Learning by Fuzzified Neural Networks*

Hisao Ishibuchi and Kouichi Morioka
University of Osaka Prefecture, Osaka, Japan
I. B. Turksen
University of Toronto, Toronto, Ontario, Canada

ABSTRACT

We derive a general learning algorithm for training a fuzzified feedforward neural network that has fuzzy inputs, fuzzy targets, and fuzzy connection weights. The derived algorithm is applicable to the learning of fuzzy connection weights with various shapes such as triangular and trapezoid. First we briefly describe how a feedforward neural network can be fuzzified. Inputs, targets, and connection weights in the fuzzified neural network can be fuzzy numbers. Next we define a cost function that measures the difference between a fuzzy target vector and an actual fuzzy output vector. Then we derive a learning algorithm from the cost function for adjusting fuzzy connection weights. Finally we show some results of computer simulations.

KEYWORDS: fuzzification of neural networks, fuzzy inputs, fuzzy targets, fuzzy connection weights, learning algorithm, feedforward neural networks

1. INTRODUCTION

Multilayer feedforward neural networks can be fuzzified by replacing real-number connection weights with fuzzy-number connection weights [1, 2]. A fuzzified neural network can handle fuzzy input vectors as well as real
input vectors. Buckley and Hayashi [3] classified fuzzified neural networks into the following three types:

1. FNN\textsubscript{1} with real number inputs and fuzzy weights: Fuzzified neural networks of this type map a real number input vector to a fuzzy output vector. These neural networks can be used as approximators of fuzzy-number-valued nonlinear functions. They are also applicable in nonlinear fuzzy regression analysis [4].

2. FNN\textsubscript{2} with fuzzy inputs and real number weights: Fuzzified neural networks of this type map a fuzzy input vector to a fuzzy output vector. These neural networks can be applied to the interpolation of fuzzy if-then rules [5, 6], the classification of fuzzy pattern vectors [5], and learning from incomplete training data [5, 7].

3. FNN\textsubscript{3} with fuzzy inputs and fuzzy outputs: Fuzzified neural networks of this type map a fuzzy input vector to a fuzzy output vector. These neural networks can be used for modeling fuzzy expert systems [3] and interpolating fuzzy if-then rules [8].

The capability of fuzzified neural networks as approximators of nonlinear fuzzy mappings was studied by Buckley and Hayashi [9], who showed that fuzzified neural networks of type FNN\textsubscript{3} are not universal approximators. The capabilities of fuzzified neural networks of type FNN\textsubscript{1} and type FNN\textsubscript{2} were examined by Buckley [10, 11].

Several approaches have been proposed for learning by fuzzified neural networks of type FNN\textsubscript{3} (for details, see the survey by Buckley and Hayashi [3]). Hayashi et al. [1] proposed a fuzzy back-propagation algorithm, which can be viewed as a direct fuzzification of the standard back-propagation algorithm [12]. The algorithm was obtained by replacing real numbers used for inputs, outputs, targets, and weights in the standard back-propagation algorithm with fuzzy numbers. It was reported in [3] that the algorithm converged to the wrong weights. Hayashi et al. [1] also discussed a back-propagation algorithm for the individual \(\alpha\)-cuts of fuzzy weights. Because the algorithm independently updates the \(\alpha\)-cuts of a fuzzy weight, one is not sure that the updated fuzzy weight in fact forms a fuzzy set (see [1, 3]). Ishibuchi et al. [8] proposed a \(\alpha\)-cut based backpropagation algorithm, which can be viewed as a back-propagation algorithm for the supports of symmetric triangular fuzzy weights. The algorithm was derived from a cost function defined for the \(\alpha\)-cuts of fuzzy outputs and fuzzy targets. The \(\alpha\)-cut based back-propagation algorithm for symmetric triangular fuzzy weights in [8] was extended to the case of nonsymmetric trapezoid fuzzy weights in [13].

In the above-mentioned studies, back-propagation learning algorithms were proposed for adjusting fuzzy weights. On the other hand, Krishnamraju et al. [14] proposed a genetic learning algorithm where genetic algorithms [15, 16] were employed for adjusting triangular fuzzy weights.
Because genetic algorithms have high optimization ability and high flexibility, the genetic learning algorithm may be applicable to learning by fuzzified neural networks of the above three types. Buckley and Hayashi [3] also discussed the application of fuzzy chaos to learning by fuzzified neural networks.

The main aim of this paper is to derive a back-propagation learning algorithm that can be applied to the learning of fuzzy weights of various shapes such as nonsymmetric triangular and trapezoidal types. The learning algorithm derived in this paper is a generalization of the former work that was applicable only to symmetric triangular fuzzy weights [8] or nonsymmetric trapezoidal fuzzy weights [13]. That is, while the learning algorithms in [8, 13] were derived for fuzzy weights of special shapes, this paper derives a general learning algorithm that is applicable to a fuzzy weight of any shape if its membership function is specified by a finite number of parameters.

In this paper, first we describe an architecture of fuzzified neural networks of type FNN3. Our fuzzified neural networks are three-layer feedforward networks with multiple inputs and multiple outputs. Next we define a cost function that measures the difference between a fuzzy target vector and an actual fuzzy output vector. Then we derive a learning algorithm from the cost function for adjusting fuzzy weights in a similar manner to the standard back-propagation algorithm. Finally we show that the derived learning algorithm can handle fuzzy weights, fuzzy inputs, and fuzzy targets of various shapes by computer simulations.

2. FUZZIFICATION OF NEURAL NETWORKS

In this section, we show how a three-layer feedforward neural network can be fuzzified. The fuzzified neural network has fuzzy weights, fuzzy inputs, and fuzzy targets.

2.1. Standard Feedforward Neural Network

Before describing fuzzified neural networks, we briefly review the standard (i.e., nonfuzzy) feedforward neural networks. Let us consider a three-layer feedforward neural network with \( n_I \) input units, \( n_H \) hidden units, and \( n_O \) output units. When an \( n_I \)-dimensional input vector \( \mathbf{x}_p = (x_{p1}, x_{p2}, \ldots, x_{pn_I}) \) is presented to the neural network, the input-output relation of each unit can be written as follows:

**Input units:**

\[
o_{pi} = x_{pi}, \quad i = 1, 2, \ldots, n_I.
\]
Hidden units:

\[ o_{pj} = f(\text{net}_{pj}), \quad j = 1, 2, \ldots, n_H, \]

\[ \text{net}_{pj} = \sum_{i=1}^{n_i} w_{ji} o_{pi} + \theta_j, \quad j = 1, 2, \ldots, n_H. \]

Output units:

\[ o_{pk} = f(\text{net}_{pk}), \quad k = 1, 2, \ldots, n_O, \]

\[ \text{net}_{pk} = \sum_{j=1}^{n_H} w_{kj} o_{pj} + \theta_k, \quad k = 1, 2, \ldots, n_O. \]

Here \( w_{ji} \) and \( w_{kj} \) are connection weights, \( \theta_j \) and \( \theta_k \) are biases, and \( f(x) = \frac{1}{1 + e^{-x}} \).

Let us denote the target vector corresponding to the input vector \( x_p \) by \( \tau_p = (\tau_{p1}, \tau_{p2}, \ldots, \tau_{p_{no}}) \). Then a cost function to be minimized in the learning of the neural network can be written as

\[ e_p = \sum_{k=1}^{n_O} \frac{(\tau_{pk} - o_{pk})^2}{2}, \]

where \( o_{pk} \) is the actual output from the \( k \)th output unit that is calculated by the input-output relation of the neural network in (1)-(5).

In the back-propagation algorithm [12], the weights \( w_{ji}, w_{kj} \) and the biases \( \theta_j, \theta_k \) are updated to decrease the cost function \( e_p \) in (6). For example, the weight \( w_{ji} \) is changed according to the following rule:

\[ w_{ji}(t + 1) = w_{ji}(t) + \Delta w_{ji}(t), \]

\[ \Delta w_{ji}(t) = -\eta \frac{\partial e_p}{\partial w_{ji}}, \]

where \( t \) indexes the number of adjustments and \( \eta \) is a constant positive real number (e.g., \( \eta = 0.1 \)). The amount of adjustment \( \Delta w_{ji}(t) \) is usually defined by adding a momentum term as

\[ \Delta w_{ji}(t) = -\eta \frac{\partial e_p}{\partial w_{ji}} + \alpha \Delta w_{ji}(t - 1), \]

where \( \alpha \) is a constant positive real number less than 1.0 (e.g., \( \alpha = 0.9 \)). The weight \( w_{kj} \) and the biases \( \theta_j, \theta_k \) are changed in the same manner as in (7)-(9).
2.2. Fuzzified Feedforward Neural Network

The inputs, weights, and biases of the standard feedforward neural network defined by (1)–(5) can be extended to fuzzy numbers. In this paper, the fuzzification of neural networks means this extension. Therefore the fuzzification does not change the neural network architecture. That is, the fuzzified neural network has the same network architecture as the standard neural network in (1)–(5).

Let us denote fuzzy numbers and real numbers by uppercase letters (e.g., $A, B, \ldots$) and lowercase letters (e.g., $a, b, \ldots$), respectively. Then the input-output relation of the fuzzified neural network can be written for a fuzzy input vector $X_p = (X_{p1}, X_{p2}, \ldots, X_{pn_l})$ as follows:

**Input units:**

$$O_{pi} = X_{pi}, \quad i = 1, 2, \ldots, n_l.$$  \hspace{1cm} (10)

**Hidden units:**

$$O_{pj} = f(\text{Net}_{pj}), \quad j = 1, 2, \ldots, n_H,$$
$$\text{Net}_{pj} = \sum_{i=1}^{n_l} W_{ji} O_{pi} + \Theta_j, \quad j = 1, 2, \ldots, n_H.$$  \hspace{1cm} (11, 12)

**Output units:**

$$O_{pk} = f(\text{Net}_{pk}), \quad k = 1, 2, \ldots, n_O,$$
$$\text{Net}_{pk} = \sum_{j=1}^{n_H} W_{kj} O_{pj} + \Theta_k, \quad k = 1, 2, \ldots, n_O.$$  \hspace{1cm} (13, 14)

where $W_{ji}$ and $W_{kj}$ are fuzzy weights, and $\Theta_j$ and $\Theta_k$ are fuzzy biases. The architecture of the fuzzified neural network is shown in Figure 1.

![Figure 1. Architecture of a three-layer feedforward fuzzified neural network.](image-url)
2.3. Calculation of the Input-Output Relation

The input-output relation in (10)-(14) is defined by the extension principle of Zadeh [17]. This means that fuzzy arithmetic (see, for example, Kaufmann and Gupta [18]) is employed for calculating the input-output relation of the fuzzified neural network. In (10)-(14), the following addition, multiplication, and nonlinear mapping of fuzzy numbers are used for defining our fuzzified neural network:

\[
\mu_{A+B}(z) = \max\{\mu_A(x) \land \mu_B(y) | z = x + y\},
\]
\[
\mu_{AB}(z) = \max\{\mu_A(x) \land \mu_B(y) | z = xy\},
\]
\[
\mu_{f(\text{Net})}(z) = \max\{\mu_{\text{Net}}(x) | z = f(x)\},
\]

where \( A, B, \text{Net} \) are fuzzy numbers, \( \mu_\cdot(\cdot) \) denotes the membership function of each fuzzy number, and \( \land \) is the minimum operator. These operations are illustrated in Figure 2 and Figure 3.

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

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

where \( \mu_X(x) \) is the membership function of \( X \), and \( \mathbb{R} \) is the set of all real numbers. Because level sets of fuzzy numbers are closed intervals (see, for example, Kaufmann and Gupta [18]), we can express \([X]_h\) as

\[
[X]_h = [[X]_h^L, [X]_h^U],
\]

where \([X]_h^L\) and \([X]_h^U\) are the lower limit and the upper limit of the \( h \)-level set \([X]_h\), respectively.

![Figure 2. Illustration of fuzzy arithmetic.](image)
From interval arithmetic [19], the above operations of fuzzy numbers in (15)–(17) can be rewritten for \( h \)-level sets as follows:

\[
\begin{align*}
[A]_h + [B]_h &= [(A)_h^L, (A)_h^U] + [(B)_h^L, (B)_h^U] \\
&= [(A)_h^L + (B)_h^L, (A)_h^U + (B)_h^U], \\
[A]_h \cdot [B]_h &= [(A)_h^L, (A)_h^U] \cdot [(B)_h^L, (B)_h^U] \\
&= \left[ \min\{(A)_h^L \cdot (B)_h^L, (A)_h^U \cdot (B)_h^U, (A)_h^L \cdot (B)_h^U, (A)_h^U \cdot (B)_h^U \}, \max\{(A)_h^L \cdot (B)_h^L, (A)_h^U \cdot (B)_h^U, (A)_h^L \cdot (B)_h^U, (A)_h^U \cdot (B)_h^U \} \right],
\end{align*}
\]

(21)

\[
f([\text{Net}]_h) = f([\text{Net}]_h^L, [\text{Net}]_h^U) = \left[ f([\text{Net}]_h^L), f([\text{Net}]_h^U) \right].
\]

(22)

It should be noted that (22) is obtained from (17) because the activation function \( f(x) \) in (17) is a strictly monotonic function. If we do not use such a function, the result of the interval arithmetic (22) may be different from (17) defined by the extension principle.

If the \( h \)-level set of \( B \) is nonnegative (i.e., if \( 0 \leq [B]_h^L \leq [B]_h^U \)), the multiplication in (21) can be simplified as

\[
[A]_h \cdot [B]_h = \left[ \min\{(A)_h^L \cdot (B)_h^L, (A)_h^U \cdot (B)_h^U\}, \max\{(A)_h^L \cdot (B)_h^L, (A)_h^U \cdot (B)_h^U\} \right].
\]

(23)

In order to utilize this simplified formulation of the multiplication, we assume in this paper that the fuzzy input \( X_{pi} \) is nonnegative. The derivation of a learning algorithm without this assumption is possible by using (21), but it is much more complicated, as we can see from the comparison between (21) and (23).
The input-output relation of our fuzzified neural network in (10)–(14) is numerically calculated for the $h$-level sets of fuzzy inputs, fuzzy weights, and fuzzy biases. The input-output relation for the $h$-level sets can be written as

**Input units:**

$$[O_{pi}]_h = [X_{pi}]_h, \quad i = 1, 2, \ldots, n_I.$$  \hfill (24)

**Hidden units:**

$$[O_{pj}]_h = f([\text{Net}_{pj}]_h), \quad j = 1, 2, \ldots, n_H,$$  \hfill (25)

$$[\text{Net}_{pj}]_h = \sum_{i=1}^{n_I} [W_{ji}]_h \cdot [O_{pi}]_h + [\Theta_j]_h, \quad j = 1, 2, \ldots, n_H.$$  \hfill (26)

**Output units:**

$$[O_{pk}]_h = f([\text{Net}_{pk}]_h), \quad k = 1, 2, \ldots, n_O,$$  \hfill (27)

$$[\text{Net}_{pk}]_h = \sum_{j=1}^{n_H} [W_{kj}]_h \cdot [O_{pj}]_h + [\Theta_k]_h, \quad k = 1, 2, \ldots, n_O.$$  \hfill (28)

From (24)–(28), we can see that the $h$-level sets of the fuzzy inputs $X_{pi}$ are mapped to the $h$-level sets of the fuzzy outputs $O_{pk}$. This means that the $h$-level sets of the fuzzy outputs $O_{pk}$ can be calculated by interval arithmetic on the $h$-level sets of the fuzzy inputs, fuzzy weights, and fuzzy biases. The calculation in (24)–(28) is based on the relations in (20)–(23). The complete calculation of the $h$-level sets of the fuzzy outputs $O_{pk}$ is shown in Appendix A.

### 2.4. An Example

As an example of the fuzzified neural network, let us consider a three-layer network with two input units, two hidden units, and a single output unit. The network architecture is shown in Figure 4(a). Using this simple example, we illustrate the input-output relation of the fuzzified neural network defined by (10)–(14).

The fuzzy outputs from the input units are the same as the fuzzy inputs $X_{p1}$ and $X_{p2}$. Therefore the total fuzzy inputs $\text{Net}_{p1}$ and $\text{Net}_{p2}$ to the hidden units are calculated as follows:

$$\text{Net}_{p1} = X_{p1}W_{11} + X_{p2}W_{12} + \Theta_1,$$  \hfill (29)

$$\text{Net}_{p2} = X_{p1}W_{21} + X_{p2}W_{22} + \Theta_2.$$  \hfill (30)
Then the fuzzy outputs $O_{p1}$ and $O_{p2}$ from the hidden units are obtained as

$$O_{p1} = f(Net_{p1}),$$  \hspace{1cm} (31)

$$O_{p2} = f(Net_{p2}).$$  \hspace{1cm} (32)

These two fuzzy outputs are fed to the output unit. The total fuzzy input $Net_p$ to the output unit is

$$Net_p = O_{p1}W_1 + O_{p2}W_2 + \Theta.$$  \hspace{1cm} (33)

Finally the fuzzy output from the output unit (i.e., the fuzzy output from the fuzzified neural network) is calculated as

$$O_p = f(Net_p).$$  \hspace{1cm} (34)

These formulations of the input-output relation of each unit are almost the same as in the standard feedforward neural network. The only difference is that fuzzy arithmetic on fuzzy numbers is used in the fuzzified neural network, while real-number arithmetic is used in the standard neural network.

The numerical calculation in (29)–(34) is performed for the $h$-level sets of fuzzy numbers by using interval arithmetic (see Appendix A). Figure 4(b) shows an example of a fuzzified neural network, fuzzy inputs, and a fuzzy output. In this figure, the neural network has trapezoidal fuzzy weights, trapezoidal fuzzy biases, and triangular fuzzy inputs. The fuzzy output in this figure is depicted from the result of the interval arithmetic for 100 level sets (i.e., $h = 0.01, 0.02, \cdots, 1.00$). This means that 100 level sets of the fuzzy output $O_p$ are calculated for depicting Figure 4(b).

3. LEARNING ALGORITHM

In this section, we derive a general learning algorithm of our fuzzified neural network. The derived learning algorithm can be applied to the adjustment of the fuzzy weights of various shapes.

3.1. Cost Function

When the fuzzy input vector $X_p = (X_{p1}, X_{p2}, \cdots, X_{pn})$ is presented to the fuzzified neural network defined by (10)–(14), the actual output vector is obtained as the fuzzy vector $O_p = (O_{p1}, O_{p2}, \cdots, O_{pno})$. Let us assume that a fuzzy vector $T_p = (T_{p1}, T_{p2}, \cdots, T_{pno})$ is given as the target vector corresponding to the fuzzy input vector $X_p$. That is, we assume that the input-output pair $(X_p, T_p)$ is given for the learning of the fuzzified neural network.
The aim of the learning by the fuzzified neural network is to decrease the difference between \( O_p \) and \( T_p \). That is, it is desired that the following equality hold approximately:

\[
T_{pk} \equiv O_{pk} \quad \text{for} \quad k = 1, 2, \ldots, n_O.
\]  

(35)

A cost function to be minimized in the learning by the fuzzified neural network should measure the difference between the fuzzy target vector \( T_p \) and the fuzzy output vector \( O_p \). First we define a cost function \( e_{pkh} \) for the \( h \)-level sets of \( O_{pk} \) and \( T_{pk} \) as follows:

\[
e_{pkh} = e_{pkh}^L + e_{pkh}^U,
\]

(36)

where

\[
e_{pkh}^L = h \cdot \frac{\left([T_{pk}]_h^L - [O_{pk}]_h^L\right)^2}{2},
\]

(37)

\[
e_{pkh}^U = h \cdot \frac{\left([T_{pk}]_h^U - [O_{pk}]_h^U\right)^2}{2}.
\]

(38)

In the cost function \( e_{pkh} \) in (36), \( e_{pkh}^L \) and \( e_{pkh}^U \) can be viewed as the squared errors for the lower limit and the upper limit of the \( h \)-level sets,
respectively (see Figure 5). Those squared errors are weighted by the value of \( h \) in (37) and (38). Therefore the squared errors for a large value of \( h \) have a large effect on the learning in the fuzzified neural network. The main aim of the introduction of \( h \) in the cost function \( e_{pkh} \) in (36)-(38) is to obtain the good fit of the fuzzy output to the fuzzy target for a large value of \( h \) (see, for example, Figure 16 in Section 4). If we value the fitting for small \( h \) as much as for large \( h \), we can eliminate \( h \) in the definition of the cost function \( e_{pkh} \) in (36)-(38).

The cost function \( e_{pkh} \) for the \( h \)-level set can be summed up over the \( n_O \) output units of the fuzzified neural network as

\[
e_{ph} = \sum_{k=1}^{n_O} e_{pkh}.
\]

The cost function \( e_{ph} \) in (39) can be viewed as the squared error between the \( h \)-level set of the fuzzy target vector \( T_p \) and the fuzzy output vector \( O_p \).

The cost function \( e_p \) that measures the difference between \( T_p \) and \( O_p \) can be defined by using various values of \( h \) in (39) as follows:

\[
e_p = \sum_{h} e_{ph}.
\]

In the computer simulations of this paper, we use five values of \( h \), viz., \( h = 0.2, 0.4, 0.6, 0.8, 1.0 \) in (40).

When \( m \) input-output pairs \((X_p, T_p)\), \( p = 1, 2, \ldots, m \), are given as training data, we have the following cost function for these training data:

\[
e = \sum_{p=1}^{m} e_p.
\]
3.2. Derivation of a Learning Algorithm

The fuzzy weights $W_{ji}$, $W_{kj}$, and the fuzzy biases $\Theta_j$, $\Theta_k$ of the fuzzified neural network are adjusted by using the cost function $e_{ph}$ defined in the last subsection. Let us assume that the $m$ fuzzy input-output pairs $(X_p, T_p)$, $p = 1, 2, \ldots, m$, are given as training data. We also assume that $n$ values of $h$ (viz., $h_1$, $h_2$, $\ldots$, $h_n$) are used in the learning of the fuzzified neural network. This means that the $n$ values of $h$ are employed for measuring the difference between the fuzzy output $O_p$ and the fuzzy target $T_p$ in the cost function $e_p$ defined by (40). In this case, the outline of the learning algorithm of the fuzzified neural network can be written as follows:

1. **Step 0**: Initialize the fuzzy weights and the fuzzy biases. Let $n_{\text{iteration}} = 0$, where $n_{\text{iteration}}$ is the number of iterations (i.e., epochs) of this algorithm.

2. **Step 1**: Let $n_{\text{iteration}} := n_{\text{iteration}} + 1$. Repeat step 2 for $h = h_1, h_2, \ldots, h_n$.

3. **Step 2**: Repeat the following procedures for $p = 1, 2, \ldots, m$.
   1. **Forward calculation**: Present the $h$-level set of the $p$th fuzzy input vector $X_p$ to the fuzzified neural network. Then calculate the $h$-level set of the fuzzy output vector $O_p$.
   2. **Back-propagation**: Adjust the fuzzy weights and the fuzzy biases by using the cost function $e_{ph}$ in (39).

4. **Step 3**: If a prespecified stopping condition is satisfied, then stop the algorithm; else go to step 1. In computer simulations of this paper, we use the total number of iterations of this algorithm as a stopping condition.

In order to implement this algorithm, we should clearly specify the back-propagation procedure in step 2. Let us assume that the fuzzy weights and the fuzzy biases can be denoted by their $s$ parameters as

- $W_{ji} = (w_{ji}^{(1)}, w_{ji}^{(2)}, \ldots, w_{ji}^{(s)})$, 
- $W_{kj} = (w_{kj}^{(1)}, w_{kj}^{(2)}, \ldots, w_{kj}^{(s)})$, 
- $\Theta_j = (\theta_j^{(1)}, \theta_j^{(2)}, \ldots, \theta_j^{(s)})$, 
- $\Theta_k = (\theta_k^{(1)}, \theta_k^{(2)}, \ldots, \theta_k^{(s)})$.

For example, a nonsymmetric trapezoidal fuzzy weight $W_{ji}$ as in Figure 6(a) is denoted by its four parameters as $W_{ji} = (w_{ji}^{(1)}, w_{ji}^{(2)}, w_{ji}^{(3)}, w_{ji}^{(4)})$. A nonsymmetric triangular fuzzy weight $W_{kj}$ as in Figure 6(b) is denoted by its lower limit $w_{kj}^L$, its center $w_{kj}$, and its upper limit $w_{kj}^U$ as $W_{kj} = (w_{kj}^L, w_{kj}, w_{kj}^U)$.
Learning by Fuzzified NNs

Figure 6. Fuzzy weight $W_{kj}$ of various shapes: (a) non-symmetric trapezoidal, (b) non-symmetric triangular, (c) symmetric triangular, (d) interval.

$w^C_{kj}, w^U_{kj}$). That is, a non-symmetric triangular fuzzy weight is denoted by its three parameters. A symmetric triangular fuzzy weight as in Figure 6(c) is denoted by its two parameters. An interval weight as in Figure 6(d) is also denoted by its two parameters.

The back-propagation procedure for the fuzzy weights and the fuzzy biases is applied to their parameters. First we derive an adjustment rule for the parameters of the fuzzy weight $W_{kj} = (w_{kj}^{(1)}, w_{kj}^{(2)}, \ldots, w_{kj}^{(q)}, \ldots, w_{kj}^{(s)})$ between the hidden layer and the output layer. As in the standard back-propagation algorithm in [12], the parameter $w_{kj}^{(q)}$ of the fuzzy weight $W_{kj}$ is adjusted by the following rule:

$$w_{kj}^{(q)}(t + 1) = w_{kj}^{(q)}(t) + \Delta w_{kj}^{(q)}(t),$$  \hspace{1cm} (46)

$$\Delta w_{kj}^{(q)}(t) = -\eta\frac{\partial e_{ph}}{\partial w_{kj}^{(q)}} + \alpha \Delta w_{kj}^{(q)}(t - 1),$$  \hspace{1cm} (47)

where $t$ indexes the number of adjustments, $\eta$ is a constant positive real number (e.g., $\eta = 0.1$), and $\alpha$ is a constant positive real number less than 1.0 (e.g., $\alpha = 0.9$).

The main difference between the standard backpropagation algorithm and our adjustment rule in (46)–(47) is the derivative $\partial e_{ph}/\partial w_{kj}^{(q)}$ in (47). As shown by Rumelhart et al. [12], the standard back-propagation algo-
The algorithm is easily derived from the cost function in (6). The calculation of the derivative $\frac{\partial e_{ph}}{\partial w_{kj}^{(q)}}$ in our adjustment rule in (46)–(47) for the fuzzy weight $W_{kj}$, however, is not so simple. Let us rewrite the derivative $\frac{\partial e_{ph}}{\partial w_{kj}^{(q)}}$ as follows:

$$\frac{\partial e_{ph}}{\partial w_{kj}^{(q)}} = \frac{\partial e_{ph}}{\partial [W_{kj}]_{h}^{L}} \cdot \frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(q)}} + \frac{\partial e_{ph}}{\partial [W_{kj}]_{h}^{U}} \cdot \frac{\partial [W_{kj}]_{h}^{U}}{\partial w_{kj}^{(q)}},$$

where $\frac{\partial e_{ph}}{\partial [W_{kj}]_{h}^{L}}$ and $\frac{\partial e_{ph}}{\partial [W_{kj}]_{h}^{U}}$ are the derivatives with respect to the lower limit and the upper limit of the $h$-level set of the fuzzy weight $W_{kj}$, respectively. It should be noted that these derivatives are independent of the shape of the fuzzy weight. This means that they can be calculated without the specification of a particular shape of the fuzzy weight $W_{kj}$ (for details, see Appendix B).

On the contrary, the derivatives $\frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(q)}}$ and $\frac{\partial [W_{kj}]_{h}^{U}}{\partial w_{kj}^{(q)}}$ depend on the shape of the fuzzy weight $W_{kj}$. Fortunately, these derivatives are easily calculated from the relation between the $h$-level set of the fuzzy weight $W_{kj}$ and its parameters when a particular shape of $W_{kj}$ is given. For example, the $h$-level set $[W_{kj}]_{h} = [W_{kj}]_{h}^{L}, [W_{kj}]_{h}^{U}$ of the nonsymmetric trapezoidal fuzzy weight $W_{kj}$ in Figure 6(a) can be represented by its four parameters as

$$[W_{kj}]_{h}^{L} = (1 - h)w_{kj}^{(1)} + hw_{kj}^{(2)},$$

$$[W_{kj}]_{h}^{U} = hw_{kj}^{(3)} + (1 - h)w_{kj}^{(4)}.$$

Then the derivatives $\frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(q)}}$ and $\frac{\partial [W_{kj}]_{h}^{U}}{\partial w_{kj}^{(q)}}$ are easily calculated for each of the four parameters $w_{kj}^{(1)}, w_{kj}^{(2)}, w_{kj}^{(3)}$, and $w_{kj}^{(4)}$. That is, we have the following for the nonsymmetric trapezoidal fuzzy weight $W_{kj}$ in Figure 6(a):

$$\frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(1)}} = 1 - h, \quad \frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(2)}} = 0,$$

$$\frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(3)}} = h, \quad \frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(4)}} = 0,$$

$$\frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(1)}} = 0, \quad \frac{\partial [W_{kj}]_{h}^{L}}{\partial w_{kj}^{(2)}} = 0,$$

$$\frac{\partial [W_{kj}]_{h}^{U}}{\partial w_{kj}^{(3)}} = 0, \quad \frac{\partial [W_{kj}]_{h}^{U}}{\partial w_{kj}^{(4)}} = 1 - h.$$
In the case of the nonsymmetric triangular fuzzy weight $W_{kj} = (w_{kj}^L, w_{kj}^C, w_{kj}^U)$ in Figure 6(b), the derivatives $\partial[W_{kj}^L]/\partial w_{kj}^{(q)}$ and $\partial[W_{kj}^U]/\partial w_{kj}^{(q)}$ can be calculated as

$$
\frac{\partial[W_{kj}^L]}{\partial w_{kj}^{(q)}} = 1 - h, \quad \frac{\partial[W_{kj}^U]}{\partial w_{kj}^{(q)}} = h,
$$

$$
\frac{\partial[W_{kj}^C]}{\partial w_{kj}^{(q)}} = 0, \quad \frac{\partial[W_{kj}^L]}{\partial w_{kj}^{(q)}} = 0,
$$

$$
\frac{\partial[W_{kj}^U]}{\partial w_{kj}^{(q)}} = 1 - h.
$$

For the fuzzy weight $W_{kj}$ with other shapes (e.g., symmetric triangular), the derivatives $\partial[W_{kj}^L]/\partial w_{kj}^{(q)}$ and $\partial[W_{kj}^U]/\partial w_{kj}^{(q)}$ in (48) can be calculated in a similar manner for the case of the nonsymmetric trapezoidal fuzzy weight and the nonsymmetric triangular fuzzy weight.

From the above discussion, we can see that the amount of adjustment for each parameter of the fuzzy weight $W_{kj}$ is calculated as follows:

$$
\Delta w_{kj}^{(q)}(t) = -\eta \cdot \left( \frac{\partial e_{ph}}{\partial [W_{kj}^L]} \cdot \frac{\partial [W_{kj}^L]}{\partial w_{kj}^{(q)}} + \frac{\partial e_{ph}}{\partial [W_{kj}^U]} \cdot \frac{\partial [W_{kj}^U]}{\partial w_{kj}^{(q)}} \right) + \alpha \Delta w_{kj}^{(q)}(t - 1).
$$

The fuzzy weight $W_{ji}$ between the input layer and the hidden layer is also adjusted in the same manner as the fuzzy weight $W_{kj}$. That is, each parameter of the fuzzy weight $W_{ji} = (w_{ji}^{(1)}, w_{ji}^{(2)}, \ldots, w_{ji}^{(q)}, \ldots, w_{ji}^{(s)})$ is adjusted by the following rule:

$$
w_{ji}^{(q)}(t + 1) = w_{ji}^{(q)}(t) + \Delta w_{ji}^{(q)}(t),
$$

$$
\Delta w_{ji}^{(q)}(t) = -\eta \cdot \left( \frac{\partial e_{ph}}{\partial [W_{ji}^L]} \cdot \frac{\partial [W_{ji}^L]}{\partial w_{ji}^{(q)}} + \frac{\partial e_{ph}}{\partial [W_{ji}^U]} \cdot \frac{\partial [W_{ji}^U]}{\partial w_{ji}^{(q)}} \right) + \alpha \Delta w_{ji}^{(q)}(t - 1),
$$

where $\partial e_{ph}/\partial [W_{ji}^L]$ and $\partial e_{ph}/\partial [W_{ji}^U]$ can be calculated without the specification of a particular shape of the fuzzy weight $W_{ji}$ (for details, see Appendix B). The derivatives $\partial[W_{ji}^L]/\partial w_{ji}^{(s)}$ and $\partial[W_{ji}^U]/\partial w_{ji}^{(s)}$, which depend on the shape of the fuzzy weight $W_{ji}$, can be calculated from the relation between the $h$-level set of $W_{ji}$ and its parameters in the same
manner as in the case of the fuzzy weight $W_{kj}$ [see (49)–(54) or (55)–(57)].

When there are some constraints among the parameters of the fuzzy weights, the learning algorithm derived in this paper may be modified in order that those constraints should be satisfied by the updated fuzzy weights. For example, the nonsymmetric trapezoid fuzzy weight $W_{kj} = (w_{kj}^{(1)}, w_{kj}^{(2)}, w_{kj}^{(3)}, w_{kj}^{(4)})$ should satisfy the inequality $w_{kj}^{(1)} \leq w_{kj}^{(2)} \leq w_{kj}^{(3)} \leq w_{kj}^{(4)}$. In the computer simulations of this paper, we rearrange the values of these four parameters in increasing order when the violation of this inequality happens after updating the fuzzy weight $W_{kj}$ (also see [8] and [13] for the handling of such a constraint).

The parameters of the fuzzy biases $\Theta_k$ and $\Theta_j$ are adjusted in the same manner as the fuzzy weights $W_{kj}$ and $W_{ji}$, respectively.

### 3.3. Numerical Examples

In this subsection, we illustrate the derived learning algorithm by two simple numerical examples. In both examples, we use a fuzzified neural network with a single input unit, five hidden units, and a single output unit. Nonsymmetric trapezoidal fuzzy numbers are used for the fuzzy weights and the fuzzy biases of the fuzzified neural network.

**EXAMPLE 1** Let us consider a single-input and single-output fuzzy mapping. We assume that both the input and the output of this fuzzy mapping are trapezoidal fuzzy numbers. That is, a trapezoidal fuzzy number is mapped to a trapezoidal fuzzy number. We also assume that the following two input-output pairs are given:

$$
[[X_p, T_p]] = \{((0.0, 0.1, 0.2, 0.3), (0.1, 0.25, 0.45, 0.6)),
(0.7, 0.8, 0.9, 1.0), (0.4, 0.55, 0.75, 0.9))\},
$$

where $X_p = (x_p^{(1)}, x_p^{(2)}, x_p^{(3)}, x_p^{(4)})$ is a trapezoidal fuzzy input and $T_p = (t_p^{(1)}, t_p^{(2)}, t_p^{(3)}, t_p^{(4)})$ is a trapezoidal fuzzy output [see Figure 6(a)]. $T_p$ is used as a fuzzy target in the learning by the fuzzified neural network. These two input-output pairs are shown in Figure 7(a). In this figure, five rectangles with solid lines show the five $h$-level sets for $h = 0.2, 0.4, 0.6, 0.8, 1.0$ of the Cartesian product of each input-output pair (the rectangle with dashed lines shows the support set). The Cartesian product of the input $X_p$ and the output (i.e., target) $T_p$ is illustrated in Figure 7(b). It should be realized that the rectangles in Figure 7 correspond to the contour lines of the membership function of the Cartesian product of $X_p$ and $T_p$. 


Using the given fuzzy input-output pairs in (61), we trained the fuzzified neural network by the learning rules in (46) and (58)–(60) with $\eta = 0.5$ and $\alpha = 0.9$. In these learning rules, we used the relations in (51)–(54) because the fuzzy weights were trapezoidal fuzzy numbers. The fuzzy biases were also adjusted in the same manner as the fuzzy weights. Five $h$-level sets of each input-output pair corresponding to $h = 0.2, 0.4, 0.6, 0.8, 1.0$ were used in the learning by the fuzzified neural network. This means that the five rectangles for each input-output pair in Figure 7(a) were used for the learning. The learning was iterated 1000 times for each level of each input-output pair (i.e., 1000 epochs) according to the algorithm in Section 3.2. After the learning, the value of the cost function $e$ in (41) was 0.000004. This means that the actual fuzzy outputs from the trained neural network were almost the same as the fuzzy targets. The actual fuzzy outputs corresponding to the given two input-output pairs and a new trapezoidal fuzzy input $X_p = (0.35, 0.45, 0.55, 0.65)$ are shown in Figure 8. From the comparison between Figure 7(a) and Figure 8, we can see that the actual fuzzy outputs are almost the same as the fuzzy targets. We can also see from Figure 8 that the trained neural network interpolates the given input-output pairs.

**EXAMPLE 2** In this example, we show that the fuzzified neural network can handle real-number inputs. A real number $x$ can be viewed as a fuzzy singleton with the following membership function:

$$
\mu_x(y) = \begin{cases} 
1 & \text{if } y = x, \\
0 & \text{if } y \neq x.
\end{cases}
$$  \hfill (62)
Therefore the $h$-level set of the real number $x$ is

$$[x]_h = [x, x] \quad \text{for} \quad 0 < h \leq 1,$$

where $[x, x]$ is a degenerate interval whose upper limit and lower limit are the same. Equations (62) and (63) mean that real numbers can be treated in the same manner as fuzzy numbers.

Let us consider a single-input and single-output fuzzy mapping. We assume that the input of this fuzzy mapping is a real number while the output is a trapezoidal fuzzy number. That is, a real number is mapped to a trapezoidal fuzzy number. We also assume that the following three input-output pairs are given:

$$\{[X_p, T_p]\} = \{[0.0, (0.4, 0.5, 0.6, 0.7)], [0.5, (0.7, 0.75, 0.85, 0.9)], [1.0, (0.2, 0.3, 0.5, 0.6)]\},$$

where $X_p$ is a real number input and $T_p = (t^{(1)}_p, t^{(2)}_p, t^{(3)}_p, t^{(4)}_p)$ is a trapezoidal fuzzy output (i.e., fuzzy target). These three input-output pairs are shown in Figure 9(a). In this figure, each fuzzy target is shown after turning its original shape clockwise. Therefore the left side and the right side of each trapezoid in this figure correspond to the 0-level and the 1-level of the membership function, respectively.

The learning in the fuzzified neural network was performed in the same manner as in Example 1. After 1000 epochs, the value of the cost function $e$ in (41) was 0.000074. This means that the actual fuzzy outputs from the trained neural network were almost the same as the fuzzy targets. The actual fuzzy outputs corresponding to the given three input-output pairs and two new real-number inputs ($x_p = 0.25$ and $x_p = 0.75$) are shown in Figure 9(b). From the comparison between Figure 9(a) and Figure 9(b), we can see that the actual fuzzy outputs are almost the same as the fuzzy targets. We can also see from Figure 9(b) that the trained neural network
interpolates the given input-output pairs. The simulation results in Figure 8 and Figure 9(b) show that the fuzzified neural network can handle real-number inputs as well as fuzzy inputs.

4. COMPARISON WITH EARLIER APPROACHES

In this section, we compare the derived learning algorithm with earlier approaches that used real-number weights [5] and symmetric triangular fuzzy weights [8]. In computer simulation, we trained the following three neural networks:

1. Fuzzified neural network with real-number weights [5].
2. Fuzzified neural network with symmetric triangular fuzzy weights [8].
3. Fuzzified neural network with nonsymmetric trapezoidal fuzzy weights.
The same network architecture (i.e., a single input unit, five hidden units, and a single output unit) was used for these three neural networks. We also employed the same condition as in Section 3.3 for the learning in each of the three neural networks.

4.1. Mapping of Triangular Fuzzy Numbers

We trained the three neural networks by the following fuzzy if-then rules:

if \( x \) is small then \( y \) is small,
if \( x \) is medium then \( y \) is medium,
if \( x \) is large then \( y \) is large,

where the membership functions of the linguistic values are given in Figure 10. These three fuzzy if-then rules can be viewed as the three fuzzy input-output pairs

\[
\{(X_p, T_p)\} = \{(small, small), (medium, medium), (large, large)\}. \quad (65)
\]

These input-output pairs are shown in Figure 11(a) in the same manner as in Figure 7(a).

In the learning by the three neural networks, the three input-output pairs in Figure 11(a) were used as training data. The values of the cost function \( e \) in (41) after 10,000 epochs are shown in Table 1 for each of the three neural networks. From Table 1, we can see that similar results were obtained by the three neural networks. That is, the difference of the fuzzy weights (real numbers, symmetric triangular fuzzy numbers, nonsymmetric trapezoidal fuzzy numbers) had a small effect on the learning.

The membership functions of the actual fuzzy outputs from the trained neural network with nonsymmetric trapezoidal fuzzy weights are shown in Figure 12 together with the fuzzy targets. From this figure, we can see that

![Figure 10. Membership function of linguistic values (S: small; MS: medium small; M: medium; ML: medium larger; L: large).](image-url)
a good fit to the fuzzy targets is obtained by the trained neural network. The actual fuzzy outputs corresponding to new fuzzy inputs medium small and medium large (see Figure 10 for the membership functions of these new fuzzy inputs) are shown in Figure 11(b). From the comparison between Figure 11(a) and Figure 11(b), we can see that the trained neural network interpolates the given fuzzy data. The actual fuzzy outputs in Figure 11(b) corresponding to the new fuzzy inputs can be interpreted as medium small and medium large, respectively. Therefore we have the following new fuzzy if-then rules from the trained neural network:

if \( x \) is medium small then \( y \) is medium small,

if \( x \) is medium large then \( y \) is medium large.

Almost the same results were obtained from the other neural networks with real-number weights and symmetric triangular fuzzy weights for this example. This means that the neural networks with real-number weights and symmetric triangular fuzzy weights work well for the mapping from triangular fuzzy numbers to triangular fuzzy numbers (see also the simulation results in [6] and [8]), like the neural network with nonsymmetric fuzzy weights.

| Table 1. The Values of the Cost Function for the Mapping of Triangular Fuzzy Numbers |
|-----------------------------------------------|-----------------|-----------------|-----------------|
| Type of weights                              | Real number     | Symmetric triangular | Nonsymmetric trapezoidal |
| Value of \( e \)                             | 0.0012           | 0.0010            | 0.0009           |
4.2 Mapping of Triangular-Shape Fuzzy Numbers

In the last subsection, the membership function of each linguistic value was linear. In this subsection, we examine the fitting ability of the three neural networks for nonlinear membership functions.

Let us assume that the following three fuzzy if-then rules are given:

if $x$ is very small then $y$ is more or less small,

if $x$ is very medium then $y$ is more or less medium,

if $x$ is very large then $y$ is more or less large,

where the membership functions of the linguistic values are shown in Figure 13 and Figure 14. In Figure 13, the linguistic hedge very has the following effect on the original membership functions:

$$
\mu_{\text{very small}}(x) = (\mu_{\text{small}}(x))^2,
\mu_{\text{very medium}}(x) = (\mu_{\text{medium}}(x))^2,
\mu_{\text{very large}}(x) = (\mu_{\text{large}}(x))^2,
$$

Figure 12. Membership functions of fuzzy targets and actual fuzzy outputs.
From Figure 13, we can see that the linguistic hedge very sharpens the original membership functions. On the other hand, the linguistic hedge more or less has the following effect on the original membership functions (see Figure 14):

\[
\mu_{\text{more or less small}}(x) = \left( \mu_{\text{small}}(x) \right)^{0.5}, \tag{69}
\]
\[
\mu_{\text{more or less medium}}(x) = \left( \mu_{\text{medium}}(x) \right)^{0.5}, \tag{70}
\]
\[
\mu_{\text{more or less large}}(x) = \left( \mu_{\text{large}}(x) \right)^{0.5}. \tag{71}
\]

From the three given fuzzy if-then rules, we have the following fuzzy training data:

\[
\{(X_p, T_p)\} = \{(\text{very small, more or less small}), \quad (\text{very medium, more or less medium}), \quad (\text{very large, more or less large})\}. \tag{72}
\]

These training data are shown in Figure 15. From this figure (or from Figures 13 and 14), we can see that the fuzziness of the fuzzy targets is larger than that of the fuzzy inputs.
In the same manner as in Section 4.1, we trained the three neural networks on the fuzzy training data in (72). The values of the cost function $e$ in (41) after 10,000 epochs are shown in Table 2 for each of the three neural networks. From Table 2, we can see that the best result is obtained by the fuzzified neural network with nonsymmetric trapezoidal fuzzy weights. The actual fuzzy outputs from the trained neural network with nonsymmetric trapezoidal fuzzy weights are shown in Figure 16 together with the fuzzy targets. From this figure, we can see that a good fit to the fuzzy targets is obtained by the trained neural network. [The fit below 0.2-level is not good, while above 0.2-level it is very good. This is because we used only five levels ($h = 0.2, 0.4, 0.6, 0.8, 1.0$) in the learning by the fuzzified neural network.]

It should be noted that the results in Table 2 for the neural networks with symmetric triangular fuzzy weights and real-number weights are not bad, because the values of the cost function $e$ are small. This means that the three neural networks can handle the nonlinear membership function of triangular shape.

We also trained the three neural networks on the following three fuzzy if-then rules:

- if $x$ is *more or less small* then $y$ is *very small*,
- if $x$ is *more or less medium* then $y$ is *very medium*,
- if $x$ is *more or less large* then $y$ is *very large*.

<table>
<thead>
<tr>
<th>Type of weights</th>
<th>Real number</th>
<th>Symmetric triangular</th>
<th>Nonsymmetric trapezoidal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of $e$</td>
<td>0.0115</td>
<td>0.0052</td>
<td>0.0006</td>
</tr>
</tbody>
</table>
From these fuzzy if-then rules, we have the fuzzy training data in Figure 17. From Figure 17, we can see that the fuzziness of the fuzzy inputs is larger than that of the fuzzy targets. The simulation results for the three neural networks are shown in Table 3. From this table, we can see that the difference of the fuzzy weights (real numbers, symmetric triangular fuzzy numbers, nonsymmetric trapezoidal fuzzy numbers) has a slight effect on the learning.

From the comparison among Tables 1–3, we can see that the fuzzy weights have a good effect on the learning when the fuzziness of the fuzzy targets is larger than that of the fuzzy inputs (see Table 2). This is because the fuzzy weights increase the fuzziness of the fuzzy inputs.
Table 3. The Values of the Cost Function for the Mapping of Triangular Fuzzy Numbers (decreasing fuzziness)

<table>
<thead>
<tr>
<th>Type of weights</th>
<th>Real number</th>
<th>Symmetric triangular</th>
<th>Nonsymmetric trapezoidal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of $e$</td>
<td>0.0046</td>
<td>0.0057</td>
<td>0.0081</td>
</tr>
</tbody>
</table>

4.3. Mapping from Triangular Fuzzy Numbers to Trapezoidal Fuzzy Numbers

In this subsection, we examine the case of trapezoidal fuzzy targets. Let us assume that the following three fuzzy if-then rules are given:

- if $x$ is small then $y$ is small,
- if $x$ is medium then $y$ is medium small or medium,
- if $x$ is large then $y$ is medium or medium large or large.

We assume that the membership functions of medium small or medium and medium or medium large or large are trapezoidal:

$$medium \text{ small or medium} = (0.0, 0.25, 0.5, 0.75), \quad (73)$$

$$medium \text{ or medium large or large} = (0.25, 0.5, 1.0, 1.0). \quad (74)$$

These membership functions are shown in Figure 18.

From the given fuzzy if-then rules, we have the training data in Figure 19(a). The three neural networks were trained in the same manner as in Section 4.1. The simulation results are summarized in Table 4. From Table 4, we can see that only the neural network with nonsymmetric trapezoidal fuzzy weights can handle the given training data. The values of the cost function $e$ obtained by the neural networks with real-number weights and symmetric triangular weights are more than 20 times as large as the results in Tables 1–3. We show the fuzzy outputs corresponding to new fuzzy inputs medium small and medium large in Figure 19(b). From the comparison between Figure 19(a) and Figure 19(b), we can see that good interpolation is realized by the trained neural network with nonsymmetric trapezoidal fuzzy weights.

5. CONCLUSION

In this paper, we have derived a back-propagation learning algorithm for fuzzified neural networks with fuzzy weights. The algorithm is applicable to the adjustment of a fuzzy weight with any shape if its membership function
is specified by a finite number of parameters. This requirement of the learning algorithm is satisfied by almost all fuzzy numbers that are usually used in applications of fuzzy theory (e.g., trapezoidal and triangular). The learning algorithm in this paper can be viewed as a generalized version of the work by Ishibuchi et al. [8, 13] that was applicable only to symmetric
Table 4. The Values of the Cost Function for the Mapping from Triangular Fuzzy Numbers to Trapezoid Fuzzy Numbers

<table>
<thead>
<tr>
<th>Type of weights</th>
<th>Real number</th>
<th>Symmetric triangular</th>
<th>Nonsymmetric trapezoidal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value of $e$</td>
<td>0.2759</td>
<td>0.2502</td>
<td>0.0017</td>
</tr>
</tbody>
</table>

triangular fuzzy weights [8] or nonsymmetric trapezoidal fuzzy weights [13]. The ability of the derived learning algorithm was examined in computer simulations. By the simulations, it was shown that the fuzzified neural networks have high fitting ability for fuzzy targets and high interpolation ability for new fuzzy inputs.

APPENDIX A: INPUT-OUTPUT RELATION FOR LEVEL SETS

For simplicity let us assume that the $h$-level sets of the fuzzy inputs are nonnegative, i.e.,

$$0 \leq [X_{pi}]_h^L \leq [X_{pi}]_h^U$$

for $i = 1, 2, \ldots, n_I$.

In this case, the input-output relation of the fuzzified neural network for the $h$-level sets in (24)-(28) can be rewritten from (20)-(21) and (23) as follows:

**Input units:**

$$[O_{pi}]_h = [[O_{pi}]_h^L, [O_{pi}]_h^U] = [[X_{pi}]_h^L, [X_{pi}]_h^U].$$

**Hidden units:**

$$[O_{pj}]_h = [[O_{pj}]_h^L, [O_{pj}]_h^U] = [f([Net_{pj}]_h^L), f([Net_{pj}]_h^U)],$$

$$[Net_{pj}]_h^L = \sum_{i=1}^{n_I} [W_{ji}]_h^L \cdot [O_{pi}]_h^L + \sum_{i=1}^{n_I} [W_{ji}]_h^L \cdot [O_{pi}]_h^U + [\Theta]_h^L,$$

$$[Net_{pj}]_h^U = \sum_{i=1}^{n_I} [W_{ji}]_h^U \cdot [O_{pi}]_h^L + \sum_{i=1}^{n_I} [W_{ji}]_h^U \cdot [O_{pi}]_h^U + [\Theta]_h^U.$$
Output units:

\[ [O_{pk}]_h = \left[ [O_{pk}]^L_h, [O_{pk}]^U_h \right] = \left[ f(\text{Net}_{pk}^L_h), f(\text{Net}_{pk}^U_h) \right], \]

\[ \text{Net}_{pk}^L_h = \sum_{j=1}^{n_H} [W_{kj}]^L_h \cdot [O_{pj}]^L_h + \sum_{j=1}^{n_H} [W_{kj}]^U_h \cdot [O_{pj}]^U_h + [\Theta_k]^L_h. \]

\[ \text{Net}_{pk}^U_h = \sum_{j=1}^{n_H} [W_{kj}]^U_h \cdot [O_{pj}]^U_h + \sum_{j=1}^{n_H} [W_{kj}]^L_h \cdot [O_{pj}]^L_h + [\Theta_k]^U_h. \]

### APPENDIX B: CALCULATION OF \( \partial e_{ph}/\partial [W_{kj}]^L_h \), \( \partial e_{ph}/\partial [W_{kj}]^U_h \), \( \partial e_{ph}/\partial [W_{kj}]^L_h \), AND \( \partial e_{ph}/\partial [W_{kj}]^U_h \)

As in Appendix A, let us assume that the \( h \)-level sets of the fuzzy inputs are nonnegative. In this case, the input-output relation of the fuzzified neural network for the \( h \)-level sets is the same as in Appendix A.

The cost function \( e_{ph} \) for the \( h \)-level set of the \( p \)th input-output pair is written from (36)-(39) as follows:

\[ e_{ph} = \sum_{k=1}^{n_O} \left\{ \frac{h \cdot \left( [T_{pk}]^L_h - [O_{pk}]^L_h \right)^2}{2} + h \cdot \left( [T_{pk}]^U_h - [O_{pk}]^U_h \right)^2 \right\}. \]

Then the derivatives \( \partial e_{ph}/\partial [W_{kj}]^L_h \), \( \partial e_{ph}/\partial [W_{kj}]^U_h \), \( \partial e_{ph}/\partial [W_{kj}]^L_h \), and \( \partial e_{ph}/\partial [W_{kj}]^U_h \) can be calculated from the input-output relation in Appendix A and the cost function \( e_{ph} \) as follows.

### B.1. The Calculation of \( \partial e_{ph}/\partial [W_{kj}]^L_h \)

1. If \( [W_{kj}]^L_h \geq 0 \) then

\[ \frac{\partial e_{ph}}{\partial [W_{kj}]^L_h} = \frac{\partial e_{ph}}{\partial [\text{Net}_{pk}]^L_h} \cdot \frac{\partial [\text{Net}_{pk}]^L_h}{\partial [W_{kj}]^L_h} = -\delta_{pkh}^L \cdot [O_{pj}]^L_h, \]

where

\[ \delta_{pkh}^L = -\frac{\partial e_{ph}}{\partial [\text{Net}_{pk}]^L_h} = h \cdot \left( [T_{pk}]^L_h - [O_{pk}]^L_h \right) \cdot [O_{pk}]^L_h \cdot (1 - [O_{pk}]^L_h). \]

2. If \( [W_{kj}]^L_h < 0 \) then

\[ \frac{\partial e_{ph}}{\partial [W_{kj}]^L_h} = \frac{\partial e_{ph}}{\partial [\text{Net}_{pk}]^L_h} \cdot \frac{\partial [\text{Net}_{pk}]^L_h}{\partial [W_{kj}]^L_h} = -\delta_{pkh}^L \cdot [O_{pj}]^U_h. \]
B.2. The Calculation of $\frac{\partial e_{ph}}{\partial [W_{kj}]^U_h}$

1. If $[W_{kj}]^U_h \geq 0$ then

$$\frac{\partial e_{ph}}{\partial [W_{kj}]^U_h} = \frac{\partial e_{ph}^U}{\partial [\text{Net}_{pk}]^U_h} \cdot \frac{\partial [\text{Net}_{pk}]^U_h}{\partial [W_{kj}]^U_h} = -\delta^U_{pkh} \cdot [O_{pj}]^U_h,$$

where

$$\delta^U_{pkh} = -\frac{\partial e_{ph}^U}{\partial [\text{Net}_{pk}]^U_h} = h \cdot \left( [T_{pk}]^U_h - [O_{pk}]^U_h \right) \cdot [O_{pk}]^U_h \cdot \left( 1 - [O_{pk}]^U_h \right).$$

2. If $[W_{kj}]^U_h < 0$ then

$$\frac{\partial e_{ph}}{\partial [W_{kj}]^U_h} = \frac{\partial e_{ph}^U}{\partial [\text{Net}_{pk}]^U_h} \cdot \frac{\partial [\text{Net}_{pk}]^U_h}{\partial [W_{kj}]^U_h} = -\delta^U_{pkh} \cdot [O_{pj}]^L_h.$$

B.3. The Calculation of $\frac{\partial e_{ph}}{\partial [W_{ji}]^L_h}$

1. If $[W_{ji}]^L_h \geq 0$ then

$$\frac{\partial e_{ph}}{\partial [W_{ji}]^L_h} = -\delta^A_{ph} \cdot [O_{pj}]^L_h \cdot \left( 1 - [O_{pj}]^L_h \right) \cdot [O_{pi}]^L_h,$$

where

$$\delta^A_{ph} = \sum_{k=1}^{n_j} \delta^A_{pkh} \cdot [W_{kj}]^L_h + \sum_{k=1}^{n_I} \delta^U_{pkh} \cdot [W_{kj}]^U_h.$$  

2. If $[W_{ji}]^L_h < 0$ then

$$\frac{\partial e_{ph}}{\partial [W_{ji}]^L_h} = -\delta^A_{ph} \cdot [O_{pj}]^L_h \cdot \left( 1 - [O_{pj}]^L_h \right) \cdot [O_{pi}]^U_h.$$

B.4. The Calculation of $\frac{\partial e_{ph}}{\partial [W_{ji}]^U_h}$

1. If $[W_{ji}]^U_h \geq 0$ then

$$\frac{\partial e_{ph}}{\partial [W_{ji}]^U_h} = -\delta^B_{ph} \cdot [O_{pj}]^U_h \cdot \left( 1 - [O_{pj}]^U_h \right) \cdot [O_{pi}]^U_h,$$

where

$$\delta^B_{ph} = \sum_{k=1}^{n_j} \delta^L_{pkh} \cdot [W_{kj}]^L_h + \sum_{k=1}^{n_I} \delta^U_{pkh} \cdot [W_{kj}]^U_h.$$
2. If $[W_{ji}]_{h}^U < 0$ then

$$\frac{\partial e_{ph}}{\partial [W_{ji}]_{h}^U} = -\delta_{ph} \cdot [O_{p_j}]_{h}^U \cdot (1 - [O_{p_j}]_{h}^U) \cdot [O_{p_i}]_{h}^U.$$ 

References


