Mathware & Soft Computing 13 (2006) 89-110

Evolutionary Training for Dynamical Recurrent Neural Networks: An Application in Finantial Time Series Prediction

M. Delgado, M.C. Pegalajar, M.P. Cuéllar
Dpto. Ciencias de la Computación e Inteligencia Artificial.
ETS. Ingeniería Informática
University of Granada. 18071 Granada, Spain.
mdelgado@ugr.es, mcarmen@decsai.ugr.es, manupc@decsai.ugr.es

Abstract

Theoretical and experimental studies have shown that traditional training algorithms for Dynamical Recurrent Neural Networks may suffer of local optima solutions, due to the error propagation across the recurrence. In the last years, many researchers have put forward different approaches to solve this problem, most of them being based on heuristic procedures. In this paper, the training capabilities of evolutionary techniques are studied, for Dynamical Recurrent Neural Networks. The performance of the models considered is compared in the experimental section, in real finantial time series prediction problems.

Keywords: Recurrent Neural Networks, Evolutionary Algorithms, Time Series Prediction

1 Introduction

Neural networks are bio-inspired mathematical models, which have successfully solved many problems in the real world [17]. The neural network architectures most known are mainly feedforward models [17][22][24][27][31]. They have traditionally been trained with algorithms based on gradient and error propagation. There is a huge variety of training algorithms for feedforward networks, being the most known the BackPropagation and its derivatives [17][27]. Dynamical Recurrent Neural Networks (DRNN) [21][8][20] may be built from a feedforward model, by including recurrent connections in the network structure. The variety of training algorithms for DRNN is not as high as for feedforward models, some examples being the algorithms Real Time Recurrent Learning (RTRL) [8][9][26] and Backpropagation Through the Time (BPTT) [8][9][25]. These algorithms are also based on error propagation across the recurrent connections. Theoretical and experimental results have shown that traditional training methods for DRNN may suffer of

local optima solutions [32]. In the last years, many researchers have put forward different approaches to solve the shrinking gradient problems for DRNN training, most of them being based on heuristic procedures [1][2][3][5][14][19].

Evolutionary algorithms [10][13][18][29] are heuristic procedures that include a set of search, learning and optimization techniques, based on nature processes. In the last decade, evolutionary algorithms have been widely used to solve many real world problems, obtaining suitable results. Some evolutionary models have also been proposed to improve the neural network training. In the case of DRNN, genetic algorithms have been applied for training and topology optimization, obtaining promising results [2][3][19]. In this paper, we study the capabilities of evolutionary algorithms to train DRNN. An Elman Recurrent Neural Network [8][15] is trained using different evolutionary techniques, in finantial time series forecasting problems [20][23][24][31]. The training models studied in this work are genetic algorithms [22][19][3], using generational [2], stationary [29][30] and mixed [10] evolution schemes; the multimodal Clearing algorithm [4][6], and the CHC algorithm [13]. The original CHC scheme has been modified, to deal with real-coded variables.

This paper is structured in the following way. Section 2 introduces the Elman Recurrent Neural Network model. Section 3 explains the evolutionary algorithms considered in this work. Section 4 exposes the evolutionary training procedure, for Elman Recurrent Neural networks. Section 5 shows the experimental results and discusses the comparative study of evolutionary and traditional training algorithms. Finally, section 6 summarizes the conclusions obtained.

2 The Elman Recurrent Neural Network

Dynamical Recurrent Neural Networks may be used as input/output mapping models. They can process patterns with undetermined size, temporal properties or dynamical behaviour. The DRNN models most known are the Fully Connected Recurrent Neural Network [8], the Jordan Network [8], and the Elman Network [8][15]. In this work, we study the Elman Network model. The Elman recurrent neural network (ERNN) is a widely studied model, for which it has been proved the equivalence with Markov models and Mealy-Moore machines [7][28]. The network is provided with long and short term memory, codified in the network structure by mean of recurrent connections. The topology of an Elman network has the following structure:

- The nodes in the input layer provide the input data corresponding to the current time, and distribute them across the other layers.
- The nodes in the hidden layer makes the non-linear transformations and operations, needed to calculate the output of the network.
- The nodes in the output layer uses the results provided by the hidden neurons, and aggregates them to produce the network output.

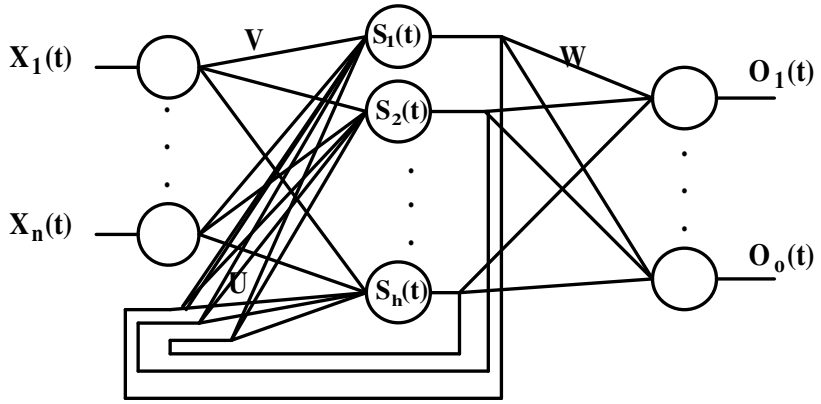

Figure 1: Structure of the Elman Network

The recurrence is carried out in the hidden layer, so that the output value of a hidden neuron, at time t, is also input for all hidden neurons, at time t+1. Figure 1 shows these ideas in the basic scheme of an Elman recurrent neural network.

The diagram in Figure 1 represents an Elman recurrent neural network with n inputs, h hidden neurons, and o outputs. $X_i(t)$ is the input data to neuron i at time t ($1 \leqslant i \leqslant n$); $H_j(t)$ is the output of hidden neuron j at time t ($1 \leqslant j \leqslant h$); and $O_k(t)$ is the k-th network output at time t ($1 \leqslant k \leqslant o$).

The values U, V, W are matrices that encode the network weights, so that V_{ji} is the weight associated to connection from input neuron i to hidden neuron j; U_{jr} is the weight associated to the recurrent connection from hidden neuron r to hidden neuron j; and W_{kj} is the weight associated to connection from hidden neuron j to output neuron k. Attending to this notation, the equations for the network dynamics are:

$$neth_j(t) = \sum_{r=1}^{h} U_{jr} H_r(t-1) + \sum_{i=1}^{n} V_{ji} X_i(t) \tag{1}$$

$$H_j(t) = f(neth_j(t)) \tag{2}$$

$$neto_k = \sum_{j=1}^{h} W_{kj} H_h(t-1) \tag{3}$$

$$O_k(t) = g(neto_k(t)) \tag{4}$$

In equations (2) and (4), the functions f(x) and g(x) are the activation functions for hidden and output neurons, respectively. In this work, we use the sigmoid and the identity functions for f(x) and g(x). Equations (5) and (6) introduce how they are calculated:

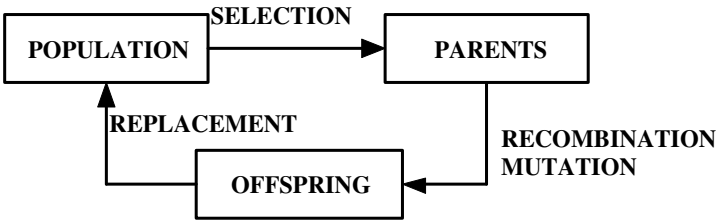

Figure 2: Main scheme of a Genetic Algorithm

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

$$g(x) = x \tag{6}$$

The traditional gradient-based algorithms to train Elman recurrent neural networks are the Truncated BackPropagation Throught the Time (TBPTT) [8], and the Real Time Recurrent Learning [8]. A complete guide about TBPTT and its application to train Elman recurrent neural networks may be found in [8][9].

3 Evolutionary algorithms

This section introduces the evolutionary algorithms used in this work, for DRNN training. Firstly, genetic algorithms are introduced in subsection 3.1. After that, subsection 3.2 explains the multimodal Clearing procedure. Finally, the CHC scheme modified is exposed in subsection 3.3.

3.1 Genetic Algorithms

The evolution process in a genetic algorithm [16][13][18][29] is based on a nature-like selection procedure, the recombination and the mutation in a set of solutions (population). The genetic evolution schemes generational [2], stationary [29][30] and mixed [10] are generated from a basic genetic algorithm procedure [13] (see Figure 2), by using different strategies for the selection, recombination, mutation and replacement of the solutions in the population. Below, algorithms 1, 2 and 3 review the main procedures for the previous schemes.

Algorithms 1, 2 and 3 show the differences in the selection, recombination, mutation and replacement operator strategies, in order to build the generational, stationary and mixed procedures, respectively.

Firstly, a generational procedure uses the selection operator to build a set of solutions, P', with size equals to the population's size. It also allows P' to include multiple instances of a solution. After that, the recombination operator is applied over the solutions in P', to build —P(t)— new solutions, H. Then, the mutation procedure alters these new individuals in H'. At the end of the evolutionary process, the replacement strategy sets the population in the following iteration of the

Algorithm 1 Genetic Procedure with Generational structure

```
1: t=0; Initialize starting population P(t)with S solutions
2: while stopping condition is not satisfied do
3:    set P'= ∅
4:    while |P'| < |P(t)| do
5:       set s= select a solution from P(t)
6:       set P'= P'+s
7:    end while
8:    set H= recombination of solutions in P'
9:    set H'= Mutation of solutions in H
10:   evaluation of solution in H'
11:   set P(t+1)= H'
12:   Apply elite strategy in P(t) and P(t+1), if required
13:   set t= t+1
14: end while
15: Return best solution in P(t)
```

Algorithm 2 Genetic Procedure with Stationary structure

```
1: set t=0; Initialize starting population P(t)with S solutions
2: while stopping condition is not satisfied do
3:    set P'= ∅
4:    while |P'| < k do
5:       set s= select a solution from P(t)
6:       set P'= P'+s
7:    end while
8:    set H= recombination of solutions in P'
9:    set H'= Mutation of solutions in H
10:   evaluation of solutions in H'
11:   set P(t+1)= P(t)
12:   replacement of solutions in P(t+1) with solutions in H'
13:   Apply elite strategy in P(t) and P(t+1), if required
14:   set t= t+1
15: end while
16: Return best solution in P(t)
```

Algorithm 3 Genetic Procedure with Mixed structure

```
1: set t=0; Initialize starting population P(t)with S solutions
2: while stopping condition is not satisfied do
3:    set P'= ∅
4:    while (|P| < k do
5:       set s= select a solution from P(t)
6:       set P'= P'+s
7:    end while
8:    set H= recombination of solutions in P'
9:    set H'= Mutation of solutions in H+P'
10:   evaluation of solutions in H'
11:   set P(t+1)= P(t)
12:   replacement of solutions in P(t+1) with solutions in H' and P'
13:   Apply elite strategy in P(t) and P(t+1), if required
14:   set t= t+1
15: end while
16: Return best solution in P(t)
```

algorithm to the new solutions in H'. On the other hand, the stationary strategy uses the selection operator to choose a fixed number of k solutions in P(t). Usually, the value for k is k=2. After that, the recombination operator generates k new solutions, which are altered using the mutation procedure. The replacement scheme in a stationary strategy also allows the designer of the algorithm to choose a replacement strategy. For instance, some of the most common replacement schemes are to replace parents with offspring, to replace the worst solutions in P(t+1) with the offspring, to replace the parents if the offspring is better, etc. Finally, the mixed strategy uses the selection and recombination schemes of a stationary procedure. However, the mutation operator is applied in both sets of parents, P', and offspring, H. The replacement scheme must also be chosen by the designer of the algorithm. For instance, one of the most common replacement schemes in a mixed procedure are to replace P(t+1) with the best —P(t+1)— solutions in H'.

This work considers all the previous evolution strategies. In the experimental section, all of them are compared to test the best strategy for the finantial time series problems approached.

3.2 Niching Clearing Procedure

Multimodal evolutionary algorithms [6] may be considered as a extension of a genetic evolution procedure. The population of a multimodal algorithm evolves covering different areas in the solution space. Then, the solutions are set to one of the areas explored, like in a clustering procedure (niching). A niche comprises a set of solutions in the same neighbourhood. In this work, two solutions are neighbors if the Euclidean distance is under a threshold. The Clearing [4][6] algorithm evolves a population of solutions. For each iteration, Clearing divides the solutions in clus-

ters (niches), and applies the selection, recombination, mutation and replacement procedures as stated in algorithm 4.

Algorithm 4 Clearing Procedure

1: set t=0; Initialize starting population P(t)with S solutions
2: **while** stopping condition is not satisfied **do**
3: compute the number of niches at iteration t, N(t)
4: niching of solutions in P(t)
5: set B= $\emptyset$
6: **for all** niche i=1..N(t) **do**
7: {sj: 1≤j≤k}= select the k best individuals of niche i
8: set B= B+{sj}
9: **end for**
10: set P'= selection of solutions from B
11: set H= recombination of solutions in P'
12: set H'= Mutation of solutions in H
13: evaluation of solutions in H'
14: set P(t+1)= P(t)
15: replacement of solutions in P(t+1) with solutions in H'
16: Apply elite strategy in P(t) and P(t+1), if required
17: set t= t+1
18: **end while**
19: Return best solution in P(t)

The Clearing main scheme in Algorithm 4 may be obtained from a genetic evolution procedure (Algorithms 1, 2 and 3). The relevant contribution in the Clearing scheme is in steps 3-9. Steps 3-4 compute the number of niches (or clusters) in the current iteration. Then, each solution in P(t) is assigned to a niche in steps 5-9. Once the niching procedure has finished, then the selection, recombination, mutation and replacement procedures are carried out following a genetic generational evolution strategy.

3.3 Algorithm CHC

The algorithm CHC [13] was firstly proposed as an alternative to solve problems with binary-encoded solutions. It combines a balance in diversity and convergence using an elite selection, the HUX recombination operator [13], incest prevention in recombination, and population reinitialization. Below, these components are reviewed.

- The elite selection is carried out at the replacement step, so that the population at the next iteration is composed by the best individuals, considering parents and offspring.
- The purpose of the HUX recombination operator is to produce new solutions that differ from the parents as much as possible, to explore new zones in

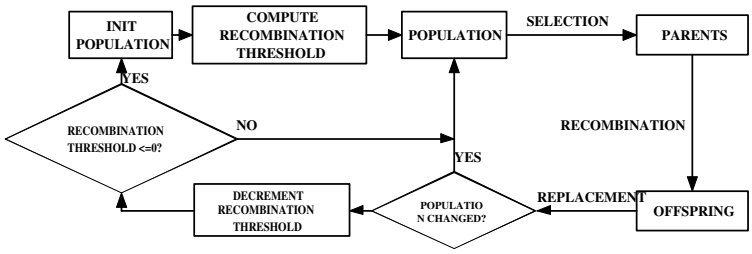

Figure 3: Main scheme for the CHC algorihm

Algorithm 5 CHC Procedure

1: set t=0; Initialize starting population P(t)with S solutions
2: set D= Average Euclidean distance in P(t)
3: set R= KÂ·D
4: **while** stopping condition is not satisfied **do**
5: set P'=∅
6: **while** |P| < |P(t)| **do**
7: set s= select a solution from P(t)
8: set P'= P'+s
9: **end while**
10: set H= recombination of solutions in P' by pairs. Two solutions a and b are not combined iif || a-b|| <D
11: evaluation of solutions in H
12: set P(t+1)= best solutions in H and P(t) : |P(t+1)|= S
13: **if** P(t) = P(t+1) **then**
14: set D= D-R
15: **end if**
16: **if** D≤0 **then**
17: re-start population P(t+1) with S-L random solutions and the L best solutions from P(t)
18: set D= Average Euclidean distance in P(t+1)
19: set R= KÂ·D
20: evaluation of new solutions in solutions P(t+1)
21: **end if**
22: set t= t+1
23: **end while**
24: Return best solution in P(t)

the search space. In this work, the HUX recombination operator is replaced with the BLX-α operator [11], so that the algorithm may be applied to real-encoded solutions.

- The incest prevention is applied in the recombination step, to avoid premature convergence. It avoids two solutions to be combined if they are similar. The similarity measure is provided by the Hamming distance for binary-encoded solutions. In this work, the similarity measure used is the Euclidean distance, for real-encoded solutions.

- The reinitialization procedure re-starts the population if it has converged to an area in the solution space. It also uses an elite strategy that preserves the best solutions found in the evolution process in the population.

Figure 3 shows the main scheme of the CHC algorithm. After that, Algorithm 5 explains in depth the adaptation of scheme in figure 3 for real-encoded solutions.

4 Evolutionary Training of Recurrent Neural Networks

This section explains the use of evolutionary algorithms to train DRNN [2]. Firstly, subsection 4.1 exposes the representation of an Elman recurrent neural network. After that, subsection 4.2 introduces the fitness function to train an Elman network in time series prediction problems.

4.1 The representation mechanism

In this work, an Elman recurrent neural network is codified into a real-valued vector. Each component in the vector is assigned to an only network connection. Then, the value of a component is the weight of the corresponding connection associated. Figure 4 shows an example of the encoding of an Elman network with one input, one output, and two hidden neurons. Considering the notation introduced in section 2, the number of components of a vector s encoding an ERNN, Ns, may be calculated using the following equation:

$$N^s = h(n + h + o) \tag{7}$$

For example, the number of components in the vector encoding the ERNN in Figure 4 is 8.

The Algorithm 6 explains the internal setting of a vector $s = (s_1, s2, ..., sN^s)$, enconding an Elman network.

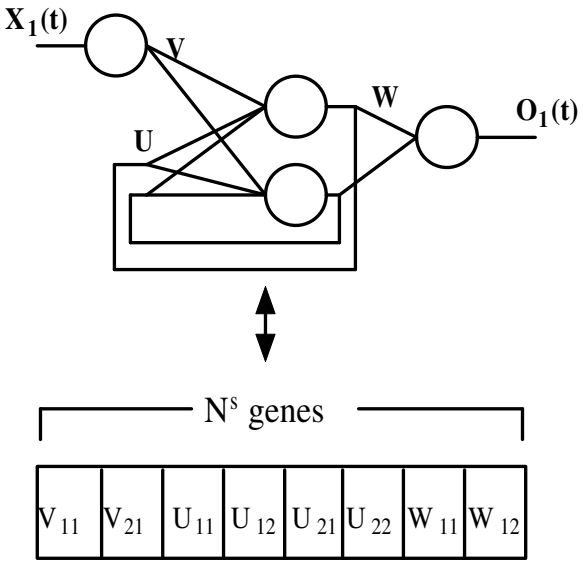

Figure 4: Encoding of an Elman network into a vector

Algorithm 6 Encoding Procedure of an Elman Network into a vector

```
set m= 1
for j = 1 to h do
   for i = 1 to n do
      set network weight V_ji = s_m
      set m= m+1
   end for
end for
for j = 1 to h do
   for r = 1 to h do
      set network weight U_jr = s_m
      set m= m+1
   end for
end for
for k = 1 to o do
   for j = 1 to h do
      set network weight W_kj = s_m
      set m= m+1
   end for
end for
```

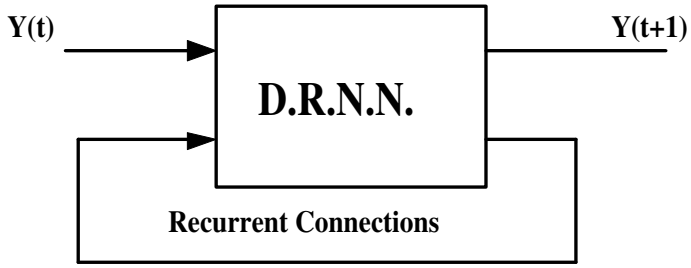

Figure 5: Main scheme for the CHC algorihm

4.2 The evaluation procedure: DRNN for time series prediction

This section introduces the use of DRNN for time series prediction, and the evaluation procedure for Elman recurrent neural networks evolutionary training. A time series [23] may be considered as a data sequence, indexed in time. The goal of time series prediction is to find the future values of the data sequence, using the knowledge provided by the previous known data. The main assumption that allows us to model and to predict a time series is that the value of the time series, at time t, depends on the T previous values. Equation (8) illustrates this idea.

$$Y(t) = F(Y(t-1), Y(t-2), ..., Y(t-T)) + e(t) \tag{8}$$

In equation (8), Y(t) is the value of the time series at time t; F is an unknown function, whose domain is the value of the time series for the previous T times, and its range is the current value of the time series. Finally, e(t) is the error while computing the current value of the time series. To simplify the problem, we may assume that e(t)=0.

Traditionally, the tools applied for time series prediction have been linear regressions and statistical models. However, when F is non-linear, other powerful techniques must be applied. In the last decade, neural networks have been often applied in time series prediction problems [23]. The problem of time series prediction may be approached using a dynamical recurrent neural network [1][20]. Assuming that the network input, at time t, is the value of the time series at time t, the network provides an output depending on the current inputs and an unknown function. This function depends on the previous network inputs in time, and it is learnt by the network and recorded into the recurrent connections. The structure of a DRNN for time series prediction is shown in the diagram of Figure 5.

The time series is presented to the network, ordered in time. The network output, at time t, must fit the value of the time series at time t+1. The training stage minimizes the mean square error between the network output at time t, and the value of the time series at time t+1, for T training patterns (equation (9)).

$$F(s) = \frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{o}(O_k^s(t) - d_k(t))^2 \quad (9)$$

In equation (9), o is the number of network outputs; $O_{sk}(t)$ is the output corresponding to the output neuron k of the ERNN associated to vector S, at time t; $d_k(t)$ is the desired output for neuron k at time t; and finally, T is the number of training patterns.

On the other hand, in the prediction stage, the network output at time $t+1$ is used as network input at time $t+2$, to predict the time series value at time $t+3$, following the scheme in figure 5. This process is cicled for times $t+1, t+2, t+3, ...$, until the prediction value for the time required is obtained.

5 Experiments

This section shows the performance of the training algorithms in section 3, to train an Elman recurrent neural network in finantial time series prediction problems. Subsection 5.1 introduces the data sets. After that, subsection 5.2 exposes the parameters for the algorithms and the network. Finally, subsection 5.3 shows the experimental results, and discusses the performance comparison of the algorithms in the ERNN training.

5.1 The data sets and the case study: prediction of gross indebtedness in the Spanish regions

The GDP data for then autonomous regions in Spain, between years 1986 and 1996, are analyzed to predict the value from year 1997 to year 2000. Figure 6 shows the data sets. Values from years 1986 to 1996 are used as training data set, and the networks trained are validated with the values from year 1997 to year 2000.

5.2 The parameters

This section exposes the parameters for the network and the algorithms. Table 1 shows the settings for the Elman recurrent neural network. Table 2 explains the configuration for the evolutionary algorithms. Finally, table 3 introduces the parameters for the traditional training algorithms (TBPTT and RTRL). The configuration in table 1 is used for the neural network.

The algorithms evolve the population until 50000 solutions have been evaluated. The number of evaluations is also the comparison criterion in the experiments of the following subsection. Each algorithm is run for 30 times in all the problems.

The evolutionary algorithms are also compared with the traditional training algorithms for DRNN. The comparison criterion is the computational time. The algorithms TBPTT and RTRL are run for 30 times, in a multi-start procedure. Each iteration of the multi-start procedure uses the highest computational time of

Table 1: Settings for the neural network

Parameter	*Value*
Number of input neurons	1 (value of the time series at time t)
Number of hidden neurons	7
Number of output neurons	1 (value of the time series at time t+1)
Activation function	Sigmoid for hidden nodes, linear for output units

the evolutionary algorithms as stopping criterion. Table 3 shows the parameters for these algorithms.

5.3 Experimental results

This subsection resumes the experimental results. Table 4 shows the average error for the solutions of the training algorithms, applied to all the data sets. Column 1 means the data set. Column 2 introduces the algorithm. Columns 3 and 4 show the mean square error of the best, in the training and test sets. Finally, column 5 prints the average computing time.

A statistical test is applied to test the relevance of the results for the algorithms. The Kolmogorov-Smirnov test has been applied to check normality conditions in the results obtained, for each algorithm. It concludes that most of results do not follow a normal distribution. Thus, the Kruskal-Wallis non-parametric test has been applied to the results of the algorithms, with 0.05 of confidence level. Table 5 shows the statistical relevance of the algorithms, according to the results of the Kruskal-Wallis test. Columns 1 and 3 mean the data sets. Columns 2 and 4 print the pValue resulting from the statistical test, starting from the best algorithm to the worst one for each data set. We mark (x) when the algorithm is statistically equivalent to the previous one, and (-) when it is worse.

Using the criterion in Algorithm 7 to compute the relevance of the evolutionary models, table 6 concludes that the best results have been found using the CHC evolutionary algorithm. On the other hand, the worst solutions have been provided by the traditional training algorithms, RTRL and BPTT. An indepth study of tables 3 and 5 may conclude that any of the evolutionary training models have provided better performance than the algorithms RTRL and BPTT. Thus, a partial conclusion of the experimental results is that evolutionary algorithms may improve the traditional training of ERNN, in the problems attached to this work.

Considering the evolutionary procedures, there is a clear difference in the statistical results. While the best solutions have been found by the algorithm CHC in all the data sets, the worst ones have been provided by the mixed genetic algorithm.

The algorithm CHC has addressed the search suitably, using the incest prevention, BLX-α recombination, the elite selection, and the reinitialization. However, the mixed genetic algorithm has made an excessive use of exploration in the search space, with the mutation operator, therefore reducing the convergence rate. This assumption is supported by the results in the stationary genetic procedure (the difference in both algorithms is the mutation operator applied to the whole popu-

Table 2: Settings for the evolutionary algorithms

Parameter	*GeneticAlgorithms*	Clearing	CHC
Scheme	Generational /stationary /mixed	generational	–
Selection operator [11]	Binary tournament selection	Binary tournament selection	Binary tournament selection
Recombination operator [12][11]	Blx-α (α= 0.5)	Blx-α (α= 0.5)	Blx-α (α= 0.5)
Mutation operator [11]	Displacement	Displacement	–
Recombination probability	0.8 (generational)	0.8	–
Mutation probability	0.08	0.08	–
Replacement	Parents are replaced with offspring (stationary and mixed)	–	–
Elite factor	The two best solutions remain in the population	The two best solutions remain in the population	–
Size of population	50	50	50
Gene bounds	[-5.0, 5.0]	[-5.0, 5.0]	[-5.0, 5.0]
Ratio of a niche	–	0.35*D (D= Maximum Euclidean distance of solutions in the population)	–
Ratio to decrease the Average Euclidean distance	–	–	K= 0.1

Table 3: Settings for the traditional training algorithms

Parameter	*TBPTT*	*RTRL*
Number of iterations	500	500
Learning rate	0.001	0.001
Time to unfold the network	2	–
Time for each multi-start iteration (sec.)	5	5

Table 4: Settings for the traditional training algorithms

DataSet	*Algorithm*	$MSE(Training)$	$MSE(Test)$	$Time(sec.)$
Andalucía	BPTT	7.09e-01	1,62	3,17
	RTRL	3,5	5,44	3,77
	GGA	4.67e-02	1.36e-01	2,23
	SGA	3.25e-01	5.28e-01	2,33
	MGA	5.36e-01	9.57e-01	1,93
	Clearing	8.30e-02	4.94e-01	3,43
	CHC	1.12e-02	2.67e-01	2,17
Aragon	BPTT	5.04e-01	7.93e-01	3
	RTRL	1,71	2,51	3,53
	GGA	1.99e-02	6.73e-02	2,2
	SGA	8.92e-02	1.35e-01	2,27
	MGA	1.52e-01	3.23e-01	1,9
	Clearing	2.75e-02	2.65e-01	3,43
	CHC	5.11e-04	4.13e-02	2,23
Asturias	BPTT	3.02e-01	4.82e-01	3,1
	RTRL	8.25e-01	1,23	3,6
	GGA	2.76e-02	5.30e-02	2,13
	SGA	9.36e-02	1.55e-01	2,33
	MGA	1.44e-01	1.93e-01	1,9
	Clearing	4.44e-02	2.53e-01	3,43
	CHC	2.07e-03	5.16e-02	2,17
Baleares	BPTT	1.88e-01	1.55e-01	3,07
	RTRL	6.25e-01	5.03e-01	3,6
	GGA	6.20e-03	9.96e-02	2,2
	SGA	5.49e-02	1.37e-01	2,33
	MGA	1.07e-01	1.75e-01	2,03
	Clearing	1.65e-02	1.07e-01	3,4
	CHC	1.78e-03	9.35e-02	2,23
Canarias	BPTT	9.06e-01	7.11e-01	3,23
	RTRL	1,4	1,14	3,73
	GGA	1.06e-01	6.21e-01	2,13
	SGA	1.72e-01	5.33e-01	2,4
	MGA	2.45e-01	4.81e-01	1,93
	Clearing	9.49e-02	9.36e-01	3,5
	CHC	1.94e-02	1,15	2,2
Cantabria	BPTT	1,02	1,07	3,33
	RTRL	1,52	1,52	4,1
	GGA	1.17e-01	5.35e-01	2,27
	SGA	4.22e-01	9.15e-01	2,4
	MGA	6.60e-01	9.99e-01	1,97
	Clearing	1.38e-01	1,07	3,53
	CHC	2.32e-02	6.23e-01	2,2

DataSet	*Algorithm*	$MSE(Training)$	$MSE(Test)$	$Time(sec.)$
Castilla-Leon	BPTT	3.10e-01	4.48e-01	3,1
	RTRL	8.07e-01	1,16	3,53
	GGA	6.14e-03	3.60e-02	2,27
	SGA	4.42e-02	9.07e-02	2,37
	MGA	8.34e-02	1.70e-01	2
	Clearing	1.17e-02	5.72e-02	3,57
	CHC	1.03e-03	7.01e-02	2,27
Catalunia	BPTT	2,81	5,96	3,23
	RTRL	3,95	7,53	3,83
	GGA	5.07e-02	1.00e-01	2,2
	SGA	2.40e-01	4.73e-01	2,37
	MGA	4.17e-01	1,01	1,97
	Clearing	5.39e-02	2.41e-01	3,53
	CHC	1.57e-03	6.39e-02	2,23
Extremadura	BPTT	8.65e-01	8.60e-01	2,93
	RTRL	3,31	3,73	3,53
	GGA	9.61e-03	4.77e-02	2,17
	SGA	2.49e-01	2.55e-01	2,3
	MGA	4.42e-01	4.11e-01	1,9
	Clearing	4.95e-02	3.46e-01	3,4
	CHC	2.25e-03	4.82e-02	2,3
Galicia	BPTT	6.73e-01	9.75e-01	3,13
	RTRL	4,67	5,67	3,7
	GGA	2.02e-02	1.01e-01	2,2
	SGA	3.48e-01	4.05e-01	2,3
	MGA	6.23e-01	7.13e-01	1,93
	Clearing	5.72e-02	1.40e-01	3,43
	CHC	1.14e-02	9.05e-02	2,27
Madrid	BPTT	2.00e-01	4.18e-01	3,4
	RTRL	8.11e-01	1,38	3,7
	GGA	1.50e-02	8.38e-02	2,17
	SGA	7.30e-02	1.52e-01	2,47
	MGA	1.02e-01	1.99e-01	2
	Clearing	2.25e-02	2.65e-01	3,4
	CHC	3.11e-03	1.74e-01	2,33
Castilla-La Mancha	BPTT	2.88e-01	3.46e-01	2,97
	RTRL	6.30e-01	7.96e-01	3,63
	GGA	1.03e-02	1.09e-02	2,17
	SGA	6.73e-02	7.05e-02	2,27
	MGA	1.16e-01	1.42e-01	1,9
	Clearing	2.62e-02	6.31e-02	3,4
	CHC	7.73e-04	7.78e-03	2,37
Murcia	BPTT	2.16e-01	1.75e-01	3,23
	RTRL	8.60e-01	6.23e-01	3,87
	GGA	1.47e-02	1.55e-01	2,2
	SGA	7.15e-02	2.05e-01	2,33
	MGA	1.10e-01	2.03e-01	1,93
	Clearing	2.71e-02	2.17e-01	3,4
	CHC	6.00e-03	1.13e-01	2,17

DataSet	*Algorithm*	$MSE(Training)$	$MSE(Test)$	$Time(sec.)$
Rioja	BPTT	2.00e-01	1.83e-01	3,13
	RTRL	7.94e-01	5.67e-01	3,67
	GGA	2.76e-02	3.14e-01	2,17
	SGA	6.54e-02	2.85e-01	2,27
	MGA	1.08e-01	3.13e-01	1,9
	Clearing	2.73e-02	3.18e-01	3,47
	CHC	1.02e-02	3.34e-01	2,2
Valencia	BPTT	5.07e-01	2,56	3,27
	RTRL	2,54	6,2	3,7
	GGA	2.25e-02	6.85e-01	2,2
	SGA	1.43e-01	9.68e-01	2,33
	MGA	2.92e-01	1,42	1,9
	Clearing	4.08e-02	7.76e-0	3,43
	CHC	5.87e-03	5.24e-01	2,2

lation, in the mixed one). In all the problems, the ranking of the S. GA scheme has provided better results than the mixed one. It has improved the convergence rate, but it has not explored the search space enough. On the other hand, the generational genetic scheme and the Clearing procedure have improved the exploration and exploitation of the stationary genetic algorithm, in the search space. Table 5 shows that the genetic procedure may improve the Clearing scheme, in some cases. However, there may be situations in which both return similar solutions.

Figure 6 plots both training and test network outputs. Points plotted with O are the real data, and points plotted with + are the network outputs.

6 Conclusions

This work has studied the training capabilities of evolutionary algorithms, based on population evolution, for Dynamical Recurrent Neural Networks. The experimental results have shown that evolutionary training may improve the traditional training of Elman recurrent neural networks, in the time series prediction problems attached in the experimental section.

On the other hand, the comparative statistical study of the algorithms proposed in this work, has concluded that algorithms with a better balance in diversity/convergence may provide better results: The algorithm CHC has found the best solutions in the problems of GDB prediction for the Spanish regions. The experiments have concluded that Elman recurrent neural networks are suitable models for time prediction problems. The network is able to learn the dependencies in time of the training data sets, and may predict the future values suitably in the test data. However, the use of evolutionary algorithms for DRNN training may help to improve the network performance, in time series prediction problems.

Table 5: Results of the statistical tests

Problem	*Algorithms*	*Problem*	*Algorithms*
Andalucía	GGAMEM	Cataluna	CHC
	CHC 0.4688 x		GGAMEM 1,95e-4 -
	Clearing 0.0003666 -		Clearing 0.0009775 -
	SGAMEM 0.0001948 -		SGAMEM 0.0006373 -
	MGAMEM 0.00579 -		MGAMEM 0.0006373 -
	BPTT 0.00037 -		BPTT 3,88e-8 -
	RTRL 2.87e-8 -		RTRL 0.0001073 -
Aragón	CHC	Extremadura	GGAMEM
	GGAMEM 8.7e-08 -		CHC 0.2871 x
	Clearing 0.008875 -		Clearing 1,53e-4 -
	SGAMEM 0.0228 -		SGAMEM 0.003585 -
	MGAMEM 0.000858 -		MGAMEM 1,69e-2 -
	BPTT 2.95e-5 -		BPTT 3,34e-6 -
	RTRL 5.23e-8 -		RTRL 2,87e-8 -
Asturias	CHC	Galicia	CHC
	GGAMEM 0.01801 -		GGAMEM 0.05099 x
	Clearing 1,24e-3 -		Clearing 0.0004337 -
	SGAMEM 0.1602 x		SGAMEM 9,31e-7 -
	MGAMEM 0.01355 -		MGAMEM 5,10e-2 -
	BPTT 1,02e-6 -		BPTT 0.003419 -
	RTRL 1,27e-7 -		RTRL 2,87e-8 -
Baleares	CHC	La Rioja	BPTT
	Clearing 0.4077 x		CHC 0.0005411 -
	GGAMEM 0.3831 x		Clearing 0.7901 x
	SGAMEM 0.001480 -		SGAMEM 0.7562 x
	MGAMEM 0.02760 -		MGAMEM 0.8016 x
	BPTT 0.6048 x		GGAMEM 0.1882 x
	RTRL 7,04e-7 -		RTRL 1,44e-3 -
Canarias	Clearing	Madrid	CHC
	SGAMEM 0.7788 x		GGAMEM 0.4333 x
	MGAMEM 0.2939 x		Clearing 0.01247 -
	GGAMEM 0.2428 x		SGAMEM 0.7901 x
	BPTT 0.03205 -		MGAMEM 0.01801 -
	CHC 0.03089 -		BPTT 3,39e-4 -
	RTRL 0.2089 x		RTRL 6,37e-8 -
Cantabria	GGAMEM	Murcia	CHC
	CHC 0.7562 x		BPTT 0.001205 -
	Clearing 0.2036 x		GGAMEM 525 x
	SGAMEM 0.02463 -		SGAMEM 0.4779 x
	MGAMEM 0.1242 x		MGAMEM 0.1984 x
	BPTT 0.0005715 -		Clearing 0.03847 -
	RTRL 2,87e-8 -		RTRL 4,40e-7 -

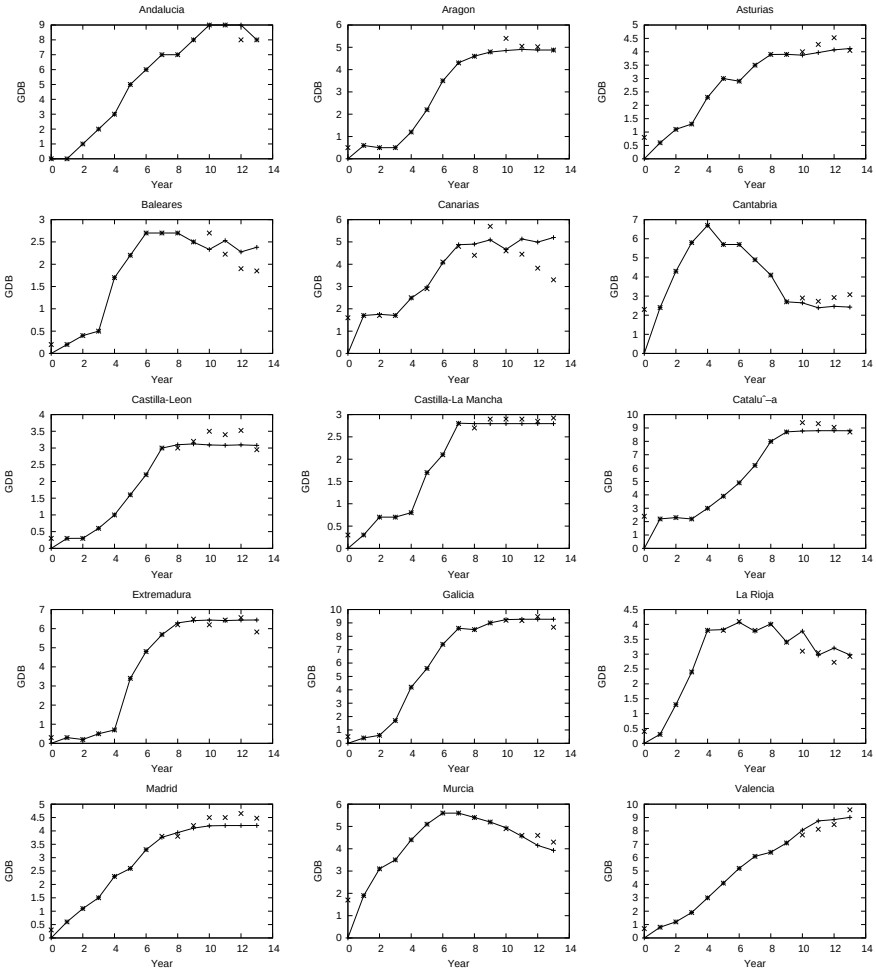

Figure 6: Predition of GDB for the Spanish regions.

Problem	*Algorithms*	*Problem*	*Algorithms*
Castilla-León	GGAMEM	Valencia	CHC
	CHC 0.3077 x		GGAMEM 0.002002 -
	Clearing 0.006236 -		Clearing 0.3671 x
	SGAMEM 0.0001538 -		SGAMEM 0.02559 -
	MGAMEM 0.0002189 -		MGAMEM 0.005445 -
	BPTT 1,77e-5 -		BPTT 2,47e-4 -
	RTRL 2,05e-7 -		RTRL 2,87e-8 -
Castilla-La Mancha	CHC		
	GGAMEM 0.005445 -		
	Clearing 4,22e-2 -		
	SGAMEM 2,58e-3 -		
	MGAMEM 3,70e-3 -		
	BPTT 6,81e-6 -		
	RTRL 5,32e-7 -		

References

[1] A. Aussem. Dinamical Recurrent Neural Networks towards prediction and modeling of dynamical systems. *Neurocomputing*, 28(1-3):207-232, 1999.

[2] A. Blanco, M. Delgado and M.C. Pegalajar. A Real-Coded genetic algorithm for training recurrent neural networks. *Neural Networks*, 14:93-105, 2001.

[3] A. Blanco, M. Delgado and M.C. Pegalajar. A genetic algorithm to obtain the optimal recurrent neural network. *International Journal of Approximate Reasoning*, 23:67-83, 2000.

[4] A. Pétrowski. A Clearing Procedure as a Niching Method for genetic Algorithms. In *IEEE International Conference of Evolutionary Computation*, Nagoya, Japan. pp. 798-803, 1996.

[5] A. Tettamanzi, M. Tomassini, J. Janben. *Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems*. ed. Springer, 2001.

[6] B. Sareni, L. Krahenbuhl. Fitness sharing and niching methods revised. *IEEE transactions on Evolutionary Computation*, 2:97-106, 1998.

[7] C. Stefan, P. Kremer, P. Baldi. Hidden Markov models and neural networks. In *Genetics, Genomics, Proteomics and Bioinformatics*, 4, John Wiley and Sons Ltd., 2005.

[8] D. P. Mandic, J. Chambers. *Recurrent Neural Networks for Prediction*. Wiley, John & Sons Inc., 2001.

[9] H. Jaeger. A tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF, and the echo state network, GMD-report, 43p. 1999.

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

[11] F. Herrera, M. Lozano, J.L. Verdegay. Tackling Real-Coded Genetic Algorithms: operator and tools for behavioural analysis. *Artificial Intelligence review*, 12:265-319, 1998.

[12] F. Herrera, M. Lozano, A.M. SÃ¡nchez. A Taxonomy for the Crossover Operator for Real-Coded Genetic Algorithms: An Experimental Study. *International Journal of Intelligent Systems*, 18:309-338, 2003.

[13] G.J.E. Rawlins. *Foundations of Genetic Algorithms*. ED. Morgan Kauffman, 1991.

[14] J. Schmidhuber. A fixed Size Storage O(n^3) Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks. *Neural Computation*, 4:243-248, 1992.

[15] J.L. Elman. Finding Structure in Time. *Cognitive Science*, 14:179-211, 1990.

[16] L.J. Eshelman, J.D. Scahffer. Real-coded genetic algorithms and interval-schemata. In *Foundations of genetic algorithms 2*. Morgan Kaufmann publishers, 1993.

[17] L.P.J. Veelenturf. *Analysis and Applications of Artificial Neural Networks*, Ed. Prentice Hall, 1995.

[18] L. Schmitt. Fundamental Study: Theory of genetic algorithms. *Theoretical Computer Science*, 259(1-2):1-61, 2000.

[19] M. Delgado, M.C. Pegalajar. A Multiobjective Genetic Algorithm for obtaining the optimal size of a Recurrent Neural network for Grammatical Inference. *Pattern Recognition, Special Issue of Grammatical Interence*. Vol 38(9):1444-1456, 2005.

[20] M. Husken, P. Stagge. Recurrent Neural Networks for Time Series classification. *Neurocomputing*, 50:223-235, 2003.

[21] M.P. Cuéllar, M.A. Navarro, M.C. Pegalajar and R. Pérez. A FIR Neural Network to model the autonomous indebtedness. In *SIGEF'03 Congress*, León , vol 2, pp. 199-209, 2003.

[22] M.P. Cuéllar, M. Delgado, M.C. Pegalajar and R. Pérez. Predicción del endeudamiento económico espanol utilizando modelos bioinspirados. In *SIGEF'03 Congress*, León, vol. 1. pp. 489-510, 2003.

[23] N. Davey, S.P. Hunt, R.J. Frank. Time Series Prediction and Neural Networks, In *proc. 5th International Conference on Engineering Applications of Neural Networks*, pp. 93-98, 1999.

[24] R. Zemomi, D. Racaceanu, N. Zerhonni. Recurrent Radial Basis fuction network for Time Series prediction. *Engineering appl. Of Artificial Intelligence*, 16(5-6):453-463, 2003.

[25] R.J. Williams, D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270-280, 1989.

[26] R.J. Williams, J. Peng. An efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network trajectories. *Neural Computation*, 2:491-501, 1990.

[27] S. Haykin. *Neural Networks (a Coprehensive foundation). Second Edition.* Prentice Hall, 1999.

[28] S.C. Kremer. On the computational power of Elman-style recurrent networks. *IEEE Trans. Neural Networks* 6(4):1000-1004, 1995.

[29] T. Back. *Evolutionary Algorithms in Theory and Practice.* Oxford, 1996.

[30] T. Back, D. Fogel, Z. Michalewicz. *Handbook of Evolutionary Computation.* Institute of Physics Publishing and Oxford University Press, 1996.

[31] Wan. *Finite Impulse Response Neural Networks with applications on time seres prediction.* PhD Dissertation. Stanford University, 1993.

[32] Y. Bengio, P. Simard, P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Trans. on Neural Networks*, 5(2):157-166, 1994.