Computing and Informatics, Vol. 20, 2001, 429–449

OPTIMIZING THE NUMBER OF LEARNING CYCLES IN THE DESIGN OF RADIAL BASIS NEURAL NETWORKS USING A MULTI-AGENT SYSTEM

José M. Molina, Inés M. Galván, José M. Valls, Andrés Leal

Computer Science Department
Carlos III University of Madrid
Avenida de la Universidad, 30. 28911 Leganés, Spain
e-mail: molina@ia.uc3m.es

Manuscript received 22 May 2001; revised 5 November 2001
Communicated by Pavol Návrat

Abstract. Radial Basis Neural (RBN) network has the power of the universal approximation function and the convergence of those networks is very fast compared to multilayer feedforward neural networks. However, how to determine the architecture of the RBN networks to solve a given problem is not straightforward. In addition, the number of hidden units allocated in an RBN network seems to be a critical factor in the performance of these networks. In this work, the design of RBN network is based on the cooperation of $n + m$ agents: n RBN agents and m manager agents. The $n + m$ agents are organized in a Multi-agent System. The training process is distributed among the n RBN agents, each one with a different number of neurons. Each agent executes a number of training cycles, a stage, when the manager decides about that is the best RBN agent and sends it the corresponding message. The m manager agents have in charge to control the evolution of each problem. Each manager agent controls one problem. Manager agents govern the whole process; each one decides about the best RBN agent in each stage for each problem. The results show that the proposed method is able to find the most adequate RBN network architecture. In addition, a reduction in the number of training cycles is obtained with the proposed Multi-agent strategy instead of sequential strategy.

Keywords: Multiagent systems, distributed systems, distributed learning, neural networks, radial basis NN

1 INTRODUCTION

RBN networks [13, 15] originated from the use of radial basis functions, as the Gaussian functions, in the solution of the real multivariate interpolation problem. RBN networks are composed of a single hidden layer with local activation functions distributed in different neighborhoods of the input space. The significance of this is that RBN networks construct local input-output approximation, which allows to obtain a very fast convergence of the learning algorithm [5, 13].

Despite these advantages RBN networks are not as widely used as they should be. One of the main reasons seems to be that it is not straightforward to design an appropriate RBN network to solve a given problem. The design of a successful RBN network involves to determine the topology, i.e., the number of radial basis units or hidden units, which is a critical factor in the performance of these networks.

The design of RBN networks is generally carried out using a tedious trial and error process, i.e., different architectures of RBN networks varying the number of hidden units are completely trained. From those, the topology being able to obtain the best performance of error is chosen as the definite topology to solve the problem. There are no general systematic ways to design a near optimal topology for a given task. The trial and error process, generally, implies a large computational cost, in terms of learning cycles, because the learning process must be completed for each RBN network architecture.

During the last years, the interest of the scientific community to develop methods or algorithms to automatically design appropriate topologies of artificial neural networks has increased. The main attention in this field has been paid to the multilayer feedforward neural networks, for instance [4, 7, 9]. Only a few works concerning determination of the architecture of RBN network appear in the literature. One line research in this field is focused on use evolutionary techniques, as genetic algorithm, to determine the optimal number of hidden units in an RBN network [8, 18, 19]. It has been shown that those strategies can be successful in some applications. However, they generally imply a large computational cost because a large set of RBN network must be trained. Other works in a different research line are found in the literature. For instance, in Platt [14], an algorithm is developed to determine the number of clustering or hidden units necessary to solve the problem. The number of hidden neurons is initialized to one and the algorithm allocates a new computational unit whenever an unknown pattern is presented to the network. Variants of this method are presented in Kadirkamanathan and Niranjan [10]; Yingwei et al. [21], where pruning strategy is incorporated. These methods imply, generally, sophisticated algorithms that might identify useless hidden neurons and in many cases the performance of methods depends of the quality of training data.

Most of the methods mentioned above treat not only to find the optimal number of hidden units in an RBN network, but they also calculate the parameters of the RBN network proposing new methods and strategies to determine them. Taking into account that the most usual learning method to train RBN network — described in Section 2 — is very simple, faster and efficient when the number of hidden neurons is appropriate. The interest of this work is not to modify that learning mechanism, but to propose other strategies that allow to determine the best architecture. In addition, the goal of this paper is also to avoid a trial and error process to obtain adequate RBN networks and to propose a strategy that requires less learning cycles than the trial and error mechanism without modifying the usual learning algorithm.

In this work, the proposed automatic system to determine an appropriate RBN network for a given problem is based on the cooperation of different agents which are organized in a Multiagent system [11, 12, 17]. The proposed system is composed of $n + m$ agents: n RBN agents and m manager agents. The RBN agents are RBN networks with different topology, i.e., with different number of hidden neurons. Each manager agent receives information about the evolution of the training process in each RBN agent for a problem. Each manager agent is in charge to control the evolution of each problem and it decides what topology is most suitable to solve its problem in terms of being able to reach the minimum error in the least number of learning cycles. In this way, in the Multiagent system, only one RBN agent (that defines a particular topology) is training at a time for a problem and the training process is distributed among the n RBN. Each agent executes a number of training cycles, a stage, when the manager decides that is the best RBN agent for a given problem (each manager agent solves a problem). The whole process is governed by manager agents, each one determines the best RBN agent in each stage for each problem and sends the corresponding message to it.

The number of learning cycles executed by the proposed system until finding the most adequate RBN network is measured. In order to show the effectiveness of the Multiagent system, the number of learning cycles that would involve a trial and error process is also measured. In this case, the trial and error mechanism is organized as a sequential procedure without manager agents, i.e., all architectures candidates — the same as the number of RBN agents — are trained simultaneously one learning cycle at a time until one of them reaches the minimum error.

The Multiagent system allows to reduce the number of training cycles of all RBN networks compared with a sequential training process. In addition, the proposed Multiagent system is a general architecture that could be useful to execute several tasks. Although in this work it is only used in the training task for a given problem, the Multiagent system can also be used for different tasks. For instance, agents could execute different training tasks (each task for each problem), or test tasks: while some networks are trained, tests could be carried out by other networks simultaneously.

The structure of the rest of the paper is as follows: Section 2 presents an overview of RBN networks. The general characteristics of Multiagent systems are presented in Section 3. In Section 4, a description of the Multiagent system Architecture and the communication protocol is included. The experimental results are presented in Section 5. Some conclusions derived from this work are presented in Section 6.

2 RADIAL BASIS NEURAL NETWORKS

RBN network involves three functionally distinct layers (see Figure 1). The input layer is simply a set of sensor units, which transmits the external input without connections. The second layer is the hidden layer, which performs a non-linear transformation of the input space. The third and final layer consists of one neuron that combines the hidden units linearly. The equation describing the output of a radial basis networks is as follows:

$$\tilde{y}_x = w_0 + \sum_{i=1}^{k} w_i \cdot \phi_i(x), \tag{1}$$

where $w_i, i = 1, \ldots, k$, are the weights of the network; w_0 is the threshold associated to the output node; $x = (x_1, \ldots, x_n)$ is the external input to the network; and $\phi_i, i = 1, \ldots, m$, are the radial basis or activation functions. The radial basis functions are often taken to be translated dilations of a prototype function, ϕ, as follows:

$$\phi_i(x) = \phi\left(\frac{\|x - c_i\|}{d_i}\right), \tag{2}$$

where $c_i = (c_{i1}, \ldots, c_{in}) \in R^n$ is the center of the function ϕ, d_i is the dilation, width or scaling factor for the radius $|x - c_i\|$ and $|\cdot\|$ is typically the Euclidean norm. The most general class of prototype functions is the Gaussian function:

$$\phi(r) = e^{-r^2/2}. \tag{3}$$

The activation of the hidden units have an appreciable value only in the neighborhood of the vector c_i; for any given input, only the small fraction of processing units with centers very close (in the input space) to the input will respond with activation which differs significantly from zero. Thus, only those weights that belong to hidden units with centers close enough to the input need to be evaluated and trained. In this sense, it is said that radial basis neural networks have a local character.

Learning in RBN network involves determination of the centers $c_i, i = 1, \ldots, k$, the dilations $d_i, i = 1, \ldots, k$, and the weights $w_i, i = 1, \ldots, k$. The learning of RBN networks can be carried out using two different styles of learning [13]: a fully supervised method and a hybrid method which combines supervised and unsupervised methods. The first approach uses an error measure defined at the output and the problem is solved as a single global nonlinear optimization problem. The learning of RBN network can be also carried out by breaking it into two phases. The first phase determines the centers c_i and the radial dilation factors, d_i, in an unsupervised manner using self-organizing methods, while the second phase performs the optimization via supervised training to determine the weights w_i. In this work, the learning of RBN networks is carried out according to the second learning approach, as follows:

1. The values c_i are determined by using the standard k-means algorithm. That algorithm is used to find k receptive field centers in the input training examples.

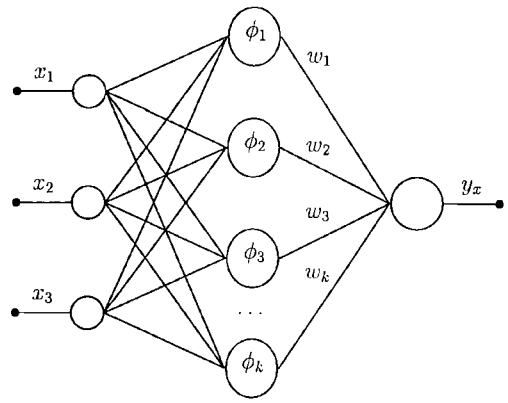


Fig. 1. Radial basis neural network

The center of the cluster determines the values c_i for the basis function (center of each hidden neuron). The number of k value is equal to the number of hidden neurons established in the network.

2. The width of each field or dilation coefficients d_i are calculated as the square root of the product of the Euclidean distances from the center c_i to its two nearest neighbours.

3. The determination of the weights is -in this work- properly the training stage of the RBN network because the centers and dilations are previously calculated. The weights of the RBN network, w_i, are found from the minimization of the mean square error:

$$E = \frac{1}{2N} \sum_{k=1}^{N} (y_x - \tilde{y}_x)^2 \tag{4}$$

where y_x and $\tilde{y}_x$ are the desired and actual network output for the input pattern x, respectively and N is the number of training samples. The minimization of the error function is carried out by an iterative process. Thus, the weights are updated for each input-output pair (x, y_x) following the negative gradient direction of the error function E, as follows:

$$w_i^{(n+1)} = w_i^{(n)} + \alpha \cdot (y_x - \tilde{y}_x) \cdot \phi_i(x), \quad i = 1, \ldots, k. \tag{5}$$

3 MULTIAGENT SYSTEM

Multiagent system proposed in this paper allows training the best RBN for a given problem concurrently with other problems. The development of a Multiagent sys-

tem is based on Distributed Artificial Intelligence (DAI) techniques [6]. The general philosophy of DAI is the decomposition of a problem to be solved by several subsystems. A distributed system could be defined through the following characteristics [2]:

- The system consists of a collection of subsystems (agents). Each agent has various skill, including sensing, communication, planning and acting.
- The group as a whole (the system composed of agents) has a set of assigned tasks (goals) to be solved.
- Each subsystem (agent) has only limited knowledge, there is no place in which all the knowledge is contained.
- Each subsystem (agent) has different capabilities and, then, differing appropriateness for a given problem or subproblem.

Compared to a traditional centralized system, in a Multiagent system (MAS) [1] the global problem is decomposed into subproblems that are solved dynamically by different agents as functions of its appropriateness. Several characteristics about control and data must be taken into account in a MAS (different from a centralized architecture):

- A global system control does not exist. There is no central agent that monitors the execution of all tasks in each agent (in a centralized architecture the control of the global solution is placed in one subsystem and the execution of any task is controlled by this subsystem).
- Data is not centralized, so all agents must share data. Each agent is able to sense the environment to obtain its own data, or input data could be received from any other agent. Any agent has a global vision of all data in MAS, each agent takes its own decision as function of its local data.
- System execution is asynchronous; any agent can be working while it receives queries any time. The synchronization of the system is related with the interchange of messages among agents (in a centralized architecture the control subsystem imposed a synchronization of tasks).

As mentioned above, a MAS system is based, fundamentally, on two ideas: the agent concept and the coordination of agents to achieve a common goal(s). An agent can be defined as a system with the following properties [20]:

- Autonomy: an agent is able to decide what to do using its local control over its actions and internal state.
- Social Ability: agents interact with other agents using some kind of agent communication language.
- Reactivity: an agent is able to sense the environment and respond to the changes on it.

- Pro-activeness: agents have their own goals.

The architecture of an agent could be decomposed into three components [11]: task solving component, the cooperation super-strate and the communication funcionality. These components could be defined for each type of agent for a given problem in order to specify its capacity in two dimensions: what can the agent do (skills) and how can the agent coordinate with other agents (control and communication)?

The coordination of those different agents is a complex problem [16]. In a distributed architecture, the coordination to obtain the solution of global goals is obtained by means of its local control and the communication among agents. Basically two dimensions, the control and the communication, ought to be addressed [3]. The control defines the cooperation, the organization of agents and the dynamics of the control organization in time. The communication specifies the protocols, the contents of the message among agents and the paradigm by which the communication takes place.

The advantages of MAS are mainly related in the characteristic of distributed problems:

- They are able to solve large size problems, especially those where classical systems are not successful.
- They allow different systems to work in connection and cooperation.
- They provide efficient solutions where information is distributed among different places.
- They allow software reusability and flexibility adopting different agent capabilities to solve problems.

4 RBN MULTIAGENT SYSTEM

The Multiagent system proposed in this paper is designed to find out the optimal topology of the radial basis neural networks being able to solve several problems simultaneously. The system is composed of two types of agents: RBN agents and manager agents (MA).

RBN agents are able to execute an RBN network. Each RBN agent can train a topology (different for each agent), i.e., an RBN with a pre-defined number of hidden neurons. There is no communication among RBN agents; they only communicate with manager agents.

MA agents are able to solve a real problem using RBN agents. The initial definition of this problem and the function to evaluate the learning process of each RBN is known by each MA. Each MA has knowledge about all RBN agents. Using that knowledge, in the end of each stage each manager agent decides which is the best RBN agent, this is, which one is the most appropriate to solve the problem. Below, the tasks of the agents involved in the Multiagent system and their architecture are described.

The responsibilities of a manager agent are as follows:

- Send the initial information of the problem. This information is the number of cycles, the credit evaluation function to be used (from a set of possible functions), etc.
- Data collection from RBN agents.
- Monitoring the evolution of the problem, in order to ensure that the system goals are achieved.
- Choosing the RBN agent with the best performance, this is associated with the best credit value, which is explained in Section 4.1.1.

On the other hand, an RBN agent has to carry out the following tasks for each problem:

- Establishing the communication to the manager agent (receiving the initial information).
- Executing a fixed number of learning cycles when indicated by the manager agent; the number of cycles is named a stage (this information is received initially).
- Evaluating the credit value associated to its radial basis neural network when the number of learning cycles is reached, in order to specify the optimal architecture (see Section 4.1.1). The credit evaluation function used depends on the problem and is received as initial information.
- Checking whether the goal has been reached.
- Checking when the training process must be stopped.

4.1 RBN Agents Architecture

As shown in Figure 2a, RBN agents have three different modules:

a) **Communication module** — This module performs the communications to manager agents using a simple protocol following the KQML one, described in Section 4.3.

When several MA send a message in order to execute the RBN for its problem, this module decides the priority of petitions as function of the evolution of results.

b) **Control module** — The control module stores information about the training state of the RBN network for each problem. That information is useful to measure the fitness of the neural network to solve the given problem.

- The number of learning cycles accomplished until the current instant in order to measure the computational effort involved.

- The mean square error reached until the current instant to measure the fitness of the RBN network.
- The derivative of the mean square error with respect to the number of learning cycles to measure the speed of the convergence.
- The mean square error desired to solve the problem.
- The credit value associated to the RBN network.

On the other hand, the control module performs the following operations:

- Starting a training stage.
- If the goal is achieved, the training is stopped.
- Computing the credit value, as shown in 4.1.1., using the credit evaluation function sent by the corresponding MA.

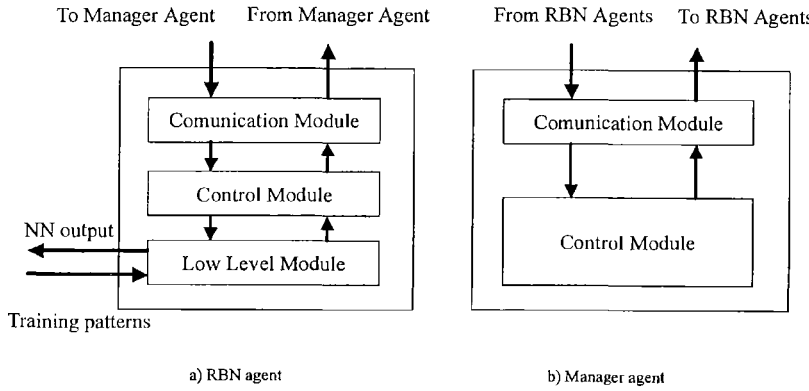


Fig. 2. Agents architecture

c) **Low level module (NN Module)** — The low level module is the radial basis neural networks in the strict sense. It receives the training input patterns and produces the output network.

4.1.1 Credit Evaluation Function

One of the most important aspects of the Multiagent System proposed in this work is the way to assign the credit value to each RBN agent. The manager agent uses that value to decide which RBN agent must start a new training stage. Hence, the result of the Multiagent System depends widely on the credit value chosen. The simplest function that a MA could send uses the number of the learning cycles (n) and the mean square error given by eq. (4) (E) to evaluate the credit value because

they are the main parameters to reflect the fitness of an RBN network. In this work the credit function selected is as follows:

$$C = \frac{1}{nE}. \tag{6}$$

Several considerations have been taken into account, which are briefly described below:

- The smaller the error is, the better is the RBN network. So, C should be inversely proportional to E.
- The sooner the agent achieves a small error, the better. So, C should be inversely proportional to the number of training cycles n.

In any case, if the derivative of the mean square error is very small (< 0.000001), and this error is bigger than the goal (defined by the problem), the agent must be discarded. A value of $C = -1$ has been chosen to represent this situation.

4.2 Manager Agent Architecture

As shown in Figure 2b, the manager agent has two different modules:

a) **Communication module** This module performs the communication with RBN agents. If all RBN agents are able to solve the problem, the protocol followed is explained in Section 4.3. When the RBN agent rejects an ORDER message, the communication module selects the next RBN agent to perform the learning stage.

b) **Control module**

- Stores the information about the credit value of each RBN agent.
- Decides which of the RBN agents is the best.

4.3 Communication Protocol

A manager agent (MA) initiates the process (see Figure 3). Its communication module generates a REQUEST.ORDER(Init, GP) act, and a message is sent to each RBN agent (RA). GP is a data structure containing the general parameters of the system: Epsilon (Goal Error), MaxCycles (max. number of training cycles), N (number of training cycles in each training stage) and the credit evaluation function.

Each RA acknowledges this message generating an ACCEPT(N_j) act which, when processed, sends an answer message to the manager agent, and starts the initial training stage. That initial training stage differs form the rest of training stages because it involves, on one hand, the determination of the centers and dilatations coefficients, and on the other hand, N learning cycles over the weights applying eq. (5). If the RA is not ready to perform this task, it generates a REJECT(N_j) act which, when processed, sends a message to the MA.

The MA waits to have all the answers from the RA From now on, the communication is established among the MA and the RA's which accepted the first message.

When the RA's finish their initial training stage, their control module computes the credit value, and sends it to the communication module which generates an INFORM (N_i, C) act which produces a message to the MA. The MA control module writes the value of C into a table indexed by the RBN Agent number (Credit Table).

Once the MA has received the messages from all the RA's, it selects the best value from the credit table, takes its index j, and generates a REQUEST.ORDER (Train, N_j) act which produces a message sent to the j-th RA. This agent acknowledges it with an ACCEPT act, and starts a new training stage — N learning cycles over the weights to minimize the global error function E using eq. (5). If this agent is working in another problem, the agent sends a REJECT act and the MA selects the next best agent. When the MA receives an ACCEPT message, it means that the RBN agents are running a stage of learning and the result will be received later. Meanwhile, the MA stays on a wait state, until an INFORM(N_j, C) message from the j-th agent is received. When this happens, the MA writes the new value of C into the table, and, again, selects the best value from the credit table, takes its index j, and generates a REQUEST.ORDER(Train, N_j).

The negotiation stops when:

1. All the credit values are -1, so none of the RA's is suitable. In this case, the MA communication module generates a REQUEST.ORDER(Kill, Fail) act, which, when processed, sends a message to every RA, to stop the system. The result is that none of the agents is able to perform the task.
2. The k-th RA reaches an error E that EK <= Egoal. In this case, this agent stops its training stage, and generates an INFORM(Success, Status, N_k) act which produces the respective message to the MA which generates a REQUEST. ORDER(Success, N_k) act that implies a message to be sent to all the RA's to stop the system, informing that the goal has been achieved by the k-th agent.

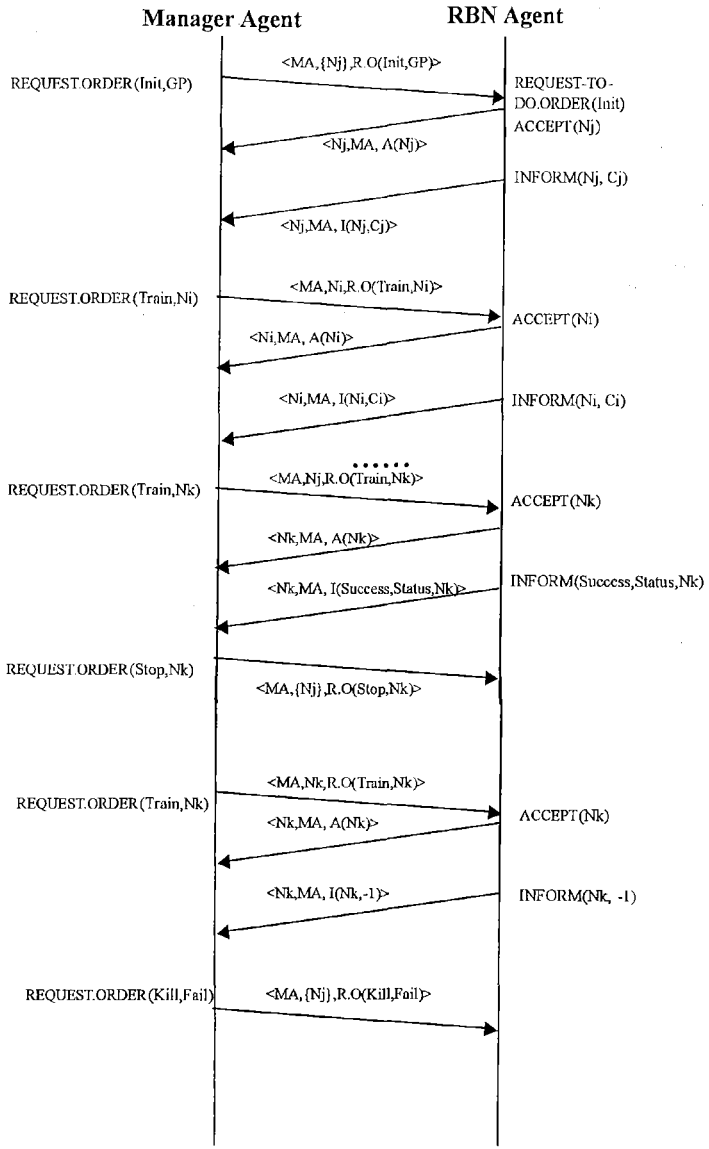


Fig. 3. Communication protocol

5 EXPERIMENTAL RESULTS

The Multiagent System proposed in this work is developed in a UNIX environment. Agents are executed in parallel (each agent is a different UNIX process) and are communicated using the socket paradigm. This system has been applied to three different approximation domains: the logistic time series, the Hermite polynomial and a piecewise-defined function. The aim for each experimental case is to find the most appropriate RBN network topology, which is measured in terms of finding the RBN network topology being able to reach the minimum error in the least number of learning cycles. In this work, several simulations varying the minimum mean square error that must be reached by the networks have been carried out.

With this purpose, a set of RBN agents with different number of hidden neurons is established for each domain and the Multiagent system has to determine the most adequate architecture. In order to verify whether the architecture obtained by the Multiagent system is certainly the best, all RBN networks of the set are trained and the evolution of the mean square errors during a large number of learning cycles is obtained.

In addition, for each domain, the number of learning cycles involved in the Multiagent system has been compared with the number of learning cycles required when all RBN networks are simultaneously trained. In the simultaneous training an adaptation step is carried out at a time in each agent. The simultaneous training stops when an RBN network reaches the desired mean square error. Then, the number of cycles is obtained by adding all the training cycles carried out by all the RBN networks.

5.1 Logistic Time Series

The logistic map given by the following equation describes a chaotic time series:

$$x(t) = \lambda\, x(t-1)[1 - x(t-1)], \tag{7}$$

where $\lambda = 3.97$ and $x(0) = 0.5$. Data of the logistic time series from $t = 0$ to $t = 100$ have been used as training patterns.

The simulations have been conducted using 10 RBN agents, in which the topology of the networks is fixed to 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 hidden neurons, respectively. In Table 1, the RBN networks obtained by the proposed system for three representative minimum errors are shown.

Minimum error	Optimal RBN network
0.001	10 hidden neurons
0.0007	15 hidden neurons
0.0005	15 hidden neurons

Table 1. Optimal RBN networks obtained by the MA System for the logistic time series

As can be seen in Figure 4 -- in which the evolution of the 10 RNN networks is shown — the Multiagent system is able to find the most adequate architecture for each minimum error. When the minimum error is fixed to 0.001, the best topology is composed of 10 hidden neurons. However, for a minimum error of 0.0005, 15 or 20 hidden neurons provide the best performance.

In Table 2 the Multiagent system is compared to a trial and error process. The number of learning cycles involved in the Multiagent system and in a simultaneous training process for different minimum error are shown. The performance of the Multiagent system is very similar to the simultaneous process when the minimum error is large. However, as the minimum error required decreases, the Multiagent system needs smaller number of cycles.

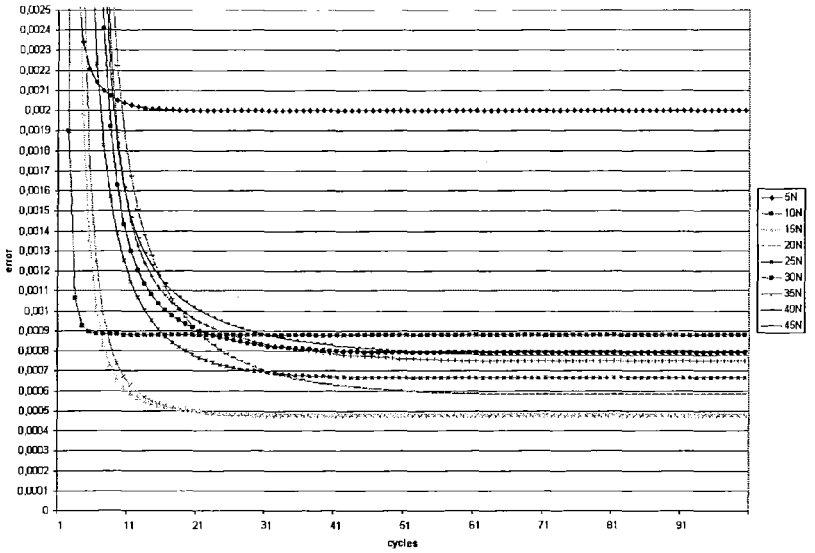


Fig. 4. Evolution of the mean square error for different RBN network topologies: logistic time series

5.2 Hermite Polynomial

The Hermite polynomial is given by the following equation:

$$f(x) = 1.1\left(1 - x + 2x^2\right)\exp\left(-\frac{1}{2}x^2\right). \tag{8}$$

Logistic time series		
Minimum error	Simultaneous training	Multiagent system
0.001000	29	20
0.000950	29	20
0.000900	38	21
0.000850	57	37
0.000800	66	38
0.000750	66	38
0.000700	75	39
0.000650	84	40
0.000600	92	41
0.000550	117	44
0.000500	164	51

Table 2. Multiagent system vs. simultaneous training for the logistic time series

A random sampling of the interval $[-4, 4]$ is used to obtain 100 input-output points for the training set. Data are normalized in the interval $[0, 1]$.

As for the logistic time series case, 10 RBN agents -- with 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 hidden neurons — have been used by the Multiagent system. The most appropriate RBN network obtained by the Multiagent system for three representative values of minimum error is shown in Table 3. In Figure 5 the evolution of the mean square error for the 10 architectures of RBN network is also displayed.

Minimum error	Optimal RBN network
0.001	20 hidden neurons
0.0007	20 hidden neurons
0.0002	35 hidden neurons

Table 3. RBN networks obtained by the MA System for the Hermite polynomial

Figure 5 shows that the best topology of RBN network is composed of 20 or 35 hidden neurons. The Multiagent system has also provided those architectures as the best. In addition, the Multiagent system requires less learning cycles to find it than a simultaneous training process, as shown in Table 4. In this case, the advantage of the Multiagent system in terms of learning cycles is more significant when the minimum error required is reduced.

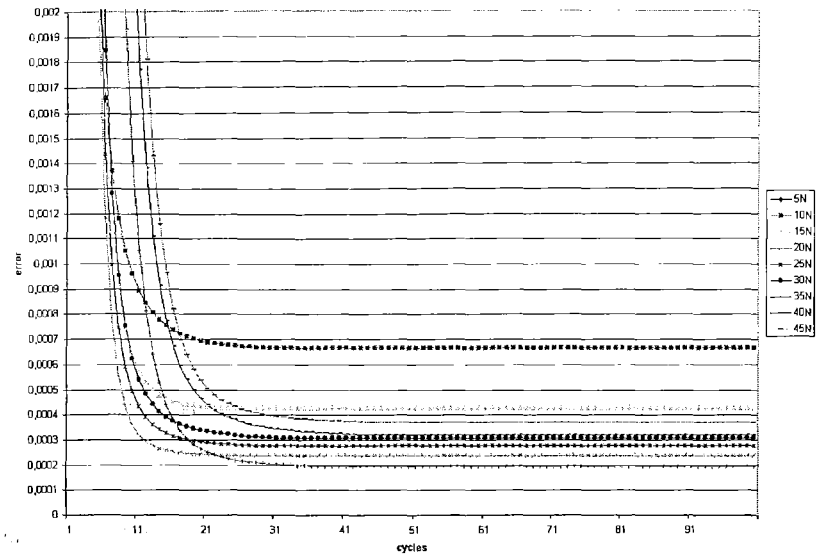


Fig. 5. Evolution of the mean square error for different RBN network topologies: Hermite polynomial

5.3 A Piecewise-Defined Function

The Multiagent strategy has also been applied to the approximation of the single variable piecewise-defined function. This function is given by the following equation:

$$f(x) = \begin{cases} -2.186x - 12.864, & \text{if } -10 \leq x < -2, \\ 4.246x, & \text{if } -2 \leq x < 0, \\ 10e^{(-0.05x-0.5)} \sin[(0.03x + 0.7)x], & \text{if } 0 \leq x \leq 10. \end{cases} \quad (9)$$

The training set is composed of 120 points randomly generated by the uniform distribution in the interval $[0, 1]$. In this case, the number of hidden neurons for the candidate RBN networks had to be increased in order to get an acceptable minimum error. Thus, 14 RBN agents have been incorporated to the Multiagent system, in which the topology of the networks is fixed to 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70 and 75 hidden neurons, respectively. The optimal topologies obtained by the Multiagent system for the different minimum errors are shown in Table 5.

As observed in Figure 6, the best architecture is composed of 55 hidden neurons for each minimun error. The Multiagent system is able to find this architecture.

In addition, the number of learning cycles required by the Multiagent system is smaller than if a simultaneous training process is used, as observed in Table 6. In

Hermite polynomial		
Minimum error	Simultaneous training	Multiagent system
0.001000	58	47
0.000950	58	47
0.000900	58	47
0.000850	58	47
0.000800	58	47
0.000750	67	48
0.000700	67	48
0.000650	67	48
0.000600	67	48
0.000550	76	49
0.000500	76	49
0.000450	76	49
0.000400	85	50
0.000350	94	51
0.000300	103	52
0.000250	157	58
0.000200	276	198

Table 4. Multiagent system vs. Simultaneous for the Hermite polynomial

Minimum error	Optimal RBN network
0.001	55 hidden neurons
0.0007	55 hidden neurons
0.00035	55 hidden neurons

Table 5. RBN networks obtained by the MA System for the piecewise-defined function

that table, the number of cycles needed by the Multiagent system and the simultaneous training for different minimum errors are presented. For the piecewise-defined function, the difference of learning cycles for both methods is larger than in previous experimental cases where the minimum error is more restrictive.

6 CONCLUSION

The design of optimal RBN networks for a given problem is generally based on the experience of the developer and in most cases the trial and error process is used. On the other hand, the performance of RBN networks depends strongly on the number of hidden neurons allocated in the network. Consequently, it is interesting to develop automatic methods that allow to determine the most appropriate architecture of RBN networks — in terms of reaching the minimum error in the minor number of learning cycles — requiring the least number of learning cycles that a exhaustive

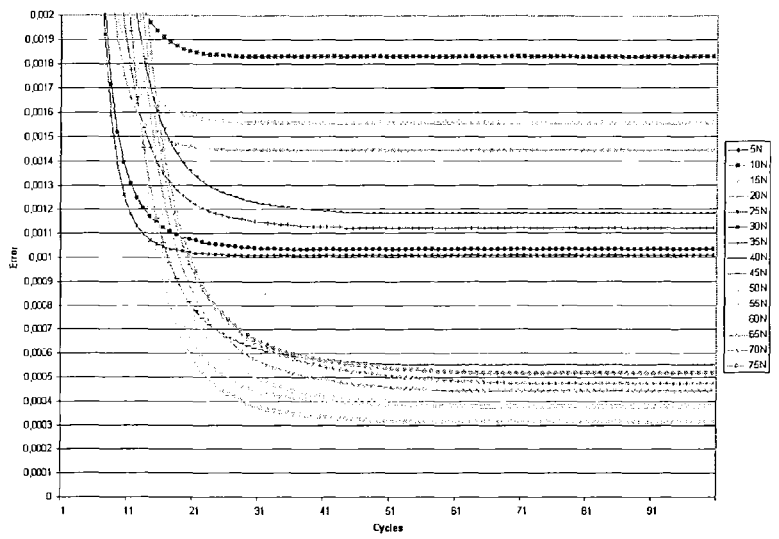


Fig. 6. Evolution of the mean square error for different RBN network topologies: Piecewise-defined function

Piecewise-defined function		
Minimum error	Simultaneous training	Multiagent system
0.001000	236	208
0.000950	236	208
0.000900	251	209
0.000850	251	209
0.000800	266	210
0.000750	266	210
0.000700	281	211
0.000650	296	212
0.000600	311	213
0.000550	326	214
0.000500	341	215
0.000450	371	217
0.000400	426	221
0.000350	503	228

Table 6. Multiagent system vs. Simultaneous for the piecewise-defined function

search. This work proposes the use of different RBN networks and the cooperation among them in a Multiagent system in order to automatically detect the best suited RBN network.

The Multiagent system proposed is composed of $n + m$ agents: n RBN agents and m manager agents. Each manager agent controls the evolution of the solution for each problem. In this way, the proposed architecture allows to train several problems (m, one for each manager agent) simultaneously, optimizing the use of the RBN agents. The protocol used in this MAS follows the KQML one, adding the initial information that the manager agent sends to RBN agents in order to define the problems exactly.

This architecture and the developed communication protocol has been tested in different experiments. The results have demonstrated the validity of the proposed system compared with an exhaustive search of the best RBN network. In most of the experimental simulations, the architecture provided by the MAS is the most appropriate to solve the given problem and the number of learning cycles involved is significantly less that if a simultaneous learning process of the candidate architecture is carried out.

In addition, it is interesting to point out that the proposed Multiagent system is a general architecture that could be useful to execute several tasks. Although in this work it is only used in the training task for a given problem, the Multiagent system can also be used for different problems. This would imply better exploitation of system resources.

REFERENCES

[1] Brenner, W.—Zarnekow, R.—Wittig, H.: Intelligent Software Agents. Foundations and Applications. Springer-Verlag, 1998, ISBN 3-540-63411-8.

[2] Cammarata, S.—McArthur, D.—Steeb, R.: Strategies of Cooperation in Distributed Problem Solving. In Readings in Distributed Artificial Intelligence, Ed. Alan H. Bond and Les Gasser, Morgan Kaufmann 1988.

[3] Decker, K. S.: Distributed Problem-Solving Techniques: A Survey. IEEE Transactions on Systems, Man, and Cybernetics, 1987.

[4] Fogel, D. B.—Fogel, L. J.—Porto, V. W.: Evolving Neural Network. Biological Cybernetics, 63, pp. 487–493, 1990.

[5] Galván, I. M.—Zaldivar, J. M.—Hernández, H.—Molga, E.: The Use of Neural Networks for Fitting Complex Kinetic Data. Computer Chemical Engineering, Vol. 20, 1996, No. 12, pp. 1451–1465.

[6] Gasser, L.: An Overview of DAI. Distributed Artificial Intelligence: Theory and Practice. Kluwer Academic Publishers, 1992.

[7] Gruau, F.: Automatic Definition of Modular Neural Networks. Adaptive Behaviour, Vol. 2, No. 3, 1995, pp. 151–183.

[8] Gruau, F.: Genetic Synthesis of Modular Neural Networks. Proceedings of the 5th International Conference on Genetic Algorithms, San Mateo CA, pp. 318–325, 1995.

[9] HARP, S.—SAMAD, T.—GUHA, A.: Designing Application-Specific Neural Networks using the Genetic Algorithm. Advances in Neural Information Processing Systems, Vol. 2, 1990, pp. 447–454.

[10] KADIRKAMANATHAN, V.—NIRANJAN, M.: A Function Estimation Approach to Sequential Learning with Neural Networks. Neural Computation 5, pp. 954–975, 1993.

[11] LUX, A.—STEINER, D.: Understanding Cooperation: an Agent's Perspective. Proc. International Conference on Multiagent Systems, ICMAS-95, AAAI Press, San Francisco (CA), 1995.

[12] MOLINA, J. M.—JIMNEZ, F. J.—CASAR, J. R.: Cooperative Management of Netted Surveillance Sensors. IEEE International Conference on Systems, Man and Cybernetics, pp. 845–850. Orlando, EEUU, 1997.

[13] MOODY, J. E.—DARKEN, C. J.: Fast Learning in Networks of Locally-Tuned Processing Units. Neural Computation 1, pp. 281–294, 1989.

[14] PLATT, J.: A Resource-Allocating Network for Function Implementation. Neural Computation 3, pp. 213–225, 1991.

[15] POGGIO, T.—GIROSI, F.: Networks for Approximation and Learning. Proceedings of the IEEE, 78, pp. 1481–1497, 1990.

[16] SYCARA, K. P.: Multiagent Systems. AI Magazine, 1998.

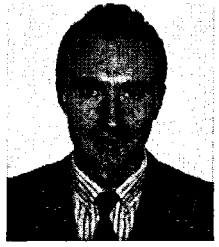
[17] WESSON, R.: Network Structures for Distributed Situation Assessment. Readings in Distributed Artificial Intelligence, Ed. Alan H. Bond and Les Gasser, Morgan Kaufman 1988.

[18] WHITEHEAD, B. A.—CHOATE, T. D.: Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction. IEEE Transactions on Neural Networks 7, 4, pp. 869–880, 1996.

[19] WHITEHEAD, B. A.—CHOATE, T. D.: Evolving Space-Filling Curves to Distribute Radial Basis Functions Over an Input Space. IEEE Transactions on Neural Networks 5, 1, pp. 15–23, 1994.

[20] WOOLDRIDGE, M.—JENNINGS, N. R.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review, 1995.

[21] YINGWEI, L.—SUNDARARAJAN, N.—SARATCHANDRAN, P.: A Sequential Learning Scheme for Function Approximation using Minimal Radial Basis Function Neural Networks. Neural Computation 9, pp. 461–478, 1997.



José M. MOLINA received his Ph.D. in Telecommunication Enginnering from Universidad Politecnica de Madrid, in 1997. He was a member of the System, Signal and Radio Coomunications of the University Politecnica of Madrid from 1992. He also joined the Computer Science Department in 1993 of the University Carlos III of Madrid, being enrolled in the Systems, Complex and Adaptive Laboratory. He is author up to 9 journal papers and 60 conference papers. His current research focuses in the aplication of soft computing techniques (NN, Evolutionary Computation, Fuzzy Logic and Multiagent Systems) to engineering problems as RADAR, robot control and vision.

Inés M. GALVÁN received a doctorate-fellowship, as research scientist, in the European Commission, Joint Research Centre Ispra (Italy) from 1992 to 1995. She received her Ph.D. in Computer Science at Universidad Politecnica de Madrid (Spain), in 1998. She has joined the Computer Science Department at the University Carlos III of Madrid in 1995, being enrolled in the Systems, Complex and Adaptive Laboratory. She current research focuses in Artificial Neural Networks and other soft computing techniques, as Evolutionary Computation and Multiagent Systems. Her research interests cover also applications fields, as time series prediction and control of dynamic process.

José M. VALLS received his degree in Computer Science from Universidad Pontificia de Salamanca, in 1996. He joined the Computer Science Department of the University Carlos III of Madrid in 1998, being enrolled in the Complex and Adaptive Systems Laboratory. He is author up to 2 journal papers and 5 conference papers. His current research focuses in the aplication of Neural Networks.

Andrés LEAL is a student of the last course of Computer Science degree. He works as analyst in Telefónica I+D (Investment and Development) of an internet mobile platform. His research interest is mainly focused in Intelligent Agents.