Pattern Analysis & Applic. (1998)1:52–61
© 1998 Springer-Verlag London Limited

Resolution of Pattern Recognition Problems using a Hybrid Genetic/Random Neural Network Learning Algorithm

J. Aguilar and A. Colmenares
CEMISID, Dpto. de Computación, Facultad de Ingeniería, Universidad de los Andes, Mérida, Venezuela

Abstract: Gelenbe has proposed a neural network, called a Random Neural Network, which calculates the probability of activation of the neurons in the network. In this paper, we propose to solve the patterns recognition problem using a hybrid Genetic/Random Neural Network learning algorithm. The hybrid algorithm trains the Random Neural Network by integrating a genetic algorithm with the gradient descent rule-based learning algorithm of the Random Neural Network. This hybrid learning algorithm optimises the Random Neural Network on the basis of its topology and its weights distribution. We apply the hybrid Genetic/Random Neural Network learning algorithm to two pattern recognition problems. The first one recognises or categorises alphabetic characters, and the second recognises geometric figures. We show that this model can efficiently work as associative memory. We can recognise pattern arbitrary images with this algorithm, but the processing time increases rapidly.

Keywords: Associative memory; Evolutionary learning; Genetic algorithm; Gradient descent rule; Pattern recognition; Random Neural Network

1. INTRODUCTION

In its complete form, neural network induction entails both parametric and structural learning, i.e. learning both weight values and an appropriate topology of nodes and links. Current connectionist methods to solve this task fall into two broad categories. Constructive algorithms initially assume a simple network and add nodes and links as warranted, while destructive methods start with a large network and prune off superfluous components [1,2]. As a consequence, these algorithms tend to force a task into an assumed architectural class, rather than fitting an appropriate architecture to the task.

Thus, how to improve the learning performance of an ANN is currently an important research problem. One approach, inspired by human brain neurons performing many operations simultaneously, is the development of learning algorithms on general-purpose parallel computers, with the objective of reducing the overall computing time [3]. Another approach is the development of more effective neural network learning algorithms, with the objective of reducing the learning time [1,4,5]. A third approach is the development of hybrid learning algorithms by integrating a Genetic Algorithm (GA) with neural network learning algorithms [2,3,6–8]. GAs have recently been applied to ANNs in two main ways: there have been attempts to use genetic search instead of learning to find appropriate connection weights in fixed architectures [6,8]; alternatively, GAs have been used to find network architecture themselves, which are then trained and evaluated using some learning procedure [2,3,7].

The RNN was proposed by Gelenbe in 1989 [9,10]. This model does not use a dynamic equation, but a scheme of interaction among neurons. It calculates the probability of activation of the neurons in the network. Signals in this model take the form of impulses which mimic what is presently known of inter-neural signals in biophysical neural networks. The ability of the

Received: 29 *September* 1997
Received in revised form: 7 *February* 1998
Accepted: 3 *March* 1998

RNN model to act as associative memories has been shown elsewhere [11]. The RNN has been used to solve optimisation [8,12–14] and pattern recognition problems [5,15]. A supervised learning procedure has been proposed [4] for the recurrent RNN model, which is mainly based on the minimisation of a quadratic error function. We [13,14,16] have explored the relationship between the RNN applied to optimisation and network learning. We have also applied the evolutionary learning on the RNN model to solve optimisation problems [8].

The problem addressed in this paper concerns the proposition of a hybrid learning approach based on the RNN and GA to solve the recognition problem. This new algorithm is presented for training the RNN by integrating a GA with the gradient descent rule-based learning algorithm of the RNN. The algorithm simultaneously acquires both network topology and weight values while making minimal architectural restrictions and avoiding structural hill climbing. We introduce two elements in the standard GA to help the exploration/exploitation abilities provided by the search space evolution: a cooperative local optimising genetic operator, and a coding granularity parameter to use different length strings (individuals).

This work is organised as follows. In Section 2, the theoretical bases of the RNN are reviewed. Then, we present an introduction to evolutionary learning and our hybrid algorithm. In Section 4, we present applications. Remarks concerning future work and conclusions are provided in Section 5.

2. RANDOM NEURAL NETWORKS

2.1. Model

The Random Neural Model was introduced by Gelenbe in 1989 [9,10]. The model consists of a network of n neurons in which positive (excitatory) and negative (inhibitory) signals circulate. Each neuron i of the network accumulates signals as they arrive, and is represented at any time t by its input signal potential $k_i(t)$. The arrival of a negative signal reduces $k_i(t)$ by 1 (inhibition) or has no effect on $k_i(t)$ if it is already zero, while an arriving positive signal adds 1 to $k_i(t)$ (excitation). Signals can either arrive at a neuron from the outside of the network or from other neurons. Each neuron can fire if its total signal count at a given instant of time is positive ($k_i(t) > 0$). Firing then occurs at random according to an exponential distribution of constant rate $r(i)$, and signals are sent out to other neurons or to the output of the network. Each time a neuron fires, a signal leaves it, depleting the total input potential of the neuron. A signal which leaves neuron i heads for neuron j with probability $p^+(i,j)$ as a positive signal (excitation), or as negative signal with probability $p^-(i,j)$ (inhibition), or it departs from the network with probability $d(i)$. Clearly, we shall have

$$\sum_{j=1}^{n} [p^+(i,j) + p^-(i,j)] + d(i) = 1 \quad \text{for } 1 \le i \le n$$

Positive signals arrive at the ith neuron according to a Poisson process of rate $\Lambda(i)$ (external excitation signals). Negative signals arrive at the ith neuron according to a Poisson process of rate $\lambda(i)$ (external inhibition signals). The main property of this model is the excitation probability of a neuron i, $q(i)$, which satisfy a non-linear equation

$$q(i) = \lambda^+(i)/(r(i) + \lambda^-(i)) \qquad (1)$$

where

$$\lambda^+(i) = \sum_{j=1}^{n} q(j)r(j)p^+(j,i) + \Lambda(i)$$

$$\lambda^-(i) = \sum_{j=1}^{n} q(j)r(j)p^-(j,i) + \lambda(i)$$

The synaptic weights for positive ($w^+(i,j)$) and negative ($w^-(i,j)$) signals are defined as:

$$w^+(i,j) = r(i)p^+(i,j) \quad w^-(i,j) = r(i)p^-(i,j)$$

and

$$r(i) = \sum_{j=1}^{n} [w^+(i,j) + w^-(i,j)]$$

Let $k(t)$ be the vector of neuron potentials at time t, and $k = (k_1, \ldots, k_n)$ be a particular value of the vector. It has been proved [9,10] that, if all the steady state excitation probabilities $q(i)$ *are such that* $0 \le q(i) \le 1$, the stationary probability distribution of the network's state given by

$$p(k) = \lim t \to \infty \text{ prob } [\underline{k}(t) = \underline{k}]$$

exists, and can be expressed by the product form

$$p(\underline{k}) = \prod_{i=1}^{n} (1 - q(i))q(i)^{k_i}$$

To guarantee the stability of the RNN, the following is a sufficient condition for the existence and uniqueness of the solution in Eq. (1)

$$\lambda^+(i) < [r(i) + \lambda^-(i)]$$

2.2. Recognition Procedure

The recognition procedure is based on an heteroassociative memory technique [5,11,15]. An heteroassociative memory is a system in which an arbitrary set of input patterns is paired with another arbitrary set of output patterns. To design such a memory, we have used a single-layer random neural of n fully interconnected neurons. For every neuron i, the probability that emitting signals depart from the network is $d(i) = 0$, because we are not interested in emitting external signals.

2.2.1. Learning Phase. Synaptic weights and firing rates are determined during the learning phase. Gelenbe has proposed an algorithm [4] for choosing the set of network parameters $w^-(i,j)$ and $w^+(i,j)$ in order to learn a given set of m input-output pairs (X, Y), where the set of successive inputs is denoted by

$$X = \{X_1, \dots, X_m\} \quad \text{where } X_k = (x_{1k}, \dots, x_{nk})$$

and

$$x_{ik} = \{\Lambda_k(i), \lambda_k(i)\}$$

The successive desired outputs are the vector $Y = \{Y_1, \dots, Y_m\}$, where $Y_k = (y_{1k}, \dots, y_{nk})$ and $y_{ik} \in [0,1]$ correspond to the desired output vectors. The learning algorithm is a gradient descent rule-based neural network algorithm. The network approximates the set of desired output vectors in a manner which minimises an error function E_k:

$$E_k = \frac{1}{2} \sum_{i=1}^{n} (q_k(i) - y_{ik})^2$$

The algorithm lets the network learn both n by n weight matrices $W_k^+ = w_k^+(i,j)\}$ and $W_k^- = \{w_k^-(i,j)\}$ by computing for each input-output pair (X_k, Y_k), a new value W_k^+ and W_k^-, using gradient descent. Let us denote by the generic term $w(u,v)$ either $w(u,v) = w^-(u,v)$, or $w(u,v) = w^+(u,v)$. The rule to update the weights may be written as

$$w_k(u,v) = w_{k-1}(u,v) - \mu \sum_{i=1}^{n} (q_k(i) - y_{ik}) \left[\frac{\partial q(i)}{\partial w(u,v)}\right]_k \qquad (2)$$

where $\mu > 0$ is the learning rate (some constant), $q_k(i)$ is calculated using X_k and $w_k(u,v) = w_{k-1}(u,v)$ in Eq. (1), and, $[\partial q(i)/\partial w(u,v)]_k$ is evaluated of the values $q(i) = q_k(i)$ and $w(u,v) = w_{k-1}(u,v)$ in Eq. (2).

The complete learning algorithm for the network is:

- Initiate the matrices W_0^+ and W_0^- in some appropriate manner. Choose a value of μ in Eq. (2).
- For each successive value of m:
  - Set the input-output pair (x_k, Y_k)
  - Repeat until the change in the new values of the weights is smaller than some predetermined valued.
  - Solve Eq. (1) with these values
  - Using Eq. (2) and the previous results, update the matrices W_k^+ and W_k^-.

Every input information is represent by a binary vector $X_k = (x_{1k}, \dots, x_{nk})$, where x_{ik} is associated with neuron i. Hence, we can translate each input vector X_k in terms of the arrival rates of exogenous signals as follows:

$$y_{ik} = 1 \rightarrow (\Lambda_k(i), \lambda_k(i)) = (\Lambda, 0) \qquad (3)$$
$$y_{ik} = 0 \rightarrow (\Lambda_k(i), \lambda_k(i)) = (0, \lambda)$$

where the values Λ and λ provide the network stability for every case. Considering Eq. (1) and the vector encoding Eq. (3), the network stability condition becomes for each vector X_k:

$$\text{if } y_{ik} = 1 \rightarrow \Lambda < r0(i) + \sum_{j=1}^{n} w0^-(i,j) - \sum_{j=1}^{n} w0^+(i,j)$$

$$\text{if } y_{ik} = 0 \rightarrow \lambda > \sum_{j=1}^{n} w0^+(i,j) - r0(i) - \sum_{j=1}^{n} w0^-(i,j)$$

Let

$$\Lambda_k = \min_i,\, y_{ik} = 1 \left[r0(i) + \sum_{j=1}^{n} w0^-(i,j) - \sum_{j=1}^{n} w0^+(i,j) \right]$$

$$\lambda_k = \max_i,\, y_{ik} = 0 \left[\sum_{j=1}^{n} w0^+(i,j) - r0(i) - \sum_{j=1}^{n} w0^-(i,j) \right]$$

To have the network damped for all the m training vectors, we choose the common values $\Lambda \leq \min_{k[}{}^{\Lambda}{}_{k]}$ and $\lambda \geq \max_{k[}{}^{\lambda}{}_{k]}$. This condition is sufficient, but not necessary to provide the network with stability. We must first initialise the connection weight matrices $W^+0 = [w^+0(i,j)]$ and $W^-0 = [w^-0(i,j)]$ either by small positive random variables which are uniformly distributed between 0 and $V_{\max}$, or according to some more appropriate rule. Simulations have lead us to initialise W_0^+ referring to the Hebbian law:

$$w^+0(i,j) = \begin{cases} \sum_{k=1}^{m} (2y_{ik} - 1)(2y_{jk} - 1) & \text{if } w^+0(i,j) > 0 \\ 0 & \text{otherwise} \end{cases}$$

On the other hand, W_0^- is initialized with small positive random variables, uniformly distributed between [0, 0.2].

2.2.2. Retrieval Phase. Once the learning phase is completed, the network must perform the completion of noisy versions of the training vectors as well as possible. In principle, for the retrieval process we do not have to modify the values of the arrival rates of exogenous signals Λ and λ which we have used to provide the network with stability during the learning procedure. If learning is perfect, the global error is practically zero. Thus, the steady-state probability $q(i)$ that each neuron i is excited is such that $0 < q(i) < 1$ for $i = 1, \ldots, n$ and the network remains damped.

Let $X' = (x'_1, \ldots, x'_n)$ be any binary input vector. To determine the corresponding output vector $Y = (y_1, \ldots, y_n)$, we first compute the vector of probabilities $Q = (q(1), \ldots, q(n))$ from the non-linear Eq. (1). This is a fixed-point equation which can be solved iteratively after initialising every probability $q(i)$ to 0.5. Theoretically, the real value of $q(i)$ is very close to 1 (or 0) if the ith component of the target output vector is equal to 1 (or 0). However, when the distortion rate applied on the training data is important, the $q(i)$ of certain neurons can be around 0.5. The state of these neurons is then considered doubtful. We can then consider that the $q(i)$ values such that $1 - b < q(i) < b$ with, for instance, $b = 0.6$, belong to the uncertainty interval Z. When the network stabilises to an attractor state, the number of neurons (NB_Z) whose $q(i) \in Z$, is equal to 0. Hence, we first treat the neurons whose state is considered certain to obtain the output vector $Y^{(1)} = (y_1^{(1)}, \ldots, y_n^{(1)})$, with

$$y^{(1)}{}_i = F_Z(q(i)) = \begin{cases} 1 & \text{if } q(i) \geq b \\ 0 & \text{if } q(i) \leq 1 - b \\ x'_i & \text{otherwise} \end{cases}$$

where F_Z is the thresholding function by intervals. If $NB\_Z = 0$, this phase is terminated and the output vector is $Y = Y^{(1)}$. Otherwise, Y is obtained after applying the thresholding function f_α as follows:

$$y_i = f_\alpha(q(i)) = \begin{cases} 1 & \text{if } q(i) > \alpha \\ 0 & \text{otherwise} \end{cases}$$

where α is the selected threshold. Eventually, Z can be reduced by decreasing b (for $b > 0.5$). Each value $q(i) \in Z$, and also the lower bound $1 - b$ are considered as potential thresholds. For each potential value of α, we present to the network the vector $X'^{(1)}(\alpha) = f_\alpha(Q)$. Then, we compute the new vector of probabilities $Q^{(1)}(\alpha)$ and the output vector $Y^{(2)}(\alpha) = F_Z(Q^{(1)}(\alpha))$. We keep the cases where $NB\_Z = 0$ and $X'^{(1)}(\alpha) = Y^{(2)}(\alpha)$. If these two conditions are never satisfied, the initial X' is considered as too different to any training vector, and α is set to 0.5. On the other hand, if several thresholds are candidate, we choose the one which provides the minimal error (α') according to Eq. (4), and the output vector of this phase is $Y = Y^{(2)}(\alpha')$:

$$Er(a) = \frac{1}{2} \sum_{i=1}^{n} (q(i)^{(1)}(a) - y_i^{(2)}(a))^2 \tag{4}$$

3. EVOLUTIONARY LEARNING

3.1. Introduction

Genetic Algorithm (GA), invented by J. H. Holland, emulates biological evolution in the computer and tries to build programs that can adapt by themselves to perform a given function [17,18]. A GA follows an 'intelligent evolution' process for individuals based on the utilisation of evolution operators such as mutation, inversion, selection and crossover. Optimisation is a major field of GA's applicability. They belong to the class of probabilistic algorithms, yet they are very different from random algorithms as they combine elements of directed and stochastic search. Because of this, GAs are also more robust than existing directed search methods. Another important property of such genetic-based search methods is that they maintain a population of potential solutions; all other methods process a single point of the search space. The population undergoes a simulated evolution: at each generation the 'good' solutions reproduce, while the 'bad' solutions die. To distinguish between different solutions we use a cost function.

Recently, there has been a good deal of interest in using GAs for machine learning problems. The variety and complexity of learning systems make it difficult to formulate a universally accepted definition of learning. However, a common denominator of most learning systems is their capability for making structural changes to themselves over time, with the intention of improving performance on tasks defined by their environment, discovering and subsequently exploiting interesting concepts, or improving the consistency and generality of internal knowledge structures. Given this perspective, one of the most important means for understanding the strengths and limitations of a particular learning system is a precise characterisation of the structural changes that are permitted, and how such changes are made. This perspective also lets one state more precisely the goal of the research in applying GAs to learning, namely, to understand when and how GAs can be used to explore spaces of legal structural changes in a goal-directed manner.

The idea is to define a space of admissible structures to be explored via GAs. The learning strategy involves maintaining a population of tested structures, and using GAs to generate new structures with better performance expectations. In considering the kinds of structural changes that might be made to the system, there are a variety of approaches of increasing sophistication and complexity. The simplest and most straightforward approach is for the GAs to alter a set of parameters that control the behaviour of a system. A second approach involves changing more complex data structures that control the behaviour of the system. A third approach involves changing the system itself.

The use of evolutionary learning based on a GA for ANNs has received much attention recently [2,3,6–8]. Most of the applications regard the use of evolutionary algorithms as a means to learn connection weights; some applications also consider a learning of the network topology; and only a few use evolutionary algorithms to define the parameters of a learning algorithm that will be used in the training phase. Some studies attempt to co-evolve both the topology and weight values within the GA framework, but as in connectionist systems, the network architectures are restricted.

3.2. Our Approach

In this paper, we propose to use a GA to design both the network's topology and the associated weighted connections of the RNN. In this way, we propose a hybrid learning algorithm by integrating a GA with the gradient descent rule-based neural network learning algorithm proposed by Gelenbe. Our hybrid algorithm consists of two learning stages using M RNNs. The backpropagation algorithm performs the first learning process until the terminal condition (error minimisation) is satisfied on each RNN. Then, the second learning stage is used to optimise the topology (connections, weights) of the networks (evolving the network's topology) by using a GA. The GA performs a global search and seeks a near-optimal initial point (connection and weight vectors) for the first stage. In this stage, each individual is used to encode the topology of one of the M RNNs used in the first stage. The fitness (objective) function for the GA is defined as the average squared system error of the corresponding neural network. After performing several iterations and meeting one of the stopping criteria (a predefined number of consecutive iterations, homogeneous individuals), the second learning stage is terminated and the individuals are considered as the new initial topologies of the M RNNs in the next iteration. If one of the next stopping criteria is met, the hybrid algorithm is terminated (system convergence): the new initial topologies of the M RNNs are the same as the initial topologies of the previous iteration, or a given number of consecutive iterations, or if the initial individuals used for the GA do not change in consecutive iterations. The general hybrid algorithm is the following:

- Initiate the parameters for the M RNNs (W_0^+, W_0^-, Λ, λ).
- Repeat until hybrid algorithm convergence
  - for i = 1 to M concurrently
  - Perform the gradient descent rule based neural network learning algorithm.
  - Generate individuals using every RNN,
  - Optimise the individuals (network's topology: connections, weights) using GA.

The main computational advantage of evolving the network's topology is the possibility to autonomously identify minimal search spaces containing reachable solutions to the problem. The dimension of search spaces can be affected by a net coding feature: the bit length of weight coding, i.e. the coding granularity [2]. In our work, coding granularity is a parameter that evolves concurrently to the net structure.

An extended direct encoding scheme is used, where each connection is represented directly by its binary definition. The representation chosen uses network nodes as basic functional units, and encodes all the information relevant for a node in nearby positions, including its input connectivity patterns and the relative weight distribution. Connectivity is coded by the presence/absence bits (connectivity bits). When connection is present, inmediately after each connectivity bit there is the binary encoding of the relative weight (defined in scientific notation). The first byte of the string specifies the number of bits (the granularity) according to which the weights of the present connections have been codified (i.e. mantissa and exponent parts). Thus, coding granularity is a control parameter

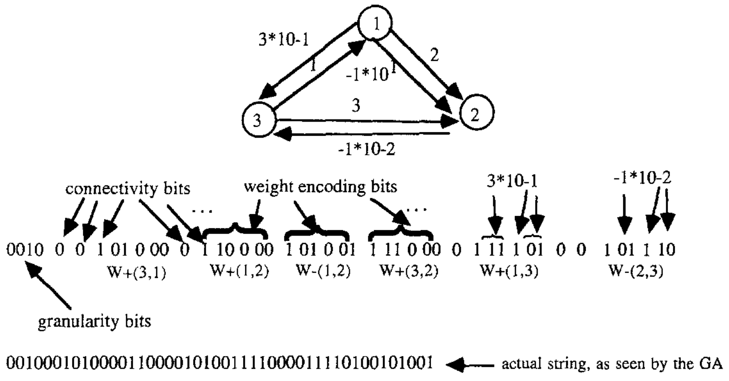


Fig. 1. Network encoding of a complete topology graph.

whose value is coded within the string where it is used. It gives the possibility of having strings of different length. New links are initialised randomly. For example, Fig. 1 presents a simple network and its coding.

We use the commonly used genetic operators (crossover and mutation) and a cooperative local optimising genetic operator. These operators make two types of learning: parametric modifications alter the value of parameters (link weights) currently in the network; whereas structural modifications alter the presence of links in the network. The crossover used is standard, a single cutting point chosen with uniform probability over the string length and a swap of the genetic material following it. Our implementation allows the mating of networks with different connectivity and/or different granularity, with no modification of the operator according to the following ideas (Fig. 2): (a) coded networks are implemented using fixed-length arrays (bit strings), defined with the maximum possible length (all connection present and maximal granularity); (b) crossover is applied to those strings, and therefore it has no problems in application.

Note that the invisible bits do not affect the search process; the only moments when they are important for the search are when they become visible, following an increment of granularity. In this case, the new bits are randomly initialised. The mutation operator is the standard, which negates each bit with probability p_m.

The cooperative local optimising genetic operator is a classical optimisation method for local optimisation of the GA individuals. In our case, local search is performed in two ways: by applying our gradient descent rule-based learning algorithm; and by using the cooperative local optimising operator. The latter is a ternary operator (three parents strings a, b and c) that tries to exploit the fitness landscape identified by the extant solutions. The algorithm for this operator is:

```
• Rank the three parents by fitness value (suppose
  fitness(a)>fitness(b)>fitness(c))
• for each ith bit of the strings
    – if $a_{1i} = b_{1i}$ then
        – $offspring_{1i} = c_{1i}$
      else
        – $offspring_{1i}$ =negate ($c_{1i}$)
```

Thus, we have introduced two elements in the standard GA to help the exploration/exploitation abilities provided by the search space evolution: a coding

```
Parents

10  0   1 01 0 00   0   1 10 0 00
11  0   0   1 011 0 001   1 001 1 011

Data Structure              cutting point

10  0 *** * ***   1 *01 0 *00 | 0 *** * ***   1 *10 0 *00
11  0 *** * ***   0 *** * *** | 1 011 0 001   1 001 1 011

Offspring

10  0   1 01 0 00   1 11 0 01   1 01 1 11
11  0   0   0   1 *10 0 *00
```

Fig. 2. Two individuals with different length undergo crossover (*can be randomly initialised, or they can be zero or one).

Fig. 3. Alphabetic characters.

granularity parameter and a cooperative local optimising genetic operator.

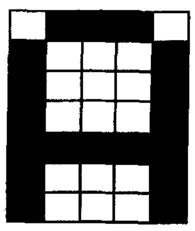
4. PERFORMANCE EVALUATION

We apply the hybrid genetic/random neural network learning algorithm to two pattern recognition problems. The first one recognises or categorises alphabetic characters and the second one recognises geometric figures. We compare the performance of our hybrid learning algorithm (*re_ev*) with the gradient descent learning algorithm (*re_rnn*) [5,15] for different learning sets S_L = {image 1, ... image L} composed of L different images chosen among those of Figs 3 and 8. That is, during testing operations we considered an increasing number of training images (S_1, S_2, ..., S_8) which were chosen from among the eight black and white images of Figs 3 and 8. We have used processors of the SP2-IBM of CeCalCULA (High Performance Computing Center of Venezuela) to test the algorithms.

4.1. Recognition of Alphabetic Characters

The problem that is presented in this section is to recognise or categorise alphabetic characters (Fig. 3). We will input various alphabetic characters to a RNN and train the network to recognise these as separate categories.

Each character is represented by a 5*7 grid of pixels. For example, to represent the letter A we must use the pattern show in Fig. 4. Here the blackened boxes represent value 1, while empty boxes represent a zero. We can represent all characters this way, with a binary map (Y) of 35 pixel values. Thus, we used a single-layer network of n = 35 neurons. The working parameters for the first learning stage are as follows: μ = 0.6, number of iterations = 15, weights error rate = 0.01. The working parameters for the second learning stage are as follows: population size (M) = 10, number of iterations = 20, mutation rate = 0.6, crossover rate = 0.3. The total number of iterations for the learning phase (hybrid algorithm) is 10.



$$\Rightarrow \quad \begin{matrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{matrix}$$

Fig. 4. Representation of the letter A with a 5*7 pattern.

To evaluate the performance of the learning algorithms, we show the minimal errors reached during the learning phase and their execution times (Fig. 5). These values represent the average of 8-(i-1) processes for each set S_i. The learning of the sets S1, S2 and S4 remains good for the hybrid learning algorithm. This algorithm provides a better error convergence of the learning phase. Concerning S7, S8 error costs are important for the gradient descent learning algorithm. In general, the hybrid learning algorithm appears to give the best results, but with a substantially large execution time. That is because the evolutionary algorithm is very slow to converge (Fig. 5).

The system errors (E_k) during the learning phase for a group of six images, using the gradient descent learning algorithm and the hybrid learning algorithm, are shown in Fig. 6. After a total of eight iterations of the learning process, the system error in the hybrid learning algorithm converges to a value of 0.006118. After the same number of iterations of the learning process, the gradient descent learning algorithm converges to a value of 3.183351.

To test heteroassociative memories, we have evaluated the recognition rates of distorted versions of the training patterns. We generated 20 noisy images used as inputs, for each training image and for a given distortion rate. The result of the learning stage is used as the initial neural network of this second phase (retrieval stage). We have corrupted them by reasonable noise rates equal to 0%, 13%, 22% and 30% distortion by modifying bit values at random. A pattern is recognised if the residual error rate is less than 3%. The results we have obtained are presented in Fig. 7.

The progressive retrieval process with an adaptive threshold value that we proposed [5,15] provides satisfactory recognition rates. The performance results obtained are lower when the noise rate is important (memories are then more discriminant). Recognition performances have considerably improved with *re_ev* in comparison with those obtained by *re_rnn*.

4.2. Recognition of Geometric Figures

In this example we recognise six by six binary images of geometric figures (Fig. 8). Thus, we used a single-layer network of n = 36 neurons. The working parameters for the first learning stage are as follows: μ =

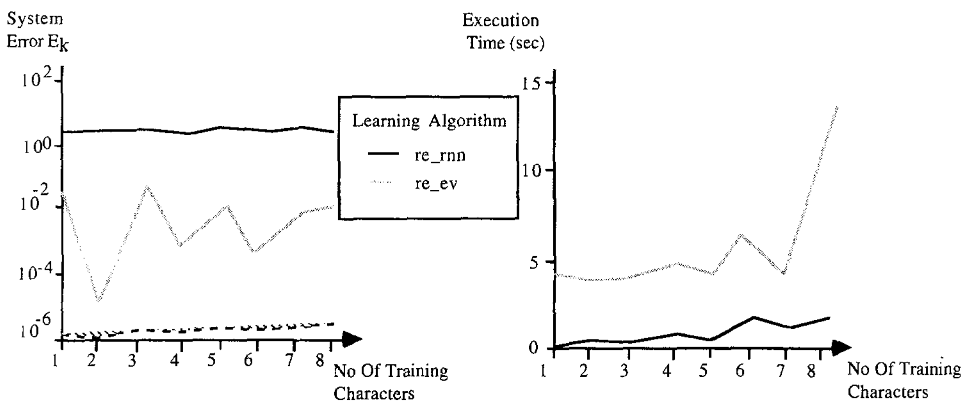


Fig. 5. System error and execution time of the learning algorithms for a different set of images.

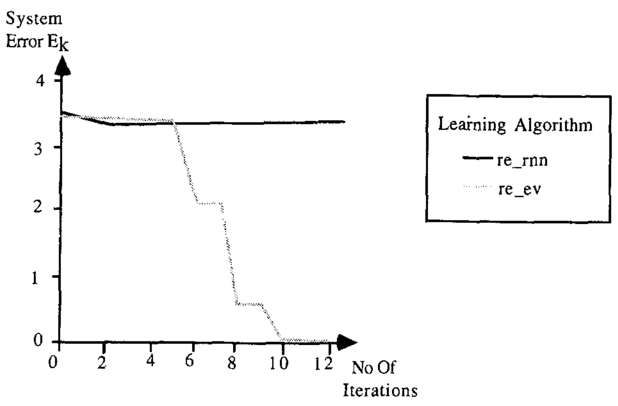


Fig. 6. System error for a set of six alphabetic characters.

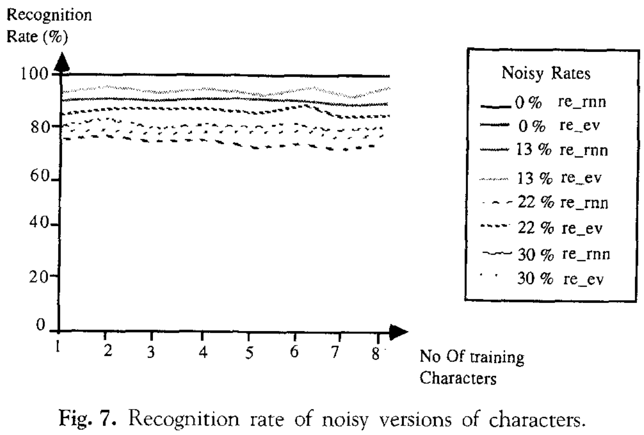


Fig. 7. Recognition rate of noisy versions of characters.

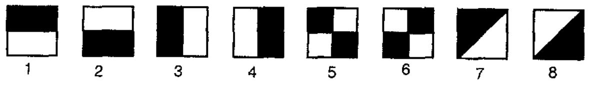


Fig. 8. Geometric figures.

0.8, number of iterations = 10, weights error rate = 0.01. The working parameters for the second learning stage are as follows: population size (M) = 10, number of iterations = 20, mutation rate = 0.6, crossover rate = 0.3. The total number of iterations for the learning phase (hybrid algorithm) is 8.

Figure 9 shows the minimal errors reached during the learning phase and their execution times. The learning of the sets S2, S3, S5, S6, S7 remains good for the hybrid learning algorithm. In general, the hybrid learning algorithm gives the best results, but with a large execution time.

To evaluate the exact recognition rates, we follow the same procedure proposed in the previous example. In general, *rec_ev* appears to give the best recognition performances (Fig. 10).

5. CONCLUSIONS

This paper presents an evolution learning algorithm that optimises a neural network on the basis of its topology and its weights distribution. This algorithm is based on the gradient descent rule-based learning algorithm of the RNN and on GA. Different length strings (individuals) are used in connection with codifying a control parameter, the coding granularity. A cooperative locally optimising operator was also used.

We have shown that this model can efficiently work as associative memory. We can recognise pattern arbitrary images with this algorithm, but the processing time increases rapidly. With this approach, the following advantages, scope and limitations are observed:

(a) The results of neural network learning are sensitive to the initial topology (connections and weights). A GA is employed to perform a global search, and to seek a good starting topology for

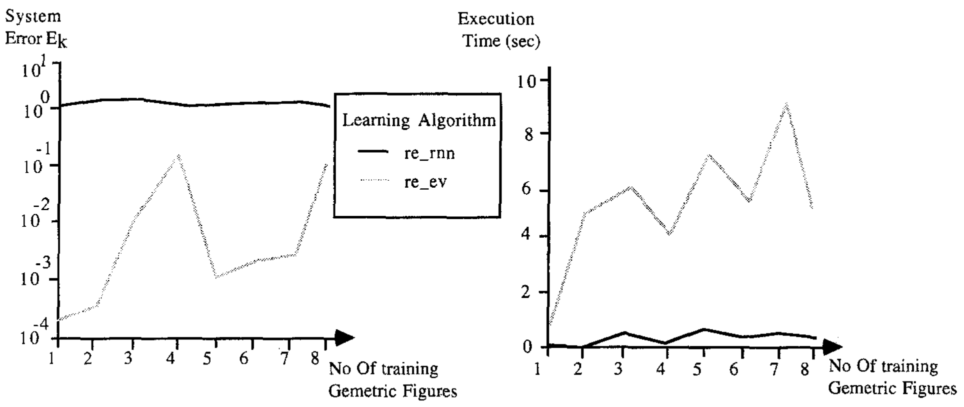


Fig. 9. System error and execution time of the learning algorithms for different set of images.

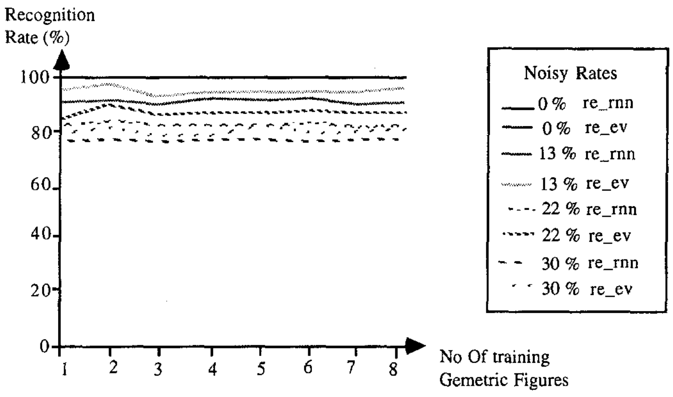


Fig. 10. Recognition rate of noisy versions of characters.

the neural network learning algorithm. The result is an improvement in the error minimisation of the algorithm.

(b) Our hybrid algorithm requires a substantial computing processing time in order to converge to an acceptably small system error value. That is, because during the first learning phase, which mainly consists of the minimisation of a quadratic error function, we have met the classical problem of the large execution time to converge to a minimal error for every RNN; and during the second learning phase, the evolutionary algorithm is very slow to converge, because generation calculations take a relatively large execution time. In this case, it is necessary to determine the better combination of genetics operators so as to decrease the number of necessary generations to reach suboptimal solutions.

(c) In the hybrid algorithm, the problem of entrapment in a local minimum encountered in gradient descent-based learning algorithms is circumvented by using a GA which is guided by the fitness function of a population. The simultaneous structural and parametric modifications based on fitness allows the algorithm to discover appropriate networks quickly, and investigate several differing architectures in parallel while avoiding over-commitment to a particular network topology.

(d) Complete network induction is approached with respect to the complex interaction between network topology, parametric values and task performance. By fixing topology, gradient descent methods can be used to discover appropriate solutions. But the relationship between network structure and task performance is not well understood, and there is no backpropagation through the space of network architectures. Instead, the network induction problem is approached with heuristics that often restrict the available architectures, the dynamics of the search mechanisms, or both. Our hybrid algorithm strikes a middle ground between these two extremes, allowing the network's complexity and behaviour to emerge in response to the requirements of the task.

(e) Most of the computations are intrinsically parallel. That is, this hybrid algorithm is easy to implement on a parallel machine, which will reduce the execution time and improve the results. By developing an efficient parallel version of our algorithm, we can increase the computational speedup.

(f) In general, the results of the hybrid learning algorithm are better than the results obtained for the algorithm presented earlier [5], but with a substantially large execution time. The number of iterations of our approach to obtain a minimal system error is smaller than the number of iterations for the approaches presented earlier [3,5].

(g) Future work will examine other recognition problems where the number of patterns to be stored is orders of magnitudes higher than the problems discussed in this work (voluminous data). Prior to that, we will develop parallel versions of our approach.

References

1. Chen D, Giles C, Sun S, Chen H, Less Y, Goudreau M. Constructive learning of recurrent neural network. Proceedings IEEE International Conference Neural Networks 1993; 1196–1201
2. Maniezzo V. Genetic evolution of the topology and weight distribution of neural networks. IEEE Transactions Neural Networks 1994; 5(1): 39–53
3. Hung S, Adeli H. A parallel genetic/neural network learning algorithm for MIMD shared memory machines. IEEE Transactions Neural Networks 1994; 5(6): 900–909
4. Gelenbe E. Learning in the recurrent Random Neural Network. Neural Computation 1993; 5(5): 584–596
5. Aguilar J. A recognition algorithm using the Random Neural Network. Proceedings 3rd International Congress on Computer Science Research 1996; 16–20
6. Fogel D, Fogel L, Porto V. Evolving neural networks. Biological Cybernetics 1990; 63: 487–493
7. Angeline P, Saunders G, Pollack J. An evolutionary algorithm that constructs recurrent neural network. IEEE Transactions Neural Network 1994; 5(1): 54–64
8. Aguilar J. Evolutionary learning on Recurrent Random Neural Network. Proceedings World Congress on Neural Networks 1995; 232–236
9. Gelenbe E. Random neural networks with positive and negative signals and product form solution. Neural Computation 1989; 1(4): 502–510
10. Gelenbe E. Stable random neural networks. Neural Computation 1990; 2(2): 239–247
11. Gelenbe E, Stafylopatis A, Likas A. Associative memory operation of the Random Network Model. Proceedings International Conference Artificial Neural Networks (ICANN 91) 1991; 307–315
12. Aguilar J. Using the general energy function of the Random Neural Networks to solve the graph partitioning problem. Proceedings IEEE International Conference Neural Networks 1996; 2130–2135
13. Aguilar J. A general method to solve combinatorial optimization problems with the Random Neural Networks. In: Cerrolaza M, Gajardo C, Brebbia C. (eds), Numerical Methods in Engineering Simulation. Computational Mechanics Publications, 1996; 349–356
14. Aguilar J. An energy function for the Random Neural Networks. Neural Processing Letters 1996; 4: 17–27
15. Aguilar J, Colmenares A. Recognition algorithm using evolutionary learning on the Random Neural Networks. Proceedings IEEE International Conference Neural Networks 1997; 1023–1028
16. Aguilar J. An approach for combinatorial optimization problem based on learning in the recurrent random neural network. Proceedings World Congress on Neural Networks 1994; 420–425
17. Mulhenbein H, Georges-Schleuter M, Kramer O. Evolution algorithms in combinatorial optimization. Parallel Computing 1988; 7(1): 65–88
18. Golberg D. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989

Jose Aguilar was born in Valera, Venezuela, in 1964. He received the computer science engineering degree from the University de los Andes, Mérida, Venezuela, in 1987. He received his Masters in computer science (DEA) from the University Paul Sabatier, Toulouse, France, in 1991. He received his PhD from the University Rene Descartes, Paris, France, in 1995. Dr Aguilar is a Professor of Computer Science at the University de los Andes. His research interests include parallel processing, evolutionary techniques, artificial life and artificial neural networks. Dr Aguilar is a member of the Venezuelan Association for Artificial Intelligence and the ACM society.

Adriana Colmenares was born in Maturin, Venezuela, in 1969. He received the computer science engineering degree from the University de los Andes, Mérida, Venezuela, in 1996. She currently works at Honeywell de Venezuela. Her current research interests are in the fields of genetic algorithms, artificial neural networks and automatic control.

Correspondence and offprint requests to: J. Aguilar, CEMISID, Dpto. de Computación, Facultad de Ingeniería, Universidad de los Andes, Av. Tulio Febres, Mérida, Venezuela