition \((u + v)\) and multiplication by \( k \) as (Goetschel and Voxman, 1986; Nguyen, 1978):

\[
\begin{align*}
(u + v)(r) &= \overline{u}(r) + \overline{v}(r), \\
(u + v)(r) &= \underline{u}(r) + \underline{v}(r), \\
(ku)(r) &= k \underline{u}(r), ku(r) = k \underline{v}(r), \text{ if } k \geq 0, \\
(ku)(r) &= k \overline{u}(r), ku(r) = k \overline{v}(r), \text{ if } k < 0.
\end{align*}
\]

**Input-output relation of each unit:**

We have given a short review on learning of fuzzified feed-forward neural networks with fuzzy set input signals and real number weights (FFNN2) (Ishibuchi et al., 1995). Consider a two layer FFNN2 with \( n \) input neurons and one output neuron. In this paper we assumed that input vector, target output are triangular fuzzy numbers and weights are crisp. When a fuzzy input vector \( A = (A_1, A_2, \ldots, A_n) \) is presented to our FFNN2, then the input-output relation of each unit can be written as follows (see Fig. 1):
**Input units:**

The input neurons make no change in their inputs, so:

\[ O_i = A_i, \ i = 1, 2, \ldots, n. \]  \hspace{1cm} (3)

**Output unit:**

\[ Y = f(\text{Net}), \]

\[ \text{Net} = \sum_{j=1}^{n} w_j O_j, \]  \hspace{1cm} (4)

where \( A_1, \ldots, A_n \) are triangular fuzzy numbers and \( w_j \) is crisp weight. The relations between input neurons and output neuron in Eqs. (3)-(4) are defined by the extension principle (Zadeh, 1975) as in (Hayashi et al., 1993; Ishibuchi et al., 1995).

![Neural Network Diagram](image)

**Fig. 1:** The proposed neural network

**Calculation of fuzzy output:**

The fuzzy output from a neuron in the second layer is numerically calculated for crisp weights and level sets of fuzzy inputs. The input-output relations of our fuzzy neural network that as shown in Fig. 1 can be written for the \( \alpha \)-level sets as follows:

\[ [O_i]^{\alpha} = [A_i]^{\alpha}, i = 1, \ldots, n. \]  \hspace{1cm} (5)

**Output units:**

\[ [Y]^{\alpha} = f([\text{Net}]^{\alpha}), \]

\[ [\text{Net}]^{\alpha} = \sum_{j=1}^{n} w_j [O_j]^{\alpha}. \]  \hspace{1cm} (6)

From Eqs. (5)-(6), we can see that the \( \alpha \)-level sets of the fuzzy output \( Y \) are calculated from those of the fuzzy inputs and crisp weights. From Eqs. (1)-(2), the above relations are written as follows:

**Input units:**

\[ [O_i]^{\alpha} = [O_i]^{\alpha}, [O_j]^{\alpha} = [A_i]^{\alpha}, [A_j]^{\alpha}, i = 1, \ldots, n. \]

**Output unit:**

\[ [Y]^{\alpha} = [Y]^{\alpha}, [Y]^{\alpha} = f([\text{Net}]^{\alpha}, f([\text{Net}]^{\alpha})), \]

\[ [\text{Net}]^{\alpha} = [\text{Net}]^{\alpha}, [\text{Net}]^{\alpha}, \]

\[ [\text{Net}]^{\alpha} = \sum_{j=1}^{n} w_j [O_j]^{\alpha} + \sum_{j=1}^{n} w_j [O_j]^{\alpha}, \]

\[ [\text{Net}]^{\alpha} = \sum_{j=1}^{n} w_j [O_j]^{\alpha} + \sum_{j=1}^{n} w_j [O_j]^{\alpha}. \]
where $M = \{ j \mid w_j \geq 0 \}$, $C = \{ j \mid w_j < 0 \}$ and $M \cup C = \{ 1, \ldots, n \}$.

**Fuzzy polynomial:**

In this section we concentrated on solving fuzzy polynomial (if exist)

$$A_0 x + \ldots + A_n x^n = A_0,$$  \hspace{1cm} (8)

for $x \in R$ when $A_i \in E_i$ ($i = 0, \ldots, n$). The fuzzy numbers $A_i$ will always be real triangular fuzzy numbers. An architecture of FFNN2 (Ishibuchi et al., 1995) solution to Eq. (8) is given in Fig. 1. The modeling scheme is designed with the simple and versatile fuzzy neural network architecture (Ishibuchi et al., 1995). In above we explained the structure and the behavior of FFNN2.

**Cost function:**

Now let $A_0$ be the target output corresponding to the fuzzy input vector $A = (A_1, \ldots, A_n)$. We introduced how to deduce the learning algorithm. We defined a cost function for $\alpha$–level sets of the fuzzy output $Y$ and the corresponding target output $A_0$:

$$e^\alpha = e_\ell^\alpha + e_u^\alpha,$$  \hspace{1cm} (9)

Where

$$e_\ell^\alpha = \alpha \cdot \frac{(\langle A_0 \rangle_\ell - \langle Y \rangle_\ell)^2}{2},$$  \hspace{1cm} (10)

$$e_u^\alpha = \alpha \cdot \frac{(\langle A_0 \rangle_u - \langle Y \rangle_u)^2}{2}.$$  \hspace{1cm} (11)

In the cost function (9), $e_\ell^\alpha$ and $e_u^\alpha$ can be viewed as the squared errors for the lower limits and the upper limits of the $\alpha$–level sets of the fuzzy output $Y$ and target output $A_0$, respectively. Then the cost function for the input-output pair $\{A; A_0\}$ is obtained as (Ishibuchi et al., 1995).

$$e = \sum_\alpha e^\alpha.$$  \hspace{1cm} (12)

**Learning algorithm of the FFNN2:**

Let a real quantity $x_0$ is initialized at random value for variable $x$. We want to update the crisp weights $w_j$ (for $j = 1, \ldots, n$) such that $w_j = x^j$. We adjusted the parameter $x_0$ and then updated weights $w_j$ (for $j = 1, \ldots, n$) using $x_0$. For crisp parameter $x_0$, adjust rule can be written as follows:

$$x_0(t + 1) = x_0(t) + \Delta x_0(t),$$  \hspace{1cm} (13)

$$\Delta x_0(t) = -\eta \frac{\partial e^\alpha}{\partial x_0} + \gamma \Delta x_0(t - 1),$$  \hspace{1cm} (14)

where $t$ is the number of adjustments, $\eta$ is the learning rate and $\gamma$ is the momentum term constant. Thus our problem is to calculate the derivative $\frac{\partial e^\alpha}{\partial x_0}$ in (14). The derivative $\frac{\partial e^\alpha}{\partial x_0}$ can be calculated from the cost function $e^\alpha$ using the input-output relation of our fuzzy neural network for the $\alpha$–levels sets in (6)-(7). We calculated $\frac{\partial e^\alpha}{\partial x_0}$ as follows:

$$\frac{\partial e^\alpha}{\partial x_0} = \frac{\partial e^\alpha}{\partial x_0} + \frac{\partial e^\alpha}{\partial x_0},$$  \hspace{1cm} (15)
where
\[
\frac{\partial e_i^a}{\partial x_0} = \left( \frac{\partial e_i^a}{\partial [Y]^a_0} \frac{\partial [Y]^a_0}{\partial \text{Net}^a_0} \frac{\partial \text{Net}^a_0}{\partial w_j} \frac{\partial w_j}{\partial x_0} \right) + \ldots \\
+ \left( \frac{\partial e_i^a}{\partial [Y]^a_j} \frac{\partial [Y]^a_j}{\partial w_j} \frac{\partial w_j}{\partial x_0} \right) + \ldots + \left( \frac{\partial e_i^a}{\partial [Y]^a_n} \frac{\partial [Y]^a_n}{\partial w_n} \frac{\partial w_n}{\partial x_0} \right),
\]
and
\[
\frac{\partial e_{iu}^a}{\partial x_0} = \left( \frac{\partial e_{iu}^a}{\partial [Y]^a_0} \frac{\partial [Y]^a_0}{\partial \text{Net}^a_0} \frac{\partial \text{Net}^a_0}{\partial w_j} \frac{\partial w_j}{\partial x_0} \right) + \ldots \\
+ \left( \frac{\partial e_{iu}^a}{\partial [Y]^a_j} \frac{\partial [Y]^a_j}{\partial w_j} \frac{\partial w_j}{\partial x_0} \right) + \ldots + \left( \frac{\partial e_{iu}^a}{\partial [Y]^a_n} \frac{\partial [Y]^a_n}{\partial w_n} \frac{\partial w_n}{\partial x_0} \right),
\]
if \(w_j \geq 0\)
\[
\frac{\partial e_i^a}{\partial w_j} = -\alpha([A]^a_i - [Y]^a_i) [A]^a_j \quad \text{and} \quad \frac{\partial e_{iu}^a}{\partial w_j} = -\alpha([A]^a_i - [Y]^a_i) [A]^a_j, j = 1, \ldots, n,
\]
otherwise
\[
\frac{\partial e_i^a}{\partial w_j} = -\alpha([A]^a_i - [Y]^a_i) [A]^a_j \quad \text{and} \quad \frac{\partial e_{iu}^a}{\partial w_j} = -\alpha([A]^a_i - [Y]^a_i) [A]^a_j, j = 1, \ldots, n.
\]
Consequently
\[
\Delta x_0(t) = \eta \alpha \sum_{j \in M} \left\{ (((A)^a_0 - [Y]^a_0) [A]^a_j] + ([A]^a_0 - [Y]^a_0) [A]^a_j \cdot (j(x_0(t)^{i-1})) \right\} + \\
\eta \alpha \sum_{j \in C} \left\{ (((A)^a_0 - [Y]^a_0) [A]^a_j] + ([A]^a_0 - [Y]^a_0) [A]^a_j \cdot (j(x_0(t)^{i-1})) \right\} + \gamma \Delta x_0(t-1),
\]
where \(M = \{ j \mid w_j \geq 0 \} \) and \(C = \{ j \mid w_j < 0 \} \).

After adjusting \(x_0\) by (13)-(14), connection weights \(w_j\) for \(j = 1, \ldots, n\) are updated with the FFNN2 model as follows:
\[
w_j(t + 1) = f_j(x_0(t + 1)), j = 1, \ldots, n.
\]

Let us assume that input-output pair \(\{A; A_0\}\) where \(A = (A_1, \ldots, A_n)\) are given as training data and also \(m\) values of \(\alpha\) - level sets (\(\alpha_1, \alpha_2, \ldots, \alpha_m\)) are used for learning of the fuzzy neural network. Then the learning algorithm can be summarized as follows:

**Learning algorithm:**

**Step 1:** \(\eta > 0, \gamma > 0, \text{Emax} > 0\) are chosen. Then crisp quantity \(x_0\) is initialized at random value.

**Step 2:** Let \(t := 0\) where \(t\) is the number of iterations of the learning algorithm. Then the running error \(E\) is set to 0.

**Step 3:** Calculate the crisp connection weights as follows,
\[
w_j(t) = (x_0)^j, i = 1, \ldots, n.
\]

**Step 4:** Let \(t := t + 1\). Repeat step 5 for \(\alpha = \alpha_1, \ldots, \alpha_m\)

**Step 5:**

i. **Forward calculation:** Calculate the \(\alpha\) - level set of the fuzzy output \(Y\) by presenting the \(\alpha\) - level set of the fuzzy input vector \(A\).

ii. **Back-propagation:** Adjust crisp parameter \(x_0\) using the cost function (9) for the \(\alpha\) -level sets of the fuzzy output \(Y\) and the target output \(A_0\).
Then update crisp connection weights as has been described in above.

**Step 6:** Cumulative cycle error is computed by adding the present error to $E$.

**Step 7:** The training cycle is completed. For $E < E_{\text{max}}$ terminate the training session. If $E > E_{\text{max}}$ then $E$ is set to 0 and we initiate a new training cycle by going back to **Step 4**.

**Numerical examples:**

To show the behavior and properties of this method, two examples have been solved in this section. For each example, the computed values of the approximate solution are calculated over a number of iterations, in addition the cost function is plotted over a number of iterations.

**Example 1:**

Consider the following fuzzy equation:

\[(1,2,3)x + (-1,0,2)x^2 + (2,3,5)x^3 = (2,5,10),\]

where $x \in \mathbb{R}$ and the exact solution is $x = 1$. In this example, we applied the proposed method to approximate the solution of this fuzzy equation. We use this training data,

\[\{A; A_b\} = \{(1,2,3), (-1,0,2), (2,3,5); (2,5,10)\}.

We trained $\text{FFNN2}$ with three input unit and single output. The training starts by $x_0 = 0.1$, $\eta = 0.001$ and $\gamma = 0.001$. Table 1 shows the approximated solution over a number of iterations and Fig. 2 shows the accuracy of the solution $x_0(t)$ where $t$ is the number of iterations.

**Table 1:** The approximated solutions with error analysis for Example 1.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x_0(t)$</th>
<th>$e$</th>
<th>$t$</th>
<th>$x_0(t)$</th>
<th>$e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.23800</td>
<td>12917.0469</td>
<td>7</td>
<td>0.99396</td>
<td>0.37516681000000</td>
</tr>
<tr>
<td>2</td>
<td>0.45373</td>
<td>286.587181</td>
<td>8</td>
<td>0.88918</td>
<td>0.0355284856000000</td>
</tr>
<tr>
<td>3</td>
<td>0.66676</td>
<td>189.884038</td>
<td>9</td>
<td>0.99946</td>
<td>0.0032131652400000</td>
</tr>
<tr>
<td>4</td>
<td>0.83993</td>
<td>86.7793470</td>
<td>10</td>
<td>0.99984</td>
<td>0.0002848850600000</td>
</tr>
<tr>
<td>5</td>
<td>0.93952</td>
<td>22.7695007</td>
<td>11</td>
<td>0.99995</td>
<td>0.0000254328900000</td>
</tr>
<tr>
<td>6</td>
<td>0.98029</td>
<td>3.45294641</td>
<td>12</td>
<td>0.99999</td>
<td>0.00000225515224</td>
</tr>
</tbody>
</table>

**Fig. 2:** The cost function for Example 1 on the number of iterations.

**Example 2:**

Let fuzzy equation

\[(-1,0,2)x + (0,1,3)x^2 + (1,3,4)x^3 = (-6,2,3),\]

with the exact solution $x = -1$. We trained the fuzzy neural network as described in last example. Before starting calculations, we assumed that $x_0 = -2.5$, $\eta = 0.001$ and $\gamma = 0.001$. Numerical result can be found in table 2. Fig. 3 shows the accuracy of the solution $x_0(t)$ where $t$ is the number of iterations.
Table 2: The approximated solutions with error analysis for Example 2.

<table>
<thead>
<tr>
<th>t</th>
<th>$x_0(t)$</th>
<th>$e$</th>
<th>t</th>
<th>$x_0(t)$</th>
<th>$e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2.0731</td>
<td>7942.1253</td>
<td>50</td>
<td>-1.0007</td>
<td>0.000373303660</td>
</tr>
<tr>
<td>2</td>
<td>-1.8185</td>
<td>2778.8567</td>
<td>51</td>
<td>-1.0006</td>
<td>0.000284124700</td>
</tr>
<tr>
<td>3</td>
<td>-1.6472</td>
<td>1238.4738</td>
<td>52</td>
<td>-1.0005</td>
<td>0.000216258640</td>
</tr>
<tr>
<td>4</td>
<td>-1.5235</td>
<td>849.4381</td>
<td>53</td>
<td>-1.0005</td>
<td>0.000164608940</td>
</tr>
<tr>
<td>5</td>
<td>-1.3578</td>
<td>212.6784</td>
<td>54</td>
<td>-1.0004</td>
<td>0.000125298780</td>
</tr>
<tr>
<td>6</td>
<td>-1.3578</td>
<td>212.6784</td>
<td>55</td>
<td>-1.0003</td>
<td>0.000095378845</td>
</tr>
</tbody>
</table>

Fig. 3: The cost function for Example 2 on the number of iterations.

REFERENCES