Neural Processing Letters **13:** 81–91, 2001.

Learning Algorithm and Retrieval Process for the Multiple Classes Random Neural Network Model

JOSE AGUILAR
CEMISID. Dpto. de Computación, Facultad de Ingeniería, Universidad de los Andes, Av. Tulio Febres. Mérida-Venezuela; e-mail: aguilar@ing.ula.ve

Abstract. Gelenbe has modeled neural networks using an analogy with queuing theory. This model (called Random Neural Network) calculates the probability of activation of the neurons in the network. Recently, Fourneau and Gelenbe have proposed an extension of this model, called multiple classes random neural network model. The purpose of this paper is to describe the use of the multiple classes random neural network model to learn patterns having different colors. We propose a learning algorithm for the recognition of color patterns based upon non-linear equations of the multiple classes random neural network model using gradient descent of a quadratic error function. In addition, we propose a progressive retrieval process with adaptive threshold values. The experimental evaluation shows that the learning algorithm provides good results.

Key words: color pattern recognition, learning algorithm, multiple classes random neural network, retrieval process.

1. Introduction

Coming up with effective learning algorithms for recurrent networks is a major objective in neural network theory [7]. There are numerous examples where recurrent networks constitute a natural approach to problems: image processing, pattern analysis and recognition, etc., where local interactions between picture elements lead to mutual interactions between neighboring neurons which are naturally represented by recurrent networks. In such cases, it is clear that effective learning algorithms enhance the value of the neural network methodology.

The Random Neural Network (RNN) has been proposed by Gelenbe in 1989 [8–11]. This model does not use a dynamic equation, but uses 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 that mimic what is known of inter-neural signals in biophysical neural networks. The RNN has been used to solve optimization [1, 2, 4] and pattern recognition problems [3, 5, 6]. Gelenbe has considered a learning algorithm for the recurrent RNN model [11]. We have proposed modifications of this algorithm for combinatorial optimization problems [3, 4] and evolutionary learning for combinatorial optimization and recognition problems [1, 5]. Fourneau and Gelenbe have proposed an extension of the RNN, called multiple classes RNN model [12].

In this paper, we propose a learning algorithm and a retrieval procedure for the multiple classes RNN on color pattern recognition problems. We shall use each class to model a color. We present a backpropagation type learning algorithm for the recurrent multiple classes RNN model using gradient descent of a quadratic error function when a set of input–output pairs is presented to the network. Our model is defined for nC parameters for the whole network, where C is the number of primary colors, n is the number of pixels of the image, and each neuron is used to obtain the color value of each pixel in the bit map plane. Combinations of primary colors create different colors according to the RGB model. Thus, our learning algorithm requires the solution of a system of nC non-linear equations each time the n-neuron network learns a new input–output pair (n-pixel image with C primary colors). In addition, we propose a progressive retrieval process with adaptive threshold values. This work is organized as follows: in Section 2 we present the multiple classes RNN. Sections 3 and 4 present our learning algorithm and retrieval process for multiple classes RNN. In Section 5, we present applications. Remarks concerning future work and conclusions are provided in Section 6.

2. The Multiple Classes Random Network Model

We describe the multiple classes random network model introduced in [12]. The network is composed of n neurons and receives exogenous positive (excitatory) and negative (inhibitory) signals as well as endogenous signals exchanged by the neurons. As in the classical model, excitatory and inhibitory signals are sent by neurons, when they fire, to other neurons in the network or to outside world. In this model, positive signals may belong to several classes and the potential at a neuron is represented by the vector $K_i = (K_{i1}, \ldots, K_{iC})$, where K_{ic} is the value of the 'class c potential' of neuron i, or its 'excitation level in terms of class c signals', and negative signals only belong to a single class. The total potential of neuron i is $K_i = \Sigma_{c=1}{}^{C} K_{ic}$. The arrival of an excitatory signal of some class increases the corresponding potential of a neuron by 1, while an inhibitory signal's arrival decreases it by 1. That is, when a positive signal of class c arrives at a neuron, it merely increases K_{ic} by 1, and when a negative signal arrives at it, if $Ki > 0$, the potential is reduced by 1, and the class of the potential to be reduced is chosen randomly with probability Kic/Ki for any $c = 1, \ldots, C$. A negative signal arriving at a neuron whose potential is zero has no effect on its potential.

Exogenous positive signals of class c arrive at neuron i in a Poisson stream of rate $\Lambda(i, c)$, while exogenous negative signals arrive at it according to a Poisson process of rate $\lambda(i)$. A neuron is excited if its potential is positive. It then fires at exponentially distributed intervals, so it sends excitatory signals of different classes or inhibitory signals to other neurons or to the outside of the network. That is, the neuron i can fire when its potential is positive ($Ki > 0$). The neuron i sends excitatory signals of class c at rate $r(i, c) > 0$, with probability Kic/Ki. When the neuron fires at rate $r(i, c)$, it deletes by 1 its class c potential and sends to neuron j a class φ positive

signal with probability $p^+(i, c; j, \varphi)$, or a negative signal with probability $p^-(i, c; j)$. On the other hand, the probability that the deleted signal is sent out of the network, or that it is 'lost', is $d(i, c)$. Thus,

$$\Sigma_{(j,\varphi)}\, p^+(i, c; j, \varphi) + \Sigma_j\, p^-(i, c; j) + d(i, c) = 1 \qquad \text{for } \forall\ i = 1, n \text{ and } c = 1, C,$$

The complete state of the network is represented by the vector (of vectors) $\underline{K}(K_1, \ldots, \underline{K}_n)$, and we shall denote by $p(\underline{K}, t) = P[\underline{K}(t) = \underline{K}]$ the probability distribution of its state. Let $\underline{K}(t)$ be the vector representing the state of the neural network at time t, and $q(i, c)$ and $0 < q(i, c) < 1$ be the solution of the system of non-linear equations:

$$q(i, c) = \lambda^+(i, c)/(r(i, c) + \lambda^-(i)), \tag{1}$$

where $\lambda^+(i, c) = \Sigma_{(j,\varphi)} q(j, \varphi) r(j, \varphi) p^+(j, \varphi; i, c) + \Lambda(i, c)$

$$\lambda^-(i) = \Sigma_{(j,\varphi)} q(j, \varphi) r(j, \varphi) p^-(j, \varphi; i) + \lambda(i).$$

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

$$w^+(j, \varphi; i, c) = r(j, \varphi) p^+(j, \varphi; i, c) \qquad w^-(j, \varphi; i) = r(j, \varphi) p^-(j, \varphi; i)$$

and

$$r(j, \varphi) = [\Sigma_{(i,c)} w^+(j, \varphi; i, c) + \Sigma_{(i,c)} w^-(j, \varphi; i)].$$

3. Learning Algorithm

Now, we define a learning algorithm for the multiple classes RNN model [11]. We assume values equal to 0, 0.5 or 1, for each class on every neuron. We propose a gradient descent algorithm for choosing the set of network parameters $w^+(j, z; i, c)$ and $w^-(j, z; i)$ in order to learn a given set of m input–output pairs (X, Y) where the set of successive inputs is denoted by:

$$X = \{X_1, \ldots, X_m\} \qquad \text{where } X_k = \{X_k(1, 1), \ldots, X_k(n, C)\},$$
$$\text{and } X_k(i, c) \text{ is the } c\text{th class on the neuron } i$$
$$\text{for the pattern } k$$

$$X_k(i, c) = \{\Lambda_k(i, c),\ \lambda_k(i)\}$$

and the successive desired outputs are the vector

$$Y = \{Y_1, \ldots, Y_m\} \qquad \text{where } Y_k = \{Y_k(1, 1), \ldots, Y_k(n, C)\},$$
$$\text{and } Y_k(1, 1) = \{0, 0.5, 1\}.$$

The values $\Lambda_k(i, c)$ and $\Lambda_k(i)$ provide network stability. In our model, $\Lambda_k(i, c)$ and

$\Lambda_k(i)$ are initialized as they have been defined previously. Normally, arrival rates of exogenous signals are chosen as follows:

$$Y_k(i, c) > 0 \Rightarrow X_k(i, c) = (\Lambda_k(i, c), \lambda_k(i)) = (Kc, 0),$$
$$Y_{ik}(i, c) > 0 \Rightarrow (\Lambda_k(i, c), \lambda_k(i)) = (0, 0).$$

The network approximates the set of desired output vectors in a manner that minimizes a cost function E_k:

$$E_k = 1/2 \quad \Sigma_{i=1}^{n} \Sigma_{c=1}^{C} [q_k(i, c) - Y_k(i, c)]^2.$$

The rule to update the weights may be written as:

$$\begin{aligned} w_k^+(u, p; v, z) &= w_{k-1}^+(u, p; v, c) - \mu \Sigma_{i=1}^{n} \Sigma_{c=1}^{C} (q_k(i, c) \\ &\quad - y_k(i, c))[\delta q(i, c)/\delta w^+(u, p; v, z)]_k \\ w_k^-(u, p; v) &= w_{k-1}^-(u, p; v) - \mu \Sigma_{i=1}^{n} \Sigma_{c=1}^{C} (q_k(i, c) \\ &\quad - y_k(i, c))[\delta q(i, c)/\delta w^-(u, p; v)]_k, \end{aligned} \tag{2}$$

where $\mu > 0$ is the learning rate (some constant).

$q_k(i)$ is calculated using X_k, $w_k^+(u, p; v, z) = w_{k-1}^+(u, p; v, z)$
and $w_k^-(u, p; v) = w_{k-1}^-(u, p; v)$ in (1)
$[\delta q(i, c)/\delta w^+(u, p; v, z)]_k$ and $[\delta q(i, c)/\delta w^-(u, p; v)]_k$ are evaluated using the values $q(i, c) = q_k(i, c)$, $w_k^+(u, p; v, z) = w_{k-1}^+(u, p; v, z)$
and $w_k^-(u, p; v) = w_{k-1}^-(u, p; v)$ in (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 (2).*
- *For each successive value of m:*
  Set the input-output pair (X_k, Y_k)
  Repeat
    Solve Equation (1) with these values
    Using (2) and the previous results
    update the matrices W_k^+ and W_k^-

  Until the change in the new values of the weights is smaller than some predetermined value.

4. Retrieval Procedure

Once the learning phase is completed, the network must perform as well as possible the completion of noisy versions of the training vectors. In this case, we propose a progressive retrieval process with adaptive threshold values. Let $X' =$

$\{X'(1,1), \ldots, X'(n,C)\}$ be any input vector with values equal to 0, 0.5 or 1, for each $X'(i,c), i = 1, \ldots, n$ and $c = 1, \ldots, C$. In order to determine the corresponding output vector $Y = \{Y(1,1), \ldots, Y(n,C)\}$, we first compute the vector of probabilities $Q = (q(1,1), \ldots, q(n,C))$. We consider that $q(i,c)$ whose values are between $1 - T < q(i,c) < T/2$ or $1 - T/2 < q(i,c) < T$, for $T = 0.8$, belong to the uncertainty interval Z. When the network stabilizes in an attractor state, the number of neurons (NB_Z) whose $q(i,c) \in Z$ is equal to 0. Hence, we first treat the neurons whose state is considered certain to obtain the output vector $Y^{(1)} = (Y^{(1)}_{(1,1)}, \ldots, Y^{(1)}_{(n,C)})$:

$$Y^{(1)}_{(i,c)} = Fz(q(i,c)) = \begin{cases} 1 & \text{if } q(i,c) > T \\ 0 & \text{if } q(i,c) < 1 - T \\ 0.5 & \text{if } T/2 \leqslant q(i,c) \leqslant 1 - T/2 \\ X'(i,c) & \text{otherwise} \end{cases}$$

where Fz 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_β as follows:

$$Y(i,c) = f_\beta(q(i,c)) = \begin{cases} 1 & \text{if } q(i,c) > \beta \\ 0.5 & \text{if } \beta/2 < q(i,c) < \beta \\ 0 & \text{otherwise} \end{cases}$$

where β is the selected threshold. Each value $q(i,c) \in Z$ is considered as a potential threshold. That is, for each $q(i,c) \in Z$:

$$\beta = \begin{cases} q(i,c) & \text{if } q(i,c) > 0.666 \\ 1 - q(i,c) & \text{otherwise} \end{cases}$$

Eventually, Z can be reduced by decreasing T (for $T > 0.666$). For each potential value of β, we present to the network the vector $X'^{(1)}(\beta) = f_\beta(Q)$. Then, we compute the new vector of probabilities $Q^{(1)}(\beta)$ and the output vector $Y^{(2)}(\beta) = Fz(Q^{(1)}(\beta))$. We keep the cases where NB_Z $= 0$ and $X'^{(1)}(\beta) = Y^{(2)}(\beta)$. If these two conditions are never satisfied, the initial X' is considered too much different from any training vector. If several thresholds are candidates, we choose the one which provides the minimal error (difference between $q(i,c)$ and $Y(i,c)$, for $i = 1, n$ and $c = 1, \ldots C$):

$$E(\beta) = 1/2 \quad \Sigma^n_{i=1}[q(i,c)^{(1)}(\beta) - Y^{(1)}_{i,c}(\alpha)]^2.$$

5. Experimental Results

We show how the multiple classes RNN can be used to solve the Color Pattern Recognition problem. A 'signal class' represents each color. The recognition procedure is based on an associative memory technique [3]. To design such a

memory, we have used a single-layer RNN of n fully interconnected neurons. For every neuron i the probability that emitting signals depart from the network is $d(i, c) = 0$. We assume a pattern composed of n points (j, k) in the plane (for $j = 1, \ldots, J$ and $k = 1, \ldots, K$). We associate a neuron $N(i)$ to each point (j, k) in the plane (for $i = 1, \ldots, n$; $j = 1, \ldots, J$ and $k = 1, \ldots, K$), such as $n = J^*K$. The state of $N(i)$ can be interpreted as the color intensity value of the pixel (j, k). That is, each pixel is represented by a neuron. On the other hand, we assume three classes to represent the primary colors (red, green, and blue) according to the RGB model. This model allows us to create different colors with the combination of different intensities of the primary colors. For example, to represent a pixel with red color the neuron value is (1, 0, 0), the black color is (1, 1, 1), the pink color is (0.5, 0, 0), and so forth. We assume values equal to 0, 0.5 and 1 for each class on every neuron. In this way, we can represent geometric figures with different combinations of colors. The parameters of the neural network will be chosen as follows:

(a) $p^+(j, \varphi; i, c) = p^+(i, c; j, \varphi)$, $p^-(i, c; j) = p^-(j, c; i)$ for any $i, j = 1, \ldots, n$ and $c, \varphi = 1, \ldots, C$.
(b) $\Lambda(i, c) = Lic$ and $\lambda(i) = 0$, where Lic is a constant for the class c of the neuron i.

5.1. PROBLEM DEFINITION

In this section, we present several examples to compare the quality of our learning algorithm for different pattern types. We will input various geometric figures to a Multiple Classes RNN and train the network to recognize these as separate categories. We test three types of examples: the first and second groups represent color patterns and the last group binary images (black and white). That is, to evaluate our learning algorithm, we use three figure groups: the first set of figures (group A) is composed by the figures shown in Figure 1, where black boxes represent red colors and white boxes represent blue colors. For the second set (group B), we use the set of figures shown in Figure 2, where black boxes represent blue colors, gray boxes represent green colors and white boxes represent red colors. The last group (group C) is composed of the same black and white figures used in [3] to compare our learning approach with the results obtained with the learning algorithm based on RNN proposed in [3], and the evolutionary learning approach proposed in [5]. Each pixel is represented by a neuron and we assume three classes to represent the primary colors (red, green, and blue) according to the RGB model.

Each figure is represented by a 6*6 grid of pixels. For example, to represent the seventh geometric figure of Figure 1 as a black and white figure, we must use the pattern shown in Figure 3. According to the RGB model, the black boxes are represented as (1,1,1), while white boxes are represented as (0, 0, 0). In this way, we can represent geometric figures with different combinations of colors (for example, in Figure 3, if we suppose black boxes correspond to red colors, and white boxes to blue colors, neurons for black boxes are equal to (1, 0, 0) and for white

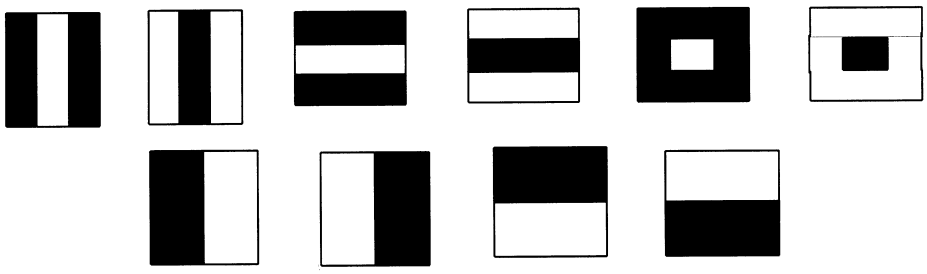

Figure 1. Geometric Figures with two colors (Group A).

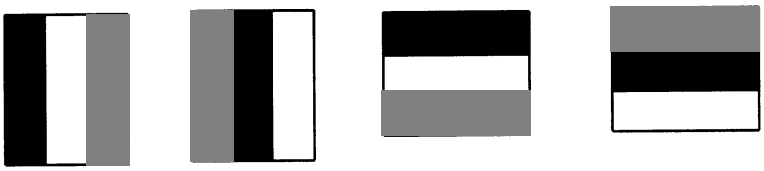

Figure 2. Geometric Figures with three colors (Group B).

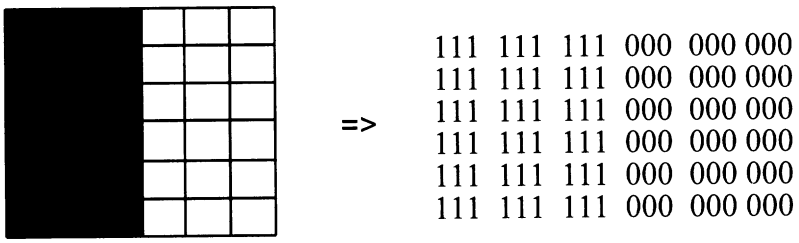

Figure 3. Representation of a geometric figure with a 6*6 pattern.

boxes to (0, 0, 1)). Thus, we use a single-layer multiple classes RNN composed of 36 neurons ($n = 36$) and 3 classes ($C = 3$).

5.2. RESULTS ANALYSIS

The results for the first group are presented in Figure 4. To evaluate the performance of the learning algorithms, we show the minimal errors reached during the learning phase and their execution times (Figure 4). These values represent the average of 8 processes for each set S_i of images. This algorithm provides a good error convergence for the learning phase. Particularly, the learning of the sets S_4 and S_6 remains good for our learning algorithm. Concerning S_2 and S_8, the error cost increases.

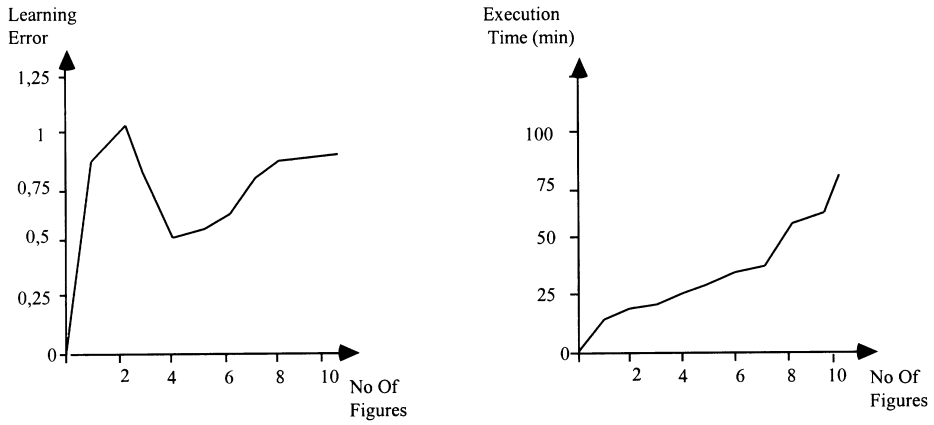

Figure 4. Learning error and execution time of the learning algorithms for Group A.

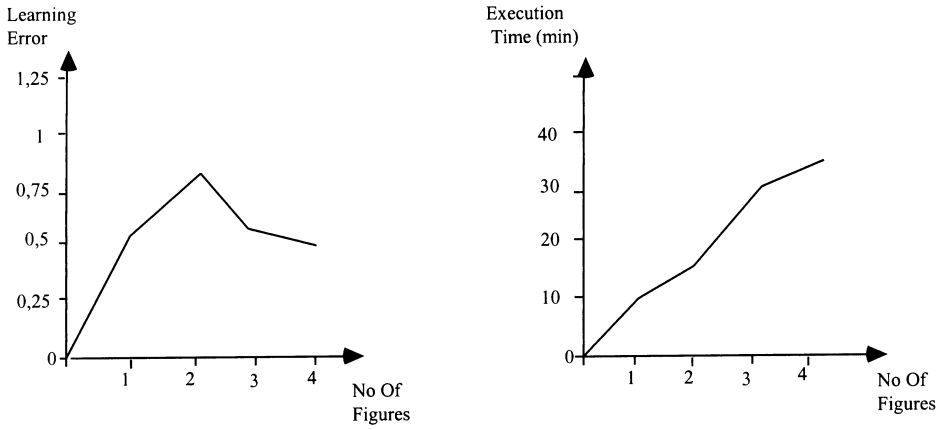

Figure 5. Learning error and execution time of the learning algorithms for Group B.

The results for the group B are presented in Figure 5. The learning of the sets S1 and S4 remains good for our learning algorithm. For the group B, we obtain a similar error convergence and execution time as for the previous group. That is, the number of colors does not imply more learning error.

Figure 6 shows the system errors during the learning phase for the last group of images (Group C), using the classical gradient decent learning algorithm (*Cl*), the evolutionary learning algorithm (*Evol*) and our Multiple Classes learning algorithm (*Mult*). In general, *Evol* appears to give the best results, but with a substantially larger execution time. That is because *Evol* is very slow to converge. The learning error remains good for our learning algorithm (*Mult*). This algorithm provides a better error convergence of the learning phase than *Cl*. Concerning *Cl*, error costs are important.

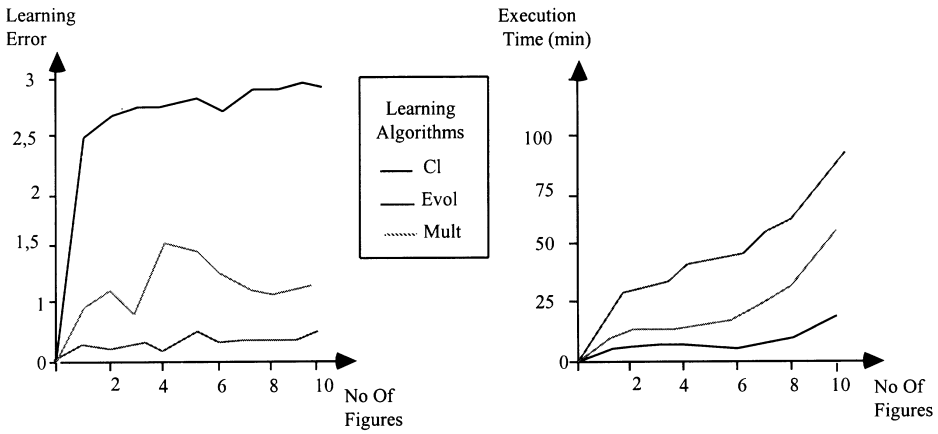

Figure 6. Learning error and execution time of the learning algorithms for Group C.

In order to test associative memories, we have evaluated recognition rates of distorted versions of the training patterns (Tables I and II). These values represent the average of 8 processes for each set S_i of images. 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 according to reasonable noise rates equal to 0%, 10%, 20% and 30%. They are distorted by modifying bit values randomly. A pattern is recognized if the residual error rate is less than 3. The performance results obtained are lower when the noise rate is large (memories are then more discriminating). The results for the first and second groups are presented in Table I. Our algorithm provides a good recognition rate. Particularly, the recognition rate for the sets S_4 and S_6 remains good for our approach. For S_{10} and 30% of noise, the recognition rate decreases.

Table II shows the recognition rate for the last group of images (Group C), using the classical gradient recognition algorithm (*Cl*), the hybrid Genetic/RNN learning algorithm (*Evol*) and our Multiple Classes learning algorithm (*Mult*). In general, *Evol* continues to give the best results. That is because *Evol* has a small learning error. The recognition rate remains good for our algorithm (*Mult*). This algorithm provides a better recognition rate than *Cl*. The recognition rate of *Cl* is bad.

6. Conclusions

In this paper, we have propose a learning algorithm and a retrieval algorithm based on the Multiple Classes Random Neural Model. We have shown that our approach can work efficiently as associative memory. We can learn arbitrary color images with this algorithm, but the processing time will increase rapidly according to

Table I. Recognition rate of noisy versions of groups A and B

Noisy Rate	0%			10%			20%			30%		
Number of Figures	4	6	10	4	6	10	4	6	10	4	6	10
Group A	99%	99%	97%	94%	94%	90%	84%	84%	81%	73%	71%	70%
Group B	99%	99%	97%	94%	93%	89%	83%	82%	81%	73%	71%	67%

Table II. Recognition rate of noisy versions of group C

Noisy Rate	0%			10%			20%			30%		
Number of Figures	4	6	10	4	6	10	4	6	10	4	6	10
Cl	99%	95%	96%	93%	91%	85%	83%	80%	77%	68%	66%	62%
Evol	99%	99%	99%	96%	95%	92%	87%	85%	81%	75%	74%	72%
Mult	99%	99%	99%	95%	94%	90%	85%	84%	81%	73%	72%	70%

the number of pixels and colors used. The number of neurons is dictated by the image resolution (in our case, we test for 6*6 pixels). During the learning phase, we have encountered classical problems like the existence of local minimal and large learning times. At the level of the retrieval algorithm, we have obtained good performance but with a large execution time. However, most of the computations are intrinsically parallel and can be implemented on SIMD or MIMD architectures.

Acknowledgements

This work was partially supported by CONICIT grant AP-97003817, CDCHT-ULA grant I-621-98-02-A and CeCalCULA (High Performance Computing Center of Venezuela).

References

1. Aguilar, J.: Evolutionary Learning on Recurrent Random Neural Network, In: *Proc. of the World Congress on Neural Networks*, International Neural Network Society (1995), pp. 232–236.
2. Aguilar, J.: An Energy Function for the Random Neural Networks, *Neural Processing Letters* **4** (1996), 17–27.
3. Aguilar, J.: A Recognition Algorithm using the Random Neural Network, In: *Proc. of the 3rd. International Congress on Computer Science Research* (1996), pp. 15–22.
4. Aguilar, J.: Definition of an Energy Function for the Random Neural to Solve Optimization Problems, *Neural Networks* **11**(4) (1998), 731–738.
5. Aguilar, J. and Colmenares, A.: Resolution of Pattern Recognition Problems Using a Hybrid Genetic/Random Neural Network Learning Algorithm, *Pattern Analysis and Applications* **1**(1) (1998), 52–61.
6. Atalay, V., Gelenbe, E. and Yalabik, N.: The Random Neural Network Model for Texture Generation, *Intl. J. Pattern Recognition and Artificial Intelligence* **6**(1) (1992), 131–141.
7. Chen, D., Giles, C., Sun, S., Chen, H., Less, Y. and Goudreau, M.: Constructive Learning of Recurrent Neural Network, In: *Proc. of the IEEE International Conference on Neural Networks* (1993).
8. Gelenbe, E.: Random Neural Networks with Positive and Negative Signals and Product Form Solution, *Neural Computation* **1**(4) (1989), 502–511.
9. Gelenbe, E.: Stability of the Random Neural Networks, *Neural Computation* **2**(2) (1990), 239–247.
10. Gelenbe, E.: Theory of the Random Neural Network Model, In: E. Gelenbe (ed.), *Neural Networks: Advances and Applications*, North-Holland, Pays-Bas, (1991).
11. Gelenbe, E.: Learning in the Recurrent Random Neural Network, *Neural Computation* **5**(5) (1993).
12. Fourneau, M., Gelenbe, E. and Suros, R.: G-networks with Multiple Classes of Negative and Positive Customers, *Theoretical Computer Science* **155** (1996), 141–156.