GENETIC ALGORITHM FOR TRAINING ARTIFICIAL NEURAL NETWORKS WITH BACKPROPAGATION

Ing. Petr ŠMÍD, Doctoral Degree Programme (2)
Dept. of Radio Electronics, FEEC, BUT
E-mail: xsmidp01@stud.feec.vutbr.cz

Supervised by: Prof. Zbyněk Raida

ABSTRACT

The paper deals with training neural models of microwave structures. The first, an artificial neural network (ANN) is trained with a basic genetic algorithm (GA). Training abilities are discussed. Further, the modification of GA and an approach to learning artificial neural networks (ANN) with backpropagation is described. Neural networks are implemented in MATLAB. Results of training abilities are compared. Finally, some ideas for improving the training process are mentioned.

1 INTRODUCTION

The design of electromagnetic (EM) structures is usually based on exploiting their numerical models. The numerical model requests a high computational power. The evaluation of model parameters has to be repeated many times in the optimization cycle: each update of the optimized parameters has to be followed by a new analysis. Replacing a numeric model of the designed EM structure by a neural one is one of ways to reduce CPU-time demands. If the ANN is trained to behave as a microwave system, it can simulate output parameters of the modeled system faster than high CPU-time demanding numerical methods. The high speed of the ANN is caused by a rich parallel connection between neurons in the ANN. Even if the ANN is implemented using a sequential computer, the ANN model evaluates parameters of the modeled system more quickly [1]. The problem is how to train the ANN.

Neural Network Toolbox of MATLAB is a useful tool for creating the neural models of microwave structures, but it contains gradient methods for updating synaptic weights in the ANN mainly [2]. Since the minimized error function has a lot of local lows, the gradient methods can fall in a local minimum. In this case, the neural model exhibits a bad ability to model the desired microwave structure. Therefore, the next part is focused on the genetic algorithm, which is resistant to fall in a local minimum [1].

2 BASIC GENETIC ALGORITHM

For simplicity, assume a wire dipole as the modeled structure. Using the method of moments, we can obtain the dipole impedance if the arm length, wire radius and frequency are known. The problem is to find an antenna size (length and radius) if any dipole impedance is desired at a specific frequency.

Next, the dipole radius is assumed to be constant. We therefore have two input variables: the dipole arm length and frequency. In order to build a fast neural model of the dipole antenna, several training patterns consisting of [arm length; frequency] at the ANN input and [input resistance, input reactance] at the ANN output have to be computed with the moment method. Obviously, the ANN has two input neurons and two output neurons. The number of hidden layers could be estimated experimentally or in accordance to [3]. Since the architecture of the ANN and the training patterns are established, the genetic optimization can start.

If the basic GA is exploited for ANN training, synaptic weights and biases are encoded into one chromosome. So a generation (a set of chromosomes) represents a set of ANN. Each chromosome is decoded and the error function of ANN, which the chromosome represents, is calculated. In order to train the ANN, the output error over all training patterns has to be minimized by setting proper synaptic weights and biases. The error function [1, 4]

$$E = f(\mathbf{W}_a^{(1)}, \mathbf{W}_a^{(2)}, \ldots, \mathbf{W}_a^{(M)}) \tag{1}$$

is put together due to the necessity of minimizing the quadratic deviation between the ANN output and the desired output over all training patterns, given [4]

$$E = \sum_{k=1}^{K} \sum_{j=1}^{m} \left[d_j(k) - y_j(k) \right]^2 , \tag{2}$$

where the elements of output vector has been established [4]

$$\mathbf{y} = \sigma\left(\mathbf{W}_a^{(M)} \sigma_a\left(\mathbf{W}_a^{(M-1)} \sigma_a\left(\cdots \sigma_a\left(\mathbf{W}_a^{(1)} \mathbf{x}_a\right)\right)\right)\right). \tag{3}$$

In (1), E is the squared output error over all training patterns (cost function) dependant on ANN weight matrices $\mathbf{W}_a^{(1)}$ (the first hidden layer weight/bias matrix) to $\mathbf{W}_a^{(M)}$ (the output layer weight/bias matrix). In (2), K means the number of training patterns, m denotes the number of output neurons of ANN, $d_j(k)$ is a value of the j-th desired output of ANN for k-th training pattern and $y_j(k)$ is the value of j-th output response of ANN for k-th training pattern $\mathbf{x}(k)$. In the equation (3), $\mathbf{y}(k) = [y_1(k), y_2(k), \ldots, y_m(k)]^T$ mean the output response of ANN corresponding to input one $\mathbf{x}(k)$ and σ_a denotes an output mapping function of a neuron [1], [4]. Note that the biases are included in weight matrices for simpler scientific notation.

3 BASIC GENETIC ALGORITHM-TRAINING ABILITIES

The basic GA was applied on ANN 2 - 5 - 5 - 2 (two inputs, two hidden layers, both hidden layers contains five neurons, and two neurons in the output layer). The first input parameter (the length of the dipole arm) can vary from 0,5 m to 1,5 m. The second input parameter (frequency) was changed from 75 MHz to 250 MHz. The training set consisted of 9 training patterns (3 lengths × 3 frequencies).

The algorithm was run with the following parameters: 50 iterations (epochs), 8 bits per

synaptic weight/bias, 24 chromosomes, elite selection strategy, probability of mutation is 40 %, probability of crossover equals to 100 %. As the selection strategy, the tournament selection was applied. Due to the stochastic principle of the GA, and due to the randomly initialized ANN at the beginning of the training, each training process was run five times. The charts in fig. 1 show the time behavior of the mean squared training error among the training patterns over 50 iterations.

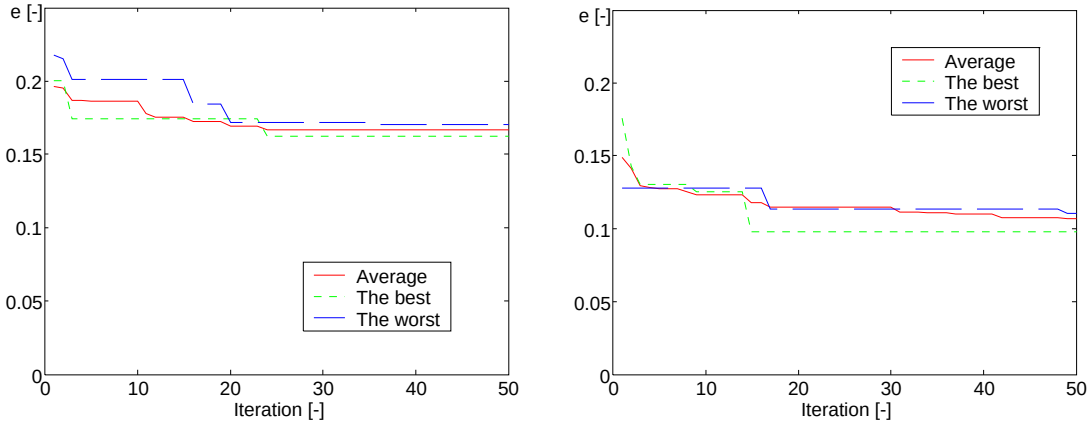

Fig. 1: *Training error of the ANN – the basic GA. Left: the complete training set (nine patterns), right: two training patterns.*

In the chart (fig. 1), we can see three curves: the squared error of the best ANN (the dotted line), the squared error of the worst ANN (the dashed line), and the average squared error computed over five realizations (the solid line). The best ANN and the worst ANN are chosen according to the value of the learning error in the 50th iteration.

The left chart (fig. 1) shows the training ability of the trained ANN by the full set of training patterns – in our example nine. The right chart (fig. 1) shows the training ability of the same ANN trained with two training patterns only.

4 GA APPLIED ON ANN WITH BACKPROPAGATION

The genetic algorithm is to overstep a local minimum, but on the other hand, it "doesn't know what direction to go", so it takes a lot of time. A variation of the GA with exploiting the backpropagation algorithm is focused below.

For instance, the ANN above consists of 57 optimized variables (weights and biases). Using the backpropagation algorithm, we know the error of each neuron. Since the error is known, each neuron can be trained separately. Thus the dimension of the problem can be reduced. The process of weight updating is changed: instead of the simultaneous weight update (if the basic GA is used), the weight values are updated particularly for each neuron. The number of optimized variables is lower and it depends on the number of neuron inputs: for example – a neuron in the first hidden layer has two synaptic weights and one bias (three optimized) variables.

Evaluating the backpropagation error is described in detail in [4]. After evaluating the error values, the desired output of all the neurons is calculated. The desired output of a neuron is given by a sum of its current error and its current output. In this case, each neuron has its own generation of chromosomes. Thus, the GA doesn't work with a set of ANN, but it works

only with a single neuron and every neuron has its own generation of chromosomes.

5 RESULTS OF GA TRAINING WITH BACKPROPAGATION

The results are given in fig. 2. The left chart shows training ability of above ANN with the same parameters as parameters set if the ANN was trained with the basic GA. A new parameter was added: the number of epochs (for each neuron), in one main iteration. The value was set equal one (low value – faster, high value – slower process-run).

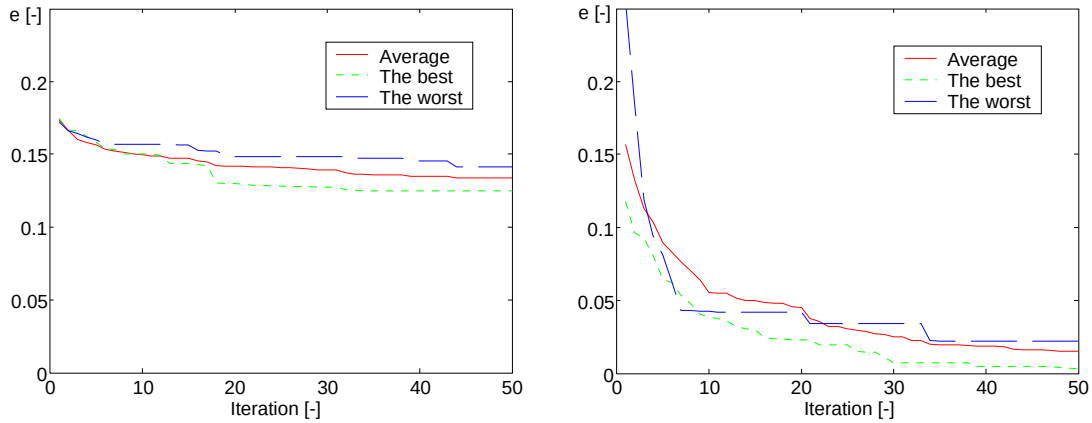

Fig. 2: *The training error of ANN – GA + BP. Left: the complete training set (nine patterns), right: two training patterns*

6 CONCLUSION

We can see that the genetic algorithm converges slowly (fig. 1). Using the error backpropagation, the training ability gets a bit better (compare fig. 1 left and fig. 2 left), but there is still a problem. Comparing the left chart to the right one in Fig. 1 and the same in Fig. 2, we can see improvement of training abilities if the number of training pattern is very low. Two patterns in example above are very low number of training patterns and it is unusable in practical case, but it helps us to find the problem.

Comparing the method of weight updating of GA and gradient methods, we can notice a difference. The gradient method usually calculates quite small differences of the weights (resulting to optimal values for all training patterns), whereas GA produces completely a new set of weigh values in each iteration. A weight setting proper for one training pattern can very differ from the weight values, which are optimal for the different training pattern).

Since averaging of calculated weights does not produce satisfactory results, some better way to obtain synaptic weights will be found. Exploiting the fact, that GA produces a set of possible optimums, and searching a group of weight values in that set with minimal distances is subject of the next work.

ACKNOWLEDGEMENTS

The paper has been prepared as a part of the solution of the GAČR projects No. 102/03/H086 and No. 102/04/1079.

REFERENCES

[1] Černohorský, D., Raida, Z., Škvor, Z., Nováček, Z.: Analýza a optimalizace mikrovlnných struktur, Brno: VUTIUM Publishing, 1999, ISBN 80-214-1512-6.

[2] Demuth, H., Beale, M.: Neural Network Toolbox For Use with MATLAB, User's Guide (nnet.pdf in MATLAB help)

[3] Raida, Z.: Modeling EM structures in the Neural Network Toolbox of MATLAB, IEEE Antennas & Propagation Magazine. 2002, vol. 44, no. 6, p. 46–67, ISSN 1045-9243

[4] Gupta, M., M., Jin, L., Homma, N.: Static and Dynamic Neural Networks From Fundamentals to Advanced Theory, New Jersey: John Wiley & Sons, 2003, ISBN 0-471-21948-7