Pattern classification with evolved RBF Neural Networks

V.M. Rivas a,b, M.G. Arenas a, P.A. Castillo c, J.J. Merelo c, A. Prieto c and G. Romero c

aDepartamento de Informática de la Universidad de Jaén (Spain)
bhttp://www.VictorRivasSantos.com
cDepartamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada (Spain)

Abstract

This paper shows the latest results of the EvRBF algorithm applied to classification problems. EvRBF is an Evolutionary Algorithm intended to automatically design Radial Basis Functions Neural Networks (RBFNN). EvRBF has been programmed using Evolutionary Objects (EO) library, so that it works with data structures that represent the net directly, instead of binary codifications. This allows the use of specific operators that interchange information between individuals, and produce new ones, relying on the specific features of RBFNN. Experiments using machine learning benchmarks show good behavior of EvRBF with respect to the quality and size of the solutions and the execution time, despite being an Evolutionary Algorithm that does not perform local tuning. These results are compared to other evolutionary neural net algorithms in the literature, showing some improvement over them.

Key words: Radial basis function neural networks, evolutionary algorithms, evolutionary operators, EvRBF, classification, evolutionary algorithms representation.

1 Introduction

Radial Basis Function Neural Networks (RBFNN), introduced by Broomhead and Lowe in [1], are two-layer, fully-connected, feed-forward networks, in which hidden neuron activation functions are Radial Basis Functions (RBF), usually Gaussian.
RBFNN output is given by eq. 1.

\[ s_j(\vec{x}_k) = \lambda_{0j} + \sum_{i=1}^{p'} \lambda_{ij} \phi_i(\vec{x}_i, \vec{c}_i, \vec{r}_i) \ ; k = 1..p, j = 1..n', s_j \in \mathbb{R}, \vec{x}_k \in \mathbb{R}^n \]  

(1)

where:

- \( \phi_i \) is the RBF assigned to hidden neuron \( i \)
- \( \lambda_{0j} \) is a bias term
- \( \lambda_{ij} \) represents the weight between hidden neuron \( i \) and output neuron \( j \)
- \( \vec{c}_i \) and \( \vec{r}_i \) are called, respectively, the center and radii (or widths) of the RBF
- \( n \) and \( n' \) are the input and output space dimensions, respectively
- \( p' \) is the number of hidden neurons, and \( p \) is the number of patterns to which \( s_j \) is going to be applied.

RBFNN’s main advantage is that optimal biases and weights (i.e., \( \lambda_{0j} \) and \( \lambda_{ij} \)) can be efficiently computed for a certain set of desired output, once the number of hidden neurons, centers and radii have been set. This is shown in eqs. 2 to 4.

\[
\begin{pmatrix}
  f_{11} & \cdots & f_{1n'} \\
  \vdots & \ddots & \vdots \\
  f_{p1} & \cdots & f_{pn'}
\end{pmatrix}
= 
\begin{pmatrix}
  1 & A_{11} & \cdots & A_{1p'} \\
  \vdots & \ddots & \ddots & \ddots \\
  1 & A_{p1} & \cdots & A_{pp'}
\end{pmatrix}
\begin{pmatrix}
  \lambda_{01} & \cdots & \lambda_{0n'} \\
  \vdots & \ddots & \ddots & \vdots \\
  \lambda_{p1} & \cdots & \lambda_{p'n'}
\end{pmatrix}
\]

(2)

which can also be expressed in matricial form as in eq. 3:

\[ F = A\lambda \]  

(3)

whose solution is given by eq. 4:

\[ \lambda = A^{-1}F \]  

(4)

where \( F \) is the set of desired outputs; \( A \) is the so-called design matrix (where \( A_{ij} \) represents the output of hidden neuron \( j \) when input pattern \( i \) is applied to the net); \( \lambda \) is the set of weights and biases (being \( \lambda_{jk} \) the weight of the connection between hidden neuron \( j \) and output neuron \( k \)); and \( A^{-1} \) represents the pseudo-inverse of matrix \( A \).
Biases and weights calculated using $A^{-1}$ yield the minimum mean square error (MSE). Using less hidden neurons than values to be approximated (i.e., $p' << p$), singular value decomposition (SVD)[2] or any gradient descent method can be used to compute $A^{-1}$.

This paper shows the latest results obtained by EvRBF, which was designed to dynamically build the optimal RBFNN that solves a classification problem. EvRBF is an evolutionary algorithm that includes RBFNN-specific operators that modify the RBFNN structure: number of hidden neurons, and centers and radii related to them. No limit is a priori imposed to the size of nets.

EvRBF does not use a binary alphabet for codification, therefore, the Scheme Theory [3,4] does not apply. Nevertheless, extended representations are supported by the Forma Analysis Theory [5] that extends the former with chromosomes equivalence classes, instead of schemas. The Forma Analysis Theory ensures that using operators independent of the chromosome representation allows the definition of independent evolutionary algorithms. These algorithms will perform as well as standard genetic algorithms. For this reason, EvRBF has been programmed using the C++ library EO (Evolutionary Objects) [6–8]. EO is based on the idea that any object that can be compared and mutated, can be evolved. Evolution does not depend on the way the object is codified. This property allows EvRBF to handle directly the data structure containing the RBFNN, instead of a codification or representation of it. Thus, the algorithm looks for a RBFNN with few restrictions, instead of the whole space of nets represented by a given codification.

The rest of the paper is organized as follows: Section 2 shows the state of the art, reviewing some of the methods in literature intended to generate RBFNN. Section 3 describes EvRBF algorithm, paying a special attention to the specific operators created to deal with RBFNN. Section 4 shows the experiments carried out to test EvRBF in classification problems; results obtained by EvRBF have been compared to others found in the scientific literature. Finally, section 5 describes conclusions and future research lines.

2 State of the art

Methods to automatically design RBFNN can be divided in evolutionary and non-evolutionary methods. Following [9], non-evolutionary methods can be classified according to the number of steps needed to train a net.
2.1 Local search methods

One-step methods select points from the set of available patterns to act as centers of the RBF; radii are fixed to a pre-established value; then, connection weights are computed. Support Vector Machines (SVM) [10] turn into RBFNN when Gaussian functions are used as activation function, and are trained according to this method.

Two-step methods are the most popular. The first step uses any kind of algorithm to choose points in the input space that will be set as the centers of the RBF. These points might differ from available in the training set. Methods such as center clustering [11], learning vector quantization [12,13], and decision trees [14], have been used to select the centers.

Establishing radii, which is a critical task [15], is also done in this step. Frequently, radii are set proportionally to the average distance from one hidden neuron to the $k$ closest ones. The ratio goes from 0.5 to 1.75 times the distance, while $k$ varies from 1 to $p' - 1$ (being $p'$ the number of hidden neurons).

Whichever the method used to set centers and radii, the second step calculates the corresponding set of weights.

Three-step methods [9,16] are intended to iteratively modify centers and radii, using a back-propagation algorithm based on the error yielded by the net. Thus, first step uses an algorithm to set centers and radii; second step computes the weights; and third step uses second order gradient descent to tune the centers and radii.

These methods have mainly two disadvantages: size of hidden layer must be set a priori, and, gradient methods that tune weights, centers and radii can be trapped in local minima.

Hybrid methods can modify the number of neurons composing the hidden layer. In this sense, incremental methods start with a few neurons (or even only one), adding new ones in order to reduce as much as possible the error yielded by the net. Methods from this group are Orthogonal Least Squares (OLS) [17], Regularised OLS [18], Resource Allocating Networks (RAN) [19], and Cell Structure Growing [20]. The differences between them are the way they determine whether the growing process must be stopped or not, and the mechanism to select the new neuron to be added in its case.

Decremental or pruning methods start with an over-specified net, removing hidden neurons iteratively. The best example of this mechanism applied to RBFNN is [21] in which, according to the Minimum Description Length
measure (MDL), a set of neurons describing the training data with the shortest possible encoding are selected among a larger set.

Both, incremental and decremental methods tend to be very restrictive, giving more importance to the number of neurons than to its accuracy, and thus finding suboptimal networks.

2.2 Evolutionary Algorithms

Evolutionary algorithms have been successfully applied to neural net design in many different ways [22]. Many methods try to optimize only the synaptic weights or the structure of the net, but there are also others intended to optimize the whole net, such as [23] for Learning Vector Quantization (LVQ) nets or [24] for multilayer perceptrons.

RBFNN have also been optimized by means of EAs: first papers [25,26] used binary codification and were constricted by the number of hidden neurons, that had to be set a priori. Both methods differ in they way they build the net. Carse [25] works with a population of nets, while Whitehead [26] works with a population of neurons (thus, only one RBFNN was build and evaluated) that compete but also cooperate to find the optimal net.

Subsequent papers [27] presented an algorithm, based on real number representation, with some advantages, like fast operators, dynamic optimization of hidden layer size, easy computation of fitness function, and fast convergence to a valid solution. Nevertheless, the empirical imposition of a limit to the number of neurons, no optimization of radii but only centers for RBF, and a badly defined penalization term for the number of times an individual can reproduce, are the main drawbacks to be considered for this algorithm.

Recent applications of evolutionary algorithms to RBFNN design [28,29] try to overcome the disadvantages of the preceding methods. González [28] starts with an improved clustering method for nets initialization, and then uses a series of mutation operators, combined with a local search method, to efficiently search the RBFNN parameters. Rivera [29] offers a different starting point, as many neurons compete and are modified using fuzzy evolution, i.e., using a fuzzy rules table that specifies the operator that must be applied to a neuron in order to improve its behavior.

The algorithm presented herein falls into this last kind of algorithms, intended to estimate the full RBFNN that solves a problem. It is a fast algorithm (despite being an evolutionary algorithm), that includes straightforward operators and easy fitness computation, and is guided by the nets gener-
alization power in its searching task.

3 The EvRBF algorithm

EvRBF is an iterative procedure in which creation and evaluation of new generations of individuals leads to finding RBFNN with good generalization power. It is a steady state algorithm, which includes elitism. Individuals size is variable but population size remains the same during all the process.

3.1 Genetic Operators

EO simplifies the task of programming new operators, since both individuals and operators are objects that can interact by means of their respective interfaces. EvRBF includes operators to specifically deal with RBFNN. Some operators are intended to escape from local minima, while others try to tune previously found solutions.

EvRBF’s operators can be divided in three groups: recombination or crossover, center and radii modification, and hidden layer size modification. Recombination is useful as a mechanism to share information between individuals, and is crucial in cooperative models, like RBFNN or self organizing maps [12]. Modification of centers, radii and number of neurons belong to the so-called mutation operators. They try to introduce a certain variability on the genetic pool, so that new areas of the search space be explored. Following subsections describe these operators grouped on these three categories.

3.1.1 Recombination: X_FIX, X_MULTI and X_AVERAGE

These operators interchange information between individuals, trying to find the building blocks of the solution.

X_FIX reemplaces a sequence of hidden neurons of RBFNN $R_1$ by a sequence of the same size taken from RBFNN $R_2$, as shown in fig. 1.

Operator X_MULTI reemplaces with probability $p_{x\_multi}$ every hidden neuron of RBFNN $R_1$ by a randomly chosen neuron coming from net $R_2$ (see fig. 2). This operator corresponds to uniform crossover in genetic algorithms with binary chromosomes.
Fig. 1. X_FIX application: neurons are interchanged without affecting to the size of the resulting net, $R_1$.

Fig. 2. X_MULTI application: neurons from $R_1$ are replaced by neurons from $R_2$, without changing the resultant net’s size, $R_1$.

Finally, X_AVERAGE selects every neuron from RBFNN $R_1$ with probability $p_{x\text{-average}}$, and makes it closer to a randomly chosen neuron from RBFNN $R_2$ by averaging values with it, as shown in fig. 3 and eqs. 5 y 6.

Fig. 3. X_AVERAGE application: values for center ($c_{1i}$) and radii ($r_{1i}$) of randomly chosen neurons from $R_1$ are set to averaged values using the center and radii of neurons from $R_2$.

$$c_{1i} = \frac{(c_{1i} + c_{2i})}{2}$$  \hspace{1cm} (5)
\[ r_{1i} = \frac{(r_{1i} + r_{2i})}{2} \] (6)

where \( c_{1i} \) is the value of the \( i \) – esime component of the central point of neuron taken from \( R_1 \), \( c_{2i} \) correspond to that component in neuron coming from \( R_2 \); analogously, \( r_{1i} \) and \( r_{2i} \) represent the \( i \) – esime components of the vectors of radii of neurons taken from \( R_1 \) and \( R_2 \), respectively.

### 3.1.2 Centers and radii modification: C\_TUNER, C\_RANDOM, R\_TUNER, R\_RANDOM

These operators use randomness to increase diversity among the individuals of the population; thus, new individuals are generated and local minima can be avoided. The differences between C\_TUNER and C\_RANDOM, and between R\_TUNER and R\_RANDOM are related to the distribution function used to randomly choose the new values for centers and radii, respectively.

Operators C\_TUNER and C\_RANDOM modify RBF centers. The exact number of neurons affected by the operators is determined by the operators internal application probabilities: \( p_{c\_tuner} \) and \( p_{c\_random} \). Subsection 3.3 describes these values.

C\_TUNER perturbs the current value \( c_i \), using a Gaussian probability function, centered on \( c_i \) itself, and as wide as the radius of the neuron, \( r_i \). The new value, \( c'_i \), will probably be close to \( c_i \), allowing the tuning of solutions generated by EvRBF.

C\_RANDOM is intended to explore the input space, since it swaps the current value, \( c_i \), for a random value following an uniform probability function. Every component of the new central point, \( c'_i \), is taken from the range \([\text{min}_i, \text{max}_i]\) that are the minimum and maximum values of the \( i \) – esime input space dimension. Both \( \text{min}_i \) and \( \text{max}_i \) can be calculated from available patterns.

R\_TUNER modifies radii using also a Gaussian probability function, centered on the current value, \( r_i \), and as wide as the \( i \) – esime dimension of input space.

Finally, R\_RANDOM modifies the radii using the input space dimension width, but applying an uniform probability function.

As it happened to C\_TUNER and C\_RANDOM, the number of neurons really affected by R\_TUNER and R\_RANDOM depends on internal application probabilities \( p_{r\_tuner} \) and \( p_{r\_random} \). Subsection 3.3 describes the values.
that must be used to make the algorithm work properly.

3.1.3 Hidden layer size modification: ADDER and DELETER

EvRBF has tries to determine the correct size of hidden layers. In this sense, operator ADDER increments the number of hidden neurons creating new ones, while operator DELETER reduces that number removing neurons.

ADDER adds a single neuron random center and radii, using an uniform probability function in the range \([\text{min}_i, \text{max}_i]\). Ultimately, selection pressure will reward nets containing the new neuron (if it results in fitness increase) or will penalize them (if it produces a negative action).

On the other hand, DELETER removes neurons from the RBFNN to which is applied in a pure random way. The number of neurons to be removed depends on the value of \(p_{\text{del\_many}}\) that determines a neuron’s probability of being removed. Subsection 3.3 shows the value found for this parameter.

3.2 Rest of components

EO library allows reutilization of components to build new algorithms, as in the case of EvRBF. Mechanism to select individuals for reproduction, to select operators to be applied, to extract information of the process, or the iterative part of this evolutionary algorithm have been borrowed directly from this library. Fig. 4 shows the skeleton of EvRBF.

Training, validation and test sets are loaded at first time. They are needed to estimate the input space ranges, as well as the input and output space dimensions. Training set is used to compute the synaptic weights (see section 1). Validation set is used to compute a fitness for every individual, once it has been trained. Training and validation sets do not share samples. Test set is used at the end of training to qualify every individual in the last generation.

According to [30], input values have been rescaled to \([0, 1]\) range. This way, every input dimension has the same weight when computing the distance between the RBF center and the point being evaluated.

Fitness function measures the generalization capability of the individual as the percentage of samples if correctly classifies. When comparing two individuals, the one with the higher generalization power is considered better. If both individuals have exactly the same error rate, the one with less neurons is said to be better. This is used as opposed to a vectorial fitness,
(1) Load training, validation and test sets.

(2) Create initial population. Train, evaluate and assign fitness to every individual.

(3) Instantiate operators, stop condition testers and state monitors.

(4) Instantiate the evolutionary algorithm with the precedent components.

(5) While stop condition is not reached, do the following:
   
   (a) Select individuals from current population and make copies of them.
   (b) Apply operators to these copies; train, evaluate and assign them a fitness.
   (c) Replace worst individuals of current generation by the these copies generated.

(6) Train every individual in last generation using training and validation data sets together.

(7) Use test data set to obtain the generalization power of every individual.

Fig. 4. General skeleton of EvRBF.

since generalization capability is the most important parameter to optimize. However, it could be faced as a multiobjective optimization problem.

The population size used by EvRBF is constant. Every individual is a complete RBFNN, and the data structure used to represent it includes methods to be created, copied, and destroyed, to access and modify its components, and to compute an output value from an input pattern. First population is created using a given process (see subsection 3.3), subsequent generations are created by removing individuals from the population and adding copies of existing individuals to which operators have been applied.

Stop conditions included by EvRBF check the number of generations, error threshold, and individuals convergence. They can be used in combination or independently. In this work, the algorithm has been stopped when a prespecified number of generations has been reached. This allows a better comparison with methods found in literature.

Finally, selection, reproduction and substitution are applied to populations. The selection operator chooses individuals to be reproduced. EvRBF uses fixed length tournament method as selection operator, giving a small opportunity to bad individual to reproduce. Reproduction consist on copying the selected individual and, then, applying operators so that copy be different from its parent. Substitution sorts individuals according to their fitness, removes the worst of them, and adds the new individuals generated by reproduction.
3.3 Execution parameters

EvRBF needs to be given a set of parameters to work properly. A series of systematic experiments have been carried out to establish the best value that should be assigned to every parameter. Parameters considered are the followings: limit for hidden neurons on first population, method to initialize centers, method to initialize radii, percentage of population being replaced, number of individuals involved in tournament selection, and application rate and internal application probability (when applied) for every operator. Experiments were carried out setting the whole set of parameters to default values, and changing one of them in turn making it vary along a wide range of feasible values. The value that achieved the best results (taking into account the final error, the nets size and the time used to get the results) is the value chosen for the parameter.

Once finished the long set of experiments, values selected for parameters are the shown in tables 1 and 2. These values have been used to run the experiments of section 4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td>Generations</td>
<td>{10, 20, 50, 75, 100}</td>
</tr>
<tr>
<td>Limit for neurons on first population</td>
<td>(10%N_{Tr})</td>
</tr>
<tr>
<td>Center initialization</td>
<td>IC_PAT</td>
</tr>
<tr>
<td>Radii initialization</td>
<td>IR_MIN_DIS2</td>
</tr>
<tr>
<td>Replaced population</td>
<td>30%</td>
</tr>
<tr>
<td>Tournament length</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1
Execution parameters for EvRBF. See also table 2.

Table 1 shows that nets in the first generation can have a random number of hidden neuron, going from 1 to 10\% of training set size. After this generation, no limit to the number of neurons is imposed. Furthermore, centers are initialized according to the IC\_PAT method, which uses randomly selected patterns from the training set as RBF centers. This method turned to be better than IC\_RND (center values were set randomly) and IC\_KM (K-means was used to select a representative set of points). On the other hand, radii are initialized using the IR\_MIN\_DIS2 method. This method computes the distance from the center of the neuron being initialized to the rest of neurons in the net, and sets the radii as 1.75 times the minimum distance found. Other methods considered to set the radii were IR\_RND\_CTE (a single randomly chosen value set as the radii for every net), IR\_RND\_VAR (many
randomly chosen values, one per radius), and \textit{IR\_MIN\_DIS} (performing as \textit{IR\_MIN\_DIS2} but using 1 times the minimum distance found instead of 1.75 times). Finally, in every generation, 30 individuals are replaced by 30 new individuals created by reproduction.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Application rate</th>
<th>Internal application probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_FIX</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>X_MULTI</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>X_AVERAGE</td>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>C_TUNER</td>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>C_RANDOM</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>R_TUNER</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>R_RANDOM</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>ADDER</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>DELETER</td>
<td>2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 2: Application rates and internal application probability for operators.

Table 2 shows \textit{application rate} and \textit{internal application probability} for every operator. Application rate of one operator divided by the sum of all application rates gives its application probability. Internal application probability is used to decide which neurons from that RBFNN will be affected by the action of the operator (they correspond to \(p\_x\_\text{multi}\), \(p\_x\_\text{average}\), \(p\_c\_\text{tuner}\), \(p\_c\_\text{random}\), \(p\_r\_\text{tuner}\), \(p\_r\_\text{random}\) and \(p\_\text{deleter}\), cited along subsection 3.1.)

4 Experiments and results

Evaluation of EvRBF on classification tasks has been developed using 3 well known real databases: the Iris plants, heart disease and cancer [31].

Every problem has been run independently for different numbers of generations: 10, 25, 50, 75 and 100. For any of these values, the algorithm has been run 5 times, starting with different initial populations and different training and validation sets. Results related to EvRBF, shown in following subsections, are averaged values along the 5 runs per number of generations, taking into account only the best individual created in every run.
4.1 Iris plants

Iris database [31] patterns are composed of 4 input numeric values (related to sepal width and length and petal width and length), and 1 output corresponding to the kind of plant (3 different plants being considered). Every class is represented by about 50 patterns. There are no null values and patterns are correctly classified. The data base has 156 patterns divided in 50% for test, 37.5% for training and 12.5% for validation.

EvRBF is compared with Whitehead and Choate’s method [26], and Burdassall and Giraud’s methods [27], since both of them use evolutionary algorithm to create RBFNN that solve this problem. Furthermore, [27] provides results yielded by an hand-optimized RBFNN (referred by RBF in table 3), by a nearest-attracting prototype (NAP) classifier (based on k-means and developed by Burdsall and Giraud themselves), and by a multilayer perceptron (MLP).

Results obtained by EvRBF for this problem are independent of the number of generations, except in runtime. Thus, the percentage of badly classified patterns is 1% ± 1, and hidden layer size is 6 ± 1 In fact, most executions found individuals that could classify 100% of the patterns in training and validation sets, so that individuals were chosen according to their size, once fitness had converged. The easiness in learning training patterns does not lead to overfitting in this problem, because training and validation sets are good enough for representing the whole database. Thus, nets showing a high generalization power over validation set, are able to correctly classify test set patterns.

Compared to other method, table 3 shows that EvRBF classifies better (1% error compared to Whitehead and Choate’s 3%) using a small number of neurons (between 5 and 7). Second best method is Whitehead and Choate’s one (an evolutionary algorithm too); Burdsall and Giraud’s method does not improve results obtained by the hand-optimized RBFNN; and, finally, MLP and NAP provide the worst results. No information about minima and maxima obtained by EvRBF is going to be shown along this work; nevertheless, nets that correctly classified the whole test set have been found for this particular problem.

4.2 Heart Disease Database

Next problem concerns diagnosis of heart disease, and has been taken from [31]. This database is composed of 270 samples, each one with 13 input variables (7 of them taking numerical values, 3 binary values and 3 nom-
Algorithm | Error % | RBFNN Size
--- | --- | ---
Whitehead-Choate [26] | 3 | 15
Burdsall-Giraud [27] | 4 | 6-9
RBF [27] | 4 | -
MLP [27] | 4 | -
NAP [27] | 4 | -
EvRBF | 10 gen. | 1 ± 1 | 6 ± 1
EvRBF | 25 gen. | 1 ± 1 | 6 ± 1
EvRBF | 50 gen. | 1 ± 1 | 6 ± 1
EvRBF | 75 gen. | 1 ± 1 | 6 ± 1
EvRBF | 100 gen. | 1 ± 1 | 6 ± 1

Table 3
Comparison of methods on Iris database classification

<table>
<thead>
<tr>
<th>Generations</th>
<th>Error %</th>
<th>RBFNN size</th>
<th>Parameters %</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11 ± 2</td>
<td>12 ± 4</td>
<td>12.15%</td>
<td>33 ± 4</td>
</tr>
<tr>
<td>25</td>
<td>13 ± 4</td>
<td>12 ± 3</td>
<td>12.15%</td>
<td>78 ± 17</td>
</tr>
<tr>
<td>50</td>
<td>14 ± 2</td>
<td>13 ± 8</td>
<td>13.16%</td>
<td>169 ± 55</td>
</tr>
<tr>
<td>75</td>
<td>16 ± 3</td>
<td>14 ± 5</td>
<td>14.17%</td>
<td>272 ± 76</td>
</tr>
<tr>
<td>100</td>
<td>11 ± 3</td>
<td>18 ± 5</td>
<td>17.21%</td>
<td>383 ± 115</td>
</tr>
</tbody>
</table>

Table 4
EvRBF’s results on Heart Disease database classification

This database allows comparison of EvRBF with Leonardis and Bischof’s method [21], intended to automatically generate RBFNN. This pruning method can modify centers, radii and weight by means of gradient descent, and uses MDL to remove neurons from an initial over-specified net.

Table 4 shows results obtained by EvRBF for this problem. Columns show the percentage of patterns badly classified, nets’ sizes, percentage of free parameters (i.e., number of values composing the RBFNN with respect to the number of values composing the test, training and validation sets), and execution time in seconds. As can be seen, the classification error and network size increase with the number of generations. This trend stops for
Algorithm Error % RBFNN size

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Error %</th>
<th>RBFNN size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leonardis-Bishop[21]</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>Gaussian Masking[21]</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>EvRBF 10 gen.</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>EvRBF 25 gen.</td>
<td>13</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5: Comparison of methods on Heart Disease database classification

100 generations, where the sizes are bigger. Thus, for this problem, the algorithm produces overfitted nets that iteratively improve their results over the training and validation sets, while reducing their generalization capabilities. Even more, when 50, 75 or 100 generations are used, nets become bigger than necessary, being the number of free parameters unacceptably large.

For this problem, EvRBF takes about 5 times more than for Iris database to reach the final generation. This is due to the higher dimension of the input space, what leads to a bigger number of neurons affected by operators, as well as a larger cost on computing the synaptic weights.

Table 5 compares results yielded by EvRBF, Leonardis-Bischof’s method [21], and Gaussian Masking [32] (GM; it uses clustering and linear programming to build neural nets). Results related to GM have been taken from [21]. No results for EvRBF when using 50, 75 and 100 generations are shown, since nets produced were invalid models (bigger than necessary).

Results show that EvRBF slightly improves the error provided by the rest of algorithms, using smaller nets at the same time. Furthermore, even when nets become overfitted (see rows belonging to 50, 75 and 100 generations on table 4), the generalization power of the nets is almost equal to Leonardis and Bischof’s nets, and better to GM’s ones.

### 4.3 Cancer-1a database

Cancer-1a database, initially proposed by Prechelt [33], is related to breast cancer diagnosis. The problem consists of determining whether a given cancer is malignant or benign, according to the analysis of cells via electronic microscopy. Every pattern is composed of 9 inputs and 1 output (0 -benign, 1 -malign). 65.5% of patterns belong to benign tumors. The training set contains 524 patterns, while the test set contains 124. This is the problem initially considered by Grönroos [34]. As in the previous case, it has been borrowed from UCI’s automatic learning repository [31].
Table 6
EvRBF’s results on Cancer-1a database classification

<table>
<thead>
<tr>
<th>Generations</th>
<th>Error %</th>
<th>RBFNN size</th>
<th>Parameters %</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.4 ± 0.5</td>
<td>27 ± 8</td>
<td>10.18%</td>
<td>65 ± 20</td>
</tr>
<tr>
<td>25</td>
<td>1.5 ± 0.3</td>
<td>31 ± 13</td>
<td>11.69%</td>
<td>227 ± 71</td>
</tr>
<tr>
<td>50</td>
<td>1.6 ± 0.5</td>
<td>23 ± 10</td>
<td>8.67%</td>
<td>255 ± 77</td>
</tr>
<tr>
<td>75</td>
<td>1.7 ± 0.9</td>
<td>18 ± 9</td>
<td>6.78%</td>
<td>1142 ± 605</td>
</tr>
<tr>
<td>100</td>
<td>1.3 ± 1.3</td>
<td>21 ± 21</td>
<td>7.92%</td>
<td>1527 ± 1238</td>
</tr>
</tbody>
</table>

Table 7
Comparison of methods on Cancer-1a database classification

Results from EvRBF are compared to those yielded by G-Prop [35] and others (Prechelt, Grönnroos, Quick-Propagation and SA-Prop) cited in the same paper. G-Prop is an evolutionary algorithm similar to EvRBF, but designed to evolve multilayer perceptrons.

EvRBF yields the results shown on table 6. As can be seen, the percentage of error is very similar independently the number of generations. In fact, the Student test gives no significative differences between those values. Sizes are affected by a higher level of variation, although differences are not significant. The number of parameters sizes is acceptable in all the cases, since is close or less than 10%. Nevertheles, runtime is quite different in this experiment, since 75 and 100 generations take more than 1000 seconds to finish, what is unusual. This has been be due to the presence of very large individuals along the evolution (although those individuals were not the fittest ones), resulting in high time consuming when computing their weights.
In table 7 can be seen the different percentages of error yielded by the methods of Prechelt and Grönroos, as well as the multilayer perceptrons trained with Quick-Propagation, or evolved by SA-Prop and G-Prop. Except those related to EvRBF, results have been borrowed from [35].

In general, EvRBF’s behaves is as good as the rest of methods being considered. Worst result yielded by EvRBF (1.7%) is slightly worst than Prechelt (1.4%), but better than Grönroos (2.0%) and Quick-Propagation (3.0%), which is the worst method.

EvRBF yields the best results for 100 generations. This result is better than all the previously cited, but worse than Castillos’s SA-Prop (1.1%) and G-Prop (0.9%). The latter is also the best according to net sizes: it can solve the problem with only 14 neurons, instead of between 18 and 31 used by EvRBF or 38, used by Quick-Propagation.

4.4 General remarks

Setting the individuals fitness using only their classification skills leads sometimes to unacceptable solutions, i.e., RBFNN with high generalization power but also with an excessive number of neurons. Penalization of big RBFNN and early stop of the EA when RBFNN size is around the maximum allowed for a given number of consecutive generations could be used to control RBFNN sizes. Moreover, using a single value for radii would drastically decrease the number of free parameters, but would make EvRBF useful only for that kind of nets.

Keeping EvRBF as general as possible can lead to overspecified nets in some problems, but makes it suitable to be directly used in any kind of classification problem. Thus, no decisions have to be taken a priori concerning to the structure of the net, or to the zone of search space where solutions are supposed to be. EvRBF provides good solutions that can also be used as entries to local search algorithms in order to tune them.

5 Conclusion

This paper shows EvRBF, an algorithm that evolves RBFNN, applied to the task of solving classification problems.

EvRBF automatically determines both topology and configuration of RBFNN. Thus, EvRBF finds the size of the net (number of hidden neurons)
and the parameters that configure each neuron: center and radii of its activation function. EvRBF does not start from pre-established topologies, does not set number of neurons a priori, does not fix an upper limit to the number of neurons, and uses only information about generalization capability and size of nets to evaluate and evolve them.

EvRBF includes operators specifically designed to deal with RBFNN. Operators can do cross-over, addition, removing and modification of hidden neurons, as well as their components. The parameters needed by the algorithm to apply the operators have been empirically set to fixed values, and are used in every problem. Thus, no explicit pre-processing is needed to apply the algorithm to new problems.

In this paper, EvRBF algorithm has been applied to different real examples. Results have been compared to others in literature showing that can be used to efficiently configure RBFNN, since it obtains nets with high generalization power and size smaller or similar to other methods.

The weakest point of the algorithm may be considered the number of parameters that configure the final nets. This number is very often greater than desired (referred to the number of values composing the training and test pattern sets). This is mainly due to the number of radii that EvRBF has to evolve: $2n$ per neuron, where $n$ is the input dimension. Reducing this number to $n$ radii per neuron (i.e., using symmetric RBF), or a single radius per neuron, or even a single radius per net would drastically reduce the number of parameters while probably maintaining a similar or slightly worst capability of generalization. Although this could be easily introduced into the algorithm, the aim of this work was to fully configure the nets, and deal with the widest range of RBF; thus, genetic operators are intended to look for the optimal set of values for all the parameters.

Future lines of research will focus on a deeper analysis of the parameters to run the algorithm (by means of ANOVA method), the use of cross-validation to set the fitness of every new individual, as well as facing this problem as a multiobjective optimization task. Finally, we are currently obtaining the data necessary to apply this method to labor risk prevention.

Acknowledgment

This work includes some of the comments done by the members of the tribunal of the doctoral thesis [36]: C. Cotta, F. Herrera, H. Pomares, J.M. Troya and L.A. Ureña.
This work has been partially supported by projects TIC2002-04036-C05-04 and TIC2003-09481-C04.

References


URL citeeseer.nj.nec.com/cohen00global.html


URL citeeseer.nj.nec.com/burdsall1997garbf.html


