Abstract. The paper deals with training the neural models of microwave structures. The first, an artificial neural network (ANN) is trained with 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.

Introduction

The design of electromagnetic (EM) structures is usually based on exploiting their numerical models. Numerical model requests high computational power. Model parameters evaluation 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 modeled system faster than high computational demanding numerical methods. The high speed of ANN is caused by rich parallel connection between neurons in the ANN. Even if the ANN is modeled using sequential computer, the evaluation speed of modeled system parameters (considering numerical model encloses many loops and/or iterations) modeled by an ANN is faster [1]. The problem is how to train the ANN.

Neural Network Toolbox of MATLAB is useful tool for creating the neural models of microwave structures, but it contains gradient methods for updating synaptic weights in 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 bad ability to model desired microwave structure. Therefore, the next part is focused on genetic algorithm, which is resistant to fall in a local minimum [1].

Basic genetic algorithm

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

For further simplification, let the dipole radius is constant, we have two input variables: the dipole arm length and frequency. In order to build fast neural model of dipole antenna, several training patterns covering the area of arm length, frequency and input impedance values has to be computed with moment method. The ANN has two input neurons and two output neurons one (real and imaginary part of input impedance). The number of hidden layers could be estimated experimentally or in accordance to [3]. Since the architecture of ANN and 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]

\[ E = f(W_a^{(1)}, W_a^{(2)}, \ldots, W_a^{(M)}) \]  

(1)

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

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

(2)

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

\[ y = \sigma \left( W_a^{(M)} \sigma \left( W_a^{(M-1)} \sigma \left( \cdots \sigma \left( W_a^{(1)} x \right) \cdots \right) \right) \right). \]

(3)

In the equation (1), \( E \) is the squared output error over all training patterns (cost function) dependant on ANN weight matrices \( W_a^{(1)} \) (the first hidden layer weight/bias matrix) to \( W_a^{(M)} \) (the output layer weight/bias matrix). In the equation (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 \( x(k) \). In the equation (3), \( y(k) = [y_1(k), y_2(k), \ldots, y_m(k)]^T \) mean the output response of ANN corresponding to input one \( x(k) \) and \( \sigma_a \) denotes an output mapping function of a neuron [1], [4]. Note that the biases are included in weight matrices for simpler scientific notation.

**Basic genetic algorithm - training abilities**

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 algorithm was run with these parameters: 50 iterations (epochs), 8 bits per synaptic weight/bias, 24 chromosomes, elite selection strategy, mutation is 40 %. Due to stochastic principle of GA and due to 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 ANN – Basic GA, left: complete training set (nine patterns), right: two training patterns](image)
In the chart, there are three curves: squared error of the best ANN (dotted line), squared error of the worst ANN (dashed line), and the average squared error computed over five realizations (solid line). The best ANN and the worst ANN are chosen according to the value learning error in the 50th iteration.

The left chart shows the training ability of trained ANN by full set of training patterns – in our example nine doublets of values of arm length and frequency and corresponding doublets of real and imaginary part of dipole impedance. Right chart shows training abilities of the same ANN trained with two training patterns only.

**GA applied on ANN with backpropagation**

Genetic algorithm is capable not to fall in a local minimum, but on the other hand it “doesn’t know how direction to go”, so it takes a lot of time, especially if the dimension of solved problem is high. A variation of GA with exploiting the backpropagation algorithm is focused below.

For instance, ANN above has 57 optimized variables. Using the backpropagation algorithm let us 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 simultaneously weight updating, if basic GA used, the weight values are updated particularly for each neuron. The number of optimized variables is lower and it depends on 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 of neurons is calculated. The desired output of a neuron is given by sum of its current error and its current output. In this case, each neuron has its own generation of chromosomes. Thus GA doesn’t work with a set of ANN, but it works only with single neuron and every neuron has its own generation of chromosomes.

**Results of GA training with backpropagation**

The results are given in Fig. 2. Left chart shows training ability of above ANN with the same parameters as parameters set if the ANN was trained with basic GA. 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 Training error of ANN – GA + BP,**  
*left: complete training set (nine patterns), right: two training patterns*
Conclusion

We can see that 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. 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 every iteration. A weight setting proper for one training pattern can very differs from the weight values, which are optimal for the different training pattern).

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

Acknowledgements

Research described in the paper was financially supported by the grants of the Czech Grant Agency no. 102/03/H086 and 102/04/1079.

References


