Proceedings of the 7th
World Congress on Intelligent Control and Automation
June 25 - 27, 2008, Chongqing, China

Optimize Neural Network Controller Design Using Genetic Algorithm

Ariel Kopel, Xiao-Hua Yu

Department of Electrical Engineering
California Polytechnic State University
San Luis Obispo, CA 93407, USA

xhyu@calpoly.edu

***Abstract* - The size of a neural network must be pre-determined before it can be trained for any application. Choosing the correct size of a neural network can increase its speed of response and thus improve the performance of the overall system. In this paper, a genetic algorithm is employed to find the optimal number of connections of a neural network controller which is used to regulate a class of DC power supplies. Satisfactory computer simulation results are obtained.**

Index Terms – Artificial neural networks, genetic algorithm, neural network controller.

I. INTRODUCTION

Before a neural network can be trained, its size (i.e., how many neurons and connections) must be determined in advance. Generally, more complex functions require larger neural networks (i.e., more neurons and synapses). If a neural network is too small for a given application, it may never be able to learn the desired function and thus produces unacceptably larger errors. On the other hand, if a neural network is too large for a particular problem, it may learn the training samples too well and not be able to generate the appropriate output for the inputs not included in the training set (this phenomenon is known as over-fitting). Selecting the appropriate neural network dimension is more of an art than a science and usually turns into a trial-and-error ordeal, which is why the genetic algorithm is a perfect tool for solving this problem.

Genetic algorithm has its roots in nature; it is based on Charles Darwin's theory of natural selection (i.e., "survival of the fittest"). In Darwin's theory, individuals in a population of reproductive organisms inherit traits from their parents during each generation. Each individual's genome represents one's phenotype (i.e., the physical characteristics) and may contain many genes. Over time, desirable traits become more common than the undesirable ones since individuals with the desirable traits that "fit" better with the environment are more likely to reproduce.

Genetic algorithm follows the natural selection theory quite closely. In genetic algorithm, genes are encoded as a string of binary numbers to represent certain phenotypes related with a specific application. As in natural selection, a population of individuals is initially created with all of their genotypes randomly selected. Then, each individual in the population is sorted based on its fitness level; where the definition of fitness function is also application specific. During each generation, two individuals are selected to reproduce, with the more "fitted" individuals more likely to be selected. The genotypes of the two parent individuals are combined to create a new offspring in a process known as crossover.

After two individuals are selected in each generation, their genomes are crossed over to produce a new individual which may yield a higher fitness index than both parents. Two crossover methods are studied in this research, i.e., gene-level and bit-level crossover. In gene-level crossover (also known as multi-point crossover), each of the child's genes is selected from one of the parents, with certain probability of each parent's gene being used. In bit-level crossover (also known as uniform crossover), the above process is implemented on each bit. After crossover, some of the bits in the child's genome may be flipped at random which also occurs in nature (called mutation). Mutation allows individuals to be generated to have new genotypes that may be potentially better than any ones that can be found in the current population. Finally, after the child's genome is completely determined, the child's fitness index can be calculated. If the child's fitness index is better than the worst individual currently in the population, then this child replaces that individual in the population. The population continually improves its overall fitness during each generation until finally the top individual is optimized to a satisfactory level.

With today's extremely fast computer processors, running genetic algorithms for hundreds of generations is possible in a reasonable amount of time. In addition, the relationship between the genotype (parameters of the system) and phenotype (system performance) does not need to be known for a good solution to be found in genetic algorithm. This brings us to use it to optimally select the dimension of a neural network controller.

II. THE GENETIC ALGORITHM

In this research, the genetic algorithm is employed to optimize the size of fully-connected feed-forward neural networks. The genetic algorithm contains several essential components and procedures. In this section, each of those components and procedures will be described in detail.

The genome determines one's phenotype (i.e., characteristics). In this research, the genome is composed of twenty four bits which are grouped into nine genes. Each of the genes represents a different neural network parameter. For example, gene 1 is used to represent the number of hidden layers of a neural network. It consists of two bits and could therefore take four different combinations (00, 01, 10, and 11); which means that the neural network can have up to four hidden layers. Note that the option of zero hidden layer is removed because most problems are too complex for such a small network. Genes 2 - 5 represent the number of neurons to be determined in each hidden layer. For example, gene 2 represents the number of hidden neurons in the first layer; gene 3 represents the number of hidden neurons in the second layer; and so on. Zero neuron is not allowed since at least one neuron must exist in each hidden layer. Note that a hierarchical structure is implied; that is, the values of genes 2 through 5 are actually dependent on the value of gene 1. Some of the genes may be inactive; however, they can still undergo crossover and mutation throughout the generations. Inactive genes can be activated in new individuals later on, due to a change in gene 1. Other genes used in this application may include neural network learning rate, neuron activation function slope, the seed of random number generator, etc.

An integral part of genetic algorithm is the generation of new individuals by combining the genomes of two "fit" parents. In order to determine which individuals are more "fit" than others, a fitness index must be determined. The purpose of this study is to minimize the size of a neural network without sacrificing its training accuracy. Thus, at the end of every generation, all individuals in the population are sorted in a reverse order based on their RMS (root-mean-square) values of training error and their sizes (i.e., number of synapses). We define:

$$J = r_e \cdot \eta_e + r_s \cdot \eta_s \qquad (1)$$

where J is the fitness index, r_e is the reverse rank of the RMS error, η_e is the error scaling factor ($0 \le \eta_e \le 1$), r_s is the reverse rank of the neural network size, and η_s is the size scaling factor ($0 \le \eta_s \le 1$). Note that

$$\eta_e + \eta_s = 1 \qquad (2)$$

Depending on the requirement of a certain application, different values of scaling factors can be chosen. For example, if minimizing the RMS error is more important than minimizing the size of the neural network, then the error scaling factor should be greater than the size scaling factor.

Once the fitness index J is obtained, it can be used to rank the "degree of fitness" for each individual. For example, the individual with a rank of 1 (i.e., with the maximum value of the fitness index) represents the most "fit" individual.

To promote the reproduction of "fit" individuals, the probability of being selected (to reproduce the next generation) is determined by each individual's degree of fitness (or rank). Each individual's probability of being selected (p) can be calculated as:

$$p = \frac{R_i}{\sum_{i=1}^{N} R_i} \qquad (3)$$

where R_i is the rank (degree of fitness) of the i-th individual and N is the total number of the rank of degree of fitness in the population. Obviously, the higher the degree of fitness of an individual is, the higher the probability of being selected for reproduction.

In gene-level crossover, the genomes of both parents are crossed over on a gene by gene basis with a 50% chance of each parent's gene being selected. This means that each of the new individual's genes has a 50% chance of coming from either parent.

III. SIMULATION

In this section, the genetic algorithm described in the section two is simulated and applied to find the optimal size of the neural network controller which is used to control a phase-shifted full-bridge DC-DC converter.

It is well known that a DC voltage regulator is an important part in many electronic devices nowadays, since all semi-conductor components are powered by DC sources. In this research, we consider a class of DC voltage regulators, i.e., DC-DC converter with PWM (pulse-width modulation). The controller's task is to maintain a stable output voltage by varying the duty cycle in pulse-width modulation when different loads are placed on the converters terminals, and/or when the supply voltage fluctuates.

The conventional approaches assume that the converter is operated around its equilibrium state; then a set of linear equations are derived based on this assumption. However, the control law is actually highly nonlinear and thus cannot guarantee its performance under this simple assumption. To improve the controller response to dynamical changes, neural network controller has been chosen as an alternative to classic methods. The controller has three inputs: load current (in amps), input voltage (in volts), and the difference of the current and previous output voltages (in volts). The output of the controller is the duty cycle that can be used in pulse-width modulation. To generate the training data set, a Simulink model is developed based on circuit analysis, as shown in Fig. 1.

First, a set of neural networks with arbitrary sizes are generated as the initial group of individuals used in the genetic algorithm. The neural networks are then trained for 2000 epochs, with each sample being selected at random from the training set to allow for better generalization. Back-propagation algorithm is employed. The weight of neural network w can be adjusted using:

$$w_m = w_{m-1} + \Delta w_m \tag{4}$$

where m is the training iteration index.

Learning in back-propagation is performed in two steps, i.e., the forward path and the backward path. In the latter, the error signal (i.e., the difference between the neural network output and the target) is fed back from the output to input, layer by layer; and all the weights are adjusted in proportion to this error, i.e.,

$$\Delta w_m = \eta \frac{\partial E}{\partial W} \tag{5}$$

where η is called the learning/training rate; W is the weight matrix of the neural network and E is the objective function which is defined as:

$$E = \frac{1}{MK}\left[\frac{1}{2}\sum_{m=1}^{M}\sum_{k=1}^{K}(y_{km} - d_{km})^2\right] \tag{6}$$

where M is the number of input/output pairs in the training set and K is the number of outputs of the neural network; y_{km} is the *k-th* network output for the *m-th* input/output pair in the training set; d_{km} represents the *k-th* desired (target) output for the *m-th* input/output pair.

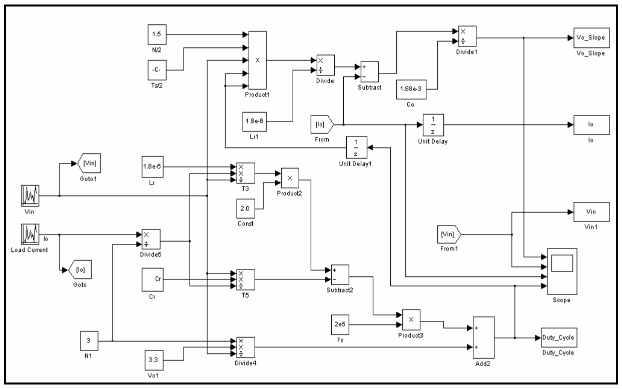

Fig. 1. The Simulink model

For sigmoid function, the neuron activation function is:

$$g(u) = \frac{1}{1+e^{-u}} \tag{7}$$

where g is the output of the nonlinear neuron and

$$u = WX \tag{8}$$

with weight matrix (W) and the input vector to neuron (X). It can be proved that:

$$\frac{\partial E}{\partial W} = (z-d)\,u(1-u)X \tag{9}$$

For the sake of simplicity, the index of neuron is not included in the above formulas.

After training is completed, each individual's RMS error is calculated and all the neural networks are ranked based on the values of their fitness indices. Crossover and mutation operations are then performed to obtain the next generation of neural networks.

Fig. 2 shows the simulation results on the relationship between the neural network RMS training error, number of connections in the neural network, and the population size. The error scaling factor is chosen to be 0.7. The population size varies between 10 to 100, in an increment of 10. The simulation runs for 50 generations to find the solution.

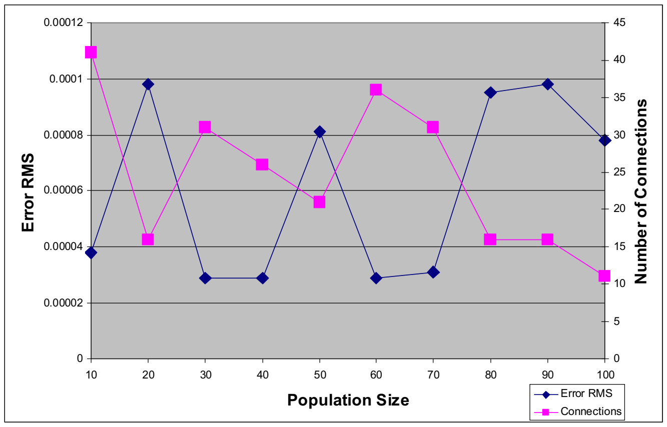

Fig. 2. Relationship between the RMS error, number of connections, and the population size

The data suggests that the RMS error does not have a correlation with population size, as it fluctuates randomly (Fig. 2). The number of connections shows a slight decreasing trend as the population size increases. The optimal population size was found to be 40 for this application and the optimal number of total connections is found to be 26.

Mutation is an important component in genetic algorithm. It allows individuals to be generated to have new genotypes that may be potentially better than any ones that can be found in the current population. The effect of the probability for bit mutation on the controller performance is also investigated. In Fig. 3, it is varied between 0.01 to 0.1, with an increment of 0.01. It is shown that by choosing the optimal value of the probability for bit mutation, we can further reduce the total number of connections of the neural network controller to 21.

Furthermore, the effect of the error scaling factor is studied. Fig. 4 shows the simulation results on the relationship between the RMS error, number of connections in the neural network, and the error scaling factor (the size scaling factor is not included; however, it can be obtained easily from Eq. (2)). The error scaling factor varies between 0 to 1, in an increment of 0.1.

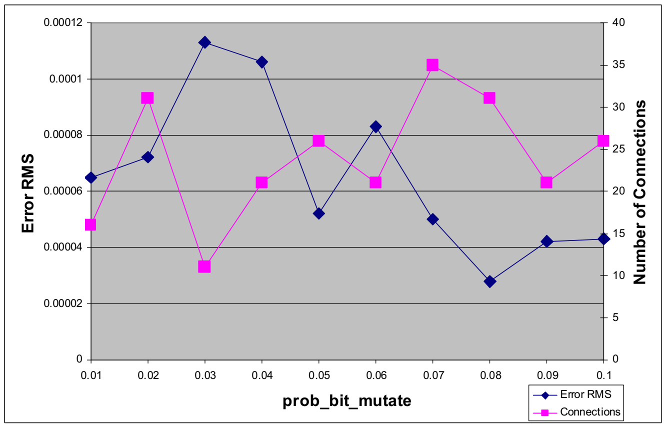

Fig. 3. Relationship between the RMS error, number of connections, and the probability of bit mutation

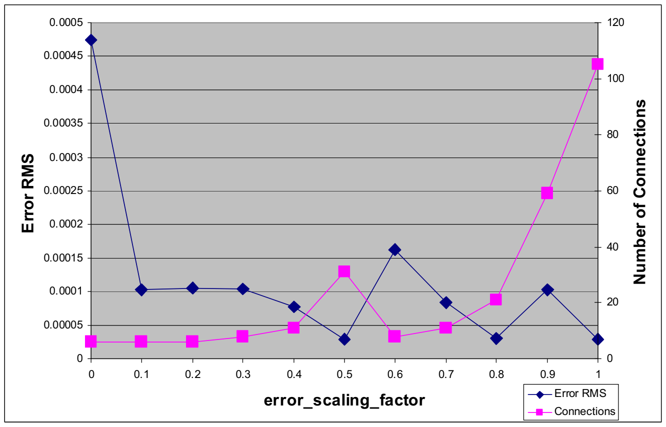

Fig. 4. Relationship between the RMS error, number of connections, and the error scaling factor (50 generations)

The data suggests, as expected, that increasing error scaling factor lowers RMS error and also increases the number of connections. RMS error declines dramatically between the error scaling factors of 0 to 0.1 and remains relatively stable as the error scaling factor increased. On the other hand, the number of connections increases dramatically at the maximum value of error scaling factor. The optimal error scaling factor is found to be 0.8 for this DC-DC converter controller application. At this point, the number of connections of the controller is about 21.

IV. Conclusions

In this paper, a genetic algorithm is studied and applied to determine the optimal size of a neural network controller for a DC power regulator. The effects of various factors/parameters in genetic algorithm on the RMS error value and the size of the neural network controller are investigated. Future research plan includes applying this approach on hardware to test its performance.

ACKNOWLEDGMENT

This work is partly sponsored by the Department of the Navy, Office of Naval Research, under Award # N00014-06-1-1111. The authors would also like to thank Cal Poly Honors Student Research Program for the support.

REFERENCES

[1] Li, W. & Yu, X.-H. 2007. A self-tuning controller for real-time voltage regulation. Forth coming in the proceedings of the IEEE International Joint Conference on Neural Networks.

[2] Yen, G. & Lu, H. 2000. Hierarchical genetic algorithm based neural network design. IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. p. 168-175

[3] Bevilacqua V., Mastronardi G., et. al. 2006. A novel multi-objective genetic algorithm approach to artificial neural network topology optimization: the breast cancer classification problem. International Joint Conference on Neural Networks. p. 1958 – 1965.

[4] P. Hancock 1992. Pruning Neural Nets by Genetic Algorithm. Proceedings of the International Conference on Artificial Neural Networks.

[5] Choi, B. & Kim, J. et al. 2002. Designing control loop for DC-to-DC converters loaded with unknown AC dynamics. IEEE Trans. On Industrial Electronics, 49(4): pp. 925-932.

[6] Choi, H.-S. & Kim, J.-W. et al. 2002. Novel zero-voltage and zero-current-switching (ZVZCS) full-bridge PWM converter using coupled output inductor. IEEE Trans. On Power Electronics, 17(5): pp. 641-648