LEARNING ALGORITHMS WITH PRESCRIBED WEIGHTS FOR ALGEBRAIC FNN

D. Arotaritei[1], H. N. Teodorescu[2]
Institute for Computer Science, Romanian Academy, Iasi, 660, Romania
e-mail: [1]dragos@iit.iit.tuiasi.ro, [2]hteodor@tuiasi.ro

ABSTRACT: The basic principles of the algebraic fuzzy neural networks with fuzzy weights using two learning algorithms (trial-and-error and adapted gradient) are presented. A modified multilayer architectures, in order to be used for economic applications are considered. The fuzzy weights can be fuzzy numbers (usually non-symmetric) and trapezoidal fuzzy numbers. The network is able to map a vector of trapezoidal (triangular) fuzzy numbers into any other vector of trapezoidal (triangular) fuzzy numbers.

KEYWORDS: algebraic fuzzy neural networks, trapezoidal fuzzy weights, trial-and-error algorithm, adapted gradient algorithm.

1. INTRODUCTION

Various approaches of fuzzification of the neural networks have been proposed. Some of them use fuzzy logic, implementing *if-then rules*, others use arithmetic fuzzy numbers [1-3]. A recent class of neuro-fuzzy system is based on neural network topology and fuzzy algebraic systems [3-5]. Two different approaches of arithmetic fuzzy neural network have been proposed in [1-2] and respectively in [3-5].

Ishibuchi [1-2] proposed an architecture of multilayer feedforward algebraic neural network for fuzzy input vectors. The output of this fuzzy neural network is a vector of fuzzy numbers. The input-output relations in this fuzzy neural network, based on max/min operators, are defined by the extension principle of Zadeh. In these papers, the authors used only symmetric triangular fuzzy numbers [1-2]. The learning algorithm, based on the cost function, uses a gradient descent method.

A potential application of the Algebraic Fuzzy Neural Network (AFNN) is the modelling, for instance economic process modelling [6-7]. In such application, the structure of the process can impose specific constraints, such as zero-valued weights, or weights having given values (during all learning stages). An AFNN with Trial-and-Error (TE) or Adapted Gradient (AG) learning algorithm can be used to fit the input-to-output existing data and moreover to perform a mapping that fits the equations of the process. In such a case, the AFNN can be used to determine the coefficients of the equations in the model [6-7]. Note that different nonlinearities with saturation can be accepted in the model of the neuron, because the nonlinearity type does not essentially changes the TE algorithm. Finally, note that even if the flow of variables in the modelled process is not exactly known, the AFNN can help in proposing models of the process, starting from the input-to-output data. This applied topic is now under investigation.

In this paper we deal with an algebraic fuzzy neural network with a modified multilayer topology. The fuzzy algebraic rules are applied in a similar manner as in [4-7]. All the operations are defined in the frame of fuzzy arithmetic and one uses triangular fuzzy numbers (usually non-symmetric) with α-cuts, or trapezoidal fuzzy numbers. We briefly present the two algorithms [4-7], and we discuss the performances for an algebraic fuzzy neural networks with modified multilayer topology for used in economic applications.

2. THE ALGEBRAIC FUZZY NEURON AND NEURAL NETWORK ARCHITECTURE

The *h*-level set of a fuzzy number $\tilde{X}$ is defined as:

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

In what follows, $\tilde{X}_h = (x_h^L, x_h^C, x_h^R)$ denote triangular fuzzy numbers for *h*-level set. Let be $\tilde{A}_h = (a_h^L, a_h^C, a_h^R)$, $\tilde{B}_h = (b_h^L, b_h^C, b_h^R)$, $\tilde{C}_h = (c_h^L, c_h^C, c_h^R)$, where $\tilde{C}_h = \tilde{A}_h * \tilde{B}_h$, $\tilde{C}_h$ is the *h*-level for a triangular fuzzy number (TFN), and * is a non-standard algebraic operation defined by:

$$\begin{aligned} c_h^L &= min\{a_h^L * b_h^L, a_h^L * b_h^R, a_h^R * b_h^L, a_h^R * b_h^R\} \\ c_h^C &= a_h^C * b_h^C \\ c_h^R &= max\{a_h^L * b_h^L, a_h^L * b_h^R, a_h^R * b_h^L, a_h^R * b_h^R\} \end{aligned} \quad (2)$$

Above, $*\in\{+, -, \bullet\}$ are the operations in classic arithmetic, and $*\in\{\tilde{+}, \tilde{-}, \tilde{\bullet}\}$ are the corresponding modified operations in fuzzy arithmetic, defined according to (1). Similar relations can be used to get the α-cuts (see below). The basic algebraic fuzzy neuron (Figure 1), is based on the operations defined above. The values $\varphi(s_h^L, s_h^C, s_h^R)$ are computed, using the modified extension principle (1)-(2), as in Fig. 2:

$$\tilde{s}_h = \tilde{x}_{1h} \tilde{\cdot} \tilde{w}_{1h} \tilde{+} \tilde{x}_{2h} \tilde{\cdot} \tilde{w}_{2h} \tilde{+} \ldots \tilde{+} \tilde{x}_{n_h} \tilde{\cdot} \tilde{w}_{n_h} \tilde{+} \tilde{\theta}_h \quad (3)$$

$$\tilde{y}_h = \varphi(\tilde{s}_h) = (\varphi(s_h^L), \varphi(s_h^C), \varphi(s_h^R)) \quad (4)$$

The algebraic fuzzy neural network (AFNN) has a multilayer feedforward graph and uses algebraic fuzzy neurons (FN) [3-6]. The symbols $\tilde{x}_{ih}$, $\tilde{w}_{jh}$ and $\tilde{\theta}_h$ stand for fuzzy inputs, fuzzy weights and fuzzy biases, respectively for a given *h*-level.

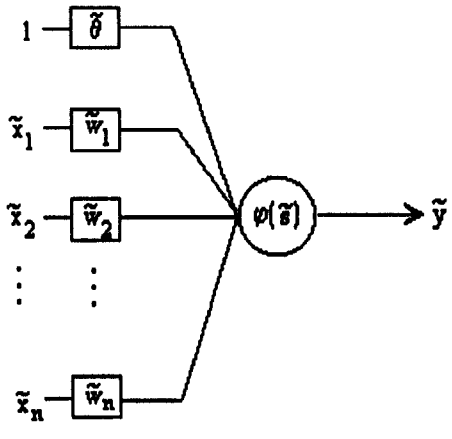

Fig. 1: The fuzzy neuron

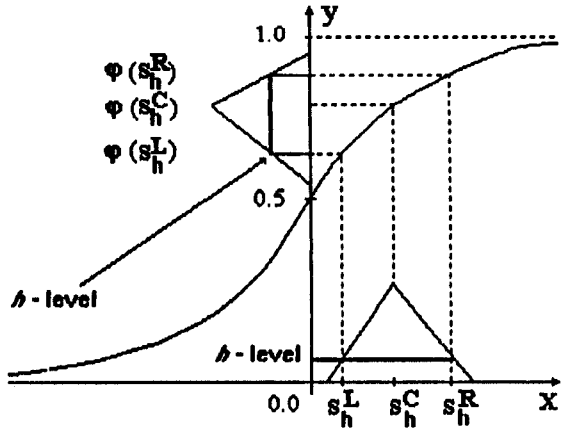

Fig. 2: The fuzzy activation function

The input-output relation in the AFNN, using triangular fuzzy numbers (TFN) and the rules (1)-(3) is straightforward to develop:

$$\tilde{x}_{j_h}^{k} = \varphi\left(\sum_{i}{}_{\text{fuzzy}}\, \tilde{x}_{i_h}^{k-1} \tilde{\cdot} \tilde{w}_{ji_h}^{k} \tilde{+} \tilde{\theta}_{j_h}\right) \tag{5}$$

where k is layer number, j is the j neuron from k layer and i is the i neuron from k-1 layer.

3. LEARNING ALGORITHMS

3.1. Trial-and-Error (TE) algorithm for *h*-level

The algorithm is as follows. First, the central (w^C) value of fuzzy weights is obtained by BP algorithm. For each *h*-level, we set the w^L , w^C and w^R values at previous *h*-1 of corresponding values obtained by a completely cycle of learning. We adapt the w^L and w^R values for this level by TE algorithm, until we reach the stop condition (for h-level). At each iteration we adapt the weight only if the condition $w_h^L < w_{h-1}^L < w_{h-1}^C = w_h^C < w_{h-1}^R < w_h^R$ is satisfied. This procedure is repeated until we reach the adaptation of the last h-level (base of TFN). The fuzzy biases θ^P, p = L, C, R, are changed in the same manner.

Remember that a trapezoidal fuzzy number can be defined as $\tilde{X}=(x^L,x^{CL},x^{CR},x^R)$, $x^L \le x^{CL} \le x^{CR} \le x^R$, $\mu(x^{CL})=1$, $\mu(x^{CR})=1$. Therefore, the learning algorithm for such fuzzy number includes an additional BP stage for the values x^{CR}. Hence, first we adapt w^L ,w^{CL} and w^R values of weights by TE algorithm as in TE for TFN. Next, we adapt the w^{CR} values by BP algorithm using BP algorithm, with the respect of relation $w^{CL} \le w^{CR} \le w^R$.

3.2. Adapted Gradient Algorithm (AG) for *h*-level

From (1), (2) and (3), it can be seen that the central values (C) of the variables operated by the AFNN are not linked to the left (L) and right (R) values of the corresponding triangular fuzzy numbers. Hence, the central values of weights (w^C) can be adapted by a classic backpropagation (BP) algorithm [2-4].

Because the w^L and w^R values are linked by *min/max* operators, the error term is not derivable, and a gradient algorithm needs a special technique. A strategy to adapt the left and right values of the weights is *adapted gradient* (AG) described below. The cost function (square error term) is classically defined by (similarly to [1]), for an *h*-level:

$$E_h = E_h^L + E_h^R = 1/2 \cdot \sum_{i=1}^{n} (d_{i_h}^L - y_{i_h}^L)^2 + 1/2 \cdot \sum_{i=1}^{n} (d_{i_h}^R - y_{i_h}^R)^2 \tag{6}$$

In the equations below, $\eta(x):\Re \to [0,1]$ is chosen to be:

$$\eta(x) = \begin{cases} 0 & \text{for } x \ge 0 \\ 1 & \text{for } x < 0 \end{cases} \quad \text{or,} \quad \eta(x) = 1/(1 + \exp(\beta x)) \quad \text{with } \beta = 500\text{-}1000 \tag{7}$$

Note that for a crisp neural network (when $w^L=w^C=w^R$ and $x^L=x^C=x^R$), the AG algorithm represents a particular case, the classic BP. The extension of fuzzy arithmetic to h-level sets is straightforward, as below:

$$w^L_{ji,h}(t+1) = w^L_{ji,h}(t) + \Delta w^L_{ji,h}(t), \quad w^R_{ji,h}(t+1) = w^R_{ji,h}(t) + \Delta w^R_{ji,h}(t) \tag{8}$$

$$\Delta w^L_{ji,h} = \lambda \cdot \delta^L_{j,h} \cdot [x^L_{i,h} + \eta(w^L_{ji,h}) \cdot [x^R_{i,h} - x^L_{i,h}]] \tag{9}$$

$$\Delta w^R_{ji,h} = \lambda \cdot \delta^R_{j,h} \cdot [x^R_{i,h} - \eta(w^R_{ji,h}) \cdot [x^R_{i,h} - x^L_{i,h}]] \tag{10}$$

where, for the output layer:

$$\delta^L_{j,h} = (d^L_{j,h} - y^L_{j,h}) \cdot y^L_{j,h} \cdot [1 - y^L_{j,h}], \qquad \delta^R_{j,h} = (d^R_{j,h} - y^R_{j,h}) \cdot y^R_{j,h} \cdot [1 - y^R_{j,h}] \tag{11}$$

and for hidden layers:

$$\delta^L_{j,h} = x^L_{j,h} \cdot [1 - x^L_{j,h}] \sum_r \delta^L_{r,h} \cdot [w^L_{rj,h} + \eta(x^L_{j,h}) \cdot [w^R_{rj,h} - w^L_{rj,h}]] \tag{12}$$

$$\delta^R_{j,h} = x^R_{j,h} \cdot [1 - x^R_{j,h}] \cdot \sum_r \delta^R_{r,h} \cdot [w^R_{rj,h} - \eta(x^R_{j,h}) \cdot [w^R_{rj,h} - w^L_{rj,h}]] \tag{13}$$

4. SIMULATION RESULTS

An example of simulation is briefly presented below. We apply the proposed method to approximate realisation of non-linear mapping of fuzzy numbers. The modified multilayer architecture is used, with 6 inputs, 3 neurons in the hidden layer and 4 neurones in the output layer (Fig. 4).

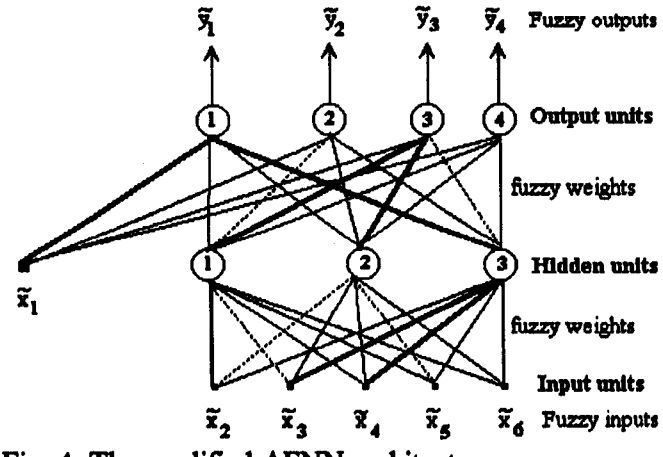

Fig. 4. The modified AFNN architecture

The dotted lines represent the missing connections (weight zeroes), the bulk lines the prescribed fuzzy weights, and the others lines represents normal fuzzy weights which must be adapted. The TFN from input space have the bases inside the interval [-1, 1] and are mapping to TFN which have the bases in [0,+1]. The learning starts with 300 iterations for adaptation of central values (C) of fuzzy weights. The learning factor, denoted by λ, is chosen 0.5. The input vector of TFN is shown in Fig. 5. The evolution of the output vector of TFN after 301 and 600 iterations are shown in Figures 6-7. After about 600 iterations, the error (square root of error term E) is generally lower than 10^{-3}.

The standard architecture of AFNN can be modified with imposed restrictions: some weights are fixed, some connections between neurons are missing, and some inputs are connected directly to

output layer. The experimental results showed the capability of AFNN and proposed algorithm to mapping, with a good performance, a VFN into another VFN. Two weights with α-cut at h=0.8 and h=0.4 are represented in Fig. 8. The learning time in this cases is greater than in classic architecture, because of restrictions (Fig. 9). The dotted line represent the error term evolution with prescribed weights, an the normal line the error term evolution with all connection.

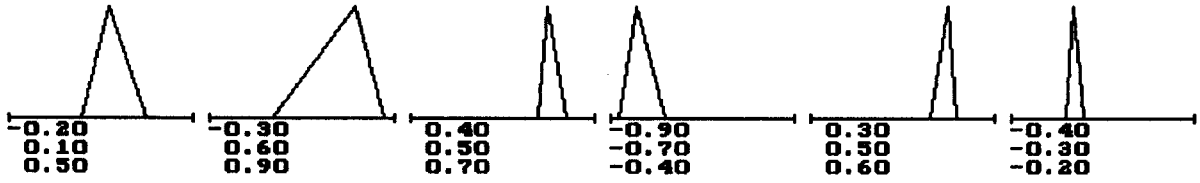

Fig. 5: The input in AFNN for TFN.

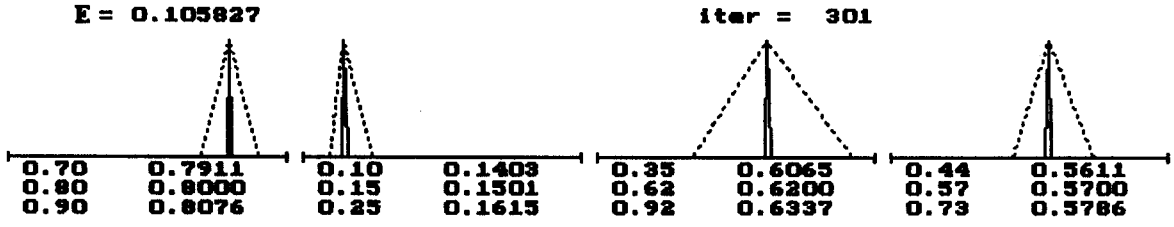

Fig. 6: The output of AFNN after 301 AG iterations for TFN.

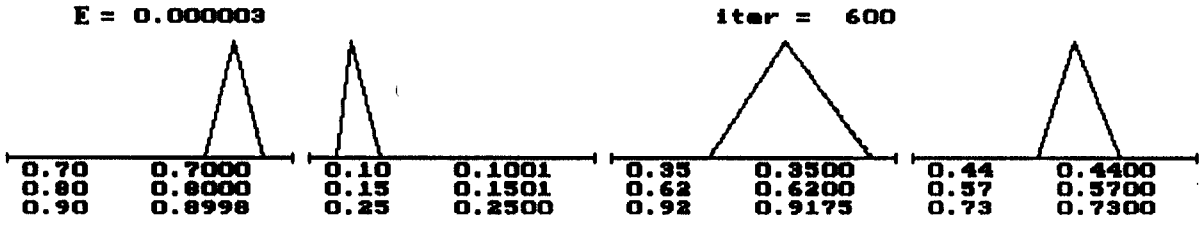

Fig. 7: The output of AFNN after 600 AG iterations, for TFN (L and R values).

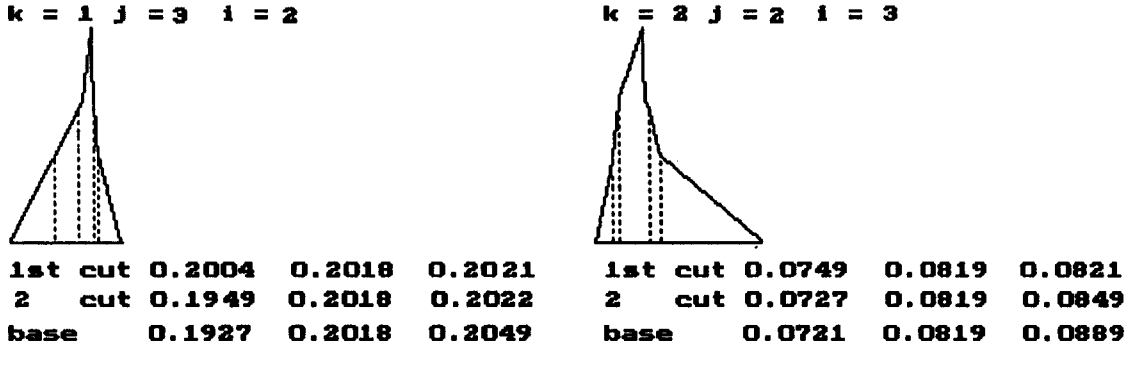

Fig. 8. The weights w_{32}^{1} and w_{23}^{2} with h=0.8 and h=0.4, after 600 AG iterations.

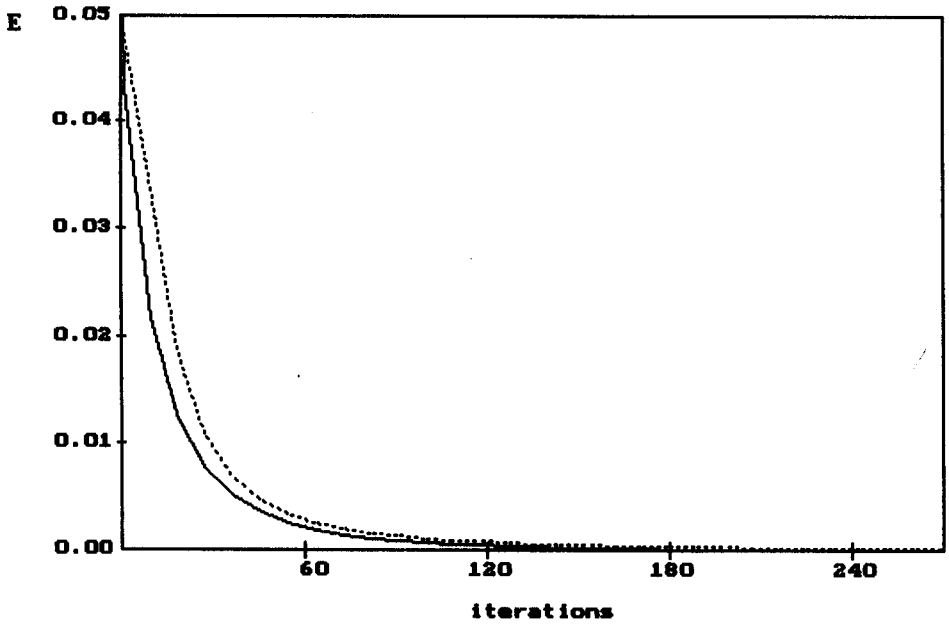

Fig. 9. The error term evolution depending for full connected AFNN and for AFN with prescribed weights.

5. CONCLUSIONS

The presented AFNN is able to deal with asymmetric triangular and trapezoidal fuzzy numbers, in contrast to the AFNN in [1-2], that uses only symmetric TFN. The TE algorithm is simpler than the one proposed by Ishibuchi [1-2]. The approximation obtained in non-linear mapping of TFN is as good as the approximation obtained by the other algorithms reported.

The computing time, in the case AFNN, may be somewhat high. One can overcome this drawback by using an appropriate learning factor (λ>5.0). In the proposed algorithms, the symmetric triangular fuzzy numbers, as used in [1-2], are a particular case. The algorithms proposed here, based on TE or AG, can be extended to any type of fuzzy numbers. The application of TE and AG with α-cuts for TFN gave a good approximation for non-linear mapping between vectors of fuzzy numbers of TFN. This assertion may be extended to fuzzy numbers with other shapes of membership functions (e.g. Gauss).

Several potential applications of the above AFNN are discussed in [7].

REFERENCES

[1] Ishibuchi, H., Kwon, K.,Tanaka, H. (1995), A learning algorithm on fuzzy neural networks with triangular fuzzy weights, Fuzzy Sets and Systems, Vol. 72, No. 3, June 23, pp. 257-264.

[2] Ishibuchi, H., Fujioka, R., Tanaka, H. (1992), An Architecture of Neural Networks for Input Vectors of Fuzzy Numbers, Proc. FUZZ-IEEE '92, San Diego, USA, pp. 643-650.

[3] Teodorescu, H.N., Arotaritei, D., Gonzales, E.P., Cuervo, G.M. (1996), Trail-and-Error Algorithm for Algebraic Fuzzy Neural Networks using Triangular Fuzzy Numbers, Proceeding of

the International Conference on Intelligent Technologies in Human-Related Sciences ITHURS' 96, León, Spain, pp. 81-86.
[4] Teodorescu, H.N., Arotaritei, D., Gonzales, E.L. (1996), A General Trial-and-Error Algorithm for Algebraic Fuzzy Neural Networks Proceedings of the Fourth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, September 2-5, 1996, vol 1, pp. 8-12.
[5] Teodorescu, H.N., Arotaritei, D., Gonzales, E.L., Cuervo, A.M. (1996), Adapted Gradient Algorithm for Algebraic Fuzzy Neural Networks, Proceedings of the 1st International Symposium/Workshop AT'96: Neuro-Fuzzy System, 29-31 August, 1996 Lausanne, Switzerland.
[6] Kaufmann, A., Aluja, J.G. (1995), Grafos neuronales para la Economia y la Gestión de Empresas, Ediciones Pirámiede, SA.
[7] Teodorescu H.N., Gonzales E.L., Cuervo, A.M., Arotaritei, D. (1996), Algebraic Neuro-Fuzzy Systems in Economic and Management Applications, 1st International Symposium/Workshop AT'96: Neuro-Fuzzy System, 29-31 August, Lausanne, Switzerland, (poster session).