Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information Sciences 165 (2004) 207–220

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Evolving RBF neural networks for time-series forecasting with EvRBF

V.M. Rivas [a,*], J.J. Merelo [b], P.A. Castillo [b], M.G. Arenas [b], J.G. Castellano [b]

[a] *Departamento de Informática, Universidad de Jaén, Escuela Politecnia Superior, Avda. de Madrid 35, E-23071 Jaén, Spain*

[b] *Departamento de Arquitectura y Tecnología de Computadores, Universidad de Granada GeNeura Team, E-18071 Granada, Spain*

Received 30 December 2001; received in revised form 30 December 2002; accepted 4 September 2003

Abstract

This paper is focused on determining the parameters of radial basis function neural networks (number of neurons, and their respective centers and radii) automatically. While this task is often done by hand, or based in hillclimbing methods which are highly dependent on initial values, in this work, evolutionary algorithms are used to automatically build a radial basis function neural networks (RBF NN) that solves a specified problem, in this case related to currency exchange rates forecasting. The evolutionary algorithm EvRBF has been implemented using the evolutionary computation framework evolving object, which allows direct evolution of problem solutions. Thus no internal representation is needed, and specific solution domain knowledge can be used to construct specific evolutionary operators, as well as cost or fitness functions. Results obtained are compared with existent bibliography, showing an improvement over the published methods.
© 2003 Elsevier Inc. All rights reserved.

Keywords: RBF; Evolutionary algorithms; EO; Functional estimation; Time-series forecasting; Currency exchange

* Corresponding author. Tel.: +34-953-01-23-44; fax: +34-953-00-24-20.
E-mail addresses: vrivas@ujaen.es (V.M. Rivas), todos@geneura.ugr.es (J.J. Merelo).
URLs: http://www.di.ujaen.es/~vrivas, http://geneura.ugr.es.

0020-0255/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2003.09.025

1. Introduction

A radial basis function (RBF), ϕ, can be characterized by a point of the input space, c, and a radius or width, r, such that the RBF reaches its optimum value (maximun/minimun) when applied to c, and decreases/increases to its opposite optimum value when applied to points far from c. The radius, r, controls how distance affects that increment or decrement. For this reason, mathematicians have used groups of RBF to successfully interpolate data. Typical examples of RBF are the Gaussian function (see Fig. 1a) and the multiquadric function (see Fig. 1b), although there are many others [1,2].

RBF are used by radial basis function neural networks (RBF NNs), which were introduced by Broomhead and Lowe in [3], being their main applications function approximation and time-series forecasting, as well as classification or clustering tasks. Traditionally, a RBF NN is thought as a two-layer, feed-forward network in which hidden neuron activation functions are RBF (see Fig. 2). Very often, the function used is the Gaussian one.

In RBF NNs each hidden neuron computes the distance from its input to the neuron's central point, c, and applies the RBF to that distance, as shows Eq. (1).

$$h_i(x) = \phi\left(\|x - c_i\|^2 / r_i^2\right) \tag{1}$$

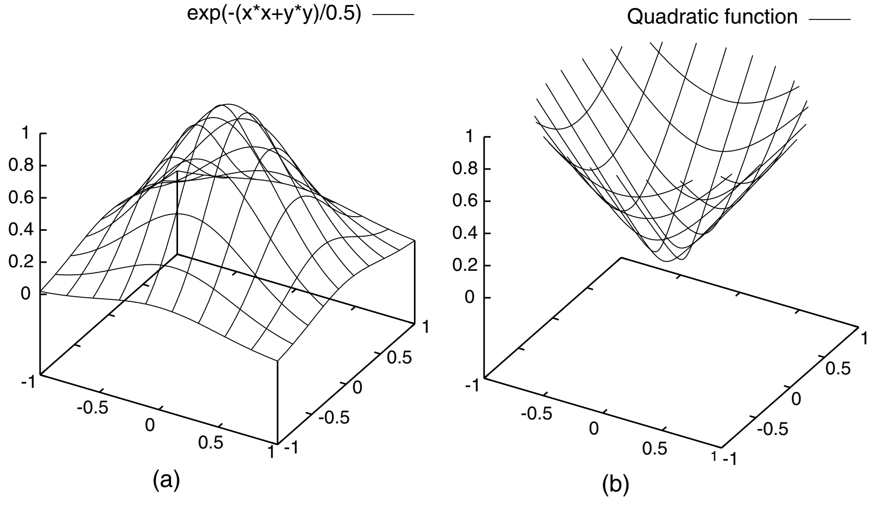

Fig. 1. Examples of radial basis functions: (a) Gaussian function, (b) multiquadric function.

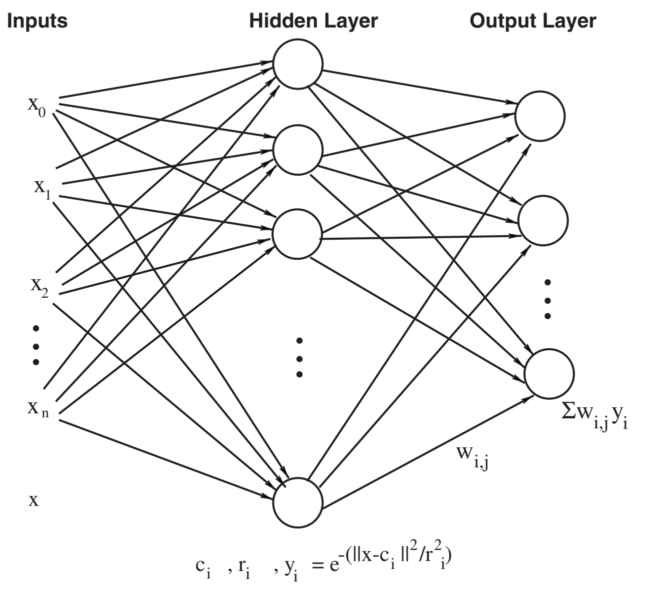

Fig. 2. RBF neural network.

where $h_i(x)$ is the output yielded by hidden neuron number i when input x is applied; ϕ is the RBF, c_i is the center of the ith hidden neuron, and r_i is its radius.

In the next step, the neurons of the output layer perform a weighted sum using the outputs of the hidden layer and the weights of the links that connect both output and hidden layer neurons (Eq. (2)).

$$o_j(x) = \sum_{i=0}^{n-1} w_{ij} h_i(x) + w_{0j} \tag{2}$$

where $o_j(x)$ is the value yielded by output neuron number j when input x is applied; w_{ij} is the weight of the links that connects hidden neuron number i and output neuron number j, w_{0j} is a bias for the output neuron, and finally, n is the number of hidden neurons.

It has been proved [4] that when enough units are provided, a RBF NN can approximate any multivariate continuous function as much as desired. This is possible because once the centers and the radii have been fixed, the weights of the links between the hidden and the outputs layers can be calculated analytically using singular value decomposition [5] or any algorithm suitable to solve lineal algebraic equations, making unnecessary the use of training algorithms, as those used in other kinds of neural networks such as multilayer perceptrons.

Thus, the main problem in RBF NNs design concerns establishing the number of hidden neurons to use and their centers and radii.

The need of automatic mechanisms to build RBF NNs is already present in Broomhead and Lowe's work [3], where they showed that one of the parameters that critically affects the performance of RBF NNs is the number of hidden neurons. When this number is not sufficient, the approximation offered by the net is not good enough; in the other hand, nets with many hidden neurons will approximate very well those points used to calculate the connection weights, while having a very poor predictive power; this is the so-called overfitting problem [6]. In consequence, establishing the number of neurons (that is, the number of centers and values related to them) that solves a given problem is one of the most important tasks researchers have faced in this field.

In this paper an evolutionary algorithm is used to find the best components of the RBF NNs that approximate a function representing a time-series. Results are compared with other techniques that also use the evolutionary approach to tune the RBF NN.

The rest of the paper is organized as follows: Section 2 describes some of the methods used to solve the cited problems; Section 3 shows our proposed solution, and describes the evolutionary computation framework we use (EO). Next section (4) shows some functional approximation experiments; and finally, our conclusions and future working lines are exposed in Section 5.

2. State of the art

There are several papers that address the problem of automatic RBF NN design. The paper by Leonardis and Bischof [7] offers a good overview. Methods found in bibliography can be divided in five classes.

(1) Methods in which *the number of radial basis functions must be given a priori*; after this, computing the values for the centers and the radii is done choosing points randomly from the training set or performing any kind of clustering method with them [8].

(2) Methods that automatically *search for the number of RBF and their centers and radii*. Most of these methods grow the final net from an initially empty one to which hidden neurons are added until a given condition is reached. One of such methods is orthogonal least squares [9], based on a mechanism called *forward selection*, that allows the addition of hidden neurons to an initially empty net until the approximation error is under a prespecified tolerance value or threshold. An improvement over this algorithm is regularised OLS [10], based in *regularised forward selection*, in which high hidden-to-output weight values are penalized using a penalty term, a process termed regularization [6]. Later, in [11], Orr newly used forward selection as the growing mechanism, and delete-1 (or leave-one-out) and generalized cross-

validation as the methods to stop adding neurons; in any of its forms, cross-validation is used to compute the generalization error of many different nets in competition one with each other, halting the algorithm when that error reaches a minimum. Other methods of the same characteristics are resource allocation networks [12], and growing cell structures [13], which try to estimate the centers using hillclimbing techniques, making them liable to fall in local optima.

(3) There are also *pruning methods* that start with an overspecified RBF NN and, iteratively, remove hidden neurons and/or weights. Leonardis and Bischof's algorithm [7] can be classified as a pruning method, in which the minimum description length measure (MDL) is used to achieve a reduction in the complexity of the RBF NN. According to the MDL principle, a set of RBFs describing the training data with the shortest possible encoding are selected among a larger set. The main drawback of pruning methods is that they tend to be very restrictive, and, thus, find suboptimal networks.

(4) *The values for the radii are also important parameters*. Very narrow radii will lead to overfitting, and radii wider than necessary can give even worst results. Thus, in order to obtain a good performance, and taking into account that each component of the points in the input space can have ranges with very different limits, many radii with many different values must be fixed. There are some algorithms intended to estimate the radii; Orr in [14] chooses a finite set of radii, and after this, the best of those radii is found (as efficiently as possible) by means of (1) constructing the RBFs, (2) using them, and (3) regarding the performance of the nets. It is obviously necessary to start with a good set of radii in order to succeed. Furthermore, it could happen that a good value for the radii were not considered because a value close to it, and present in the initial set, had produced bad results.

(5) The use of *evolutionary algorithms* to construct neural nets is also well known in the literature. Some reviews can be found in [15,16]. They usually try to optimize only one of the parameters of the net (number of neurons, topology, learning rate, and so on), leaving the other parameters fixed. A different case can be found in the work by Castillo et al. [17,18], in which evolutionary algorithms are used to create good multilayer perceptrons (MLPs), optimizing all the parameters at the same time, and obtaining very good results. A similar and previous example can be found in Merelo and Prieto [19], where a neural network called learning vector quantization (LVQ) is genetically optimized. Nevertheless, both Castillo and Merelo's methods can not be applied directly to the construction of RBF NNs since their genetic operators are geared toward their specific network architecture. An application of evolutionary algorithms to RBF NN evolution can be found in the work by Gonzlez et al. [20], where expert mutation operators are created to evolve RBF NNs.

At this point, we would like to highlight the work by Sheta and de Jong [21] given that we are going to compare our results with theirs. They proposed an autoregressive model for RBF (AR-RBF) that was tuned using GA, that is, using cross-over and mutation operators to set the weights, instead of least square error (LSE), the most commonly used method.

The AR-RBF model is described by Eq. (3):

$$o(k) = k\sum_{i=1}^{n} w_i h(o(k - t_i)) \tag{3}$$

where $o(k)$ is the exit of the NN, k is a selected constant gain, $h()$ is the RBF, w_i the weights linking hidden neurons to the output one, and n is the order of the autoregressive model as well as the number of hidden neurons.

The GA used by Sheta and de Jong coded the centers, radii and weights of the RBF NN as genes, using a string representation; thus, every individual was a vector of the form:

$$c_1 r_1 w_1 c_2 r_2 w_2 \ldots c_n r_n w_n \tag{4}$$

The string representation used for the genes leads to the conclusion that generic cross-over and mutation operators were used. It seems that more sophisticated operators (like those intended to estimate the number of hidden neurons) are not used by the authors. Despite this fact, the authors got very good results, and, they did it using only a few individuals, a few generations, and, consequently, only a few evaluations.

3. EvRBF

The method introduced in this paper uses an evolutionary algorithm, EvRBF, to build RBF NNs, optimizing their generalization error by finding the number of neurons in the hidden layer, and their centers and radii.

The evolutionary algorithm itself has been programmed using the new evolutionary computation framework, evolving objects (EO), current version is 0.9.2 [22,23]. It can be found at `http://eodev.sourceforge.net` and is available under open source license.

This new framework is the result of the cooperative work carried out by several research teams in Europe. EOs main advantage is that what can be evolved is not necessarily a sequence of genes (bitstring or floating point values), but anything to which a fitness or cost function can be assigned. In this sense, every data structure one can imagine can be converted into something evolvable, and consequently, something that EO can evolve, optimizing it.

EO allows to define operators that increase or decrease the diversity in the population, that is, cross-over-like operators and mutation-like operators. Some of these operators are generic and can be applied to many classes of evolvable objects, but new operators can also be specifically defined for each object; taking into account that, using EO, evolutionary algorithms are able to deal with the solution of the problem being optimized itself, instead of a representation of that solution, problem domain knowledge can be inserted in the operators, obtaining a better behaviour. This is also in accordance with Michalewicz's ideas published in [24].

In the present research, a standard genetic algorithm has been used with specific operators. Fixed size population, tournament selection, and elitist replacement policy have been used, and two kind of operators, mutation-like and cross-over-like, have been created. The algorithm finishes when a previously specified number of generations has been reached.

3.1. Binary cross-over operator

The binary-operator needs two RBF NNs to be applied, although it only changes one of them. This operator takes an uniformly random chosen number of consecutive hidden neurons from the first network, and another random sequence from the second; then it replaces the first of these sequences by the second one, so that the second individual remains unchanged.

3.2. Unary operators

Given that in a RBF NN the optimum weights from hidden-to-output neurons can be easily computed, the unary or mutation-like operators affect the hidden neuron components, centers and radii, in their quantities and values, but not the hidden-to-output weights.

Centers mutator. This operator changes a given percentage of the components of the center of every hidden neuron. Each component of the center is modified by adding or subtracting a random value, which is chosen following a Gaussian probability function with mean 0 and standard deviation 0.1.

Radii mutator. This operator is applied to one of the existing nets, but affects to the radii of the RBF. Radii are modified using a Gaussian function as defined previously. Radii are different for each center, and for each component of the center.

Hidden neuron deleter. This operator erases a given percentage of hidden neurons in the net it is being applied. Neurons to remove are randomly chosen.

Hidden neuron adder. As its name indicates, this operator adds random neurons to the net. Althought it has been designed is currently not used given that the binary-operator is good enough to increase the number of neurons in an efficient way.

The breeder

Every new generation is created in the following way:

(1) Select a few individuals (the elite) using tournament selection of fixed size.
(2) Delete the rest of the population.
(3) Generate new individuals by copying individuals from the elite.
(4) Apply operators to the new individuals.
(5) Set the weights of the new individuals using SVD.
(6) Remove the non-sense neurons, i.e., those whose weights to output neurons are very close to 0.
(7) Evaluate the net, and set its fitness.

Once tournament selection, performed with a parametric number of individuals, has finished and the elite has been formed, individuals are copied as many times as needed until the population reaches its fixed size. Later, crossover-like and mutation-like operators are applied only to new individuals. Operators are applied with a given fixed probability (not necessarily the same for all of them) to each neuron of each RBF NN, so that if one operator is used with a probability of 0.01, it will modify 1% of the RBF on average.

A RBFF NN fitness is calculated using leave-one-out cross-validation method. The training data set composed of k pairs input–output, is split k times in 2 disjoint subsets: the first one is composed of $k-1$ data points, and the second one is composed of the remaining point. The net is trained and validated k times using the subset having $k-1$ data to train, and the other point to validate. The fitness is calculated as the average of the inverse of the root mean square error (Eq. (5)) over the k different subsets having only 1 point.

$$\text{fitness} = \frac{\sum_{j=1}^{k}\left(\sqrt{\frac{\sum_{j=1}^{n}(y_{ik}-o(x_{ik}))^2}{n}}\right)^{-1}}{k} \tag{5}$$

where y_{ik} is the expected output, $o(x_{ik})$ is the output calculated by the net when input x_{ik} is applied, and n is the number of input–output pairs in the validation set.

4. Experiments and results

Previous experiments in function approximation using this new approach can be found in [25].

For this new work, the time-series being forecasted is the one used by Sheta and de Jong in [21]. This time-series is composed of real data representing the exchange rates between British pound and US dollar during the period going from 31 December 1979 to 26 December 1983, available from http://pacific.commerce.ubc.ca/xr/data.html, thanks to the work done by Prof. Werner Antweiler, from the University of British Columbia, Vancouver, Canada.

Data are composed of 208 observations. The herein proposed evolutionary algorithm uses as training data set only half the data (randomly chosen) using the leave-one-out method described before. Once the evolutionary algorithm has finished, and before obtaining the generalization error, all individuals in the last population are newly trained with this training set. This is the final training error showed in Table 2 on the row belonging to training data set, and has been included for comparison. Test (or generalization) error is calculated using the full data set, also to compare with Sheta and de Jong's results.

The experiment has been repeated 10 times using the parameters showed in Table 1. Nevertheless, individuals in the initial population were set randomly to avoid a biased search.

Figs. 3 and 4 graphically show the estimation provided by the best net found in one of the ten experiments once it had been optimized via the evolutionary algorithm. Fig. 3 shows the original data set (dashed line) and the forecasting done by the net (solid line). On the other hand, Fig. 4 shows the differences between real and forecasted values for the same net, as well as the average value of these differences.

Table 2 shows the error rates achieved by the EvRBF algorithm and compares with results found in literature. Given that experiments have been run 10 times, averaged values and standard deviations are showed.

There is no information about RBF NN size to compare with. The best net found by EvRBF during the evolutionary process had only 8 neurons, and its MSE over the training set was 1.6018×10^{-4}. The best result found in the generalization task was yielded by a net with only 7 neurons; its MSE was 5.3090×10^{-4} over the full data set and 4.0899×10^{-4} over the training set. On

Table 1
Parameters used to run the evolutionary algorithm

Parameter	Value
Population	15
Number of generations	50
Tournament size	3
Individuals used to generate offspring	20% of population
Cross-over rate	0.2
Center modifier mutator rate	0.2
Radii modifier mutator rate	0.2
Neuron remover mutator rate	0.4

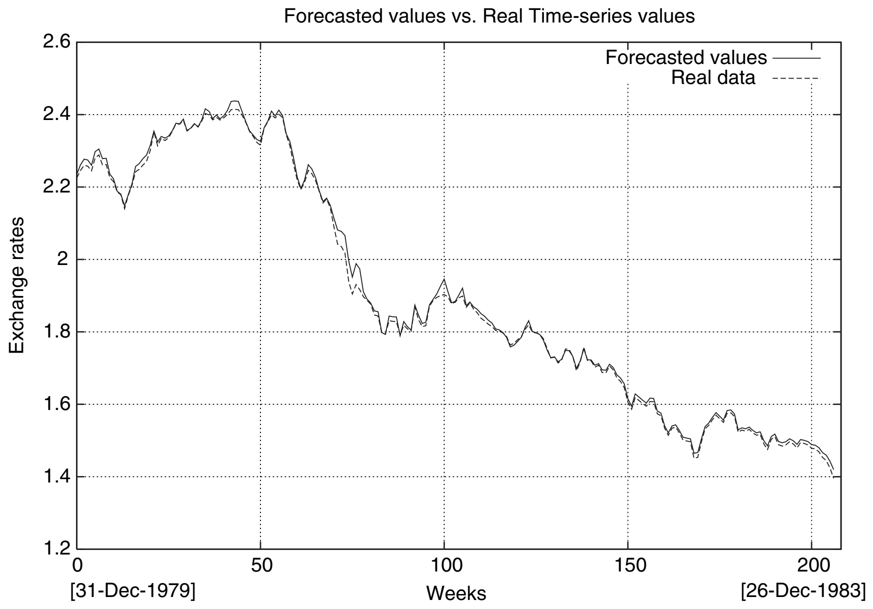

Fig. 3. Real values used in the experiments (dashed line) and values provided by the best net found in the experiments (solid line).

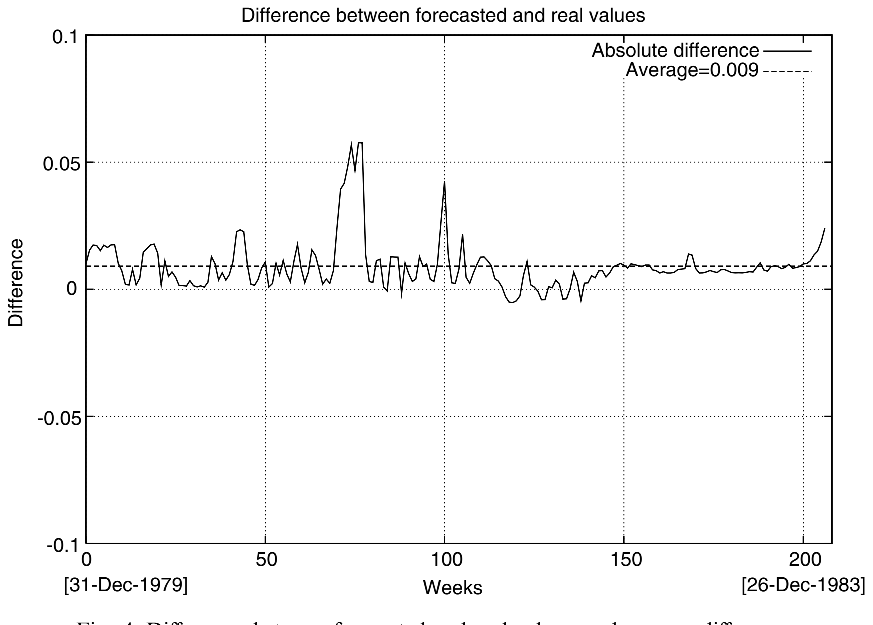

Fig. 4. Differences between forecasted and real values, and average difference.

Table 2
Comparison between results obtained by Sheta and de Jong (MSE-GAs), a previously published method (MSE-LSE), and results obtained by EvRBF (both MSE-GAs and MSE-LSE results borrowed from [21]

Data set	MSE-LSE	MSE-GA	EvRBF	
			Average	Best individual
Training	7.2601×10^{-4}	5.1407×10^{-4}	$3\times10^{-4}\pm1\times10^{-4}$	2×10^{-4}
Full data set	12.001×10^{-4}	8.7220×10^{-4}	$6\times10^{-4}\pm2\times10^{-4}$	4×10^{-4}

Values show the MSE computed over the training and full data sets, thus lower values are better. As can be seen, on average error over the full data set is half the one obtained by previous works.

the other hand, RBF NN size changed in a broad range, having as average 22 and standard deviation 10.

In terms of the number of parameters, a net with 7 hidden neurons (as it is the best net found), using only 1 input and producing 1 output has 7 values for the centers, 7 for the radii and 7 for the weights. This makes 21 parameters, what is a relatively small number of parameters to learn 208 data points.

Finally, the ascendant progression of the items being logged (maximum, minimum and average fitness) shown in Fig. 5 is an indicator of the good search strategy followed by the operators used in the GA. In this sense, the evolution of the average fitness shows that new individuals are, in average,

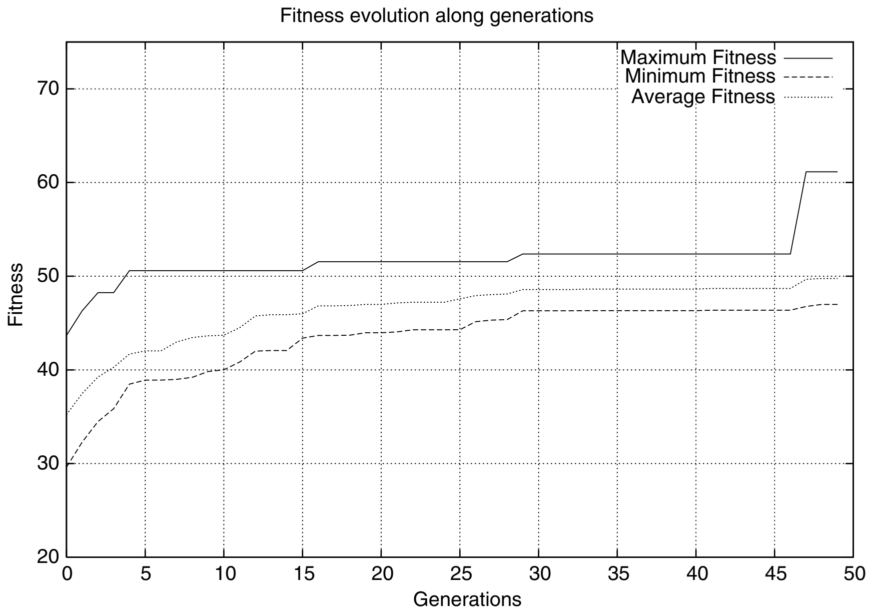

Fig. 5. Maximum, minimum, and average fitness evolution during the genetic algorithm execution.

better than those of the precedent generations, although this does not mean that finding the best individual of all be an easy task.

5. Conclusions

Creating the best RBF NN that solves a given problem is a difficult task because many parameters have to be set at the same time: number of hidden neurons, centers and radii for them. Evolutionary algorithms can help in finding the optimal values for those parameters, obtaining RBF NNs with low generalization error.

In order to use specific problem knowledge, the RBF NNs are evolved as such, without the typical use of a representation (binary or floating point vector) and a decoder. This is possible thanks to the evolutionary computation framework, EO, implemented as a C++ class library, since it allows to evolve any object to which a fitness or cost function can be assigned.

The results obtained by our algorithm, EvRBF, show that good RBF NNs are found when it is applied to time-series forecasting.

Future work includes the following research lines:

- *Creation and testing of new operators*, especially binary or cross-over-like operators, performing merges where average values are calculated between the neurons being affected, and also operators that apply a certain local-tuning process to improve the evolutionary algorithm search.
- *Application of statistical tests* to evaluate the impact produced by the parameters used to run the evolutionary algorithm.
- *Testing EvRBF with other problems* to which RBF NN have been applied: more time-series prediction tasks, classification and function approximation.

Acknowledgements

This work has been supported in part by TIC2002-04036-C05-04 and TIC2003-09481-C04.

References

[1] B.A. Whitehead, T.D. Choate, Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction, IEEE Transactions on Neural Networks 7 (4) (1996) 869–880, July.

[2] F. Schwenker, H.A. Kestler, G. Palmm, Three learning phases for radial-basis-function networks, Neural Networks 14 (2001) 439–458.

[3] D.S. Broomhead, D. Lowe, Multivariable functional interpolation and adaptative networks, Complex Systems 11 (1988) 321–355.

[4] W.A. Light, Some aspects of radial basis function approximation, Approximation Theory, Spline Functions and Applications 356 (1992) 163–190.

[5] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, second ed., Cambridge University Press, 1992.

[6] C. Bishop, Training with noise is equivalent to Tikhonov regularization, Neural Computation 7 (1) (1995) 108–116.

[7] A. Leonardis, H. Bischof, And efficient MDL-based construction of RBF networks, Neural Networks 11 (1998) 963–973.

[8] J.E. Moody, C. Darken, Fast learning in networks of locally tuned processing units, Neural Computation 2 (1) (1989) 281–294.

[9] S. Chen, C.F. Cowan, P.M. Grant, Orthogonal least squares algorithm for learning radial basis function networks, IEEE Transactions on Neural Networks 2 (2) (1991) 302–309.

[10] S. Chen, E.S. Chng, K. Alkadhimi, Regularized orthogonal least squares algorithm for constructing radial basis function networks, International Journal of Control 64 (5) (1996) 829–837.

[11] M.J.L. Orr, Regularisation in the selection of radial basis function centres, Neural Computation 7 (3) (1995) 606–623.

[12] J. Platt, A resource-allocating network for function interpolation, Neural Computation 3 (2) (1991) 213–225.

[13] B. Fritzke, Supervised learning with growing cell structures, in: J.D. Cowan, G. Tesauro, J. Aspector (Eds.), Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann, 1994, pp. 255–262.

[14] M.J.L. Orr, Optimising the Widths of Radial Basis Functions, V Brazilian Congress on Neural Networks, 1998.

[15] A.V. Adamopoulos, E.F. Georgopoulos, S.D. Likothanassis, P.A. Anninos, Forecasting the magnetoencephalogram (MEG) of epilectic patients using genetically optimized neural networks, in: Proceedings of the genetic and Evolutionary Computation Conference, GECCO'99, vol. 2, Morgan-Kaufmann Publ, 1999, pp. 1457–1462, July 1999.

[16] D. White, P. Ligomenides, GANNet: a genetic algorithm for optimizing topology and weights in neural networks design, in: Proceedings of IWANN'93, Lectures Notes in Computer Science, vol. 686, 1993, pp. 322–327.

[17] P.A. Castillo, J. Carpio, J.-J. Merelo-Guervós, V. Rivas, G. Romero, A. Prieto, Evolving multilayer perceptions, Neural Processing Letters 12 (2000) 115–127.

[18] P.A. Castillo, J.-J. Merelo-Guervós, A. Prieto, V. Rivas, G. Romero, G-prop: global optimization of multilayer perceptrons using GAs, Neurocomputing 35 (2000) 149–163.

[19] J.J. Merelo, A. Prieto, G-LVQ a combination of genetic algorithms and LVQ, in: D.W. Pearson, N.C. Steele, R.F. Albrecht (Eds.), Artificial Neural Nets and Genetic Algorithms, Springer-Verlag, 1995.

[20] J. Gonzlez, I. Rojas, H. Pomares, M. Salmerón, Expert mutation operators for the evolution of radial basis function neural networks, in: J. Mira, A. Prieto (Eds.), Connectionits Models of Neurons, Learning Processes, and Artificial Intelligence, Lecture Notes in Computer Science, vol. 2084, IWANN'2001, June 2001, pp. 538–545.

[21] A.F. Sheta, K. de Jong, Time-series forecasting using GA-tuned radial basis functions, Information Sciences 133 (3–4) (2001) 221–228, April.

[22] J.J. Merelo, M.G. Arenas, J. Carpio, P.A. Castillo, V.M. Rivas, G. Romero, M. Schoenauer, Evolving objects, in: M. Graña (Ed.), FEA2000 (Frontiers of Evolutionary Algorithms) Proceedings, Proc. JCIS'2000. P.P. Wang (Ed.), Editorial Association for Intelligent Machinery, vol. I, Atlantic City, NJ, ISBN:0-9643456-9-2, February 27–March 3, 2000, pp. 1083–1086.

[23] M. Keijzer, J.-J. Merelo-Guervós, G. Romero, M. Schoenauer, Evolving objects: a general purpose evolutionary computation library, in: P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, M. Schoenauer (Eds.), Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France, October 29–31, 2001, Selected Papers, Lecture Notes in Computer Science, vol. 2310, Springer, 2002, pp. 231–244.

[24] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, thrid ed., Springer-Verlag, New York, USA, 1999.

[25] V.M. Rivas, J.J. Merelo, P.A. Castillo, Evolving RBF neural networks, in: J. Mira, A. Prieto (Eds.), Connectionits Models of Neurons, Learning Processes, and Artificial Intelligence—IWANN'2001, Lecture Notes in Computer Science, vol. 2084, Springer, 2001, pp. 506–513, June 2001.