Evolution of Neural Networks for Classification and Regression*

Miguel Rocha[1], Paulo Cortez[2], and José Neves[1]

[1] Dep. Informática, Universidade do Minho, 4710-057 Braga, PORTUGAL
mrocha@di.uminho.pt jneves@di.uminho.pt
http://www.di.uminho.pt/~mpr
http://www.di.uminho.pt/~jneves
[2] Dep. Sistemas de Informação, Univ. do Minho, 4800-058 Guimarães, PORTUGAL
pcortez@dsi.uminho.pt
WWW home page: http://www.dsi.uminho.pt/~pcortez

Abstract. Although *Artificial Neural Networks (ANNs)* are important *Data Mining* techniques, the search for the optimal *ANN* is a challenging task: the *ANN* should learn the input-output mapping without overfitting the data and training algorithms may get trapped in local minima. The use of *Evolutionary Computation (EC)* is a promising alternative for *ANN* optimization. This work presents two hybrid *EC/ANN* algorithms: the first evolves neural topologies while the latter performs simultaneous optimization of architectures and weights. Sixteen real-world tasks were used to test these strategies. Competitive results were achieved when compared with a heuristic model selection and other *Data Mining* algorithms.
Keywords: Supervised Learning, Multilayer Perceptrons, Evolutionary Algorithms, Lamarckian Optimization, Neural Network Ensembles.

1 Introduction

Artificial Neural Networks (ANNs) denote a set of connectionist models inspired in the behavior of the human brain. In particular, the *Multilayer Perceptron (MLP)* is the most popular *ANN* architecture, where *neurons* are grouped in *layers* and only *forward connections* exist [8]. This provides a powerful base-learner, with advantages such as nonlinear mapping and noise tolerance, increasingly used in *Data Mining* due to its good behavior in terms of predictive knowledge [15].

The interest in *MLPs* was stimulated by the advent of the *Backpropagation* algorithm in 1986 and since then, several fast gradient based variants have been proposed (e.g., *RPROP*) [17]. Yet, these training algorithms minimize an error function by tuning the modifiable parameters of a fixed architecture, which needs to be set a priori. The *MLP* performance will be sensitive to this choice: a small network will provide limited learning capabilities, while a large one will induce generalization loss (i.e., overfitting).

* This work was supported by the *FCT* project POSI/EIA/59899/2004.

Thus, the correct design of the *MLP* topology is a complex and crucial task, commonly addressed by trial-and-error procedures (e.g. exploring different number of hidden nodes), in a *blind* search strategy, which only goes through a small set of possible configurations. More elaborated methods have also been proposed, such as *pruning* [22] and *constructive* [11] algorithms, although these perform *hill-climbing* and are thus prone to local minima.

Evolutionary Computation (EC) is a good candidate for *MLP* design, since it performs a global multi-point (or *beam*) search, quickly locating areas of high quality, even when the search space is very complex. The combination of *EC* and *ANNs*, called *Evolutionary Neural Networks (ENNs)*, is a suitable candidate for topology design, due to the error surface features [25]:

- the number of nodes/connections is unbounded;
- changes are discrete; and
- the mapping from the structure to its performance is indirect: similar topologies may present different performances, while distinct topologies may result in similar behavior.

Indeed, the most common approach when using *EC* to optimize *ANNs* is to evolve neural structures that are trained by gradient based algorithms. However, this presents some drawbacks since the gradient based algorithms used to perform the *MLP* training are sensitive to parameter settings and to the initial weights. Thus, very often, the error surface is rugged and the training process is trapped in local minima. Furthermore, evolving neural structures without any weight information will make it harder to access the real performance of a given topology due to the *noisy fitness evaluation* problem [25]: different random initial weights may produce quite distinct fitness values.

An alternative is to use the global search advantages of *EC* to simultaneously evolve topologies and train the *ANN*, by considering genetic operators that work both on the topology and on the values of the connection weights. Under this context, the use of *Lamarckian evolution* [2] is a promising approach, where *EC* is combined with a local search procedure (e.g. tabu search or gradient based algorithm), improving the fitness of the individuals during their lifetime.

Ensembles are another promising *Data Mining* research field, where several models are combined to produce an answer [5]. Often, it is possible to build ensembles that are better than individual learners given that the errors made by the individual models are uncorrelated, a condition easily met in *ANN Ensembles* since training algorithms are stochastic in nature. Since *ENNs* already create a population of *ANNs* during evolution, this strategy can be easily adapted to *ENNs* with no computational effort increase.

In this work, two *ENN* combinations are presented: the evolution of neural topologies and the simultaneous optimization of *ANN* topologies and weights. In both cases, a direct representation will be used as well as a *structural mutation*, which adds or deletes connections or weights. In the second *ENN*, connection weights are optimized through a Lamarckian evolution that uses a *random mutation* and a gradient-based algorithm (e.g. *RPROP*) for the local search. Both evolutionary techniques will be tested in classification and regression problems,

using single *ANN* and ensemble based models. Then, the results will be compared with a heuristic *ANN* selection procedure, as well with other *Data Mining* methods.

The paper is organized as follows. First, a description is given on the neural and evolutionary algorithms (Section 2). Then, in Section 3 the experiments performed are described and the results analyzed. Finally, conclusions are drawn in Section 4.

2 Learning Methods

2.1 *Data Mining* tasks

This work endorses two important *Data Mining* goals: *classification* and *regression* tasks. The former requires a correct labeling between input attributes and one of several predefined classes (e.g., classifying cells for cancer diagnosis). The latter deals with a mapping between a n-dimensional input vector and a real-value variable (e.g., stock market prediction). The main difference is set in terms of the output representation, since both tasks require supervised learning and can be modeled by *MLPs*. Nevertheless, the performance metrics are quite distinct and some methods can only be applied to one of these goals (e.g. *Decision Trees*) [14].

Two distinct accuracy measures will be adopted: the *Percentage of Correctly Classified Examples (PCCE)*, used in classification tasks; and the *Normalized Root Mean Squared Error (NRMSE)*, applied in the regression ones. These measures are given by the equations:

$$\begin{aligned} \Phi(i) &= \begin{cases} 1\ , & \text{if } T_i = P_i \\ 0\ , & \text{else} \end{cases} \\ PCCE &= \textstyle\sum_{i=1}^{K} \Phi(i)/K \times 100\ (\%) \\ RMSE &= \sqrt{\textstyle\sum_{i=1}^{K} (T_i - P_i)^2/K} \\ NRMSE &= \frac{RMSE}{\sum_{i=1}^{K} T_i/K} \times 100\ (\%) \end{aligned} \tag{1}$$

where K denotes the number of examples; P_i, T_i the predicted and target values for the i-th example. High $PCCE$ values suggest good classifiers, while a good regressor should present a low $NRMSE$.

2.2 Neural Networks

The *MLPs* used in this study make use of biases, sigmoid activation functions and one hidden layer with a variable number of nodes. A different approach was followed for the regression tasks, since outputs may lie out of the logistic output range ($[0, 1]$). Hence, shortcut connections and *linear* functions were applied on the output neuron(s), to scale the range of the outputs (Fig. 1) [7]. This solution avoids the need of filtering procedures, which may give rise to information loss.

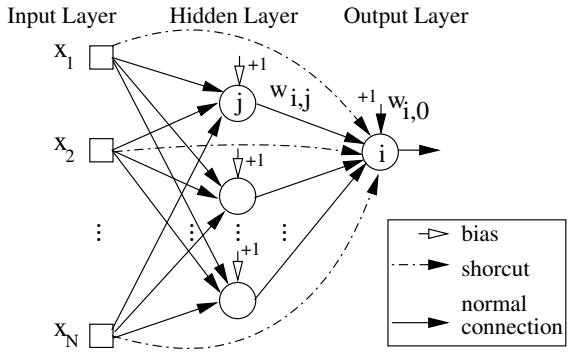

Fig. 1. A fully connected *Multilayer Perceptron* with one output neuron, bias and shortcuts.

Before feeding the *MLPs*, the data was preprocessed with a *1-of-C* encoding (one binary variable per class) applied to the nominal attributes and all inputs were rescaled within the range $[-1, 1]$. For example, the **safety** attribute from the task **car** was encoded as: *low* $\rightarrow$ (1 -1 -1), *med* $\rightarrow$ (-1 1 -1) and *high*$\rightarrow$ (-1 -1 1). Regarding the outputs, the discrete variables were normalized within the range $[0, 1]$ (using also a *1-of-C* encoding for the nominal attributes). Therefore, the predicted class is given by the nearest class value to the node's output, if one single node is used (binary variable); otherwise the node with the highest output value is considered. On the other hand, regression problems will be modeled by one real-valued output, which directly represents the *dependent* target variable.

2.3 Heuristic model selection

In order to provide a basis for comparison with the *ENNs* developed in this work, an *Heuristic* approach (*HNN*) to model selection was defined by a simple trial-and-error procedure, where $N + 1$ fully connected *MLPs*, with a number of hidden nodes ranging from 0 to N, are compared.

For each *MLP*, the initial weights were randomly set within the range $[-1, 1]$. Next, the *RPROP* algorithm [17] was selected for training, due to its faster convergence and stability. This procedure is stopped after a maximum of 500 epochs or when the error slope was approaching zero. Then, the *MLP* with the lowest validation error (computed over non training data) was selected.

A distinct approach is to use the trained *MLPs* (as before) in order to build an *ANN Ensemble*. When building ensembles, two main issues need to be contemplated: the method used to select individual models to be part of the ensemble and how to combine their outputs into a single value. Previous work by the authors [19] has shown that:

– Using individual *ANNs* with *heterogeneous topologies*, where a family of *MLPs* with distinct structures (and therefore complexities) are combined,

outperforms other heuristics such as *injecting randomness* [4] (i.e. training *MLPs* with the same topology but distinct initial weights);
- The best way to combine the outputs of each individual *MLP* is to use a simple average, where the output of the ensemble is given by:

$$E_{i,j} = (\sum_{k=0}^{N} S_{i,j,k})/(N+1) \quad (2)$$

where $S_{i,j,k}$ denotes the j output node value for the i example, given by a *MLP* with k hidden nodes; $E_{i,j}$ is the final ensemble output for the i example and j output node. For classification tasks, the interpretation of the outputs is made over the averaged values ($E_{i,j}$).

Therefore, these two options will be used in this work, in a configuration denoted by *Heuristic Neural Network Ensemble (HNNE)*, which is build of $N+1$ trained *MLPs*, with the number of hidden nodes varying from 0 to N. It is important to note that this method does not imply extra computational work when compared with the *HNN* configuration.

2.4 Evolution of Neural Network Topologies

In this work, the first *ENN* to be presented is the *Topology-optimization Evolutionary Neural Network (TENN)*, where the aim is to evolve the optimal structure of a *MLP* to a given problem. Under this approach, a *direct* representation was embraced, where the genotype is the whole *MLP*.

The data is divided into training and validation sets (TS and VS). Then, each individual is evaluated by setting the weights of its connections to a random value and training the *MLP* in its genotype, using the process described above, i.e., the *RPROP* algorithm with 500 epochs. The value of the error of the trained *MLP* in the validation set gives the fitness of the individual ($RMSE$ on VS). In both classification and regression tasks, the error metric used by the fitness function is the $RMSE$, since this is a smoother statistic than $PCCE$. Nevertheless, in the analysis of the results (next section), the $PCCE$ will be used in the classification datasets and the $NMSE$ for the regression ones.

Regarding the initialization of the population, each individual is set by choosing a random number of hidden nodes between 0 and H. Then, each possible connection is set with a probability of p. New individuals are created by the application of a structural *mutation* operator, which works by adding or deleting a random number (generated between 1 and M) of nodes or connections (Fig. 2.c and 2.d). This was the only operator used in the evolution of new solutions, since the crossover operator was discarded due to the fact that previous experiments [18] revealed no gain in its use. This behavior is probably related to the *permutation* problem [25]; i.e., several genomes may encode the same *MLP*. Nevertheless, this *ENN* is still potentially able to search through any kind of *MLP* connectivity, ranging from linear models to complex nonlinear *MLPs*.

In *TENN*, a population of P individuals evolves for G generations and a *selection* scheme involving the conversion of fitness values into rankings and the

application of a roulette wheel scheme is used. In every generation, half of the individuals are kept, and the remaining are bred using the genetic operator. The best individual is always kept, i.e. an *elitism* of 1 is used. The whole *TENN* procedure is given in by the pseudo-code:

```
BEGIN
   Split the data into TS and VS
   i ← 0
   Ψ_i ← Initialize the population with P MLPs
   Evaluate the initial population Ψ_i
   WHILE (i < G)
      A_i ← Select P/2 ancestors from Ψ_i for reproduction
      O_i ← Apply the structural mutation over all elements of A_i
      Evaluate the offspring O_i
      S_i ← Select P/2 − 1 survivors from Ψ_i
      Set the next generation (Ψ_{i+1} ← best(Ψ_i) ⋃ S_i ⋃ O_i)
      i ← i + 1
END
```

The evaluation function for a given population Ψ of size P_Ψ is given by:

```
BEGIN
   j ← 1
   WHILE (j ≤ P_Ψ)
      MLP_j ← Select the j-th MLP from the population Ψ
      Initialize the MLP_j weights
      Train the MLP_j on TS with 500 RPROP epochs
      Set the MLP_j fitness as the RMSE on VS
      j ← j + 1
END
```

A *TENN Ensemble (TENNE)* will be built using the best N individuals (the ones with lower validation errors) obtained during the evolutionary process. The ensemble's output is computed as the average of the *MLPs* (Eq. 2). This step does not imply any significant extra computational effort over *TENN*.

2.5 Simultaneous Evolution of Neural Network Topologies and Weights

In the previous *TENN* approach, an a priori architecture needs to be set before training. On the other hand, evolving neural structures without weight information will make harder the fitness evaluation due to the noisy fitness evaluation problem. A possible solution to this problem is to to simultaneous evolve topologies and weights [26].

Therefore, a *Simultaneous Evolutionary Neural Network (SENN)* algorithm is proposed. The direct representation used in *TENNE* is kept, and the evolutionary algorithm uses two different mutation operators (Fig. 2), each with an equal

probability of application (50%): the *structural mutation* presented above and a *macro mutation*, which operates over the connection weight values by replacing a random number (from 1 to M) of these with new randomly generated values within the range $[-1.0, 1.0]$.

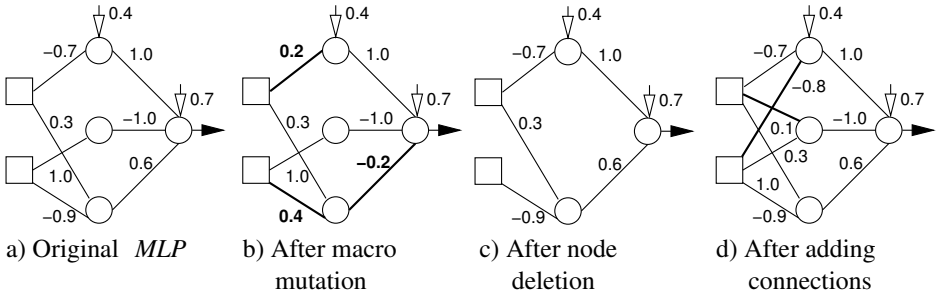

Fig. 2. Example of the application of the mutation operators.

This algorithm will also be combined with a local optimization procedure, under a *Lamarckian* evolution setting [3]. In each generation, L epochs of the *RPROP* learning algorithm are applied to every individual (*MLP*) in the population, using the examples in the training set. In past work [18], this Lamarckian approach (with macro mutation) to training outperformed eight *evolutionary algorithms* (using different crossovers and mutations) and gradient-based algorithms (e.g. *Backpropagation* and *RPROP*).

The *selection* procedure, substitution rate and termination criteria are kept from the *TENN* approach presented above. On the other hand, the initialization of the population also follows a similar process, completed by a step that randomly initializes the weights within the range $[-1.0; 1.0]$. Thus, the *SENN* procedure is given by following pseudo-code:

```
BEGIN
   Split the data into the TS, FS and MS sets
   i ← 0
   Ψ_i ← Initialize the population and weights of the P MLPs
   WHILE (i < G)
      Train the MLPs from Ψ_i on TS with L RPROP epochs
      Evaluate the current population Ψ_i
      A_i ← Select P/2 ancestors from Ψ_i for reproduction
      O_i ← Apply the macro (50%) or structural (50%) mutations on
            all elements of A_i
      S_i ← Select P/2 − 1 survivors from Ψ_i
      Set the next generation (Ψ_{i+1} ← best(Ψ_i) ⋃ S_i ⋃ O_i)
      i ← i + 1
END
```

Regarding the *fitness function*, a different strategy was used here, since the simultaneous evolution of weights and topologies is very sensitive to overfitting. Thus, the validation set is subdivided into: a *fitness* set (FS), used for the fitness evaluation (the $RMSE$ is also used), and a *model selection* set (MS), used to select the best individual at the end of the process. This evaluation function is performed using the pseudo-code:

```
BEGIN
   j ← 1
   WHILE (j ≤ P_Ψ)
      MLP_j ← Select the j-th MLP from the population Ψ
      Set the MLP_j fitness as the RMSE on FS
      j ← j + 1
END
```

As before, a *SENN Ensemble (SENNE)* will be built using the best N individuals (in this case the ones with lower errors in the second validation set) obtained during the evolutionary process.

3 Results and Discussion

In this work, sixteen real-world datasets were selected from the UCI repository, which is commonly used to benchmark learning algorithms [21]. The main features are listed in Table 1, namely: the number of **num**eric, **bin**ary and **nom**inal (i.e. discrete with three or more distinct labels) input attributes, as well as the number of **ex**amples and classes. The regression tasks are identified by the symbol $\Re$ (last eight rows).

In the last decades, and due to increase attention given to the *Data Mining* field, several models and algorithms have been proposed, with each one presenting its own purposes and capabilities [14]. In this work, five of the most popular *Data Mining* models/algorithms will be selected, as basis of comparison with the neural models, namely:

- a classification *Decision Tree* based on the C4.5 algorithm (*J48*);
- a regression *Decision Tree* (M5 algorithm);
- a *k-Nearest Neighbor* (*IB5*);
- an *Instance Based Algorithm* (*KStar*); and
- a *Support Vector Machine (SVM)*.

The *ANN/EC* experiments were conducted using a software package developed in *JAVA* by the authors, while the other *Data Mining* techniques were computed using the *WEKA* environment with its default parameters [24]. For each model, 30 runs of a 5-fold cross-validation process [9] (stratified in the classification tasks) were executed. This means that in each of these 150 experiments, 80% of the data is used for learning and 20% for testing.

For all setups, the learning data was divided into *training* (50% of the original dataset) and *validation* sets (30%). In case of the *SENN/SENNE* algorithms,

Table 1. A summary of the datasets used.

Task	Description	Inputs			Ex.	Classes
		Num.	Bin.	Nom.		(K)
balance	balance scale weight and distance	4	0	0	625	3
bupa	BUPA liver disorders	6	0	0	345	2
car	car evaluation database	0	0	6	1728	4
cmc	contraceptive method choice	5	3	1	1473	3
dermat	dermatology database	34	0	0	366	6
ionosp	radar returns from the ionosphere	34	0	0	351	2
sonar	sonar classification (rocks vs mines)	60	0	0	104	2
yeast	protein localization sites	7	1	0	1484	10
abalone	age of abalone	7	0	1	4177	$\Re$
autos	auto imports database	17	3	5	205	$\Re$
breast	breast cancer time to recur	1	4	4	286	$\Re$
heart	Cleveland heart disease database	6	3	4	303	$\Re$
housing	housing prices in suburbs of Boston	12	1	0	506	$\Re$
mpg	Auto-Mpg (miles per gallon)	5	0	2	398	$\Re$
servo	rise time of a servomechanism	2	0	2	167	$\Re$
wpbc	Wisconsin prognostic breast cancer	32	0	0	194	$\Re$

the *validation* set was subdivided into *fitness* (15% of the original dataset) and *model selection* (15%) sets. Finally, the $SENN/TENN$ parameters were set to $N = 20$, $P = 20$, $p = 50\%$, $H = 10$, $L = 50$, $M = 5$ and $G = 20$.

Tables 2 and 3 show the average of the errors measured over the 30 runs, for each method and instance. The 95% t-student confidence intervals [6] are also shown for the evolutionary approaches. The last row of each table averages the global performance of each learning strategy, measured over all instances.

Some evidences are clearly shown from the results. In the classification tasks, it is obvious that the *ANN*-based learning models (the last six columns) are quite competitive, outperforming the other *Data Mining* algorithms. In fact, the two exceptions to this behavior are the **dermat**ology and **sonar** instances, where the *SVM* and *KStar* get the best results.

Regarding the single *ANN* approaches, both evolutionary algorithms (*TENN* and *SENN*) excel the heuristic method (*HNN*), with a difference of 0.8% and 1% in the average performance. When analysing each particular dataset, the *TENN* (*SENN*) shows a significant improvement in 3 (4) of the 8 tasks ($p-$value $<$ 0.05). On the other hand, the *ANN Ensemble*-based versions (*HNNE*, *TENNE* and *SENNE*) obtain better results when compared with the correspondent single *ANN* ones, with improvements ranging from 1.2% to 1.5%. At the dataset level, the improvements of *TENNE/SENNE* are significant in 6/5 cases. Overall, the evolutionary ensembles (*TENNE* and *SENNE*) clearly show the best accuracy over all alternatives considered in this study.

For the regression tasks, the decision tree (*M5P*) is quite competitive, outperforming all the other *Data Mining* algorithms and *ANN* approaches except

Table 2. The classification results ($PCCE$ values, in %).

Task	J48	IB5	KStar	SVM	HNN	TENN	SENN	HNNE	TENNE	SENNE
balance	78.3	87.6	88.3	87.6	94.9	95.2$_{\pm 0.4}$	96.4$_{\pm 0.5}^{\dagger}$	95.7	96.0$_{\pm 0.4}$	**96.7**$_{\pm 0.4}^{\ddagger}$
bupa	64.6	61.1	65.6	57.9	68.3	67.9$_{\pm 1.0}$	67.8$_{\pm 1.0}$	69.6	**70.9**$_{\pm 0.8}^{\ddagger}$	70.4$_{\pm 0.8}^{\ddagger}$
car	91.3	92.3	87.1	93.4	97.3	98.3$_{\pm 0.2}^{\dagger}$	98.7$_{\pm 0.2}^{\dagger}$	98.4	98.9$_{\pm 0.1}^{\ddagger}$	**99.0**$_{\pm 0.1}$
cmc	51.3	47.0	49.7	48.4	50.6	54.7$_{\pm 0.5}^{\dagger}$	53.6$_{\pm 0.5}^{\dagger}$	52.3	**55.5**$_{\pm 0.4}$	54.8$_{\pm 0.4}^{\ddagger}$
dermat	96.0	96.7	94.4	**97.5**	96.1	95.7$_{\pm 0.4}$	95.3$_{\pm 0.5}$	96.5	96.8$_{\pm 0.3}^{\ddagger}$	96.5$_{\pm 0.4}^{\ddagger}$
ionosp	89.9	84.5	83.7	88.0	89.9	89.8$_{\pm 0.7}$	90.5$_{\pm 0.7}$	91.5	**92.8**$_{\pm 0.5}^{\ddagger}$	92.2$_{\pm 0.6}^{\ddagger}$
sonar	72.9	81.0	**84.4**	77.0	78.2	79.0$_{\pm 1.1}$	79.5$_{\pm 1.0}$	81.8	81.3$_{\pm 0.9}^{\ddagger}$	81.6$_{\pm 1.1}$
yeast	55.7	57.0	53.2	56.5	58.5	59.3$_{\pm 0.4}^{\dagger}$	59.7$_{\pm 0.5}^{\dagger}$	60.1	**60.3**$_{\pm 0.3}^{\ddagger}$	**60.3**$_{\pm 0.4}$
Mean	75.0	75.9	75.8	75.8	79.2	80.0	80.2	80.7	**81.5**	81.4

† - Statistically significant (p-value< 0.05) under pairwise comparison with *HNN*.
‡ - Statistically significant under pairwise comparison with the non ensemble version.

TENNE and *SENNE*. However, as before, the evolutionary models excel the heuristic one, with average improvements from 1.6% to 2.6%. The differences are statistically significant in 5 datasets for *TENN* and in 3 problems for *SENN*. The ensembles also behave better than their single based *ANN*'s counterparts (from 0.8% to 3.4% in average), presenting the best overall results. There is a statistical difference for *TENNE* over *TENN* in 5 cases, while this number increases to 7 when comparing *SENNE* over *SENN*.

Table 3. The regression results ($NRMSE$ values, in %).

Task	M5P	IB5	KStar	SVM	HNN	TENN	SENN	HNNE	TENNE	SENNE
abalone	21.5	23.0	22.4	22.7	21.2	21.1$_{\pm 0.1}$	21.1$_{\pm 0.1}$	**21.0**	21.1$_{\pm 0.0}$	**21.0**$_{\pm 0.1}$
autos	13.6	19.5	19.5	**12.5**	13.2	14.6$_{\pm 0.5}$	13.9$_{\pm 0.6}$	13.5	13.4$_{\pm 0.4}^{\ddagger}$	12.6$_{\pm 0.5}^{\ddagger}$
breast	**41.0**	44.3	44.5	41.9	43.0	42.5$_{\pm 0.5}^{\dagger}$	43.9$_{\pm 0.8}$	48.6	41.7$_{\pm 0.5}$	41.7$_{\pm 0.4}^{\ddagger}$
heart	**21.0**	22.3	25.6	21.6	22.1	21.6$_{\pm 0.3}$	22.4$_{\pm 0.4}$	22.6	21.1$_{\pm 0.2}$	21.3$_{\pm 0.2}^{\ddagger}$
housing	17.5	22.2	18.2	22.6	19.0	17.6$_{\pm 0.4}^{\dagger}$	17.4$_{\pm 0.5}^{\dagger}$	16.2	**16.0**$_{\pm 0.3}^{\ddagger}$	**16.0**$_{\pm 0.4}^{\ddagger}$
mpg	12.0	14.8	14.3	13.2	13.9	12.5$_{\pm 0.2}^{\dagger}$	12.8$_{\pm 0.3}^{\dagger}$	12.0	**11.7**$_{\pm 0.1}^{\ddagger}$	11.9$_{\pm 0.2}^{\ddagger}$
servo	43.7	59.6	66.8	69.8	60.6	45.5$_{\pm 2.8}^{\dagger}$	48.0$_{\pm 2.3}^{\dagger}$	48.8	**40.0**$_{\pm 2.6}^{\ddagger}$	42.1$_{\pm 2.8}^{\ddagger}$
wpbc	73.5	73.4	97.2	71.7	76.5	73.2$_{\pm 0.9}^{\dagger}$	77.0$_{\pm 1.5}$	80.7	**70.9**$_{\pm 0.6}^{\ddagger}$	72.2$_{\pm 1.0}^{\ddagger}$
Mean	30.5	34.9	38.6	34.5	33.7	31.1	32.1	32.9	**29.5**	29.9

† - Statistically significant under pairwise comparison with *HNN*.
‡ - Statistically significant under pairwise comparison with the non ensemble version.

When comparing the average results of the best two methods, the $TENNE$ algorithm slightly outperforms $SENNE$ both in classification and regression. Yet, in the vast majority of the datasets, the differences are not statistically

significant. Since a similar level of accuracy was obtained, another dimension will be used in the analysis of both alternatives: the computational effort. In this parameter, the evaluation clearly favors the *SENN/SENNE* algorithms, whose computational burden is similar to the one required by the *HNN/HNNE*.

It should be noted that the computational load of each evolutionary approach can be approximated by the number of epochs in the *ANN* training. All other computational processes are negligible when compared with this measure. The overhead of the *TENN/TENNE* is due to the fitness evaluation process, where a complete training of the *ANN* (500 epochs of the *RPROP* algorithm) is required. In each generation, this procedure is applied to half of the population (the offspring). In contrast, in the *SENN/SENNE* approaches, the evaluation process is straightforward, involving only the calculation of the error metric. In this case, the *RPROP* is only used in the lamarkcian operator, which is applied to all individuals and implies 50 epochs. As a result, the *TENN/TENNE* algorithms requires a computational time that is 500% higher than *SENN/SENNE*.

For demonstrative purposes, Fig. 3 plots the evolution of the best and average fitnesses of the population, for a given *SENN* simulation with the **bupa** task. The figure shows a steady evolution, suggesting also diversity within the population, since the two curves do not overlap. This example was executed in a *Pentium IV 3.4 GHz* processor, demanding a computational time of 65 seconds. The best evolved *MLP* contains a total of 9 hidden nodes, 7 bias and 4 shortcut connections, denoting a strong nonlinearity for this task.

Similar work to the *SENN* approach has been reported in literature, namely the EPNet system [12], which obtained interesting results. However, this approach was only applied to five UCI datasets where the best results are obtained by low complexity *MLPs* (in some cases linear models). It is not surprising that, since EPNet heavily promotes simple models, good results were obtained for these cases. In this work, the majority of the problems demanded *MLPs* with a much higher number of hidden nodes, where is it believed that the EPNet system would not excel.

4 Conclusions

In this work, two evolutionary algorithms were proposed for *ANN* optimization: the evolution of topologies and the simultaneous evolution of weights and topologies. The performances of these methods were favorably compared with other *Data Mining* algorithms (e.g. *Decision Trees* or *Support Vector Machines*) and with heuristic approaches to *ANN* model selection. The two evolutionary models were enhanced by considering *ANN Ensembles*, which combine the outputs of the best *ANNs* obtained during evolution, in order to create more accurate models. When taking both accuracy and computational effort parameters into consideration, the best available option is to consider the simultaneous evolution strategy and to use the resulting *ANN Ensemble* as a prediction model.

The major contributions of this work were the following: i) in contrast with previous studies, an exhaustive comparison is performed over the two main evo-

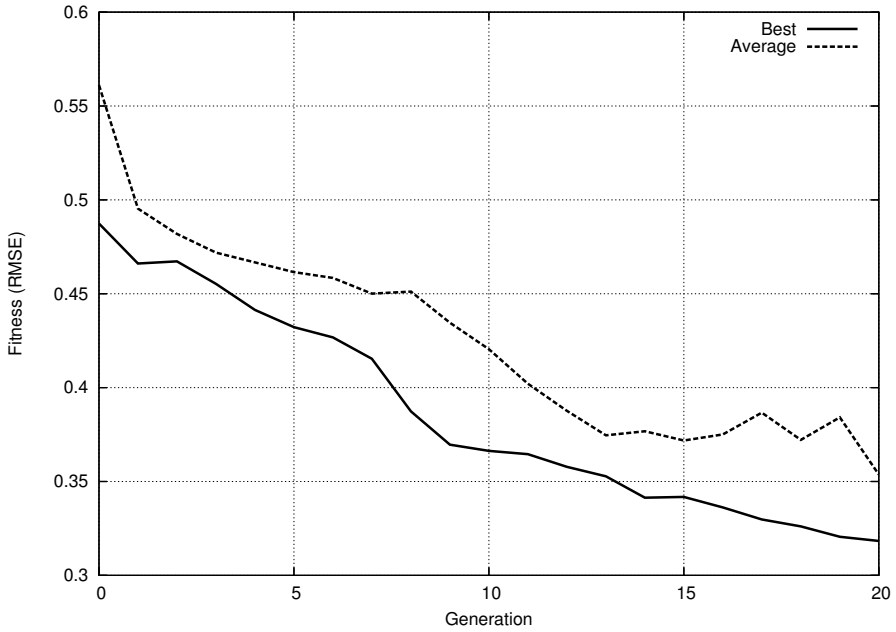

Fig. 3. Example of the fitness evolution for the **bupa** task.

lutionary approaches for *ANN* design and heuristic approaches to model selection, encompassing a total of 16 classification and regression datasets, whose underlying models denote different degrees of nonlinearity; ii) the proposal and evaluation of several methods to build *ANN* ensembles (*HNNE*, *TENNE* and *SENNE*); iii) the splitting of the *validation set* into two components in simultaneous evolution, a feature that was necessary to boost its performance.

Although the considered genetic operators (structural and random mutations) are quite simple, they present a huge flexibility. Indeed, the proposed approach can be easily extended to other *ANN* architectures (e.g., *Recurrent Neural Networks* [10]) or learning scenarios (e.g. *Reinforcement Learning* [16]). In the former case, the *RPROP* local learning could be substituted by an appropriate learning algorithm (e.g. *Backpropagation Through Time* [23]), while in the latter, the rewards from the environment can be used to implement the local search heuristics.

In future work, the modeling of biological processes (fed-batch fermentations) will be taken as a case study of this technology [20]. In this task, a number of variables evolves over time (e.g. glucose, biomass, weight of the fermenter, dissolved oxygen, etc), and an accurate model to predict this behavior over time is a major requirement to perform adequate optimization and online control of the process.

One other direction in future work will be the development of more elaborated strategies to build *ANN Ensembles*, namely by designing fitness functions which reward specialization [13], with the aim that individual models can focus in certain areas of the search space. Furthermore, the incorporation of other parameters of the *ANN* structure, such as the transfer function and the learning algorithm (and specific parameters) to use in Lamarckian evolution, will also be considered [1].

References

1. A. Abraham. Meta learning evolutionary artificial neural networks. *Neurocomputing*, 56:1–38, January 2004.
2. D. H. Ackley and M. L. Littman. *A case for Lamarckian evolution*, pages 3–10. Addison-Wesley, Reading, MA, 1994.
3. P. Cortez, M. Rocha, and J. Neves. A Lamarckian Approach for Neural Network Training. *Neural Processing Letters*, 15(2):105–116, April 2002.
4. T. Dietterich. Machine Learning Research: Four Current Directions. *AI Magazine*, 18(4):97–136, 1997.
5. T. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems, LNCS 1857*, pages 1–15. Springer, 2001.
6. A. Flexer. Statistical evaluation of neural networks experiments: Minimum requirements and current practice. In *Proceedings of the 13th European Meeting on Cybernetics and Systems Research*, volume 2, pages 1005–1008, Vienna, Austria, 1996.
7. T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, NY, USA, 2001.
8. S. Haykin. *Neural Networks - A Compreensive Foundation*. Prentice-Hall, New Jersey, 2nd edition, 1999.
9. R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1143, Montreal, Quebec, Canada, August 1995. Morgan Kaufmann.
10. Kim Ku, Man Mak, and Wan-Chi Siu. A study of the lamarckian evolution of recurrent neural networks. *IEEE Transactions on Evolutionary Computation*, 4(1):31–42, April 2000.
11. T. Kwok and D. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems: A survey. *IEEE Transactions on Neural Networks*, 8(3):630–645, May 1999.
12. Y. Liu and X. Yao. Evolving Modular Neural Networks Which Generalize Well. In *Proc. of the 1997 IEEE Intern. Confer. on Evolutionary Computation, Indianapolis*, pages 670–675, New York, 1997. IEEE Press.
13. Y. Liu, X. Yao, and T. Higuchi. Evolutionary Ensembles with Negative Correlation Learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.
14. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
15. S. Mitra, S. Pal, and P. Mitra. Data Mining in Soft Computing Framework: A Survey. *IEEE Trans. on Neural Networks*, 13(1):3–14, January 2002.
16. David Moriarty, Alan Schultz, and John Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence*, 11:199–299, August 1999.

17. M. Riedmiller. Advanced Supervised Learning in Multilayer Perceptrons - From Backpropagation to Adaptive Learning Algorithms. *Int. Journal of Computer Standards and Interfaces*, 16:265–278, 1994.
18. M. Rocha, P. Cortez, and J. Neves. Evolutionary Neural Network Learning. In F. Pires and S. Abreu, editors, *Progress in Artificial Intelligence, EPIA 2003 Proceedings, LNAI 2902*, pages 24–28, Beja, Portugal, December 2003. Springer.
19. M. Rocha, P. Cortez, and J. Neves. Ensembles of Artificial Neural Networks with Heterogeneous Topologies. In *Proceedings of the 4th Symposium on Engineering of Intelligent Systems (EIS2004)*. CD-ROM edition, ICSC Academic Press, March 2004.
20. M. Rocha, J. Neves, I. Rocha, and E. Ferreira. Evolutionary Algorithms for Optimal Control in Fed-batch Fermentation Processes. In G.Raidl et al., editor, *Proceedings of the Workshop on Evolutionary Bioinformatics - EvoWorkshops 2004, Lecture Notes in Computer Science 3005*, pages 84–93. Springer, 2004.
21. C. Soares. Is the UCI Repository Useful for Data Mining? In F. Pires and S. Abreu, editors, *Progress in Artificial Intelligence, EPIA 2003 Proceedings, LNAI 2902*, pages 209–223, Beja, Portugal, 2003. Springer.
22. G. Thimm and E. Fiesler. Evaluating pruning methods. In *Proc. of the Int. Symp. on Artificial Neural Networks*, pages 20–25, Taiwan, December 1995.
23. P. Werbos. Backpropagation through time: what it does and how to do it. *Proc. IEEE*, 78(10):1550–1560, 1990.
24. I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, USA, 2000.
25. X. Yao. Evolving Artificial Neural Networks. In *Proc. of the IEEE*, 87(9): 1423-1447, September 1999.
26. X. Yao and Y. Liu. A New Evolutionary System for Evolving Artificial Neural Networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, 1997.