Evolutionary Strategies and Genetic Algorithms for Dynamic Parameter Optimization of Evolving Fuzzy Neural Networks

L. Minku, T. Ludermir
Federal University of Pernambuco
Center of Informatics
P.O. Box 7851, Cidade Universitária, Recife-PE, Brazil, 50732-970
{flm,tbl}@cin.ufpe.br

Abstract- Evolving Fuzzy Neural Networks are usually used to model evolving processes, which are developing and changing over time. This kind of network has some fixed parameters that usually depend on presented data. When data change over time, the best set of parameters also changes. This paper presents two approaches using Evolutionary Computation for the on-line optimization of these parameters. One of them utilizes Genetic Algorithms and the other one utilizes Evolutionary Strategies. The networks were used to Mackey-Glass chaotic time series prediction with changing dynamics. A comparative study is made with these approaches and some variations of them.

1 Introduction

Evolving[1] processes are processes that are developing, changing over time in a continuous manner. There are many real world problems that are evolving processes, *e.g.* biological data processing, electricity load forecasting and adaptive speech recognition. These processes are difficult to model because some of their parameters may not be known *a priori*, unexpected perturbations or changes can happen at certain time during development and they are not strictly predictable in the longer term [Kasabov 2003].

Evolving Connectionist Systems (ECOSs) constitute a paradigm created to make easier the evolving processes modeling task. Evolving Fuzzy Neural Networks (EFuNNs) [Kasabov 2001] form an example of a class of ECOSs that facilitate the representation and knowledge extraction using fuzzy rules.

Although ECOSs can evolve their structures as incoming data arrives, they still have some fixed parameters that are not adjusted during the learning. At this work two approaches using Evolutionary Computation (EC) to on-line optimization of these parameters were implemented. One of them uses a Genetic Algorithm (GA), the other one uses an Evolutionary Strategy (ES) and both were based in the method introduced by [Kasabov, Song and Nishikawa 2003].

Experiments were done to compare the results of the optimization of some of the fixed parameters of EFuNNs using these approaches and some variations of them. The analysis also compares the effect of the optimization using the full power of these approaches with the effect of their execution in a similar way to [Kasabov, Song and Nishikawa 2003]. The EFuNNs were used to chaotic time series prediction with changing dynamics.

[1]Notice that the term "evolving" will be used with a different mean from "evolutionary".

2 Evolving Connectionist Systems and Fuzzy Neural Networks

The ECOSs presented in [Kasabov 2003] are systems constituted by one or more neural networks. They have the following characteristics [Kasabov, Song and Nishikawa 2003]:

- They facilitate evolving processes modeling task.
- They facilitate knowledge representation and extraction.
- They have the following learning characteristics:
  - Lifelong: they learn from continuously incoming data in a changing environment during its entire existence.
  - On-line: they learn each example separately while the system operates (usually in real time).
  - Incremental: they learn new data without totally destroying the patterns learned before and without the need to make a new training on old and new data together.
  - Fast, possibly through just one pass of data propagation.
  - Local: they locally partition the problem space, allowing fast adaptation and tracing evolving processes over time.
- They learn as both individual systems, and as part of an evolutionary population of such systems.
- They have evolving structures and use constructive learning.
- They evolve in an open space, not necessarily of fixed dimensions.

EFuNNs [Kasabov 2001] are a class of ECOSs that join together the neural networks functional characteristics to the expressive power of fuzzy logic. They have a five-layer architecture as it is shown in figure 1. The first layer represents the input vector, the second represents fuzzy quantification of the input vector, the third represents associations between fuzzy input space and fuzzy output space, the fourth represents fuzzy quantification of the output vector and the fifth represents the output vector.

The learning occur at the rule nodes layer. Each node *rj* of this layer is represented by two vectors of connection weights (*W1(rj)* and *W2(rj)*). *W1* represents the coordinates of the nodes in the fuzzy input space and it is adjusted

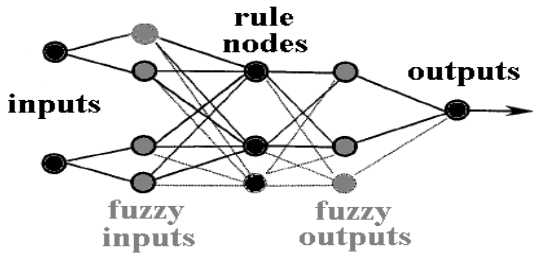

Figure 1: EFuNN architecture

through unsupervised learning. *W2* represents the coordinates of the nodes in the fuzzy output space and it is adjusted through supervised learning. The learning rules are the following:

- $W1(r_j(t+1)) = W1(r_j(t)) + lr1(r_j(t)) * (x_f - W1(r_j(t)))$
- $W2(r_j(t+1)) = W2(r_j(t)) + lr2(r_j(t)) * (y_f - A2) * A1(r_j(t))$

  Where: x_f and y_f are the fuzzy input and fuzzy output vectors; $lr1(r_j(t))$ and $lr2(r_j(t))$ are the learning rates for the $W1$ and $W2$ weights of the node r_j at time t; $A2$ is the fuzzy output activation vector and $A1(r_j(t))$ is the activation value of the rule node r_j at time t.

The EFuNN learning algorithm is briefly described below. It is recommended to read [Kasabov 2001] to get more details.

Algorithm 2.1 (EFuNN Learning Algorithm)

1. Set initial values for the following system parameters: number of membership functions; initial sensitivity threshold S of the nodes (it is also used to determine the initial radius of the receptive field of a node r_j , when it is created ($R(r_j) = 1 - S$); error threshold E; aggregation parameter $Nagg$; pruning parameters OLD and Pr; *m-of-n* value (number of highest activation nodes used in the learning); maximum radius of the receptive field $Mrad$; rule extraction thresholds $T1$ and $T2$.
2. Set the first rule node r_0 to memorize the first example (x, y):

   $$W1(r_0) = x_f \text{ and } W2(r_0) = y_f$$

   where x_f and y_f are the vectors of fuzzy quantification of the vectors x and y, respectively.
3. Repeat for each new input-output pair (x, y) presentation:
   (a) Determine the local normalized fuzzy distance D between x_f and the $W1$ weights. The distance D between two fuzzy vectors $x1$ and $x2$ is calculated as following:

   $$D(x1, x2) = subabs(x1, x2)/sumabs(x1, x2)$$

   where $subabs(x1, x2)$ is the sum of all absolute values of the vector obtained after subtraction of the fuzzy vectors $x1$ and $x2$ and $sumabs(x1, x2)$ is the sum of all absolute values of the vector obtained after sum of the fuzzy vectors $x1$ and $x2$.

   (b) Calculate the activations $A1$ of all rule nodes. An example of how it can be calculated is:

   $$A1 = 1 - D(W1(r_j), x_f)$$

   (c) Select the rule node r_k that has the smallest distance $D(W1(r_k), x_f)$ and that has activation $A1(r_k) >= S(r_k)$. In the case of *m-of-n* learning, select *m* nodes instead of just one node.

   (d) If this node does not exist

   i. Create a new rule node for (x_f, y_f).

   (e) Else

   i. Determine the activation $A2$ of the output layer and the normalized output error $Err = subabs(y, y')/Nout$, where y is the desired output, y' is the obtained output and $Nout$ is the number of nodes of the output layer.

   ii. If $Err > E$

   A. Create a new rule node for (x_f, y_f).

   iii. Else

   A. Apply the learning rules to $W1(r_k)$ and $W2(r_k)$ (in the case of *m-of-n* learning, the rules are applied to the *m* rule nodes).

   (f) Apply aggregation procedure after the presentation of Nagg examples.

   (g) Update the parameters $S(r_k)$, $R(r_k)$, $Age(r_k)$ and $TA(r_k)$. $TA(r_k)$ can be, for example, the sum of the activations $A1$ obtained for all examples that r_k accommodates.

   (h) Prune rule nodes, if necessary, according to OLD and Pr.

   (i) Extract rules, according to $T1$ and $T2$.

According to algorithm 2.1, there are parameters that are adjusted during the learning (rule nodes and their weights) and parameters that do not change during the learning, but defining it (number of membership functions, *E*, *Nagg*, *OLD*, *Pr*, *m-of-n* value, *Mrad*, *T1* and *T2*). All parameters that define the learning must be determined according to the used data set [Kasabov, Song and Nishikawa 2003].

3 On-line Optimization of ECOSs Parameters Using Evolutionary Computation

Through the use of different parameters of the kind that define the learning, ECOSs attain different performances and different weights are learned. Usually the optimal parameters set depends on the input and output presented data. The objective of ECOSs on-line parameters optimization is the determination of the best parameters set for the moment, at the same time that learning occurs.

A method that uses EC to optimize ECOSs parameters

was introduced in [Kasabov, Song and Nishikawa 2003]. It uses a fitness function based only on the Root Mean Squared Error (RMSE) generated by the prediction made by EFuNN over an data set. In this way, if the allowed values interval for the parameters being optimized is not small enough, the networks generated are very big in comparison with the number of training examples presented. Moreover, although the approach can be used with other evolutionary algorithms, [Kasabov, Song and Nishikawa 2003] made tests just with GAs, that are not always the best evolutionary technique to the problem at hand.

At this work two approaches that use EC to on-line optimization of EFuNN parameters were implemented. One of them uses a GA, the other one uses an ES and both are based on the approach introduced by [Kasabov, Song and Nishikawa 2003]. The following subsections describe these approaches.

3.1 Optimization Based on Genetic Algorithm

This approach uses a GA to do EFuNNs parameters optimization. As the GA utilized in [Kasabov, Song and Nishikawa 2003], the used representation is binary, the mutation is bitwise bit-flipping, the recombination is one-point crossover, the parents selection is fitness proportional, using the roulette wheel method and the survivor selection is generational. It is recommendable to read [Eiben and Smith 2003] for an explanation about GAs. Algorithm 3.1 shows the details of the approach implemented at the present work.

Algorithm 3.1 (Optimization Based on GA)

1. Initialize the population with individuals created randomly choosing values for each bit of the genotype (the genotype codifies parameters to be optimized).
2. Define a window with *P* time points for EFuNN learning (training time window) and, optionally, define a window with *P* time points for testing (testing time window).
3. Repeat for all training/testing data:
   (a) Apply the learning algorithm to each EFuNN of the population using all *P* points of the training time window and the parameters codified by the genotype of the individuals.
   (b) Determine each EFuNN RMSE (using the prediction over the points of the same time window used for learning or using the prediction over the testing time window points, if there is one) and use the weighted sum of the RMSE and the size of the network as fitness value to be minimized ($fitness = W_{rmse} * RMSE + W_{size} * size$). The W_{rmse} and W_{size} weights are pre-defined by the user.
   (c) Make parent selection using roulette wheel method and probabilities determined through subtraction of the biggest fitness of the generation and the fitness of the individual.
   (d) Apply crossover and mutation with probabilities *Pc* and *Pm*, respectively, to generate new individuals (parents rule nodes are inherited by children).
   (e) Apply generational survivor selection.
   (f) Shift the training and the testing (if there is one) time window (the new time windows are composed by $P - D$ old points and D new points).

3.2 Optimization Based on Evolutionary Strategy

This approach uses an ES to make EFuNNs parameters optimization. ESs have the following characteristics, desired to parameters optimization of dynamic processes:

- They are good on real values parameters optimization [Eiben and Smith 2003].
- Usually they are used with self-adaptation of mutation parameters [Eiben and Smith 2003]. This is desired because the fitness surface is not known and it can change (in accordance with variations of testing data characteristics).
- They have low takeover time [Eiben and Smith 2003]. This characteristic permit a fast adaptation to the moment characteristics. This is good for on-line optimization because it is not desired that the population has not became adequate before other changes occur in data characteristics.

The used ES utilizes gaussian perturbation mutation with self-adaptation of the standard deviations and one standard deviation for each variable representing an EFuNN parameter to be optimized. This kind of mutation is adequate for ordinal numeric parameters optimization. Random numbers produced by a normal distribution obtained by Polar method [Knuth 1998] were used to generate numbers of a gaussian distribution.

The representation uses real values, with two genes for each EFuNN parameter to be optimized. The first of these genes represents the parameter itself and the second represents the corresponding self-adapting standard deviation.

The crossover is local discrete for the variables representing the parameters to be adjusted and local intermediary for the variables representing the self-adapting standard deviations, as [Eiben and Smith 2003] suggests.

The parents selection is random with equal probabilities for all population members and the survivor selection is (μ, λ). The latter was chosen because it is better than $(\mu + \lambda)$ to follow moving optimal points of the search space, to scape from local optima of the search space and to utilize with self-adaptation of the mutation parameters [Eiben and Smith 2003].

See algorithm 3.2 for more details of this approach.

Algorithm 3.2 (Optimization Based on ES)

1. Initialize the population with individuals created randomly choosing values for each parameter to be optimized (the genotype codifies parameters to be optimized).

2. Define a window with *P* time points for EFuNN learning (training time window) and, optionally, define a window with *P* time points for testing (testing time window).
3. Apply the learning algorithm to each EFuNN of the population, using all *P* points of the training time window and the parameters codified by the genotype of the individuals.
4. Determine each EFuNN RMSE (using the prediction over the points of the same time window used for learning or using the prediction over the testing time window points, if there is one) and use the weighted sum of the RMSE and the size of the network as fitness value to be minimized ($fitness = W_{rmse} * RMSE + W_{size} * size$). The W_{rmse} and W_{size} weights are pre-defined by the user.
5. Repeat for all training/testing data:
   (a) Randomly select N parents, using equal probabilities for all individuals of the population.
   (b) Apply mutation with the self-adapting standard deviations and crossover with rate *Pc*, creating one child for each pair of parents.
   (c) Shift the training and the testing (if there is one) time window (the new time windows are composed by $P - D$ old points and D new points).
   (d) Apply the learning algorithm for each child.
   (e) Determine each child RMSE.
   (f) Make (μ, λ) survivor selection, *i.e.* make a deterministic selection of the μ (μ is the population size) best individuals among all λ children. The children number is $\lambda = N/2$ and $\lambda > \mu$.

4 Experiments

The algorithms described in sections 3.1 and 3.2 were used to make on-line optimization of some of the parameters that define the EFuNN learning. The EFuNNs were used to Mackey-Glass chaotic time series prediction with changing dynamics, like the experiments made by [Kasabov, Song and Nishikawa 2003].

Section 4.1 describes the Mackey-Glass time series, section 4.2 shows the parameters of the optimization, the executions made for the experiments and the objective of these executions, section 4.3 explain the results of the experiments.

4.1 Mackey-Glass Chaotic Time Series

The EFuNNs were used to time series prediction. The time series utilized was the benchmark chaotic time series Mackey-Glass [Mackey and Glass 1977]. It is described by the following differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t)$$

This time series has chaotic behaviour for some values of the parameters τ, a, b and for the initial value $x(0)$. In the experiments made at this work, the used values were $a = 0.2$, $b = 0.1$, $x(0) = 1.2$ and $x(t) = 0$ for $t < 0$, like the values utilized by [Kasabov, Song and Nishikawa 2003]. The τ value is changed from 17 to 19 during the execution of the algorithms. This modification causes a change in the chaotic nature and in the series attractor. Although this modification is not drastic, it can cause fails on the prediction over the series with the modified parameter [Kasabov 2003].

Two types of experiments were done. In the first one, the RMSE value to be used in the fitness function was obtained by the prediction over the same time window used for learning. This experiment used the points from 0 to 599 with $\tau = 17$ and points from 600 to 1199 with $\tau = 19$. In the second experiment, the prediction RMSE used was calculated over a testing time window different from the training time window. The experiment used the points from 0 to 599 with $\tau = 17$ and the points from 1200 to 1799 with $\tau = 19$ for training and the points from 600 to 1199 with $\tau = 17$ and from 1800 to 2399 with $\tau = 19$ for testing.

Figure 2 shows time series used.

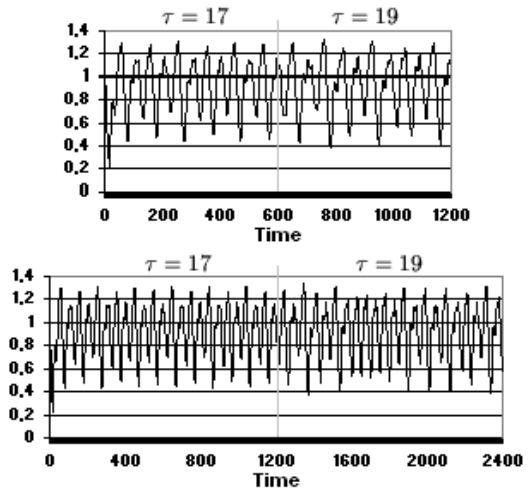

Figure 2: Mackey-Glass time series

The EFuNN prediction task is to determine $x(t+6)$ from the data vectors $[x(t-18), x(t-12), x(t-6), x(t)]$.

4.2 Optimization Parameters and Experiments Objectives

The EFuNN optimized parameters were *E, Mrad, m-of-n* and *OLD*.

The experiments were made in order to compare results of the approaches with and without considering the EFuNNs sizes in the fitness function, with and without pruning and with the RMSE obtained from the prediction over a time window equal to the training window and from the prediction over a time window different from the training window, as it was explained in sections 3.1, 3.2 and 4.1.

Two sets of allowed values intervals for the parameters to be optimized were used. The set 1 was equal to the utilized by [Kasabov, Song and Nishikawa 2003], in a try to get similar results for some of the tests with $W_{size} = 0$. In

this set, the *m-of-n* interval is $[1, 8] \in Z$, the *Mrad* interval is $[0.75, 0.95] \in R$ and the *E* interval is $[0.05, 0.25] \in R$. The experiments made by [Kasabov, Song and Nishikawa 2003] did not optimize the pruning parameter OLD and they did not use pruning. The assumed interval utilized in the experiment of this paper was $[10, 266] \in Z$. The second intervals set utilized was *m-of-n* ($[1, 15] \in Z$), *Mrad* ($[0.01, 0.8] \in R$), *E* ($[0.01, 0.6] \in R$) and OLD ($[10, 266] \in Z$). This set contains the values usually allowed by EFuNNs.

The experiments without pruning did not need to optimize the *OLD* parameter. The experiments without pruning used $Pr = 0$ and the experiments with pruning used $Pr = 1$. The *Nagg* parameter was fixed in 0 (no aggregation was used), *T1* and *T2* were fixed in 0 (no rule extraction was performed), and *S* was fixed in 0.9. The population size was 12, the RMSE weight for the calculation of the fitness was $W_{rmse} = 10$, the time window size was $P = 200$, with 90% of overlap between consecutive windows ($D = 20$). Note that the modification of τ parameter occur at generation 21.

The GA used $Pc = 0.7$, $Pm = 0.01$. The genotype for the experiments with the allowed values interval set 1 had 3 bits for *m-of-n* value, 6 for *Mrad*, 6 for *E* and 8 for *OLD*, when it was used. The genotype for the experiments with the allowed values interval set 2 had 4 bits for *m-of-n* value, 10 for *Mrad*, 9 for *E* and 8 for *OLD*, when it was used.

The ES used $Pc = 0.7$ and $N = 96$. The genotype had 6 real type variables when no pruning was utilized and 8 when pruning was utilized. Notice that this representation is more adequate for ordinal numeric parameters optimization than binary representation because in the latter small modifications on the genotype can cause big modifications on the phenotype [Eiben and Smith 2003].

Table 1 summarizes the executions. In the tables of this paper, App means approach, Prun means pruning, Test=Train means that the testing time window is equal to the training time window, W_{size} indicates the W_{size} weight utilized, Int set means allowed values intervals set and Num means number of executions.

App	Prun	Test = Train	W_{size}	Int set	Num
GA	Yes	Yes	0	2	10
			0.0005	2	10
		No	0	2	10
			0.002	2	10
	No	Yes	0	1	10
			0	2	1
			0.0005	2	10
		No	0	1	10
			0	2	1
			0.002	2	10
ES	Yes	Yes	0	2	10
			0.0005	2	10
		No	0	2	10
			0.002	2	10
	No	Yes	0	1	10
			0	2	1
			0.0005	2	10
		No	0	1	10
			0	2	1
			0.002	2	10

Table 1: Executions

4.3 Results

This section explains the main analysis of the executions showed in table 1. Sometimes the results will be considered only after generation 15, not to be affected by the fast drop on RMSE values of the first generations.

The executions showed in table 2 and with $W_{size} = 0$ generated very big[2] EFuNNs in comparison with the number of training examples (1200). Big networks make the training time bigger and need more memory to be stored, especially when various networks are used, as it is the case of an evolutionary algorithms with population size bigger than 1 individual. The executions were finalized at generation 28 because of the size of the generated networks, so their sizes in the table are concerning the EFuNNs at the final of this generation. The networks of generation 28 were also the biggest best networks of all generations until generation 28. The networks became too big because the fitness function was not penalized with the size of the network. The EFuNNs of the executions showed in table 2 and with $W_{size} \neq 0$ were smaller than those with $W_{size} = 0$. The sizes showed in the table are concerning the biggest EFuNNs of all generations of all 10 executions made. Using the allowed values intervals set 1 and $W_{size} = 0$, in a similar way to [Kasabov, Song and Nishikawa 2003], the EFuNNs did not became too big in comparison with the number of training examples, but this intervals set has small intervals in comparison with set 2.

Prun	Int set	W_{size}	App	Test = Train	Size
No	2	0	AG	Yes	1214
				No	3373
			ES	Yes	5239
				No	2275
		0.0005	AG	Yes	838
		0.002		No	201
		0.0005	ES	Yes	911
		0.002		No	196

Table 2: Executions without pruning and allowed values intervals set 2

Prun	W_{size}	Size averages	RMSE averages
No	0.0005	biggest	biggest
Yes	0.0005	smallest	
Yes	0		smallest

Table 3: Executions of ES with testing time window equal to training time window and allowed values intervals set 2

Comparing the executions from table 3, it can be ob-

[2] The terms size, big and small networks refer to the rule nodes number.

served that ES without pruning had the biggest rule nodes numbers averages and ES with pruning and $W_{size} = 0.0005$ had the smallest, as figure 3 shows. ES without pruning had the biggest RMSE averages and ES with pruning and $W_{size} = 0$ had the smallest most of the time after generation 15, as figure 3 shows. Notice that the executions with pruning and $W_{size} = 0.0005$ generated smaller networks than executions with pruning and $W_{size} = 0$, as expected, since $W_{size} = 0.0005$ penalizes the size of the networks. GA had a similar behaviour for these parameters, as figure 4 shows. T-Student statistic tests [Witten and Frank 2000] executed for ES and GA, among their own rule nodes number averages and among their own RMSE averages, with 5% of significance level, confirmed these analysis.

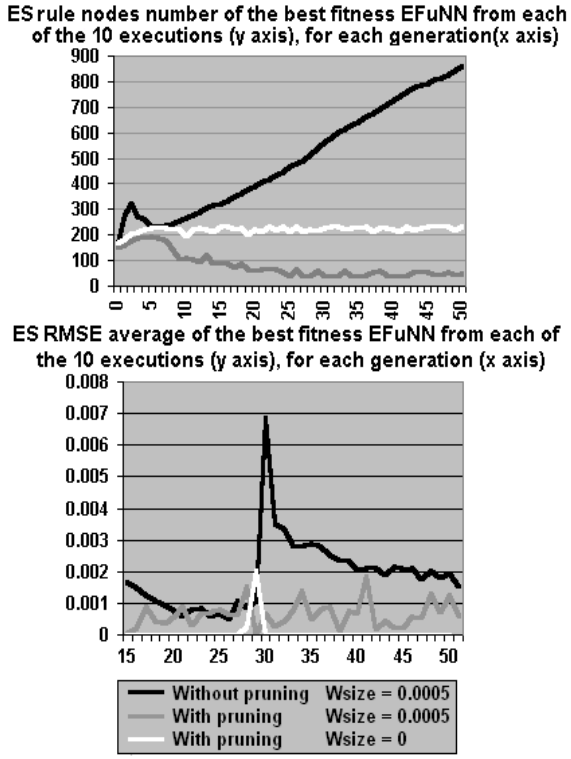

Figure 3: ES rule nodes number and RMSE averages for testing time window equal to training time window

Prun	W_{size}	Size variation	RMSE variation
No	0.002	smaller	smaller
Yes	0.002	bigger	bigger
Yes	0		

Table 4: ES executions with testing time window different from training time window and allowed values intervals set 2

Comparing executions from table 4, it can be observed that with pruning the rule nodes number averages and the RMSE averages vary more than in the executions without pruning, as it can be seen in figure 5. It can also be observed that for a similar RMSEs averages, ES with pruning and $W_{size} = 0.002$ generated EFuNNs with smaller rule nodes

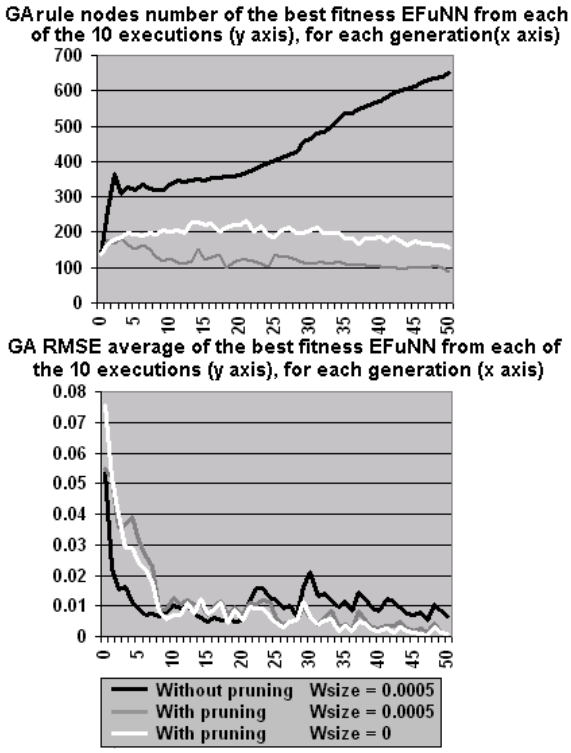

Figure 4: GA rule nodes number and RMSE averages for testing time window equal to training time window

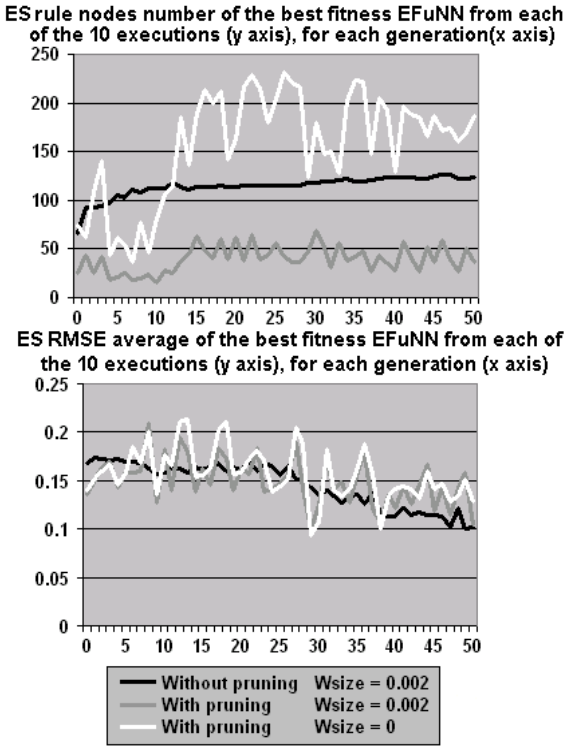

Figure 5: ES rule nodes number and RMSE averages for testing time window different from training time window

number averages than ES with pruning and $W_{size} = 0$. This was expected, since $W_{size} = 0.002$ penalizes the size of the networks. GA had a similar behaviour for these parameters, as figure 6 shows.

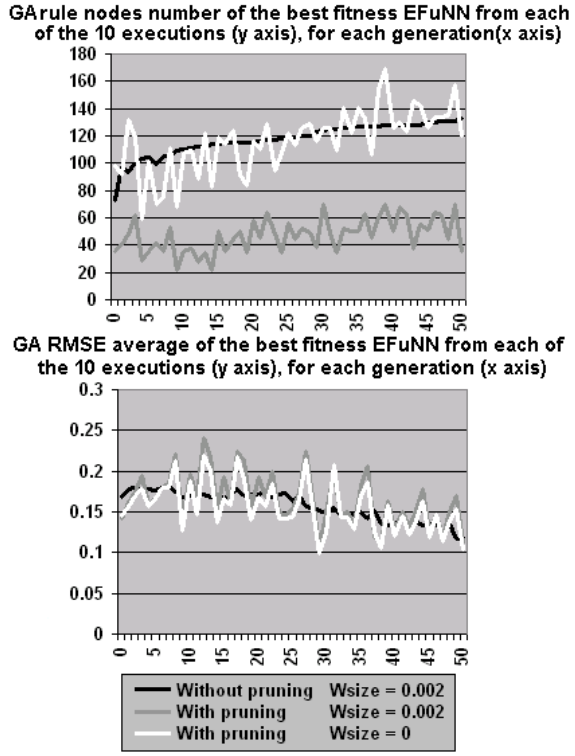

Figure 6: GA rule nodes number and RMSE averages for testing time window different from training time window

Notice that for these ESs and GAs executions, the use of testing time window different from training time window when $W_{size} = 0$ or when no pruning was performed had the effect of reducing EFuNN average sizes in comparison with the use of equal windows. This analysis was confirmed with T-Student tests with 5% of significance level. The RMSEs of the EFuNNs with testing time window different from training time window and with testing time window equal to training time window cannot be compared because they are calculated over different testing data sets. The bigger values of testing time window different from training time window occur as a direct consequence of the use of a testing data set different of that used for learning.

Table 5 shows the results of comparisons between AG and ES, all confirmed by T-Student tests with 5% of significance level. Figures 7, 8 and 9 shows the graphics of the comparisons number 1, 2 and 3, respectively.

5 Conclusions

The optimization approaches presented in this paper make the main characteristics of ECOSs, as adaptive learning, even more useful to applications that have incoming data with changing characteristics.

Changing only one parameter in the executions, there are three different ways to generate smaller EfuNNs. They are

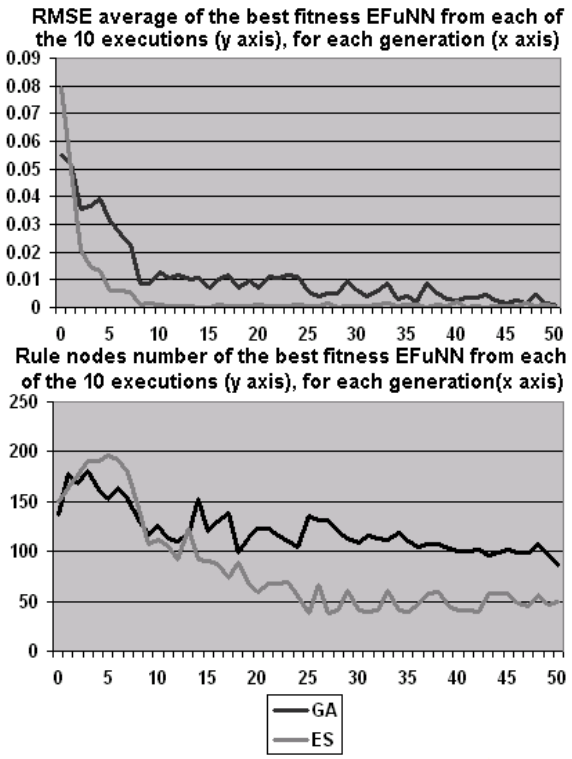

Figure 7: GA and ES RMSE and rule nodes number averages for testing time window equal to training time window, with pruning and $W_{size} = 0.0005$

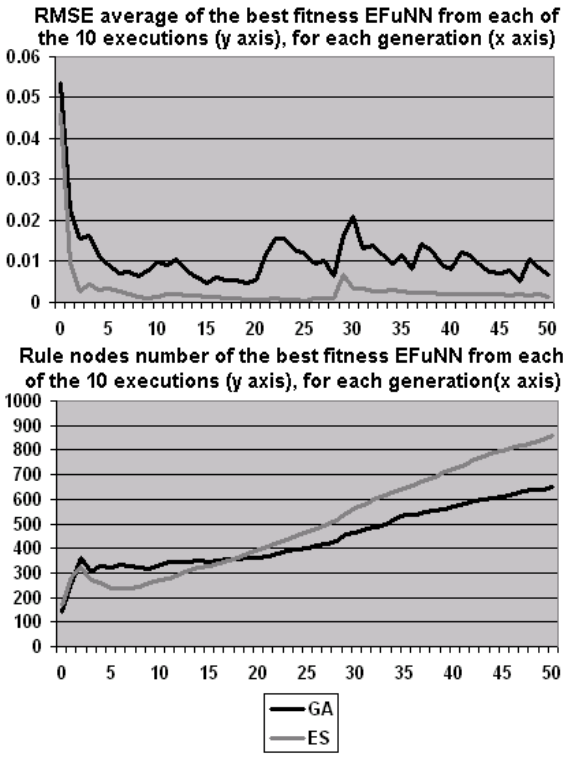

Figure 8: GA and ES RMSE and rule nodes number averages for testing time window equal to training time window, without pruning and $W_{size} = 0.0005$

Comparison number	Prun	Test = Train	W_{size}	App	RMSE averages	Size averages
1	Yes	Yes	0.0005	GA	bigger	bigger
				ES	smaller	smaller
2	No	Yes	0.0005	GA	bigger	smaller
				ES	smaller	bigger
3	No	No	0.002	GA	bigger	bigger
				ES	smaller	smaller

Table 5: Comparisons between GA and ES with allowed values intervals set 2

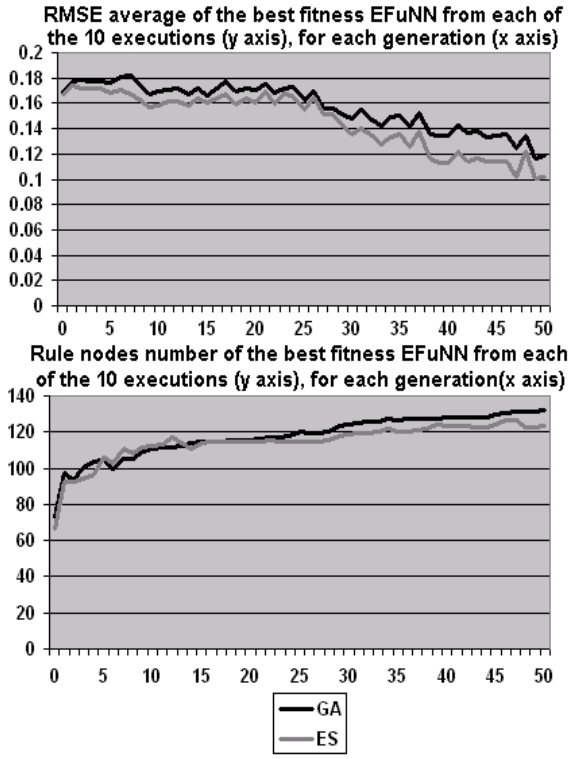

Figure 9: GA and ES RMSE and rule nodes averages for testing time window different from training time window, without pruning and $W_{size} = 0.002$

the use of:

- $W_{size} \neq 0$. It permits the use of bigger allowed values intervals for the parameters to be optimized even when no pruning is utilized.
- Training time window different from testing time window. This way is valid when either there is no pruning or $W_{size} = 0$.
- Pruning. This way is valid when the training time window is equal to the testing time window. It permits the use of bigger allowed values intervals for the parameters to be optimized even when $W_{size} = 0$. Besides, it generates not only EFuNNs with a reduced size, but also EFuNNs with reduced RMSE averages.

It is not recommended to use pruning with a testing time window different from training time window because it causes a high variation of the size and RMSE averages of the EFuNNs.

The RMSE of the EFuNNs generated using ES was smaller than the RMSE of EFuNNs generated using GA without pruning. Similar effect was observed for executions with pruning and testing equal to training time window.

Future works include an investigation of the effect of varying the fitness function weights, the creation of a method for varying the weights of the fitness function over time and the use of multi-objective techniques.

Acknowledgments

This work was supported by CNPq.

References

[Kasabov 2003] Kasabov, N. *Evolving Connectionist Systems.* Great Britain: Springer, 2003.

[Kasabov, Song and Nishikawa 2003] Kasabov, N., Song, Q., Nishikawa, I. Evolutionary Computation for Dynamic Parameter Optimization of Evolving Connectionist Systems for On-line Prediction of Time Series with Changing Dynamics. *IEEE Proceedings, IJCNN'2003*, v. 1, p. 438-443, Portland, Oregon, July 2003.

[Kasabov 2001] Kasabov, N. Evolving Fuzzy Neural Networks for Supervised/Unsupervised On-line, Knowledge-Based Learning. *IEEE Transactions on Systems, Man and Cybernetics*, Part B, v. 31, n. 6, p. 902-918, Dec. 2001.

[Mackey and Glass 1977] Mackey, M. C., Glass, L. Oscillations and Chaos in Physiological Control Systems. *Science*, v. 197, p. 287-289, Jul. 1977.

[Eiben and Smith 2003] Eiben, A.E., Smith, J. E. *Introduction to Evolutionary Computing.* Berlin: Springer, 2003.

[Witten and Frank 2000] Witten, I. H., Frank, E., Credibility: Evaluating What's Been Learned. In: *Data Mining - Pratical Machine Learning Tools and Techniques with Java Implementations.* San Francisco: Morgan Kaufmann Publishers, 2000.

[Knuth 1998] Knuth, B. E., *The Art of Computer Programming - Seminumerical Algorithms.* 3ed., v. 2. Massachusetts: Reading Mass Addison-Wesley Pub. Co., 1998.