Recognition Algorithm using Evolutionary Learning on the Random Neural Networks

Jose Aguilar & Adriana Colmenares
CEMISID. Dpto. de Computación. Facultad de Ingeniería
Universidad de los Andes. Av. Tulio Febres. 5101 Mérida-Venezuela
aguilar@ing.ula.ve

Abstract

Gelenbe has modeled the neural network using an analogy with queuing theory. This model (called Random Neural Network) calculates the probability of activation of the neurons in the network. Recently, we have proposed a recognition algorithm based on the Random Neural Network. In this paper, we propose to solve the patterns recognition problem using an Evolutionary Learning on the Random Neural Network. The Evolutionary Learning is based in a hybrid algorithm that trains the Random Neural Network by integrating a Genetic Algorithm with the gradient descend rule based learning algorithm of the Random Neural Network. This hybrid learning algorithm optimizes the Random Neural Network on the basis of its topology and its weights distribution.

1. Introduction

How to improve the learning performance of ANN is currently an important research problem. One approach, is the development of learning algorithms on general-purpose parallel computers with the objective of reducing the overall computing time [6]. Another approach, is the development of more effective neural network learning algorithms with the objective of reducing the learning time [4]. A third approach, is the development of hybrid learning algorithms by integrating Genetic Algorithm (GA) with neural network learning algorithms [5, 6, 7, 8].

In this paper, a new algorithm is present for training Random Neural Network (RNN) by integrating a GA with the gradient descend rule based learning algorithm of the RNN. The RNN has been proposed by Gelenbe in 1989 [1]. This model not uses a dynamic equation, but uses a scheme of interaction among neurons. We introduce two elements in the standard GA to help the exploration/exploitation abilities provided by the search space evolution: a cooperative local optimizing genetic operator and a coding granularity parameter to use different length individuals. This hybrid learning algorithm optimizes the RNN on the basis of its topology and its weights distribution. This work is organized 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 an application. Conclusions are provided in section 5.

2. Random Neural Networks

2.1. Model

The Random Neural Model consists of a network of n neurons in which positive and negative signals circulate. Each neuron accumulates signals as they arrive, and can fire if its total signal count at a given instant of time is positive. 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 outside of the network. Each neuron i of the network is represented at any time t by its input signal potential $k_i(t)$. A negative signal reduces by 1 the potential of the neuron to which it arrives (inhibition) or has no effect on the signal potential if it is already zero; while an arriving positive signal adds 1 to the neuron potential (excitation).

A signal which leaves neuron i heads for neuron j with probability $p^+(i,j)$ as a positive signal, or as negative signal with probability $p^-(i,j)$, or it departs from the network with probability d(i). External positive signals arrive to the i^{th} neuron according to a Poisson process of rate $\Lambda(i)$. External negative signals arrive to the i^{th} neuron according to a Poisson process of rate $\lambda(i)$. The main property of this model is the excitation probability of a neuron i, q(i):

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

where, $\lambda^+(i) = \sum^n_{j=1} q(j)r(j)p^+(j,i)+\Lambda(i)$

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

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

$$w^+(i,j) = r(i)p^+(i,j)$$
$$w^-(i,j) = r(i)p^-(i,j)$$
and, $$r(i) = \Sigma^n_{j=1} [w^+(i,j) + w^-(i,j)]$$

2.2. Recognition Procedure

The recognition procedure is based on a heteroassociative memory technique [4, 12]. This technique involves the use of supervised learning. To design such a memory, we have used a single-layer Random Neural Network of *n* fully interconnected neurons. For every neuron *i* the probability that emitting signals depart from the network is d(i)=0.

2.2.1 Learning phase. Weights and firing rates are determined during the learning phase. Gelenbe has proposed an algorithm 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) [4]:

$$X = \{X_1, ..., X_m\}$$
where, $$X_k = \{\Lambda_k, \lambda_k\}$$
and, $$\Lambda_k = (\Lambda_k(1), ..., \Lambda_k(n))$$
$$\lambda_k = (\lambda_k(1), ..., \lambda_k(n))$$

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

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

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^-. The rule to update the weights is:

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

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 (1)
$[\partial q(i) / \partial w(u,v)]_k$ is evaluated of the values
$q(i) = q_k(i)$ and $w(u,v) = w_{k-1}(u,v)$ in (2)

In our problem, we can translate each input vector X_k in terms of arrival rates of exogenous signals as follows:

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

Where the values Λ and λ provide the network stability for every case. Thus, the network stability condition is:

$$\text{if } y_{ik}=1 \rightarrow \Lambda < r_0(i) + \Sigma^n_{j=1} w_0^-(i,j) - \Sigma^n_{j=1} w_0^+(i,j)$$
$$\text{if } y_{ik}=0 \rightarrow \lambda > \Sigma^n_{j=1} w_0^+(i,j) - r_0(i) - \Sigma^n_{j=1} w_0^-(i,j)$$

Let $\Lambda_k = \min_{i,\, y_{ik}=1} [r_0(i)+\Sigma^n_{j=1} w_0^-(i,j)-\Sigma^n_{j=1} w_0^+(i,j)]$
$\lambda_k = \max_{i,\, y_{ik}=0} [\Sigma^n_{j=1} w_0^+(i,j)-r_0(i)-\Sigma^n_{j=1} w_0^-(i,j)]$

The choice of initial weights may influence the convergence of the learning process. W_0^- is initialized with small positive random variables, uniformly distributed between [0, 0.2]. Simulations have lead us to initialize W_0^+ referring to the Hebbian law:

$$W_0^+ = \begin{cases} w(i,j) = \Sigma^m_{k=1} (2y_{ik}-1)(2y_{jk}-1) & \text{if } w(i,j)>0 \\ 0 & \end{cases}$$

2.2.2 Recognition phase. Once the learning phase is completed, the network must perform as well as possible the completion of noisy versions of the training vectors. Let $X'=(x'_1, ..., x'_n)$ be any binary input vector. In order to determine the corresponding output vector $Y= (y_1, ..., y_n)$, we first compute the vector of probabilities $Q=(q(1), ..., q(n))$. We 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 stabilizes to an attractor state the number NB_Z of neurons 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 $Y^{(1)}= (y^{(1)}_1, ..., y^{(1)}_n)$, with:

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

Where *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_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. Each value $q(i) \in Z$ and also the lower bound *1-b* are considered as potential thresholds. Eventually, Z can be reduced by decreasing *b* (for b>0.5). 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 $Y^{(2)}(\alpha) = Fz(Q^{(1)}(\alpha))$. 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 too much different of any training vector and α is set to 0.5. If several thresholds are candidate, we choose the one which provides the minimal error (diference between q(i) and y(i) for i=1, n).

3. Evolutionary Learning

3.1. Introduction

The use of evolutionary learning based on GA for ANN is recent [2, 5, 6, 7, 8]. Most of the applications regard the use of evolutionary algorithms as a means to learn connection weights, some applications also regard learning of network topology; and only few uses evolutionary algorithms to define the parameters of a learning algorithm that will be used in the training phase. Some studies attempt to coevolve both the topology and weight values within the GA framework. In this paper, we propose to use GA to design both network's topology and the associated weighted connections of the RNN.

3.2. Our Approach

We propose a hybrid learning algorithm by integrating GA with the gradient descend rule based neural network learning algorithm proposed by Gelenbe. This hybrid algorithm consists of two learning stages using *M* RNN's. The backpropagation algorithm performs the first learning process until the terminal condition (error minimization) is satisfied on each RNN. Then, the second learning stage is used to optimize the topology (connections, weights) of the networks by using a GA. In this stage, each individual is used to encode the topology of one of the *M* RNN's use 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* RNN's in the next iteration. The hybrid algorithm is terminated (system convergence), if one of the next stopping criteria is met: the new initial topologies of the *M* RNN's are the same than 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.

Granularity has an impact in the design phase, too coarse may prevent the existence of a suitable solution, while too fine may lead to huge search spaces, retarding the speed of learning [7]. 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 use networks 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 presence/absence bits (connectivity bits). When connection is present, inmediately after each connectivity bit there is the binary encoding of the relative weight (defined using 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. Thus, coding granularity is a control parameter whose value is coded within the string where it is used. It gives the possibility of having strings of different length. New links are initialized randomly. Figure 1 presents a simple network and its coding.

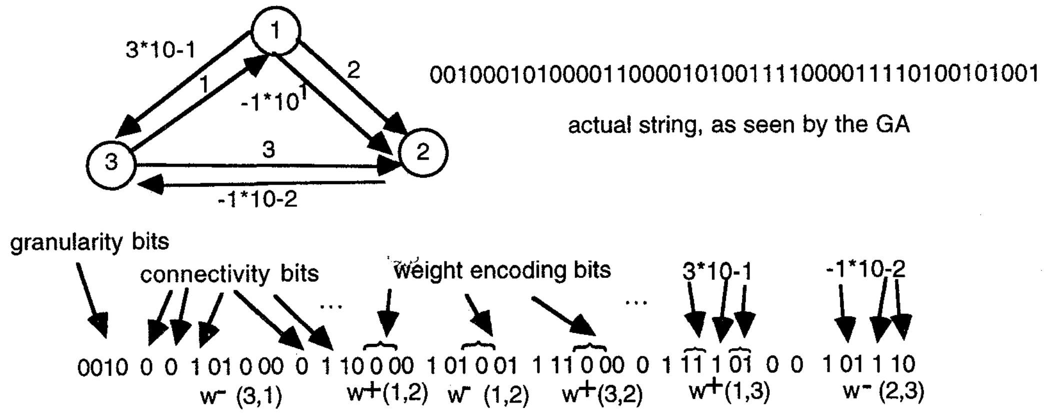


Figure 1. Network encoding of a complete topology graph

We use the commonly used genetic operators (crossover and mutation) and a cooperative local optimizing 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. A simple implementational trick allows mating of network with different connectivity and/or different granularity, with no modification of the operator (Figure 2). Coded networks are implemented using fixed-length arrays, defined with the maximum possible length (all connection present and maximal granularity). Crossover is applied to those strings and therefore it has no problems in application.

```
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
                                        Where * can be randomly initialized, or  they can be zero or one
```

Figure 2. Two individuals with different length undergo crossover.

The mutation operator is the standard, which negates each bit with probability p_m. The cooperative local optimizing genetic operator is a classical optimization method for local optimization of the GA individuals. In our case, local search is performed in two ways: by applying our gradient descend rule based learning algorithm and by using the cooperative local optimizing operator. The latter is a ternary operator (3 parents) 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 $i^{th}$ bit of the strings
   - if $a_{1i} = b_{1i}$ then
       - offspring$_{1i}$ = $c_{1i}$
     else
       - offspring$_{1i}$ =negate ($c_{1i}$)
```

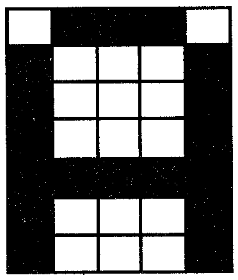
To solve the recognition problem for *N* images using this approach, we use *M* RNN's compose by *n* neurones. The complete evolution learning algorithm for this problem is:

```
- Initiate the parameters for the M RNN's ($W_0^+$, $W_0^-$, $\Lambda$,
   $\lambda$).
- For each successive image:
  - Repeat
    - for i = 1 to M concurrently
      - Perform the gradient descend rule based neural
          network learning algorithm.
    - Generate individuals using every RNN,
    - Optimize the individuals (network's topology:
        connections, weights) using GA,
  Until system convergence
```

4. Performance Evaluation

4.1. Problem Definition

The problem that is presented in this section is to recognize or categorize alphabetic characters (A, B, C, D, E). We will input various alphabetic characters to a RNN and train the network to recognize 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 figure 3. 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 35 neurons.



⇒

```
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
```

Figure 3. Representation of the letter A with a 5*7 pattern.

4.2. Results Analysis

To evaluate the exact recognition rates, we have generated 10 noisy patterns for each training image and for a given distortion rate. We have corrupted them by reasonable noise rates equal to 10% and 5%. A pattern is recognized if the residual error rate is less than 3%. We compare the performance of our approach (re_ev) with the results obtained with the recognition algorithm based on RNN (re_rnn) proposed in [12]. The results we have obtained are presented on figure 4.

The progressive retrieval process with adaptive threshold value that we proposed, provides satisfactory recognition rates. The performance results obtained are lower when the noise rate is important (memories are then more discriminant). Recognition rates remain greater than 90%. In general, rec_ev appears to give the best results, but with a substantially large execution time. That is because the evolutionary algorithm is very slow to converge.

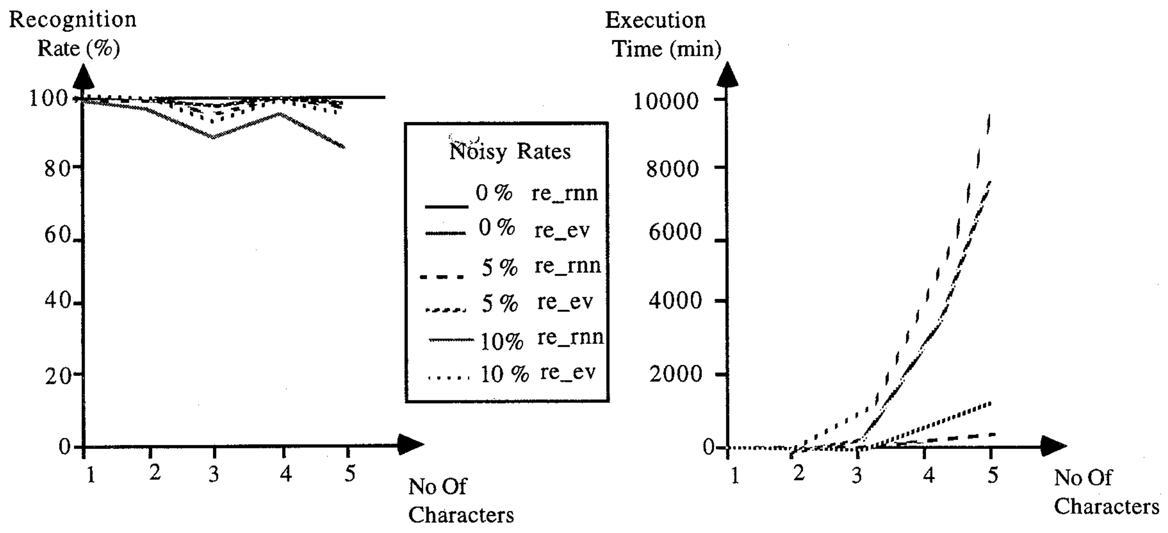


Figure 4. Recognition rate of noisy versions of characters and execution time of the learning algorithms

5. Conclusions

This paper presents an evolution learning algorithm that optimizes a neural network on the basis of its topology and its weights distribution. This algorithm is based on the gradient descend rule based learning algorithm of the RNN and on GA. Different length individuals are used in connection with codifying a control parameter, the coding granularity. A cooperative locally optimizing operator was also used. With this approach, the following observations are made: a) the results of neural network learning are sensitive to the initial topology (connections and weights). A GA is employed to perform global search and to seek a good starting topology for the subsequent neural network learning algorithm. b) The simultaneous structural and parametric modifications based on fitness allows the algorithm to discover appropriate networks quickly and investigate several differing architectures in parallel. c) Complete network induction is approached with respect to the complex interaction between network topology, parametric values and task performance. d) This hybrid algorithm is easy to implement in parallel machine which will reduce the execution time and will improve the results.

References

[1] E. Gelenbe, "Random neural networks with positive and negative signals and product form solution", Neural Computation, Vol. 1, No. 4, 1989, pp. 502-511.

[2] D. Golberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison Wesley, 1989.

[3] E. Gelenbe, A. Stafylopatis, and A. Likas, "Associative Memory Operation of the Random Network Model", Proceedings International Conference on Artificial Neural Networks, ICANN 91, 1991.

[4] E. Gelenbe, "Learning in the recurrent Random Neural Network", Neural Computation, Vol. 5, No. 5, 1993, pp. 584-596.

[5] P. Angeline, G. Saunders, and J. Pollack, "An evolutionary algorithm that constructs recurrent neural network", IEEE Trans. on Neural Network, Vol. 5, No 1, 1994, pp. 54-64.

[6] S. Hungs, H. Adeli, "A parallel genetic/neural network learning algorithm for MIMD shared memory machines", IEEE Trans. on Neural Networks, Vol. 5, No 6, 1994, pp. 900-909.

[7] V. Maniezzo, "Genetic Evolution of the Topology and Weight Distribution of Neural Networks", IEEE Trans. on Neural Networks, Vol. 5, No. 1, 1994, pp. 39-53.

[8] J. Aguilar, "Evolutionary Learning on Recurrent Random Neural Network", Proceedings of the World Congress on Neural Networks, 1995.

[9] J. Aguilar, "An Energy Function for the Random Neural Networks". Neural Processing Letters, Kluwer Academic Publishers, Vol. 4, 1996, pp. 17-27.

[10] J. Aguilar, "A Recognition Algorithm using the Random Neural Network", Proceeding of the 3rd. International Congress on Computer Science Research, 1996.