Evolutionary Synthesis of Dynamical Object Emulator Based on RBF Neural Network

S.A.Sergeev, K.V.Mahotilo

Department of Automated Electromechanical Systems
Kharkov State Polytechnic University
21 Frunze St., Kharkov, 310002, Ukraine
E-mail: sergeev@kpi.kharkov.ua

I. Introduction

In [1] we gave a general description of a "blind"-GA-based procedure of synthesizing a dynamical object emulator based on an RBF network and reported the results of solving a test problem. We coded all the network parameters into one chromosome, and even though its length reached 250 bits, the results happened to be promising. Without leaving the framework of a comparatively simple 3-8-2 network architecture, we managed to obtain quite satisfactory accurateness of the dynamical object transient process approximation and to solve the problem of predicting the object behaviour for a time interval up to 10 seconds.

Unfortunately, despite being a very effective and efficient technique, GAs take quite a long time as compared with local adjusting methods. That is why nowadays there is quite a clear tendency to merge GAs and local techniques into one computational procedure though they used to be opposed to each other as alternative ways of parameter search.

For instance, in [2] GA is reinforced with hillclimbing, in [3] it is combined with Backpropagation algorithm, though while searching, they keep on solving different subtasks, namely, optimization of the network architecture and that of its parameters.

This paper reports an RBF network training technique which joins together GA strategy and an adjusting procedure typical for RBF networks.

II. RBF networks

The desire to transform Rosenblatt's perceptron so that it was able to accomplish an arbitrary nonlinear mapping of input signals into output one might have caused the development of RBF networks. It resulted not only in adding nonlinear units to the conventional perceptron scheme, but also in completely changing the network training concept. RBF network synthesis may be considered as an approximation problem in a multi-dimensional input signal space, and then learning is equivalent to finding such a surface which best meets the training data set.

An RBF network architecture comprises three layers. The first layer acts as the input. The second, hidden layer consists of nonlinear neurones with a radial-basis activation function, and the third, output layer performs the network response to an input signal and conventionally consists of linear neurones.

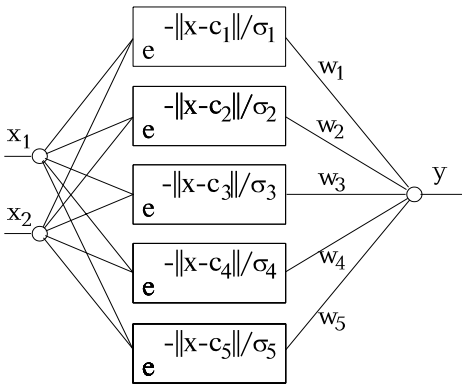

Fig.1. Architecture of an RBF neural network

The fast technique of training a k-n-1 RBF network contains two steps. First, the centres c_j, $j = \overline{1,n}$ of hidden-layer neurone activation functions are chosen at the input space E_n points for which training

patterns $\{x, d\}_i$, $i = \overline{1, m}$ are known, and the activation function window widths σ_j, $j = \overline{1, n}$ are set. Then the output linear neurone synaptic weights w_j, $j = \overline{1, n}$ are computed. The latter task is reduced to solving matrix equation in form of

$$\Phi w = d, \qquad (1)$$

where Φ is the interpolation matrix, w is the vector of the synaptic weights of an output-layer neurone and d is the vector of the output patterns, respectively.

$$\Phi = \begin{pmatrix} f_{11} & . & . & . & f_{1n} \\ f_{21} & . & . & . & f_{2n} \\ . & . & . & . & . \\ f_{m1} & . & . & . & f_{mn} \end{pmatrix}; \; w = \begin{pmatrix} w_1 \\ . \\ . \\ . \\ w_n \end{pmatrix}; \; d = \begin{pmatrix} d_1 \\ . \\ . \\ . \\ d_m \end{pmatrix},$$

where

$$f_{ij} = f(x_i, c_j) = \exp\left(- \sqrt{\sum_{p=1}^{k} (x_{ip} - c_{jp})^2} \Big/ \sigma_j \right).$$

Here x_i, $i = \overline{1, m}$ is the vector of a neural network input pattern.

Generally, solving (1) does not cause any difficulty in computation even in case of the interpolation matrix not being square. But it is undoubted that the case of the number of the hidden-layer neurones n being equal to the number of the training patterns m is the simplest. Then the interpolation matrix is square and (1) is solved in the conventional way

$$w = \Phi^{-1} d \qquad (2)$$

The solution of (2) yields the interpolation surface passing through the patterns in which the neurone centres are located. The approximation error for the rest input space points depends on how well the neurone centres have been arranged and their windows have been chosen.

One should bear in mind, nevertheless, that this approach is quite wasteful because it requires a large number of hidden-layer neurones.

Lack of precise recommendations as to choosing activation function centres results in lack of guarantee for obtaining the global solution to the general synthesis problem. Due to this, a designer usually finds it desirable to make the RBF network hidden layer as large as possible. This extensive way, however, very soon has him face such a problem as a known “curse of dimensionality” [4]. Also, in case of a large number of the ANN inputs, the problem of manually arranging the centres can hardly be solved.

III. Application of RBF networks and GA in control

The combination of GAs and ANNs has already resulted in researchers advancing in quite a few real world applications, but it is in control that this alliance yields such appreciable benefit. Evolutionary paradigm seems to be a perfect platform for developing an intelligent control system (ICS) which would possess attributes of artificial intelligence. Such a system must be capable of changing a control strategy through predicting a dynamical object behaviour. In our opinion, an RBF-based dynamical object neuroemulator may be a best choice to act as a predictor.

RBF networks have proved to be as capable of approximation as MLPs but they are distinct from the latter on a higher speed of learning due to their being inherently local, what makes them especially attractive to be applied to synthesizing a dynamical object emulator.

Reverting to the problem of training a neuroemulator, we should note that there are some nonevolutionary approaches in making a decision about choosing activation function centres. But it is the ability to reduce the number of voluntary choices not only as to locating activation function centres, but also as to the size of the hidden layer, that GAs are noted for.

What we suggest is to minimize both the hidden-layer size and the pattern set, required for the adequate neuroemulator training. We believe that the method of fast adjusting the synaptic weights, specific for RBF network, may serve as a genetic search accelerator. In this case, the hidden layer neurone activation

functions are processed through the “slow” GA, and output-layer neurone synaptic weights are adjusted with the “fast” linear method based on the solution of (2).

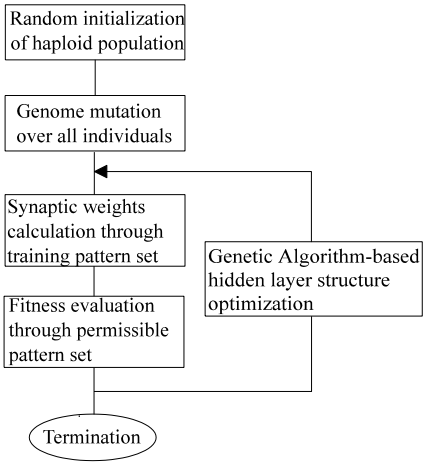

Fig.2. The developed algorithm scheme

In general, the technique being described comprises the following procedures.

1.The dynamical object data are obtained in two stages. At the first stage, it is necessary to gain information about the dynamical object operation in all reference modes. The obtained transient processes are discretized in time to split the input and output spaces into domains. Each pair of the corresponding domains is numbered to form a training pattern $\{x, d\}_i$. Thus, the dynamical object input and output vectors are treated as a pattern set.

At the second stage, the pattern set undergoes preliminary reduction not to take into accounts the patterns which are the same or quite alike. Expelling one of the twins from the pattern set prevents the centres of two neurones from occupying the same input space domain. Otherwise, the interpolation matrix may become singular to cause difficulty in inverting it.

The set of so received patterns is called a permissible testing set, while that of the patterns directly used for the parametric training of the network is called a training set.

2.A population of network hidden-layer genotypes is generated at random. To code chromosomes, we use the four-letter alphabet of **RrDd**, where **R** and **r** are, respectively, recessive 1 and 0, **D** and **d** are, respectively, dominant 1 and 0.

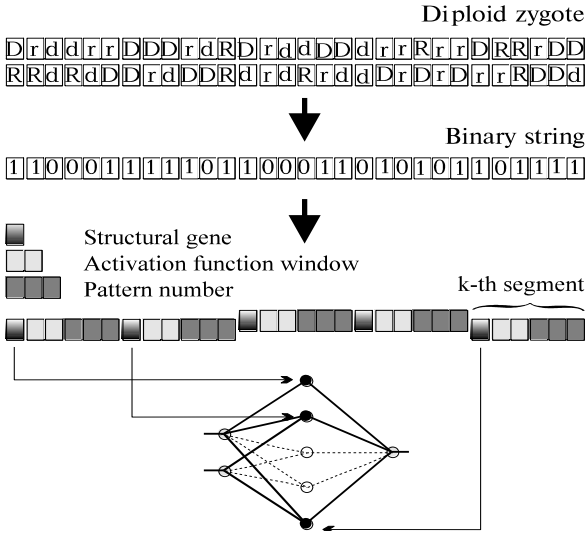

Fig.3. From diploid individual zygote to hidden layer structure

A chromosome comprises segments of identical structure and length, the number of which determines the maximum number of network hidden-layer neurones. The first gene in each segment indicates whether in the hidden layer there is a neurone specified by this segment. Then goes a domain which codes a neurone activation function window width. The last several genes in the segment code the number of a pattern from a permissible pattern set which will be produced to the network when adjusting synaptic weights.

The initial population is created as a haploid one (consisting of single chromosomes). Then each individual undergoes genome mutation to double its chromosome set. Thus, the whole population becomes diploid.

3.Growing individuals is performed in two steps. First, the individual zygote is decoded into the structure and parameters of the network hidden layer, and then, matrix equation (2) is solved to adjust the output-layer neurone synaptic weights.

The process of decoding the four-letter diploid zygote into a binary string which serves to construct the network hidden layer is shown in fig. 3.

To find out which of the parental gamete homologous genes will influence the individual phenotype, we use a dominancy map (fig.4). Our approach differs from others on allowing for fitnesses of the parents, whose gametes compose the zygote, when determining the expressiveness of contradictory genes (shaded cells in the map). The letters in the left column of the map denote the possible gene allele of the less fit parent while the letters in the upper row stand for the fitter parent's genes. The main body of the map shows the final binary state of the gene.

Better fit parent gene

Worse fit parent gene	D	d	R	r
D	1	0	1	1
d	1	0	0	0
R	1	0	1	1
r	1	0	0	0

Fig.4 Dominancy map

After a pair of the individual chromosomes has been decoded into the intermediate binary string, the whole record is split into segments, and structural gene alleles undergo analysis. It is assumed that if the gene is 0, it inactivates the rest segment part (as it has occurred to the fourth segment in fig.3) what results in not including a neurone in the hidden layer.

Similar situation may result from collapsing a neurone activation function window (the third segment in fig.3). If analysing the active segments reveals that the activation function window width has become 0, the current segment also becomes inactivated.

Thus, the way of coding a network provides two possibilities to eliminate a neurone from the hidden layer, namely, sudden expulsion via switching the structural gene allele or gradual collapsing of the window.

Neurone activation function centres are set at those input space points which contain co-ordinates of the input patterns whose numbers are coded at the end domains of the active segments.

It is worth mentioning that in this technique the interpolation matrix is always square and nonsingular to meet the requirements for existing the inverse matrix Φ^{-1}.

Adjusting the synaptic weights ends forming the network structure.

4.The fitness is evaluated by means of a special vector criterion which includes the aggregate approximation error over all the patterns and the hidden-layer size. Evaluating the fitness of each network version with the testing set, we simultaneously estimate the fitness of this training set to represent the dynamical object in all reference modes. Thus, the problem of the neuroemulator training as well as the pattern library minimization are solved in parallel.

5.The population ranking is accomplished through vector evaluation of the fitness in the framework of hierarchic approach. In case of the same approximation error, the preference is given to the individual with the smaller hidden-layer size.

Ranking the population finishes with its splitting into two groups. A certain number of individuals with worse fitness are eliminated. In the succeeding generation, their places are occupied by offspring. The individuals left make up a parental group and generate as many offspring as necessary to keep the population size unchanged. The probability of being a parent is taken equal for every individual from the parental group.

The operators (gene mutation along with inversion) providing for the offspring genetic variability are only applied to haploid gametes produced from parental chromosomes via crossover operator. The mutation operator allows equally probable change of the current gene allele for any of the rest three permitted by the alphabet.

Nevertheless, one should bear in mind that the mutation operator acting is not as simple as in case of a haploid population. In fact, changing 0 for 1, for example, in a structural gene of a haploid individual would

result in appearing an additional hidden-layer neurone (in a non-zero window case, certainly). In our scheme, the similar mutation of R-D in gametegenesis may not bring forth the same result, i.e. we have to allow for the state of the gene which occupies the same locus in the homologous chromosome.

It is important that in the approach being described, applying the crossover operator causes no trouble with the offspring structure. Actually, whatever network hidden-layer structure has been coded by structural genes of parental chromosomes, the offspring structure will always be legal.

According to its fitness, an offspring is either eliminated or given a place in the parental group.

As one can see, the suggested training technique implements the principle of "learning during evolution".

IV. Dynamical object neuroemulator synthesis

To explore the efficiency of the developed neuroemulator synthesis procedure, we have chosen, as a simulated object, a second-order dynamical link defined by the system of linear differential equations

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = \left(-2T\varsigma x_2 - x_1 + kU\right) / \sqrt{T}, \end{cases} \tag{3}$$

with parameter values $k = 1$, $T = 0.5$ and $\varsigma = 0.1$.

This link, of all elementary dynamical links, most meets our requirements due to quite a complicated transient characteristic typical for technical objects.

To train the neuroemulator, we used two different schemes shown in fig. 5. In both of them, the object and the neuroemulator receive the same control signals U. Also, in the 1st scheme, the neuroemulator receives two last co-ordinate values from the object output, and in the 2nd scheme, the neuroemulator gets those from its own output. Thus, the 1st-scheme neuroemulator learns to predict the dynamical object behaviour for one step ahead, while the 2nd-scheme neuroemulator is trained to make a long-term prediction.

The levels of these tasks are apparently very different. A neuroemulator trained to make a long-term prediction will easily cope with the task of a short-term (one-step) prediction. The question of whether the opposite is true needs resolving. To clear it up, we have carried out the following numerical experimentation.

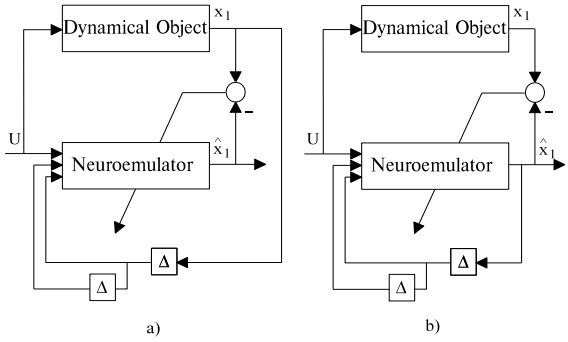

Fig.5. Neuroemulator training schemes for short-term (a) and long-term (b) predictions

To create a library of patterns required for synthesizing a short-term predictor, we chose an object transient curve consequent on a step input signal and processed it according to point 1 (see the previous chapter). We split the required interval into 200 domains to obtain a set of 200 patterns. Expelling alike patterns reduced the set to 180 patterns.

Having limited the maximum number of the hidden-layer neurones to 20, we performed the neuroemulator training according to the 1st scheme and obtained results shown in fig. 6,a. As one can see, the network satisfactorily enough predicted the object behaviour.

Having tested the ability of the synthesized neuroemulator to make a long-term prediction we were surprised that it failed to display oscillation properties. Increasing neither the number of the hidden-layer neurones up to 40, nor the training epochs made the desired effect.

Having compared the phase image of the object and that of the neuroemulator, we found an only transient curve with zero initial conditions to be too scanty source of patterns to train the neuroemulator. That is why beginning the second try to train the neuroemulator, this time via the scheme of long-term

prediction, we assembled a permissible pattern set from 25 transient curves obtained under different initial conditions.

That turned out to provide progress in coping with the task. The prediction quality substantially increased to reveal the desired oscillations on the prediction curve. Also, the network happened to make do with only 10 hidden-layer neurones. The neuroemulator-predicted transient process is demonstrated in fig. 6,b.

a) b)

Fig.6. The dynamical object (curve 1) and RBF-emulator (curve 2) outputs for short-term (a) and long-term (b) predictions

We admit that such a test problem is certainly simplified while the problem of creating the pattern library to train a real object neuroemulator is actually very complicated. In practice, there always exists a vast and even surplus bank of patterns, and which of them best do for training is difficult to determine a priori.

V. Conclusion

Similar to the mentioned example of MLP, reinforcing GA with a local search procedure appears very effective in training RBF networks.

One of the advantages of RBF+GA combination is lack of difficulties in applying the crossover operator which generally arise and are hard to get through in MLPs structural optimization.

Another merit of the above-described technique results from the possibility to simultaneously accomplish two tasks, namely, to optimize the network and work over the training set. As far as one can judge from the given example, a training data set may actually be considerably diminished what is appreciated in designing control systems.

VI. Acknowledgements

The authors would like to express deep gratitude to Professor Erik D. Goodman from Michigan State University, Chairman of the Board of ICAD/GA Consortium, for his permanent attention and all possible support to our research.

This work was supported, in part, by the International Soros Science Education Program (ISSEP) through grant N PSU062065.

VI. References

[1] V.B.Klepikov et al., ”Neural Technologies in Electrical Drive Control” *in Proceedings of the 2nd Conference SENE'95, Lodz*, Poland, 1995, pp. 336-343.

[2] S.A.Harp, T.Samad, A.Guha, “Towards the Genetic Synthesis of Neural Networks”, *in Proceedings of the 3d Int. Conference on Genetic Algorithms,* 1989, pp. 360-369.

[3] D.Whitley, S.Dominic and R.Das, “Genetic Reinforcement Learning with Multilayer Neural Networks”, *in Proceedings of the 4th Int. Conference. on Genetic Algorithms,* 1991, pp. 562-569.

[4] S.Haykin, *Neural Networks. A Comprehensive Foundation*, Macmillan College Publishing Company, 1994.