EFuNNs Ensembles Construction Using a Clustering Method and a Coevolutionary Genetic Algorithm

L. Minku and T. Ludermir

***Abstract*— This paper introduces a new approach to construct neural network ensembles called Clustering and Co-evolution to Construct Neural Network Ensemble (CONE). This approach was used to create and optimize the parameters of a particular type of Evolving Fuzzy Neural Networks (EFuNNs) Ensemble. Experimental results on four benchmark databases show that the CONE generates EFuNNs ensembles with accuracy either better or equal to the accuracy of single EFuNNs generated using a genetic algorithm. Besides, the execution time of CONE to generate EFuNNs ensembles is lower than the execution time of the genetic algorithm to produce single EFuNNs.**

I. Introduction

Several approaches have been developed to optimize parameters of Evolving Connectionist Systems (ECoSs) [1]. Among them, [2], [3], [4] and [5] can be cited. All these methods use evolutionary algorithms to make the optimization of ECoSs parameters. [2] describes a try to optimize some ECoSs parameters in an on-line manner. [3] presents a method to optimize the parameters and the order of presentation of the training patterns in an off-line manner. [4] and [5] present successfully methods to optimize ECoSs parameters in an on-line manner.

However, ensembles of learning machines have been formally and empirically shown to generalize better than single predictors [6]. Instead of utilizing just one neural network to solve a specific problem, an ensemble of neural networks combines a set of neural networks. There are many advantages of using ensembles of neural networks instead of single neural networks, e.g. an ensemble can perform more complex tasks than any of its components (i.e. individual neural networks in the ensemble), and it can make an overall system easier to understand and modify [7]. Nevertheless, it is important to observe that the components of the ensemble should have errors at least somewhat not correlated in order to constitute a successful ensemble [8]. Besides, each component of the ensemble should have small error rates.

Given the advantages of using ensembles of neural networks and the complexity of the problems to be solved, it is clear that ensembles of neural networks are an important problem solving technique. In spite of the fact that ensembles of learning machines perform better than their members, their construction is not an easy task [6]. Because of the difficulty to construct ensembles, it is important to research methods to automatically construct them. Some papers related to the construction of ensembles of neural networks are: [9], [10], [11], [12], [13], [7], [14], [15], [16], [17] and [6].

Fernanda L. Minku and Teresa B. Ludermir. Federal University of Pernambuco, Informatics Center, P.O.Box 7851, Cidade Universitria, Recife-PE, Brazil, 50732-970. Emails: {flm, tbl}@cin.ufpe.br.

In order to improve the accuracy of the outputs of a particular type of ECoSs called Evolving Fuzzy Neural Network (EFuNN), [18] has used a multi-module classifier called multiEFuNN. But the construction of EFuNNs ensembles, in the same way as the construction of other ensembles of neural networks, is not an easy task. Besides, the choice of the best EFuNN parameters set is also a difficult task and the execution time of evolutionary algorithms to optimize the EFuNN parameters is high.

This paper introduces a new approach to construct ensembles of neural networks, called Clustering and Co-evolution to Construct Neural Network Ensemble (CONE). This approach was used to create a particular type of EFuNNs ensemble and, at the same time, to optimize the parameters of its EFuNNs.

The paper is organized as follows: Section II contains an explanation about ECoSs and EFuNNs. Section III introduces the proposed approach and explains a particular instance of it (i.e. a clustering method, a particular type of EFuNNs ensemble and a coevolutionary algorithm which can be used by the approach). Section IV contains the results of the experiments made with this instance of CONE using four different benchmark databases.

II. ECoS and EFuNNs

The ECOSs presented in [1] are systems constituted by one or more neural networks. They have the following characteristics [4]:

- 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 their entire existence.
  - On-line: they learn each example separately while the system operates. Usually, a system which operates in an on-line mode is also a systems which operates in a lifelong mode, and vice-versa.
  - 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 can 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 [19] are a class of ECOSs which join together the neural networks functional characteristics to the expressive power of fuzzy logic. They have a five-layer architecture, as it is shown by figure 1. The first layer represents the input vector, the second represents the fuzzy quantification of the input vector, the third represents the associations between fuzzy input space and fuzzy output space, the fourth represents the fuzzy quantification of the output vector and the fifth represents the output vector.

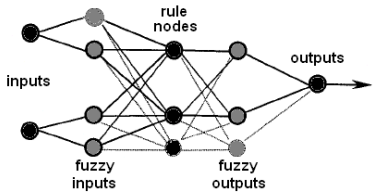

Fig. 1. EFuNN Architecture

The learning occurs 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 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 [19] 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 which are adjusted during the learning (rule nodes and their weights)

and parameters which do not change during the learning, but define it (number of membership functions, *E*, *Nagg*, *OLD*, *Pr*, *m-of-n* value, *Mrad*, *T1* and *T2*). Through the use of different parameters set, EFuNNs attain different performances and different weights are learned. The optimal parameters set usually depends on the input and output data presented. Thus, it is important to choose correctly the parameters which define the EFuNN learning according to the data presented.

III. A New Approach to Construct Neural Network Ensembles

This section presents a new approach called CONE. The general idea of the proposed approach is to construct neural network ensembles using a clustering method to partition the input space in clusters. After that, the clusters are used to separate the training and the test patterns in various subsets of training and test patterns with empty intersection. Each subset is used to train/test a different population of neural networks, which is evolved through a coevolutionary algorithm. Thus, each cluster is associated with a training subset, a test subset and a population of neural networks. The purpose of training the individuals of each population using a different subset of the training patterns is to maintain the output errors of the neural networks of different populations at least somewhat not correlated. Figure 2 illustrates the creation of clusters of the input space using the training and test patterns set.

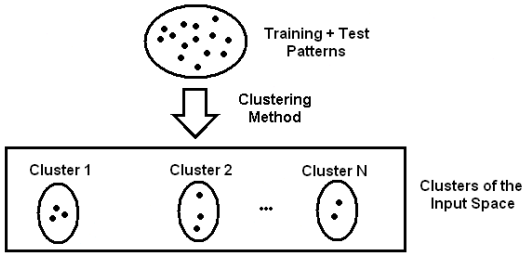

Fig. 2. Clustering

The patterns data used by the approach are divided in 3 types:

- Training patterns: used to create clusters and to train the neural networks;
- Test patterns: used to create clusters and to test the neural networks during the evolutionary process;
- Final test patterns: used to test the neural networks ensemble generated at the end of the evolutionary process.

The training and test data set is subdivided into subsets according to the clusters of the input space, as figure 3 shows. In this way, if there are N clusters, there will be N subsets of training patterns with empty intersection and N subsets of test patterns with empty intersection. If there are not enough patterns to compose a test subset correspondent to a specific cluster of the input space, the same subset used to train the neural networks of the correspondent population can be used to test them during the evolutionary process.

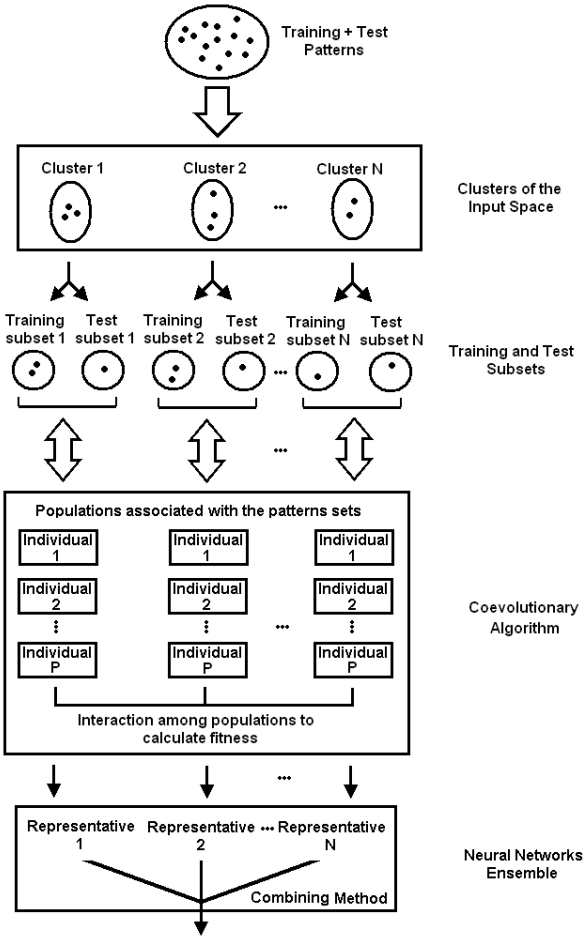

Fig. 3. Proposed approach

If there are N clusters, there will be also N populations to be evolved through a coevolutionary algorithm. The individuals of the population are neural networks and the coevolutionary algorithm can be used to optimize their parameters. These parameters can be both the architecture of the neural networks and, for example, the weights of the connections. Thus, it is possible to use the coevolutionary algorithm both to train and to optimize the architecture of the neural networks. However, it is also possible to use the specific learning algorithm of the neural networks to train them and the coevolutionary algorithm just to optimize their architectures. It is important to emphasize that each training/test subset is used to train/test the individuals of a specific population.

The evolutionary process is cooperative because the fitness of an individual of a population is calculated using a representative individual of each one of the other populations to constitute a neural networks ensemble. The representative of a population can be, for example, its best individual. There is no matching between individuals of different populations. The interaction among individuals of different populations occurs only in the calculation of the fitness value.

At the end of the evolutionary process, the representa-

tives of each population of the last generation are used to constitute the ensemble, as shown by figure 3. In order to use/test the ensemble, the clusters to which the input test pattern belongs are determined. After that, the outputs of the EFuNNs correspondent to these clusters are calculated and combined using a pre-determined combining method. Examples of combining methods can be found in [8].

The following sections explain the instance of CONE used in the experiments to produce EFuNNs ensembles. Section III-A explains the clustering method used to partition the input space, section III-B explains the EFuNNs ensembles created and section III-C explains the coevolutionary algorithm used.

A. A Clustering Method

The clustering method used in the experiments performed with the proposed approach is similar to the Evolving Clustering Method [18]. The algorithm 3.1 presents it.

In this paper, the distance between two vectors x and y denotes the General Euclidean Distance, defined as follows:

$$||x-y|| = \sqrt{\frac{\sum_0^{size-1}(x_i-y_i)^2}{size}}$$

Algorithm 3.1 (Clustering Method) Let $NumEx$ be the number of patterns and $Dthr$ be a distance threshold.

1) Create the first cluster C_0 by simply taking the position of the first pattern as the first cluster center Cc_0 and setting a value 0 for its cluster radius Ru_0.
2) For each input pattern x_i from $i = 1$ to $NumEx - 1$ do:
   a) Determine the distance between x_i and all N cluster centers Cc_j already created:
   $$D_{ij} = ||x_i - Cc_j||,\ j = 0, 1, ..., N-1$$
   b) If there is a distance value $D_{ij} \leq Ru_j$, it means that x_i belongs to a cluster C_m with the minimum distance $D_{im} = ||x_i - Cc_m|| = min(||x_i - Cc_j||)$, subject to the restriction $D_{ij} \leq Ru_j,\ j = 0, 1, ..., N-1$. In this case, neither a new cluster is created nor an existing cluster is updated.
   c) Else
      i) Find the cluster C_α with the minimum distance $D_{i\alpha} = ||x_i - Cc_\alpha|| = min(||x_i - Cc_j||),\ j = 0, 1, ..., N-1$.
      ii) If $D_{i\alpha} > Dthr$, create a new cluster, in the same way as described in the step 1.
      iii) Else update C_α: increment the number of patterns accommodated by C_α ($NExs_\alpha = NExs_\alpha + 1$); update Cc_α ($Cc_\alpha = Cc_\alpha + (x_i - Cc_\alpha)/NExs_\alpha$) and make R_α be the maximum between the following values: 1. the distance between the old Cc_α and the new Cc_α plus the old Ru_α and 2. the distance between x_i and the new center Cc_α

B. Creating EFuNNs Ensembles

As it was explained in section II, EFuNNs have parameters which are adjusted during the learning and parameters which define the learning. In the experiments performed with the proposed approach, the coevolutionary algorithm was used to optimize the parameters which define the EFuNN learning and the EFuNN learning algorithm itself is used to train the EFuNNs.

According to CONE, after the evolutionary process, the representatives of each population are used to construct the EFuNNs ensemble. In the experiments, the best fit individual of a population was considered the representative of this population. Two combining methods were used to combine the outputs of the EFuNNs that compose the ensemble. One of them is the arithmetic average of the outputs of the EFuNNs to which the pattern presented belongs. The other one is the weighted average of the outputs of the EFuNNs to which the pattern presented belongs. The value used as the weight of a cluster $C_j,\ j = 0, 1, ...N$ is $1/||x_i - Cc_j||$), where x_i is the pattern presented and Cc_j is the cluster center. If a pattern does not belong to any cluster, the output of the ensemble is the output of the EFuNN correspondent to the cluster whose center is the nearest center to the pattern.

C. Coevolutionary Algorithm

The experiments made with CONE have used a coevolutionary genetic algorithm as the coevolutionary algorithm. It is recommendable to read [20] for an explanation about evolutionary algorithms and [21] for an example of a coevolutionary approach.

The coevolutionary genetic algorithm used has a binary representation of the EFuNN parameters to be optimized, bitwise bit-flipping mutation, one-point crossover, and generational survivor selection. The parents selection used is proportional to the value determined by be following equation:

$$Prob_{i,p,g} = \frac{best\_fitness_{p,g} - fitness_{i,p,g}}{\sum_{j=0}^{pop\_size_p - 1} fitness_{j,p,g}} \quad (1)$$

where $best\_fitness_{p,g}$ is the fitness of the best individual of the population p of the generation g, $fitness_{i,p,g}$ is the fitness of the individual i of the population p of the generation g, and pop_size_p is the size of the population p.

The parents selection was made using the roulette wheel method and, according to equation 1, the fitness value is minimized. [20], a recent book about evolutionary algorithms, gives examples of applications of Genetic Algorithms (GAs) in which the fitness function is directly minimized, instead of change it into a function that has to be maximized.

Each initial population is composed by individuals created randomly choosing values for each of the EFuNN parameters to be optimized and one population is created for each cluster of the input space, according to CONE.

In the initial population, the fitness of the individuals is calculated in an isolated manner, *i.e.* to determine the fitness of an individual, the individuals of other populations are not

considered. The function used to calculate the fitness in this generation is:

$$fitness_i = W_{rmse}\ RMSE_i + W_{size}\ size_i \qquad (2)$$

where W_{rmse} and W_{size} are pre-defined weights, $RMSE_i$ is the Root Mean Squared Error (RMSE) obtained testing the EFuNN correspondent to the individual i with the test subset correspondent to its population, and $size_i$ is the size of this EFuNN.

The size component of the fitness function is used to penalize the size of the EFuNNs generated, as suggested by [5]. In this way, the execution time of the evolutionary algorithm is not so high.

In all generations after the initial one, the fitness of an individual i is calculated using not only the output error and the size of this individual, but also the output error and size of the representatives of the other populations of the previous generation. The functions used to calculate the fitness in all generations except the initial generation are:

$$RMSE = \sqrt{\frac{SSE_i + repr\_sse}{total\_test\_patterns\_number}}$$

$$size = size_i + repr\_size$$

$$fitness_i = W_{rmse}\ RMSE + W_{size}\ size$$

where W_{rmse} and W_{size} are pre-defined weights, SSE_i is the Sum of Squared Error (SSE) obtained testing the EFuNN correspondent to the individual i with the test subset correspondent to its population, $size_i$ is the size of this EFuNN, $repr\_sse$ is the sum of the SSEs and $repr\_size$ is the sum of the sizes of the representatives of all other populations in the previous generation, and $total\_test\_patterns\_number$ is the total number of test patterns, including the patterns correspondent to all populations.

Each population is evolved in a separate manner and there is an interaction among the populations only to calculate the fitness value, according to CONE. The algorithm 3.2 is the algorithm used to evolve a specific population. The stop criterium used for the evolutionary process is the number of generations.

Algorithm 3.2 (Evolutionary process of a population)

1) Create the initial population.
2) Repeat until a maximum number of generations is attained:
   a) If the EFuNNs correspondent to the individuals of the population have any rule nodes, delete them.
   b) Apply the EFuNN learning algorithm to each EFuNN of the population using the training subset correspondent to this population and the parameters codified by the genotype of the individuals.
   c) Test the EFuNNs correspondent to all individuals of the population using the test subset.
   d) Determine the fitness value of each individual of the population.
   e) Make parent selection using roulette wheel method and probabilities determined through the equation 1.
   f) Apply crossover and mutation with probabilities *Pc* and *Pm*, respectively, to generate new individuals.
   g) Apply generational survivor selection.

IV. Experiments

This section explains the experiments made with the instance of CONE presented in sections III-A, III-B and III-C. Section IV-A explains the databases and the creation of the data sets used in the experiments, section IV-B shows the parameters used and IV-C contains the results of the experiments.

A. Data Sets

The experiments have utilized four benchmark databases: Iris Plant, Wine, Glass and Cancer. These databases were obtained from the UCI Machine Learning Repository [22] and from Proben1 [23].

The Iris Plant database contains 3 classes of 50 patterns each, where each class refers to a type of iris plant (Iris Setosa, Versicolour and Virginica). Its input attributes are sepal length, sepal width, petal length and petal width in cm. Each pattern has 4 inputs and 3 outputs. One class is linearly separable from the other 2, and the latter are not linearly separable from each other.

The Wine database consist of the results of a chemical analysis of wines grown in the same region in Italy, but derived from three different cultivars. The analysis has determined the quantities of 13 constituents found in each of the three types of wines. There are 59 patterns of class 1, 71 patterns of class 2 and 48 patterns of class 3. Each pattern has 13 inputs and 3 outputs. All inputs are continuous.

The Glass database classify glass types. The results of a chemical analysis of glass splinters (percent content of 8 different elements) plus the refractive index are used to classify the sample to be either float processed or non float processed building windows, vehicle windows, containers, tableware, or head lamps. The database contains 214 patterns. The sizes of the 6 classes are 70, 76, 17, 13, 9, and 29 patterns, respectively. Each pattern has 9 inputs and 6 outputs. All inputs are continuous and two of them have hardly any correlation with the result. As the number of patterns is quite small, the problem is sensitive to algorithms that waste information.

The Cancer database consist of diagnosis of breast cancer. Its patterns try to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes are for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei. There are 699 patterns, and 458 of the patterns are

benign and 241 are malign. Each pattern has 9 inputs and 2 outputs. All inputs are continuous.

The training, test and final_test data sets utilized by the experiments were created as follows:

1) Two data sets (training+test data set and final_test data set) were created:
   - The Proben1 databases (Glass and Cancer) are already separated in training (50% of the patterns), validation (25% of the patterns) and test data sets (25% of the patterns). There are also 3 different partitions of the patterns which compose each of these sets. For each partition, the Proben1 training and validation sets were used to compose an CONE training+test data set and the Proben1 test set was used to compose an CONE final_test data set.
   - The other databases (Iris and Wine) were obtained directly from the UCI Machine Learning Repository. These sets were processed in order to create 3 different partitions of training+test and final_test data sets. Each training+test data set contains 75% of the patterns of each class and each final_test data set contains 25% of the patterns of each class.
2) Each one of the 3 partitions of the training+test and final_test data sets was used in a different execution of the CONE. For each execution:
   a) The training+test patterns set was used to create the clusters of the input space.
   b) After that, the training+test patterns set was separated according to the clusters into subsets of training+test patterns. Each pattern belongs to the subset correspondent to the cluster which has the nearest center to this pattern.
   c) Each training+test subset was then divided into 2 subsets. One of them is a training subset, with 66% of its patterns, and the other one is a test subset, with 34% of its patterns.

In this way, the total number of training patterns is always 50% of the patterns of the database, the total number of test patterns is 25% of the patterns of the database, and the total number of final test patterns is 25% of the patterns of the database.

B. Parameters and Executions

The EFuNN parameters optimized during the coevolutionary process and their intervals of allowed values were: m-of-n ($[1, 15] \in Z$), error threshold ($[0.01, 0.6] \in R$), maximum radius ($[0.01, 0.8] \in R$), initial sensitivity threshold ($[0.4, 0.99] \in R$) and membership functions number ($[2, 8] \in Z$).

The genotype was composed by 4 bits for the m-of-n value, 9 for the error threshold, 10 for the maximum radius, 6 for the initial sensitivity threshold, and 3 for the membership functions number of each input/output attribute.

The $Dthrs$ parameter of the clustering method was empirically determined and it is 0.40 for Iris database, 50 for Wine database, 0.20 for Glass database and 0.37 for Cancer database.

The parameters of the coevolutionary genetic algorithm were: population size = 12, mutation rate = 2%, crossover rate = 70%, W_{rmse} = 0.1, and stop criterium = 50 generations. Four different values for W_{size} were used: 0.005, 0.0005, 0.00005, and 0.000005.

Hence the CONE was executed with 4 different combinations of evolutionary parameters for each database. Three different partitions of the training+test and final_test data sets were also used, thus totalizing 12 combinations of configurations. Three executions with different random seeds were performed for each combination, totalizing 36 executions for each database. The same 3 seeds were used in the 3 executions of each combination for all databases.

Executions with the above combinations of parameters were also made using a GA to generate a single EFuNN. The GA utilized was the same as the coevolutionary genetic algorithm presented in section III-C, but using the fitness function 2 for all generations and just one species. In this way, 36 executions of the GA were made for each database.

The above parameters were determined empirically. Particularly, the population size was 12 because some executions were made with bigger populations and did not cause improvements in the generalization of the generated ensembles which have compensated the increase on the execution time. The stop criterion was 50 generations because in most executions of the GA, 50 generations have already caused a certain convergency.

The objective of the executions explained above was to compare:

- EFuNNs ensembles generated using CONE with weighted average combining method (weighted EFuNNs ensembles);
- EFuNNs ensembles generated using CONE with arithmetic average combining method (arithmetic EFuNNs ensembles);
- Single EFuNNs generated using GA.

The characteristics compared were the execution times of the evolutionary approaches and the output classification errors of the EFuNNs ensembles and of the single EFuNNs generated.

C. Results

In this section, the classification errors are those obtained using the final_test patterns set to test the single EFuNNs or the EFuNNs ensembles generated after the evolutionary processes.

Figure 4 shows the classification errors averages of the EFuNNs ensembles generated after the evolutionary processes of all 36 executions of the CONE using weighted average combining method and arithmetic average combining method, for each database. The classification errors averages of the single EFuNNs generated after the 36 executions of the GA, for each database, are also shown. Observe that the classification errors average of the Glass database is multiplied by 0.1 in this figure. Table I shows the classification

errors averages, standard deviations, minimal and maximum values, considering the 36 executions of each database. In this table and in the other tables of this paper, "W Ens" means Weighted EFuNNs Ensemble, "A Ens" means Arithmetic EFuNNs Ensemble, "Sing" means Single EFuNN, "Av" means Average, "SD" means Standard Deviation, "Min" means Minimal Value, and "Max" means Maximum Value.

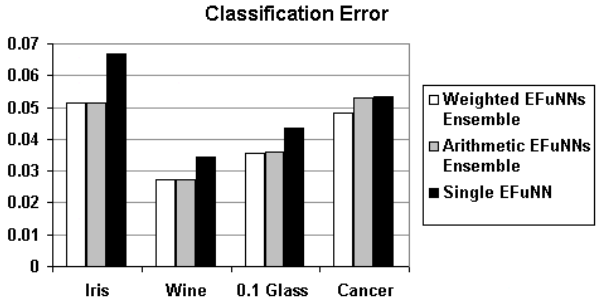

Fig. 4. Classification errors

TABLE I
MEASURES RELATED TO THE CLASSIFICATION ERRORS

		Iris	Wine	Glass	Cancer
W Ens	Av	0.0513	0.0272	0.3543	0.0484
	SD	0.0397	0.0301	0.0991	0.0152
	Min	0	0	0.1698	0.0230
	Max	0.1282	0.0889	0.5849	0.0920
A Ens	Av	0.0513	0.0272	0.3585	0.0528
	SD	0.0397	0.0301	0.0974	0.0211
	Min	0	0	0.1698	0.0172
	Max	0.1282	0.0889	0.5849	0.0920
Sing	Av	0.0670	0.0346	0.4350	0.0535
	SD	0.0384	0.0234	0.1680	0.0252
	Min	0	0	0.1887	0.0172
	Max	0.2051	0.1111	0.6792	0.1207

It can be observed that for all databases the classification errors averages of ensembles was lower than the classification errors averages of single EFuNNs. Nevertheless, just the averages of the ensembles created for Iris and Glass databases were considered statistically lower than the classification errors averages of single EFuNNs.

In the executions made with Iris and Wine databases, the classification errors averages of the weighted EFuNNs ensembles were considered statistically equal to the classification errors averages of the arithmetic EFuNNs ensembles. However, in the executions made with Glass and Cancer databases, the classification errors averages of the weighted EFuNNs ensembles were considered statistically lower than the classification errors averages of the arithmetic EFuNNs ensembles.

Table II shows the statistics of the paired T student tests [24] performed to prove the analysis made with the classification errors. The signals "=" indicate that the classification errors of all 36 executions of the compared approaches were equal.

TABLE II
T STUDENT TEST STATISTICS COMPARING THE CLASSIFICATION ERRORS AVERAGES AND USING LEVEL OF SIGNIFICANCE EQUAL TO 0.05

	Iris	Wine	Glass	Cancer
W Ens x A Ens	=	=	-3.1623	-2.2832
W Ens x Sing	-2.3052	-1.2076	-4.3775	-1.3435
A Ens x Sing	-2.3052	-1.2076	-4.0719	-0.1565

Figure 5 shows the execution times averages of the 36 executions of CONE to generate EFuNNs ensembles and of the 36 executions of GA to generate single EFuNNs, for each database. Observe that the execution times averages of Glass and Cancer databases are multiplied by 0.1 in this figure. Table III shows the execution times averages, standard deviations, minimal and maximum values, considering the 36 executions of each database. It is interesting to see that the standard deviations of the execution times of the CONE to produce EFuNNs ensembles were lower than the standard deviations of the execution times of the GA to produce single EFuNNs for all databases.

Fig. 5. Execution times in seconds

TABLE III
MEASURES RELATED TO THE EXECUTION TIMES

		Iris	Wine	Glass	Cancer
Ens	Av	32.3333s	31.8333s	142.5556s	277.1944s
	SD	2.5185s	6.4209s	64.6478s	160.3850s
	Min	29s	21s	56s	649s
	Max	38s	42s	281s	109s
Sing	Av	163.8611s	95.3611s	264.1111s	996.4444s
	SD	89.2397s	44.0488s	150.7067s	621.8672s
	Min	54s	144s	72s	403s
	Max	297s	167s	466s	2768s

For all databases, the execution times average among all 36 executions of the GA to generate a single EFuNN was statistically greater than the execution times average among all 36 executions of the CONE to generate an EFuNNs ensemble. Table IV shows the statistics of the paired T student tests made to prove this analysis.

The execution time of the CONE to generate EFuNNs ensembles was lower than the GA execution time to generate single EFuNNs possibly because in the optimization process of a single EFuNN, for each pattern presented to train/test the EFuNN, the activation levels of all rule nodes of the

TABLE IV

T STUDENT TEST STATISTICS COMPARING THE EXECUTION TIMES AVERAGES AND USING LEVEL OF SIGNIFICANCE EQUAL TO 0.05

	Iris	Wine	Glass	Cancer
Ens x Sing	-8.8184	-9.5510	-7.8229	-8.8763

EFuNN have to be calculated. When an EFuNNs ensemble is being created, just the activation levels of the rule nodes of the correspondent EFuNN (or the correspondent EFuNNs in the case of test) have to be calculated. The single EFuNN is usually bigger than each EFuNN which compose an ensemble because the single EFuNN has to accommodate all training patterns and a component of an ensemble has to accommodate only the patterns correspondent to a particular cluster of the input space.

Table V shows the number of clusters created for the executions of CONE with each partition of the data sets, for each database. The clusters number are a consequence of the $Dthrs$ used.

TABLE V

CLUSTERS NUMBER

	Iris	Wine	Glass	Cancer
Partition 1	13	7	13	7
Partition 2	15	6	17	7
Partition 3	13	7	15	6
Average	13.6667	6.6667	15	6.6667

V. CONCLUSIONS

This paper introduces a new coevolutionary approach to produce neural network ensembles, called CONE and describes a clustering method and a coevolutionary algorithm which can be used to create EFuNNs ensembles.

The experimental results on four benchmark problems and a particular instance of CONE show that the EFuNNs ensembles generated using this instance have either better or equal generalization abilities to the single EFuNNs generated using a GA. They also show that the weighted EFuNNs ensembles have either better or equal generalization abilities to the arithmetic EFuNNs ensembles generated using the instance of CONE. The execution times of the instance of CONE to produce EFuNNs ensembles are lower than the execution times of GA to produce single EFuNNs. The standard deviations of the execution times are also lower for CONE.

Future works include the use of other clustering methods and coevolutionary algorithms to create neural network ensembles using CONE.

ACKNOWLEDGMENT

This work was supported by CNPq.

REFERENCES

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

[2] M. Watts and N. Kasabov, "Dynamic optimisation of evolving connectionist system training parameters by pseudo-evolution strategy," in *CEC'2001*, vol. 2, Seoul, may 2001, pp. 1335–1342.

[3] ——, "Evolutionary optimisation of evolving connectionist systems," in *CEC'2002*, vol. 1. Honolulu, Hawaii: IEEE Press, may 2002, pp. 606–610.

[4] N. Kasabov, Q. Song, and I. Nishikawa, "Evolutionary computation for dynamic parameter optimization of evolving connectionist systems for on-line prediction of time series with changing dynamics," in *IJCNN'2003*, vol. 1, Portland, Oregon, july 2003, pp. 438–443.

[5] F. L. Minku and T. B. Ludermir, "Evolutionary strategies and genetic algorithms for dynamic parameter optimization of evolving fuzzy neural networks," in *CEC'2005*, vol. 3, Edinburgh, Scotland, september 2005, pp. 1951–1958.

[6] A. Chandra and X. Yao, "Ensemble learning using multi-objective evolutionary algorithms," *Journal of Mathematical Modelling and Algorithms (accepted)*, 2005.

[7] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 380–387, november 2000.

[8] T. G. Dietterich, "Machine-learning research: Four current directions," *The AI Magazine*, vol. 18, no. 4, pp. 97–136, 1998.

[9] X. Yao, Y. Liu, and P. Darwen, "How to make best use of evolutionary learning," *Complex Systems - From Local Interactions to Global Phenomena*, pp. 229–242, 1996.

[10] Y. Liu and X. Yao, "Towards designing neural network ensembles by evolution," in *Proceedings of the International Symposium on Artificial Life and Robotics, AROB'1996*, Beppu, Japan, february 1996, pp. 265–268.

[11] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 28, no. 3, pp. 417–425, 1998.

[12] Y. Liu and X. Yao, "A cooperative ensemble learning system," in *IJCNN'1998*, USA, 1998.

[13] ——, "Simultaneous training of negatively correlated neural networks in an ensemble," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 29, no. 6, pp. 716–725, 1999.

[14] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi, "Evolving a cooperative population of neural networks by minimizing mutual information," in *CEC'2001*. Seoul, Korea: IEEE Press, may 2001, pp. 384–389.

[15] M. M. Islam, X. Yao, and K. Murase, "A construtive algorithm for training cooperative neural network ensembles," *IEEE Transactions on Neural Networks*, vol. 14, no. 4, pp. 820–834, 2003.

[16] H. A. Abbass, "Speeding up backpropagation using multiobjective evoluctionary algorithms," *Neural Computation*, vol. 15, no. 11, pp. 2705–2726, november 2003.

[17] ——, "Evolving neural network ensembles by minimization of mutual information," *International Journal of Hybrid Systems*, vol. 1, no. 1, 2004.

[18] N. Kasabov, "Ensembles of efunns: An architecture for a multimodule classifier," in *Proceedings of the International Conference on Fuzzy Systems*, vol. 3, Australia, 2001, pp. 1573–1576.

[19] ——, "Evolving fuzzy neural networks for supervised/unsupervised on-line, knowledge-based learning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 31, no. 6, pp. 902–918, december 2001.

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

[21] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architeture for evolving coadapted subcomponents," *Evolutionary Computation*, vol. 8, no. 1, pp. 1–29, 2000.

[22] C. B. D.J. Newman, S. Hettich and C. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[23] L. Prechelt, "PROBEN1 - a set of neural network benchmark problems and benchmarking rules," Fakultt fr Informatik, Universitt Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, september 1994.

[24] I. H. Witten and E. Frank, *Data Mining - Pratical Machine Learning Tools and Techniques with Java Implementations*. San Francisco: Morgan Kaufmann Publishers, 2000.