Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 1 (March), pp. 8-20

Genetic Algorithm for Fuzzy Neural Networks using Locally Crossover

D. Arotaritei

Dragos Arotaritei
"Gr. T Popa" University of Medicine and Pharmacy
Romania, 700115 Iasi
E-mail: dragos_aro@yahoo.com

Abstract: Fuzzy feed-forward (FFNR) and fuzzy recurrent networks (FRNN) proved to be solutions for "real world problems". In the most cases, the learning algorithms are based on gradient techniques adapted for fuzzy logic with heuristic rules in the case of fuzzy numbers. In this paper we propose a learning mechanism based on genetic algorithms (GA) with locally crossover that can be applied to various topologies of fuzzy neural networks with fuzzy numbers. The mechanism is applied to FFNR and FRNN with L-R fuzzy numbers as inputs, outputs and weights and fuzzy arithmetic as forward signal propagation. The α-cuts and fuzzy biases are also taken into account. The effectiveness of the proposed method is proven in two applications: the mapping a vector of triangular fuzzy numbers into another vector of triangular fuzzy numbers for FFNR and the dynamic capture of fuzzy sinusoidal oscillations for FRNN.
Keywords: rules, figures, citation of papers, citation of books, examples.

1 Introduction

Fuzzy neural networks are usually based on neural network architecture [1] with fuzzification of inputs, outputs, weights, or rules that are applied using fuzzy systems [2]. GA is one of the alternatives to gradient techniques (see [3], [4] and [5]) that can be used to develop effective end efficient learning algorithm for fuzzy neural networks.

In [6], the authors proposed a learning algorithm for FFNN with fuzzy inputs and weights called "fuzzy delta rule". This algorithm is derived by replacing fuzzy differentiation with real differentiation, this fact being point out by the authors.

Architecture of multilayer feedforward algebraic neural network for fuzzy input vectors, fuzzy outputs and fuzzy weights is proposed in (see [7] and [8]). The input-output relations in this fuzzy neural network, using *max/min* operators, are defined by the extension principle of Zadeh. In these papers, the authors used only symmetric triangular fuzzy numbers (see [7] and [8]). A cost function for the level sets (α-cuts) of fuzzy outputs and fuzzy target is defined. The fuzzy output from each unit in the fuzzy neural network is computed for level sets of fuzzy inputs, fuzzy weights and fuzzy biases. The learning algorithm, based on the cost function, uses a gradient descent method. A heuristics method, in order to respect the shape of triangular fuzzy weights, during the learning stage is proposed (see [7] and [8]).

The same architecture but with non-symmetric and trapezoidal fuzzy numbers has been used in [9]. The authors proposed and "trail-and-error" algorithm using adaptive updated based on gradient techniques developed in the frame of fuzzy arithmetic [9]. The α-cuts and fuzzy biases are also taken also into account [9]. A learning algorithm to fully connected fuzzy recurrent neural network (FCFRNN) has been developed in [10]. The algorithm is developed for non-symmetric triangular fuzzy numbers and gradient techniques developed in the frame of fuzzy arithmetic. The symmetric fuzzy numbers and crisp numbers are considered particularly cases [10].

Few papers refer to GA for adjusting fuzzy weights. In [11] the authors present a GA mechanism for adaptation of fuzzy weighs as LR-fuzzy numbers for FFNR. The single chromosome includes all the fuzzy weights in binary format. The single point crossover (split point - SP) is applied to entire chromosome that represent the coded binary value of al the triangular fuzzy weights expressed as L-R fuzzy number with n parameters [11].

A structure of fuzzy recurrent neural network with GA learning algorithm is proposed in [12]. All the fuzzy weights are coded in a genome, a single string of bits that represent L-R fuzzy numbers. This string represents an individual,a potential solution of the problem. The connections in the recurrent structure follow a limited connection that is the structure is not fully recurrent structure in the sense of [14]. In both papers (see [11] and [12]) the fitness is the performance index and the objective is to minimize this error.

This paper presents a GA algorithm based on learning mechanism that adjusts the fuzzy weights represented as L-R fuzzy numbers. The fuzzy neural networks computing operations are described in the fuzzy arithmetic framework. The proposed method is applied to two structures: feedforward structure and fully recurrent structure.

The proposed method proved to be better than the existing algorithms in term of number of generations until the optimal solution has been reached.

2 The Fuzzy Arithmetic Framework and the Fuzzy Neuron

In this paper we used the input-output relations fuzzy neural network,using *max-min* operators, defined by the extension principle of Zadeh, and applied in previously papers (see [9] and [10]). All the nodes have a *semilinear (sigmoidal)* transfer function. We denote a triangular fuzzy number as follow:

$$\tilde{X} = (x^L, x^C, x^R) \tag{1}$$

A non-standard algebraic operation between two TFN,$\tilde{A}$ and $\tilde{B}$ is defined, as in (see [9] and [10]), by:

$$\tilde{C} = \tilde{A}\tilde{\Diamond}\tilde{B} = \tilde{X} = (c^L, c^C, c^R) = (min(P), a^C \Diamond b^C, max(P)) \tag{2}$$

$$P = a^L \Diamond b^L, a^L \Diamond b^R, a^R \Diamond b^L, a^R \Diamond b^R \tag{3}$$

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

$$[\tilde{X}_h] = \{x | \mu_x \geq h, h \in \mathbb{R}\} \quad for \quad 0 < h \leq 1 \tag{4}$$

In the equations (2)-(3), $\Diamond \in \{+, -, \cdot, /\}$ are the operations in classic arithmetic, and $\tilde{\Diamond} \in \{\tilde{+}, \tilde{-}, \tilde{\cdot}, \tilde{/}\}$ are the corresponding modified operations in fuzzy arithmetic, defined according to [7]. The division is restricted to division by a non-zero numbers, that is $\{b^L, b^C, b^R\} \neq 0$ Similar relations can be used to get the α-cuts.

We consider a fuzzy neuron (FN) that perform input-output operations. The FN get an input vector fuzzy signal $\tilde{x} = (\tilde{x_1}, \tilde{x_2}, ..., \tilde{x_N}\}$ nd transfer it after multiplication with fuzzy weights $\tilde{w} = (\tilde{w_1}, \tilde{w_2}, ..., \tilde{w_N}\}$ by activation function to output. The total input of the fuzzy neuron is given by (see [9] and [10]):

$$\tilde{s} = \tilde{w_1}\tilde{\cdot}\tilde{x_1}\tilde{+}\tilde{w_2}\tilde{\cdot}\tilde{x_2}...\tilde{+}\tilde{w_N}\tilde{\cdot}\tilde{x_N} = \sum_{i=1,N}^{fuzz} \tilde{w_i}\tilde{\cdot}\tilde{x_i}\tilde{+}\tilde{b} \tag{5}$$

$$\tilde{y} = f(\tilde{s}) \tag{6}$$

In the equations above the sum is in the fuzzy arithmetic framework (2)-(3) and $\tilde{b}$ is the fuzzy bias. The output is computed by (5) and the extension principle of Zadeh [13]. We used sigmoidal function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

Due to monotonic increasing property of the sigmoidal function f, the TFN shape is preserved for the output of the neurons. In section 3 and 4 we will apply these assertions to two types of neural networks architectures: feedforward neural networks [1] and fully connected recurrent neural networks [14].

3 Feed-Forward Fuzzy Neural Networks

There are three basically types of fuzzy neural networks depending on the type of fuzzification of inputs, outputs and weights (including biases): fuzzy weights and crisp inputs, crisp weights and fuzzy inputs and fuzzy weights and fuzzy inputs [11]. In what follows we consider the most complete fuzzification of neural networks: fuzzy inputs, fuzzy weights and fuzzy outputs.

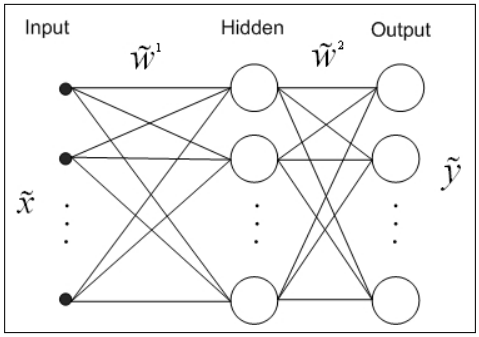

Figure 1: A three layered FFNN

In Fig. 1 we showed an example of FFNN, a three layer FFNN. FFNN propagate the signal layer by layer until the signal reach the output layer. For each k layer we have:

$$\tilde{s}_j^k = \sum_{i=1,N_k}^{fuzz} \tilde{w}_{ji}^k \tilde{\cdot} \tilde{x}_i \tilde{+} \tilde{b}_i \tag{8}$$

$$f(x) = f(\tilde{s}_k^j) \tag{9}$$

In the equations above, j is the fuzzy neuron from k layer and i-th is the fuzzy neuron from k-1 layer. The training of FFNN is to adapt the network in order to mapping the known inputs to target outputs. The objective of the learning algorithm is to minimize the error measure between the desired outputs and the real outputs. The learning algorithm must adjust the fuzzy weights based of error measure in order to achieve this objective.

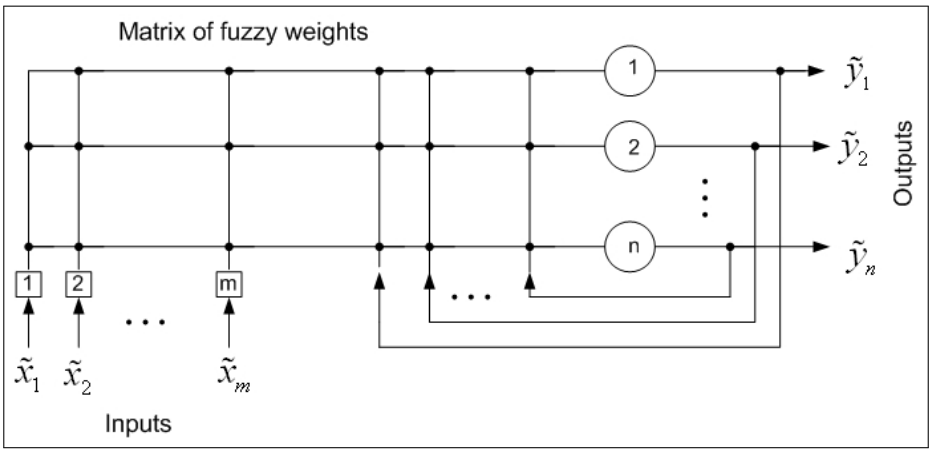

Figure 2: Fully connected FRNN

4 Fuzzy Recurrent Neural Networks

Let the FRNN network have n units and m inputs. Each bias allocation will be seeing as an input line whose value is always $\tilde{1} = (1, 1, 1,)$.

Let the FRNN network have n units and m inputs. Each bias allocation will be seeing as an input line whose value is always $\tilde{1} = (1, 1, 1,)$. We denote by (I, U, T) the set of indices for input units, output units and target units. Lets $\tilde{x}(t)$, $\tilde{y}(t)$ and $\tilde{d}(t)$ denote the inputs, the outputs and the targets (if exists) of the units in RNN at time t, respectively. We denote, similar to [14], a generalized $\tilde{z}(t)$ as follow:

$$\tilde{z}_k = \begin{cases} \tilde{x}_k(t) & \text{if } k \in I \\ \tilde{d}_k(t) & \text{if } k \in T(t) \\ \tilde{y}_k(t) & \text{if } k \in U - T(t) \end{cases} \tag{10}$$

The basic algebraic fuzzy neuron is based on the operations defined in (2)-(5), where the sum above denoted by $\sum^{fuzz}$ is the sense of the algebraic sum (4) and $\tilde{b}_k$ is the bias of the k-th unit.

$$\tilde{s}_k(t) = \sum_{r \in U \cup I}^{fuzz} \tilde{w}_{kr} \tilde{\cdot} \tilde{z}_r(t) = \tilde{w}_{k1} \tilde{\cdot} \tilde{z}_1 \tilde{+} ... \tilde{+} \tilde{w}_{k,m+n} \tilde{\cdot} \tilde{z}_{m+n}(t) \tilde{+} \tilde{b} \tag{11}$$

$$\tilde{y}_k(t+1) = f(\tilde{s}_k(t)) = f(s_k^L, s_k^C, s_k^R) \tag{12}$$

The missing connection in the RAFNN architecture is simply represented by a zero weight value (fuzzy number zero).

5 GA with Locally Crossover (GALC)

GA are optimization techniques based on principles of mechanism of natural evolution (see [3] and [4]). GA are working on possible space of solutions (usually codified by chromosomes) in order to find the best candidate suitable for a particularly problem. The fitness is a measure of performance of the individual in order to achieve to desired objectives. The objective function can be the maximization of fitness or minimization of fitness.

Two problems arise from GA usage for solving the optimization problems: the choice of the fitness function and the codification the values of parameters that must optimized into an

individual chromosome. The range of values that are part of this codification play an important role related to both convergence speed of GA and the accuracy of the results.

In our approach we will use the minimization of objective function that is a measure of output errors related to fuzzy numbers. This function is usually named the fitness function.

The general fitness function can be defined based distance among desired fuzzy numbers and actual fuzzy numbers at outputs [11]. However, in our application we will use a more practical measure based on Hamming distance. By this objective function we estimate more intuitive the difference between desired and real outputs.

$$D_{fitness} = min \sum_{i=1}^{K} (|\tilde{y}_i^d - \tilde{y}_i|) \tag{13}$$

GA uses binary encoded in order to represent genes or chromosomes. Each binary value (0 or 1) is named allele [3]. The value of solutions is mapped as binary string in the process of encoding using linear of nonlinear functions. We used the representation of TFN as LR-type fuzzy numbers with n parameters [13]. Each parameter is coded as binary string of a specified length.

$$\tilde{A} = (m, dL, dR) \tag{14}$$

In the case of no α-cuts, each fuzzy weight is represented by three chromosomes corresponding to central value w_{jiC}^k , and L, R values that represents the left and the right spread are denoted by dw_{jiL}^k and dw_{jiR}^k (Fig. 3). The values are coded into binary string in a predefined range.

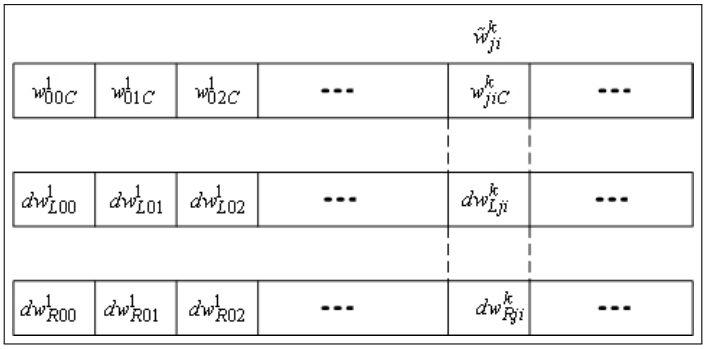

Figure 3: Chromosomes of one individual

In the case of FFNN, $\tilde{w}_{ji}^k$ is fuzzy connection weight between the *j*-th neuron from k layer with *i*-th neuron from *k*-1 layer.

Each individual has three chromosomes that represent the strings for the three types of values. We must remark that in TFN the order must be preserved:

$$w_{ji}^k - dw_{jiL}^k \leq w_{ji}^k \leq w_{ji}^k + dw_{jiR}^k, \quad dw_{jiL}^k \geq 0, dw_{jiR}^k \geq 0 \tag{15}$$

The selection process has two stages. The first stage is selection of two parents for crossover using a known schema: tournament, roulette or stochastic sampling. After this global selection, we perform a locally selection inside of the chromosome. This selection is inspired by the fact observed in crisp feedforward neural networks that not all the weights have the same contribution to forward propagation of the signal. Usually, the weights from k layer has a different contribution than the weights from k+1 layer. Conversely, it is natural to suppose that some parameters give a more contribution to best solution than others, so a fine locally adjustment can improve the global solution.

The locally selection selects the weights, the locations in the chromosome where the corresponded weight is coded in binary string for locally crossover (Fig 4).

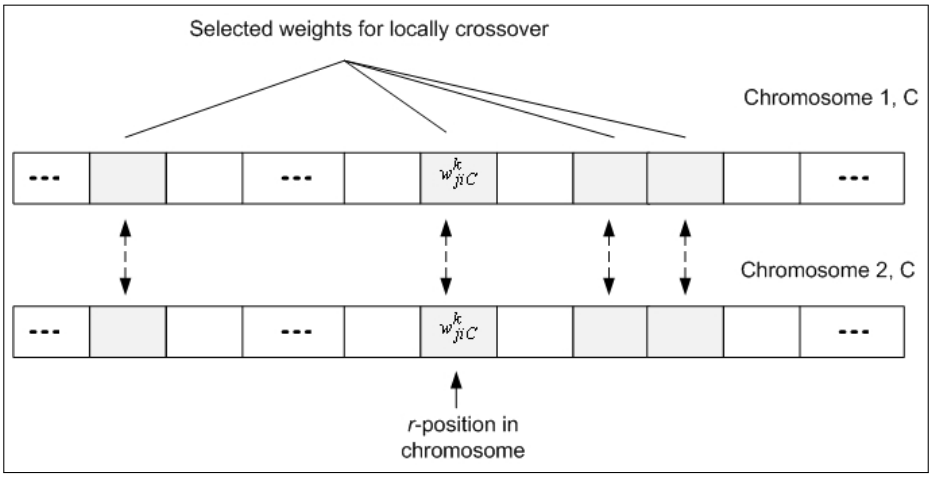

Figure 4: Selection of genes (weights) for locally crossover for C value

The same selected location are taken into account in one step for all three values, as in (13). The representation is given in Fig. 5.

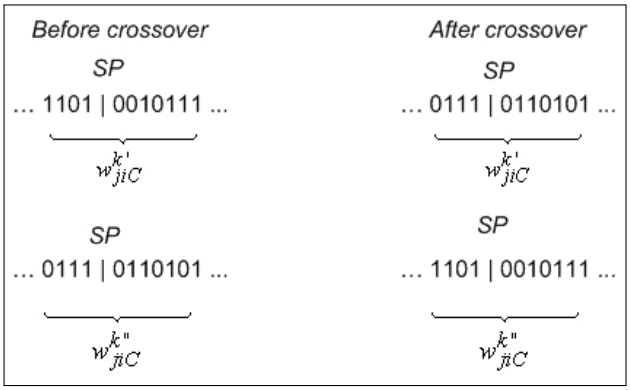

Figure 5: Selection of genes (weights) for locally crossover for dL, dR values

The crossover operation pick the selected genes and mate them in order to produce two offspring genes. The split point (*SP*) is chosen randomly, the same for both parents. The example from Fig. 6 illustrate the locally crossover. The same *SP* is applied to corresponding gene that code the left and right spread of the selected weights.

Figure 6: Locally crossover between selected genes

Mutation is made globally al level of entire chromosomes. The mutation pick and bit (*allele*) with a probability p_m and flip this value to 1 if the value is 0 and to 0 if the value is 1. Mutation

implements a random search at global level. By p_m, the level of search can be modified, the greater is p_m, the level of search in the solution space is greater.

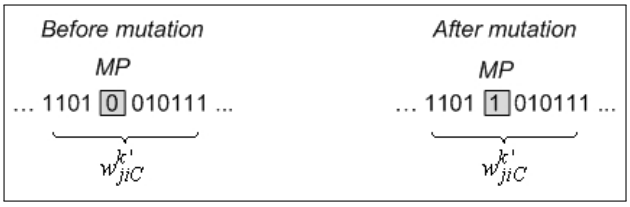

Figure 7: Global mutation

The GALC algorithm is based on the practical following observation. Due to nonlinear aspect of the transfer function of the neuron, the changes in some weights can have a greater contribution to result than the changes in the other weights. Also, the impact of this changes for FFNN can decrease or increase in the forward process (Fig. 8).

GALC propose to use this observation in order to "encourage" the more contributing weights to be selected in order to do crossover and evolve faster meanwhile the rest of weights will evolve slowly, based only on mutations. For the same dx ($dx_1 = dx_2$), we can have $dy_2 > dy_1$ in same cases with one or two magnitude order.

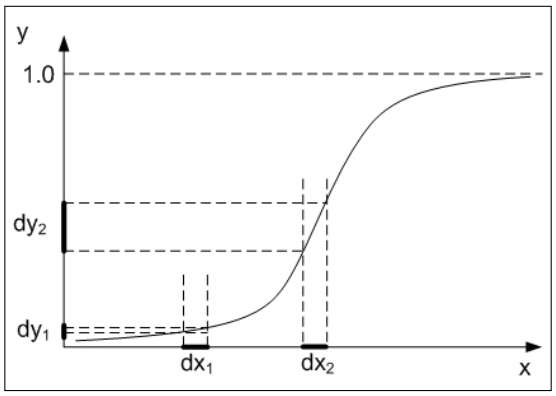

Figure 8: The influence of function of activation over quantitative adaptation of the weight

Let denote by NN the total number of neurons in the FFNN. The proposed algorithm based on basic GA procedures is summarized below.

Step 1: Linear mapping of the solution space (fuzzy weights)into chromosomes (binary string) that represent individuals of the population P.

Step 2: Initialize the population P with random solutions (uniform distribution). The population has N individuals that represents $3*N$ set of chromosomes for TFN with no fuzzy bias and $3*N + 3*NN$ set of chromosomes for TFN with fuzzy bias.

Step 3: Evaluate the fitness of the population P.

Step 4: Evaluate the stop criteria. If the stop criteria is fulfilled, go to *Step 10*.

Step 5: Store the best individual of the current generation and the best individual of all the generations.

Step 6: Generate a new population using selection operator according to selection schema based on fitness values of the chromosomes.

Step 7: Random selection of genes for locally crossover. The selected genes are a percent pr from total number of the genes that represent a chromosome.

Step 8: Locally crossover for selected genes and creation of new generation of the population. The corresponding genes are mated with probability pc and the results offspring replace the parents in the new population.

Step 9: Global mutation over chromosomes with random probability p_m.

Step 10: Map the best individual of all the generations into solution, the fuzzy weights of the FFNN.

The stop criterion can have different forms. One of the most common is a predefined number of generations. Another one can be the stop of the process, before a maximum number of generations if no significant improvement has been made in the best individual. In our application we used the first criterion.

The GALC algorithm can be easily extended to α-cuts. For each h-level we must allocate 2 chromosomes that correspond to dL and dR values al level h (Fig. 9). The central value is unchanged from one level to next level. We start with the base level (h=0) where we adapt the L, C and R values using L-R type representation (12). Next we consider the C value and we must adapt the left and right spread at the level h=1,2,3,... n.

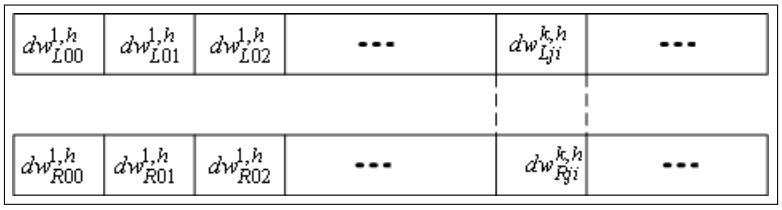

Figure 9: Chromosomes of one individual for h-level

Because of *max/min* operators and multiplication operation, the shape of membership for fuzzy weights (and biases) is a curved triangle. Restrictions must be made in order to avoid non desired occurrence in the weights adaptation process.

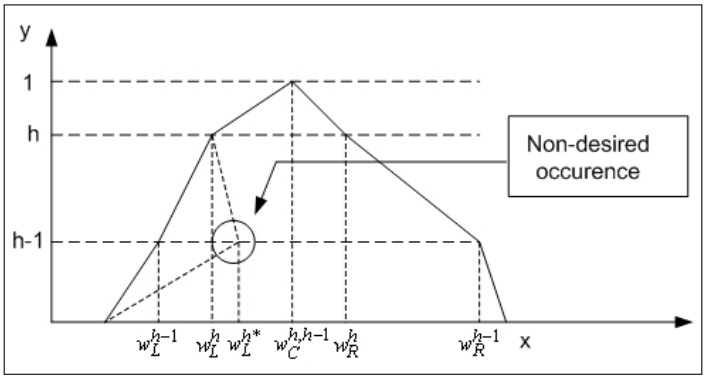

Figure 10: A non-permitted occurrence in the weights adaptation process

The restrictions are:

$$dw_{jiL}^{k,h} \leq dw_{jiL}^{k,h-1}, \quad dw_{jiR}^{k,h} \leq dw_{jiR}^{k,h-1} \tag{16}$$

These restrictions are included in the algorithm in the following way. If the after the crossover, the children accomplish the rules (15) the local crossover operation is validated and the children are selected in the next generation depending on the fitting. If the children doesn't accomplish the rules (15) the local crossover operation is canceled and the selected weights remain unchanged.

6 Applications

6.1 A mapping of non-symmetric triangular fuzzy numbers with FFNN

We apply the proposed method to approximate realisation of non-linear mapping of fuzzy numbers. The TFN from input space have the bases inside the interval [0, 1] and are mapping to TFN which have the bases in [0,+1]. The FFNN has 3 inputs, 6 neurons in the hidden layer and 3 neurons in the output layer.

The FFNNs fuzzy error can be defined as a distance between fuzzy desired output and the fuzzy real output [15]. Because we are interested in accuracy of outputs for all the points that represents the fuzzy number, we express the error measure:

$$e_i^{L,C,R} = \left| d_i^{L,C,R} - y_i^{L,C,R} \right| \tag{17}$$

$$J_{total}(t) = \sum_{q=L,C,R;i=1...N} e_i^q \tag{18}$$

The inputs and the target (desired outputs are):

$\tilde{x}$ = (0.2,0.1,0.1),(0.3,0.1,0.2),(0.6,0.2,0.1)
$\tilde{d}$ = (0.4,0.2,0.2),(0.8,0.2,0.1)

We generate an initial population of random binary strings that represent the chromosomes. The population is set to P=100 individuals, $p_c = 0.7$, $p_m = 0.3$, $gen = 1000$ (the maximum number of generations). We set $p_r = 20\%$, that is from $6 \times 3 + 6 \times 2 = 30$ weights, and at each generation we select random a number of 0.2*30 = 6 weights.

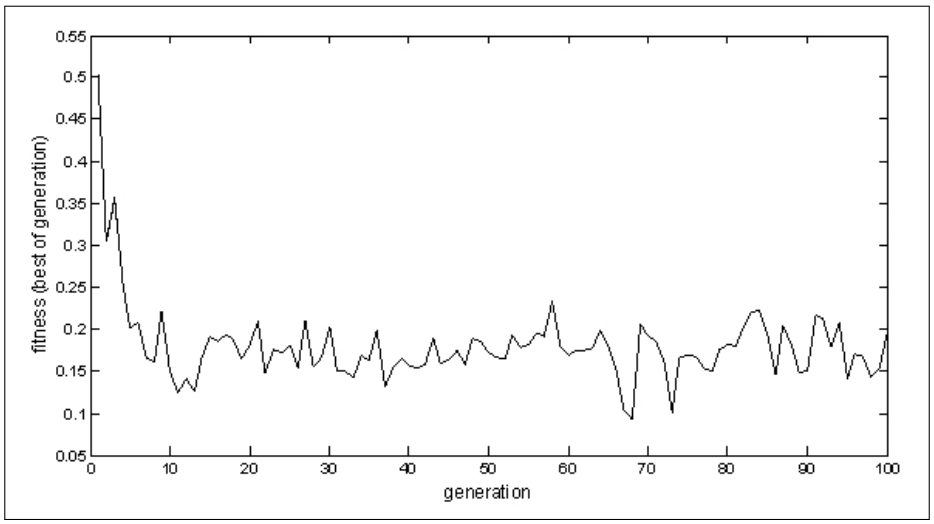

Figure 11: The error evolution (FFNN example)

The best result is obtained in gen= 68, when J = 0.0927 (Fig. 11). The best fitness individual has the lowest value of fitness (we used the best individual - minimum fitness approach). The desired and the real output values are:

$\tilde{d}$ = (0.4,0.2,0.2),(0.8,0.2,0.1)
$\tilde{x}$ = (0.4055,0.1945,0.2176),(0.7714,0.1636,0.1119)

We made experiments with α-cuts, also. In our experiment we used three α-cuts, at levels h=0, h= 0.33, and h = 0.66. The experimental results for one of the fuzzy weights (k=2, j =1, i=2) and the levels of the α-cuts mentioned above is showed in Fig. 12.

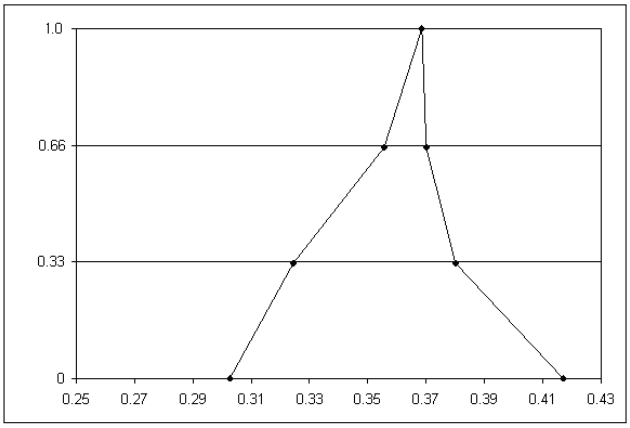

Figure 12: The fuzzy weight $\tilde{w}_{12}^2$ with three α-cuts)

6.2 FRNN - Learning of defined dynamic

The algorithm designed to train an arbitrary network dynamics can be tested using an interesting class of behaviors, the oscillations [14]. A pair of logistic units (Fig. 13) was used to learn a trajectory of a TFN, composed by three sine waves with the same frequencies and the same phase but different range corresponding to L, C and R values of the TFN (Fig. 14).

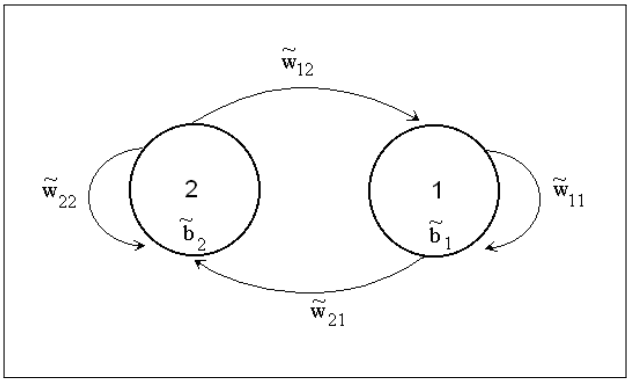

Figure 13: Fuzzy sine wave network.

The overall error measure at each sample time t is given by:

$$J(t) = \sum_{k \in U} \left[\left[\left| e_k^L(t) \right| + \left| e_k^R(t) \right| \right] + \left| e_k^C(t) \right| \right] \tag{19}$$

$$J_{total}^{fuzz}(t_0, t_1) = \begin{cases} d_k^{L,C,R}(t) - y_k^{L,C,R}(t) & \text{if } k \in T(t) \\ 0 \quad otherwise \end{cases} \tag{20}$$

The network performance measure over trajectory is defined by:

$$J_{total}^{fuzz}(t_0, t_1) = \sum_{t=t_0+1}^{t_1} J(t) \tag{21}$$

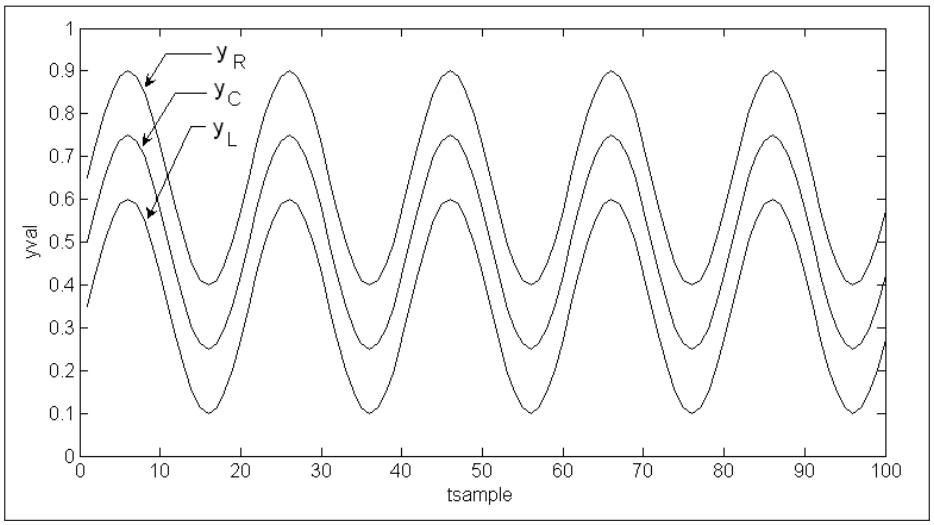

Figure 14: The training sine waves (targets).

In our particular case we consider a complete trajectory a sinus period (2π). The best individual is obtained after 176 generation after 10 starting new generations and epochs.

Usually, the stable solution is obtained in less than 500 generation with uniform random distribution of initial population made from 100 individuals.

Table 1: RAFNN parameter for sine waves

	L	C	R
$\tilde{w}_{11}$	0.3026	0.3687	0.4169
$\tilde{w}_{12}$	-0.4405	-0.2509	-0.0286
$\tilde{w}_{21}$	-0.2506	-0.0076	-0.0035
$\tilde{w}_{22}$	0.2214	0.4054	0.5207
$\tilde{b}_1$	-0.3521	-0.3028	-0.2499
$\tilde{b}_2$	-0.1499	-0.0085	0.0772

Stable, the sine like oscillation are obtained for sine frequencies above 20 network ticks per cycle with the 20,000 ticks or less. The output of the unit 1 in the absence of a teacher imposed to unit 2, after stable oscillation had been established is shown in Fig. 15

As we are the expectations, taking into account the results from [14], the frequency of the free-running logistic is around 8% lower than the trained frequency. The amplitude is smaller and it is diminishing in time.

This example proved that the FRNN is able to learn a dynamics that is the sinusoidal waves. The output of the unit 1 has at each step a triangular fuzzy number, and values y^L, y^C and y^R have a sinusoidal shape. It is clear from results that the output error is important, but out target was to learn predefined dynamic using minimization of error. The algorithm stopped when the performance not improve during the last S generations (in our case, S=3).

7 Conclusions

We proposed to use a novel method and a new learning algorithm, GALC for fuzzy neural networks with feedforward and fully recurrent architecture. The method and a learning algorithm proved by theoretical and experimental approach to be effective with acceptable stable solution.

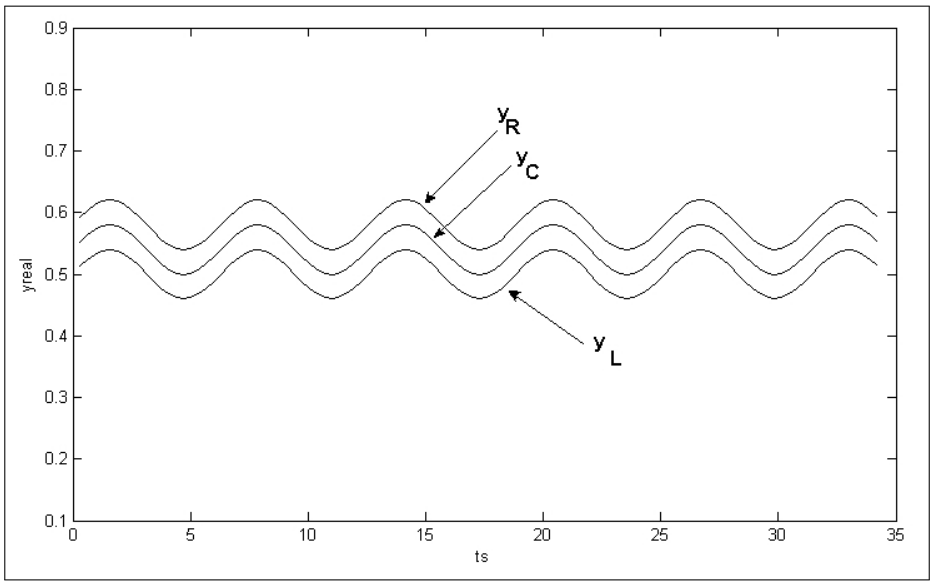

Figure 15: The output of the unit 1.

The solution is usually obtained in a shorter time (in terms of number of generation) than the actually GA based learning algorithms. Moreover, the algorithm is effective also for fully fuzzy recurrent neural networks as they has been defined in section 4.

Different from (see [11] and [12]) we proposed a complete fuzzification of the neural network in the sense that the sum of the fuzzy weighted inputs are also in the fuzzy arithmetic framework (see [11] and [12]).

The proposed method offer a possibility to work with long strings. It is know that in the case of very long chromosome, the global crossover can slow considerably the algorithm convergence. That is a very large number of generation is required in order to achieve the global solutions. The proposed method that start with a larger number of individual in the initial population and selective locally adaptation could improve the performance of convergence by pressure of these parameters that has a greater importance in the best solution.

Bibliography

[1] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1998.

[2] D.J. Dubois, H. Prade, *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, 1980.

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, 1989.

[4] J. R .Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[5] O. Cordon, V. Herrera, M. Lozano, On the Combination of Fuzzy Logic and Evolutionary Computation: a short review and Bibliography, in: W. Pedrycz (Ed.), *Evolutionary Computation*, Kluwer Academic Publishers, Dordrecht, pp.33-56, 1997.

[6] Y. Hayashi et. al., Fuzzy Control Rules in Convex Optimization, *Fuzzy Neural Networks with Fuzzy Signals and Weights IJCNN'92*, Vol. 2, pp. 165-195, 1992.

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

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

[9] H.N. Teodorescu, D. Arotaritei, E. Lopez Gonzales, A General Trail-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, Vol. 1, pp. 8-12, 1996.

[10] D. Arotaritei, Recurrent Algebraic Fuzzy Neural Networks based on Fuzzy Numbers, *Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, Vol. 5, pp. 2676 - 2680, 2001.

[11] R.A. Aliev, B. Fazlollahi, R.M. Vahidov, Genetic algorithm-based learning of fuzzy neural network, Part 1: feed-forward fuzzy neural networks, *Fuzzy Sets and Systems*, Vol. 118, Issue 2, pp. 351-358, 2001.

[12] R. A. Aliev, R. R. Aliev, B. G. Guirimov, K. Uyar, Recurrent Fuzzy Neural Network Based System for Battery Charging, *Lecture Notes in Computer Science*, Vol. 4492, pp. 307-316, 2007.

[13] L.A. Zadeh, Fuzzy Sets, *Inform. Control 8*, pp. 338-353, 1965.

[14] R.J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, Vol. 1, Issue 2, pp. 270-280, 1989.

[15] C. Chakraborty, D. Chakraborty, A theoretical development on a fuzzy distance measure for fuzzy numbers, *Mathematical and Computer Modelling*, Nr. 43, pp. 254-261, 2006.