(NASA-CR-200242) NEURAL NETWORK METHODS IN ANALYSIS OF ACOUSTO-ULTRASONIC DATA (NASA. Lewis Research Center) 15 p

N96-70914

Unclas

Z9/63 0101249

NASA-CR-200242

NAG3-1422

NEURAL NETWORK METHODS IN ANALYSIS OF ACOUSTO-ULTRASONIC DATA

K.J. Cios+, E.A. Mayer+, A. Vary*, and H. E. Kautz*

University of Toledo+ and NASA Lewis Research Center*

P-15

ABSTRACT

Acousto-ultrasonics (AU) is a nondestructive evaluation technique that was developed for the purpose of testing various types of materials. AU is sensitive to different types of damage that may occur in composites. A study done at NASA Lewis Research Center with metal-matrix composites and AU shows that after some signal processing of the raw AU signals, damage due to strain and stiffness degradation could be determined. This paper uses the data obtained from the NASA Lewis Research Center study and applies to it a combination of neural networks and genetic algorithms.

INTRODUCTION

Genetic algorithms are good at handling nonlinear and higher-dimensional problems which are characteristic of the problem of training neural networks [1]. They are also not as sensitive to local minima like gradient training algorithms such as backpropagation [2]. In addition, genetic algorithms use information from a population of neural networks instead of from a single neural network. This allows the genetic algorithm to obtain a global perspective of the solution space so as to avoid local minima to find the global minimum. In this paper, we describe the application of genetic algorithms to the building and training of neural networks to determine the mechanical properties of materials from acousto-ultrasonic data. This combination of neural networks and genetic algorithms will be called genetic gradient cascade-correlation.

Acousto-Ultrasonic Technique

A detailed description of the AU technique is described elsewhere [3, 4, 5, 6, 7]. The work by Vary [5], the originator of the method, provides the best description. A typical AU setup consists of two piezoelectric transducers, a sender and a receiver, which are coupled to the surface of the specimen under test (Figure 1). The sending transducer introduces ultrasonic pulses into the specimen which then propagate along the specimen and are picked up by the receiving transducer. The propagation of the pulses through the material is affected by various mechanical properties. It is this fact which allows us to deduce these mechanical properties by studying the waveform measured by the receiving transducer.

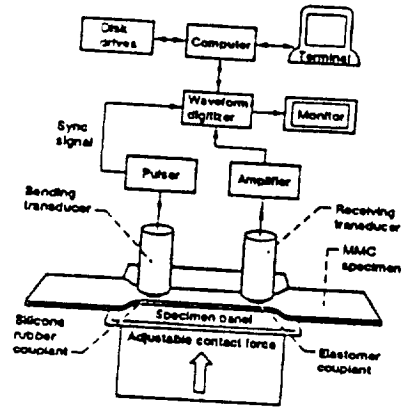

Figure 1. Acousto-ultrasonic data collection system.

In the data we use, the pulses were sent at a repetition rate of 500 Hz and the separations of the transducers was 1.91 cm for all measurements. Six measurements were taken: three with both transducers on the top of the specimen and three with both transducers on the bottom of the specimen. The metal-matrix composites we evaluate are SiC (SCS-6)-reinforced Ti-15V-3Cr-3Sn-3Al (Ti-15-3) material containing eight layers of fibers yielding a total thickness of 0.20 cm. Two types of specimens are used: a straight-sided specimen 1.27cm wide and a tapered specimen. Two types of specimens, $[0]_8$ and $[+-30]_{2S}$, are considered in our study.

Each AU signal measurement consists of 512 discrete time samples of the signal received by the receiving transducer as shown in Figure 2. A fast Fourier transform (FFT) is performed which yields 256 real and imaginary amplitudes at 256 frequencies plus a DC component. The FFT is simplified to a power spectrum and then averaged over every two samples yielding 128 frequencies. Of these 128 frequencies, we found that only 32, Figure 3, contain pertinent information so these are used for training the neural network [7].

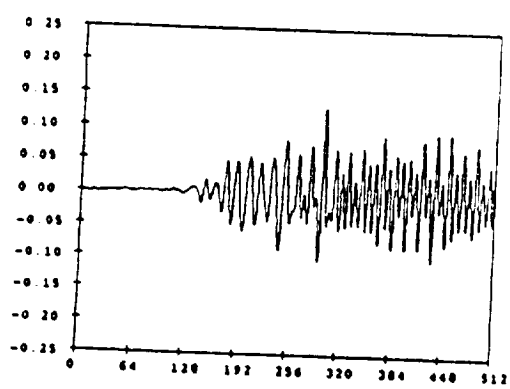

Figure 2. A typical AU signal collected on SiC/Ti-15-3 specimen.

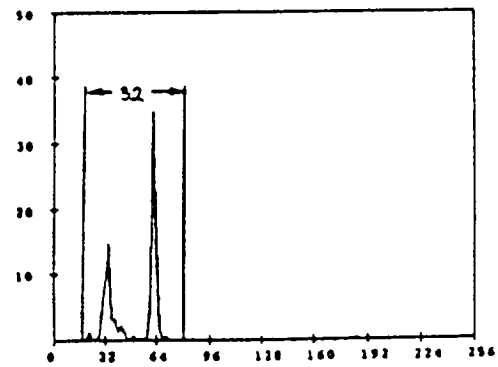

Figure 3. FFT of the signal shown in Figure 2.

BACKGROUND

Neural Networks

Neural networks are composed of fundamental units known as neurons or nodes. These neurons are like their biological counterparts in that they are connected to either sensory inputs or directly to other neurons by synapses which are weighted [2]. When the appropriate pattern of inputs are applied to the neuron, it will fire and send a signal down its axon. A negative weight will tend to inhibit the firing of the neuron while a positive weight will tend to excite the firing of the neuron. The output of a neuron is modeled by the following formula:

$$o = f(\mathbf{W}, \mathbf{X}) = (1 + e^{-(\mathbf{W} \cdot \mathbf{X})})^{-1} \tag{1}$$

The output is given by the dot product $\mathbf{W} \cdot \mathbf{X}$ where $\mathbf{X}$ is the vector of the inputs (x_0, x_1, ... x_n)$^{-1}$ and $\mathbf{W}$ is the vector of the weights (w_0, w_1, ... w_n)$^{-1}$. $\mathbf{W}$ is known as the *weight vector*. x_0 is always equal to 1 and is known as the *bias*. The dot product is applied to an activation function. The activation function is a threshold function. For the activation function, we use the sigmoidal function given by the equation:

$$f(x) = (1 + e^{-x})^{-1} \tag{2}$$

The neurons can be connected in various architectures. In this paper, we are concerned with feed-forward and cascade-correlation neural networks. In a layered feed-forward network, the neurons are arranged in multiple layers consisting of an input layer, one or more middle layers, and the output layer. The input neurons are connected to the outside world and transmit unweighted inputs to the middle layers. The neurons in each consecutive layer are connected via synapses to neurons in the previous layer. The cascade-correlation architecture is like the feedforward network except that it has only

one neuron in each middle layer and each neuron is connected to all the neurons in the previous layers.

Neural networks are trained by example. To train a neural network for solving a certain problem, the training data is arranged in vectors consisting of the input data to be applied to the network along with its desired or targeted outputs that the network will produce when the training is completed. The initial weights in the network are usually set up randomly. The input data is applied to the neural network and the data is propagated through the network via the synapses and the firing of the neurons. The outputs of the neural network are then compared with the target outputs and the weights of the neural network are then adjusted using any of a number of learning algorithms so that the next propagation of inputs will produce the target outputs. We will now examine the concept of genetic algorithms and how they can be applied to training the cascade-correlation neural network.

Genetic Algorithms

Genetic algorithms are based on Darwin's concepts of the survival of the fittest and evolution as applied to problems of optimization and classification [1]. They work with a population of proposed solutions to a problem. Darwin's theory states that animals that are fittest to their environment have better chances of surviving. Analogously, we assign each member of the population a fitness based on how well they come to solving the problem. The better they are at solving the problem, the higher the fitness they are assigned. For the genetic algorithm to work, the members of the population should be encoded in an alphabet which is a set of characters representing the various characteristics of the solution. The principle of minimal alphabets states that we should select the smallest alphabet that allows the natural expression of the problem. Since the weights can be represented as real numbers, the alphabet $\{0,1\}$ which allows for the maximum number of schema per bit is chosen.

The simple genetic algorithm consists of three operators: reproduction, crossover, and mutation. Referring to Darwin, the animals in the wild that are allowed to mate are those who are the strongest or fittest. This ensures that their offspring, and therefore the population of that animal, will be fit and will survive. We apply this principle to a population of neural networks. The members of the population are assigned probabilities to become parents based on their fitness. The roulette wheel method assigns the probability of becoming a parent as:

$$p_{parent} = \frac{fitness_{parent}}{\sum_{population} fitness_i} \tag{3}$$

Two parents are chosen and two offspring or children are produced using the two remaining operators: crossover and mutation.

Parents are represented by strings of 1's and 0's. Various mappings of the parameters of the problem to the alphabet {0,1} can be used depending on the desired range of the parameters. In the following example, let us assume there is one integer parameter. The two parents are encoded in binary. A crossover point is randomly chosen. All the bits before the crossover point are switched between the two parents. The result is two new patterns of bits which are called the children. The process can be illustrated as follows:

crossover point = 5:

Parents:	Offspring:
00101|000	00111|000
00111|100	00101|100

The reason behind crossover and binary encoding warrants some explanation. The genetic algorithm optimizes by manipulating bit patterns known as schema. Examples of schemas are as follows:

***10* 000001 10110* *****0 11*111 1**1**

where a " * " indicates a "don't care." The genetic algorithm searches for an optimal solution by combining various schemas. Schema with shorter defining lengths, or the length between defined bit positions, have a better chance of surviving during the crossover operation. The third operator, mutation, consists of the switching of a bit with a certain probability $p_{mutation}$. This serves the same purpose as the biological mutation. The mutation may allow members of the population to adapt to the problem better therefore increasing their fitness and likelihood of survival. It also serves the purpose of reintroducing schema that may be crucial in the optimization but may have been eliminated in earlier generations.

The weights of the neural network are binary encoded and then concatenated together to form a binary string for each network. Parents are randomly selected in proportion to their fitness and new children are created using crossover and mutation. A non-overlapping population is used in which a new population is generated for each generation.

GENETIC GRADIENT CASCADE-CORRELATION

The genetic gradient cascade-correlation (GGrCC) is a modification of the cascade-correlation learning architecture. To understand it, one must understand cascade-correlation. A short presentation of the cascade-correlation learning architecture follows. The GGrCC algorithm is then presented.

Cascade-Correlation Learning Architecture

The cascade-correlation architecture has an architecture that lends itself well to the use of genetic algorithms due to its asymmetric architecture. Each weight has a specific function and is not interchangeable. The cascade-correlation also has a self-determining architecture in which neurons are added as they are needed [8].

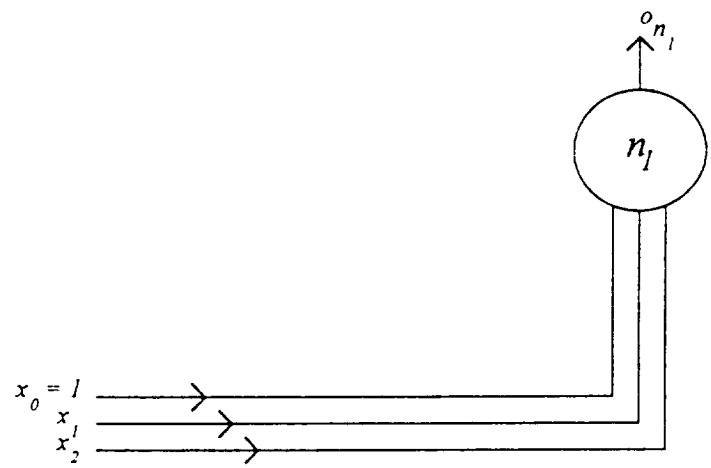

Figure 4. Initial cascade-correlation architecture.

In the cascade-correlation learning architecture, the neural network starts as a simple feedforward network with a layer of output neurons as shown in Figure 4 [8]. These output neurons are directly connected to the inputs to the network. The cascade-correlation learning algorithm is shown below.

Cascade-Correlation Algorithm.

- **(A1a) Set up initial cascade-correlation architecture, Then at step j : j = 0,1,...;**
- **(A1b) Randomize output neuron weights.**
- **(A1c) Compute gradient of output weights using (6).**
- **(A1d) Adjust output weights using Quickprop (8).**
- **(A1e) Create hidden neuron j and randomize weights.**
- **(A1f) Compute gradients of hidden neuron j using (10)**
- **(A1g) Adjust hidden neuron j weights using Quickprop (11)**
- **(A1h) Connect hidden unit j to network.**
- **(A1i) If acceptable error or maximum number of hidden nodes, stop; otherwise return to (A1b).**

The output weights are randomized in the range of -0.5 to 0.5. A set of training inputs is applied to the network and the SSE error in the network is calculated using the following equation:

$$SSE = \sum_p \sum_i (E_{n_i}^p)^2 \tag{4}$$

where:

$$E_{n_i}^p = y_{n_i}^p - o_{n_i}^p \tag{5}$$

$E_{n_i}^p$ is the residual error for training vector p and output neuron n_i, $y_{n_i}^p$ is the desired output of neuron n_i for training vector p and $o_{n_i}^p$ is the present output of neuron n_i for training vector p. The output weights are then adjusted to minimize the SSE. This is done by first calculating the gradient of the error with respect to the weights by:

$$\nabla E_{n_i}^p = \frac{\partial E_{n_i}^p}{\partial w_{n_i,x_j}} = \eta E_{n_i}^p f_p' I_{x_j}^p \tag{6}$$

where:

$$f_p' = o_{n_i}^p (1 - o_{n_i}^p) \tag{7}$$

w_{n_i,x_j} is the weight from neuron n_i to x_j which is either an input to the network or the output from a hidden neuron to be added later. η is the learning constant, f_p' is the derivative of the activation function, and $I_{x_j}^p$ is the input [illegible] output neuron n_i from x_j. We then use a number of Quickprop iterations to update the weight. Quickprop is a form

of the numerical analysis technique of Newton's method. Quickprop is represented by the following equation:

$$\Delta w_{n_i,x_j}(t) = \frac{\nabla E^p_{n_i}(t)}{\nabla E^p_{n_i}(t-1) - \nabla E^p_{n_i}(t)} \Delta w_{n_i,x_j}(t-1) \tag{8}$$

where w_{n_i,x_j} is the change in the weight and t is the iteration number.

A hidden neuron is now created and its weights randomized in the range of -0.5 to 0.5. The correlation of the hidden neuron's output to the error in the neural network is calculated using the following equation:

$$S_{h_j} = \sum_i \left| \sum_p (o^p_{h_j} - \bar{o}_{h_j})(E^p_{n_i} - \bar{E}_{n_i}) \right| \tag{9}$$

where $\bar{o}_{h_j}$ is the average output of the hidden neuron and $\bar{E}_{n_i}$ is the average error. The gradient of the correlation with respect to the hidden node weights is given by:

$$\nabla S_{h_i} = \frac{\partial S_{h_i}}{\partial w_{h_i,x_j}} = \sum_p \sum_i \sigma_{h_i} (E^p_{h_i} - \bar{E}_{h_i}) f'_p I^p_{x_j} \tag{10}$$

where σ_{h_i} is the sign of correlation between $o^p_{h_j}$ and $o^p_{n_i}$. Quickprop is then used to update the hidden node weights to maximize the correlation. The hidden nodes are updated by:

$$\Delta w_{h_i,x_j}(t) = \frac{\nabla S_{h_i}(t)}{\nabla S_{h_i}(t-1) - \nabla S_{h_i}(t)} \Delta w_{h_i,x_j}(t-1) \tag{11}$$

The output neurons are then connected to the new hidden node as shown in Figure 5 and the output weights are once again readjusted to minimize the SSE in the network. This process of optimizing the output weights, creating a hidden neuron, optimizing the hidden neuron weights, connecting it to the output neurons, and adjusting the output neuron weights is repeated until an acceptably small error is produced or a maximum number of nodes is reached. As hidden neurons are added to the network, they are linked to any previously installed hidden neurons in addition to the inputs to the network. This is demonstrated in Figure 6.

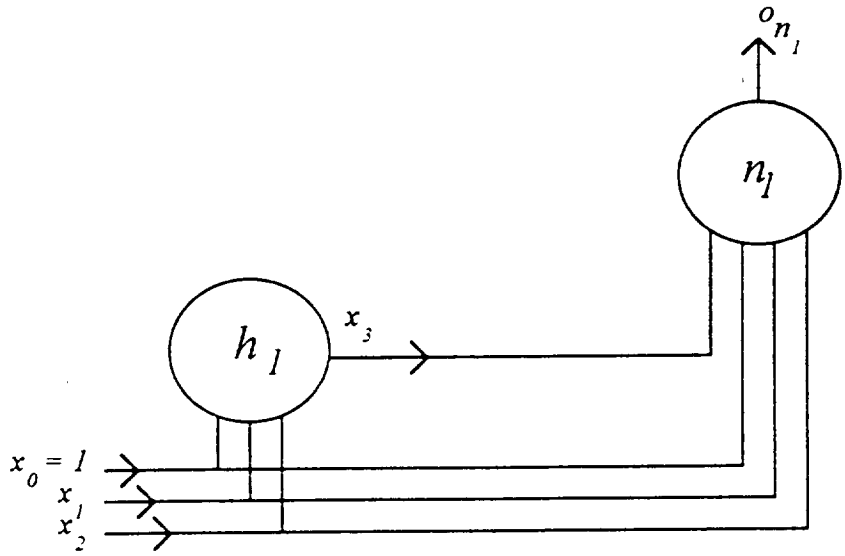

Figure 5. Cascade-correlation architecture with one hidden unit.

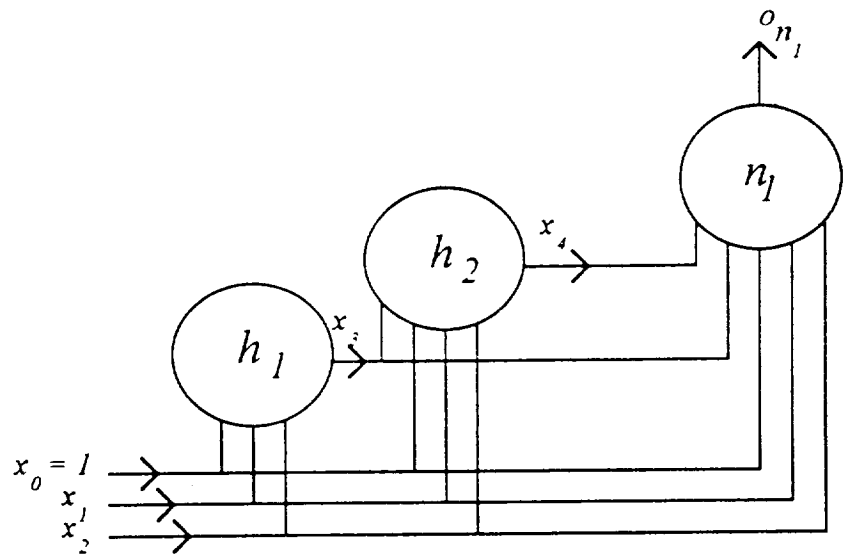

Figure 6. Cascade-Correlation architecture with two hidden units.

Genetic Gradient Cascade-Correlation Learning Architecture

In the genetic gradient cascade-correlation algorithm (**GGrCC**), the genetic algorithm is used to find the weights instead of using Quickprop. However, when genetic algorithms are used in conjunction with the cascade-correlation learning architecture, there is the problem of the genetic algorithm not converging to the optimum set of weights for the neural network. After adding new hidden units, the genetic algorithm could not train the

output weights to improve the performance of the neural network. In an attempt to overcome this problem, the genetic algorithm is first allowed to operate on the weights for a fixed number of generations. The gradients of the errors are used to focus on the range of interest and the genetic algorithm is used on this smaller range.

The genetic gradient algorithm is shown below:

Genetic Gradient Cascade-Correlation Algorithm.

- **(A2a) Set up initial cascade-correlation architecture, Then at step j : j = 0,1,...;**
- **(A2b) Perform genetic search of output neuron weight space using equation (12) for chromosome fitnesses. If search yields no or little improvement, go to (A2e).**
- **(A2c) Compute gradient of output weights using (6).**
- **(A2d) Adjust output weight to chromosome coding using (13) and go to (A2b).**
- **(A2e) Create hidden neuron j.**
- **(A2f) Perform genetic search of weight space of hidden neuron j using equation (9) for chromosome fitnesses. If search yields little or no improvement, go to (A2i).**
- **(A2g) Compute gradients of hidden neuron j using (10)**
- **(A2h) Adjust hidden neuron j weight to chromosome coding using (13) and go to (A2f).**
- **(A2i) Connect hidden unit j to network.**
- **(A2j) If error is acceptable or maximum number of hidden nodes, stop; otherwise return to (A2b).**

The output layer weights are encoded as binary strings and concatenated to form the chromosome to be used by the genetic algorithm. The chromosomes each represent a different set of weights. We chose to use output layer weights randomized in the range of -10 to 10 and we assign the fitness of each chromosome by:

$$fitness(i) = 4 \times p \times o - SSE_i \tag{12}$$

where p is the number of training vectors, o is the number of outputs, and SSE is the sum of the squares of error present in the network where the SSE is given by equation (4). The genetic algorithm utilized the non-overlapping population model.

We further optimized the weights by using information from the gradient in the weight space of the best chromosome to choose a new range for each weight. In general, we allow the population ten generations for convergence. The error gradient for the output

layer weights is calculated from equation (6). We calculate the new range for each weight by:

$$[(w_{oi} + C \cdot \min(\Delta_p w_{oi}))\ldots(w_{oi} + C \cdot \max(\Delta_p w_{oi}))] \tag{13}$$

where C is a constant. We adjusted the range of the weights twice. First, we used the value of C=1000 and adjusted the weight range. We then used the genetic algorithm for ten generations and then readjusted the weight range using C=100 followed by ten more generations of the genetic algorithm. In this way, we are using the gradient to focus the genetic algorithm's search on around the area of convergence instead of wasting time searching the remaining weight space. This greatly improves the optimization of the output weights.

The weights in the hidden neurons are initially randomized in the range of -20 to 20. The correlation given in equation (9) is used as the fitness for the candidate neurons. We allow the population twenty generations for convergence and adjust the range of the hidden nodes weights and using the genetic algorithm again using equation (13) with C=1000 and C=100 and the gradient in equation (10). The output weights are then readjusted, another hidden node created, and the process continues as outlined in the algorithm.

RESULTS

The neural networks are set up and trained to recognize different aspects of the data. The goal is to predict a certain parameter of a specimen when the preprocessed AU signal is applied to the inputs of a neural network. The parameters that are of interest to us are the elastic modulus and the strain that a specimen may have been subjected to.

Due to a limited number of specimens available, the specimens are rotated between training and testing in the following way: one specimen is set aside for testing and the rest are used for training the neural network. Then the next specimen is set aside and the remainder is used for training. This is repeated for all specimens in the data set.

Prediction of Elastic Modulus for $[0]_8$ and $[+\text{-}30]_{2S}$ Laminates

A specimen's elastic modulus, E, may be calculated from the initial gradient of its experimentally obtained stress-strain curve [3]. We attempt here to predict a specimen's elastic modulus directly from its AU measurements. Tables 1a and 1b show the values for E predicted for five specimens of type $E[0]_8$ using GGrCC and backpropagation respectively. Tables 2a and 2b show the values of E predicted for six $[+\text{-}30]_{2S}$ specimens again using GGrCC and backpropagation.

The genetic gradient cascade-correlation network starts off with 32 inputs and one output node. Seven hidden nodes are added to each network during training as shown in Figure 7.

Specimen Number	Actual E (GPa)	Tapered Specimen		Non-tapered Specimen	
		Predicted E (GPa)	Error (GPa)	Predicted E (GPa)	Error (GPa)
1	192	178.9	13.1	188.0	4.0
2	193	179.5	13.5	193.4	0.4
3	183	177.9	5.1	194.5	11.5
4	179	193.6	14.6	195.5	1.9
5	197	193.5	3.5	187.4	9.6
Average Error (GPa)			8.3		4.6

Table 1a. Estimation of E for $[0]_8$ using genetic gradient cascade-correlation

Specimen Number	Actual E (GPa)	Tapered Specimen		Non-tapered Specimen	
		Predicted E (GPa)	Error (GPa)	Predicted E (GPa)	Error (GPa)
1	192	206.8	14.8	191.5	0.4
2	193	184.5	8.5	188.1	4.9
3	183	188.0	5.0	189.7	6.7
4	179	188.5	9.5	192.5	13.5
5	197	188.7	8.3	186.6	10.4
Average Error (GPa)			9.2		7.2

Table 1b. Estimation of E for $[0]_8$ using back propagation

Specimen Number	Actual E (GPa)	Tapered Specimen		Non-tapered specimen	
		Predicted E (GPa)	Error (GPa)	Predicted E (GPa)	Error (GPa)
1	150	151.1	1.1	144.0	6.0
2	150	151.1	1.1	142.8	7.3
3	149	145.7	3.3	142.0	7.0
4	142	141.7	0.3	144.8	2.8
5	141	143.9	2.9	148.6	7.6
6	148	141.3	6.7	142.4	5.6
Average Error (GPa)			2.6		6.1

Table 2a. Estimation of E for $[+\text{-}30]_{2S}$ using genetic gradient cascade-correlation

Specimen Number	Actual E (GPa)	Tapered Specimen		Non-tapered specimen	
		Predicted E (GPa)	Error (GPa)	Predicted E (GPa)	Error (GPa)
1	150	151.6	1.6	148.9	1.1
2	150	149.3	0.7	148.0	2.0
3	149	147.9	1.1	144.8	4.2
4	142	144.8	2.8	147.0	5.0
5	141	144.9	3.9	147.7	6.7
6	148	144.0	4.0	144.1	3.9
Average Error (GPa)			2.4		3.8

Table 2b. Estimation of E for $[+\text{-}30]_{2S}$ using back propagation

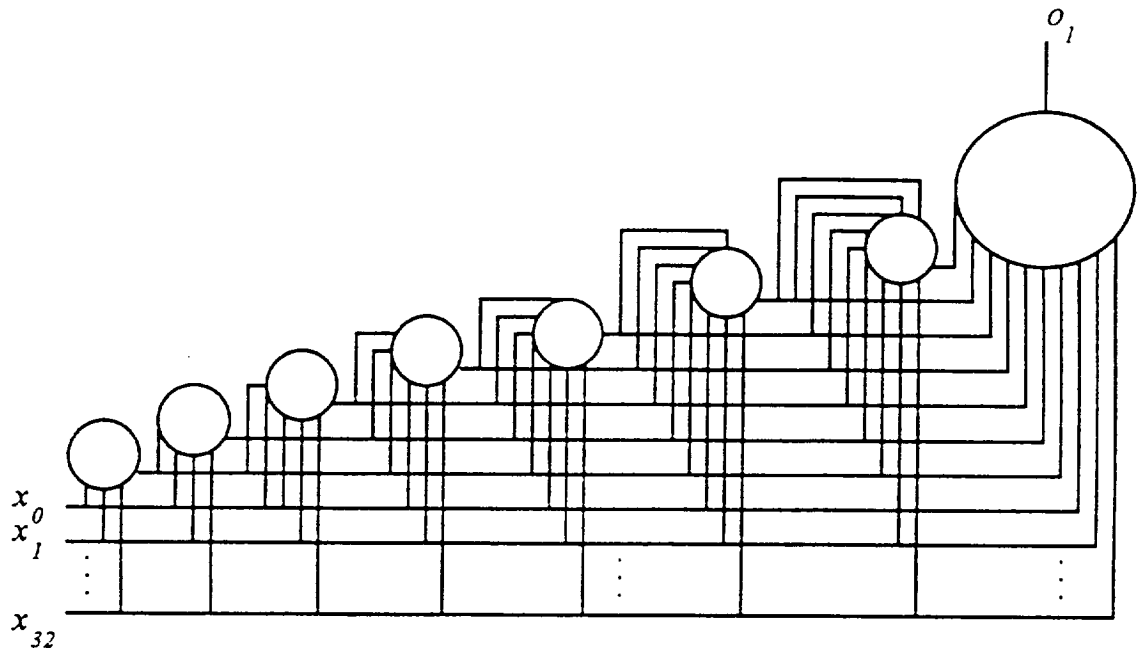

Figure 7. The GGrCC architecture for the AU data.

CONCLUSIONS

The results of this study indicate that a genetically trained neural network using acousto-ultrasonic data can predict the elastic modulus for $[0]_8$ and $[+\text{-}30]_{2S}$ specimens with a maximum error less or equal to 14.6 GPa, and the average error less or equal to 8.3 GPa. The results show that it is possible to predict certain mechanical parameters of a specimen from the fast Fourier transform of its AU waveform without any additional signal processing. Experiments with much larger data sets should be run for a more comprehensive analysis.

REFERENCES

[1] Goldberg, D. E. Genetic Algorithms in Search, Optimizing, and Machine Learning. New York, NY: Addison-Wesley, 1989.

[2] Freeman, J. A., and Skapura, D. M. Neural Networks Algorithms, Applications, and Programming Techniques, Reading, MA: Addison-Wesley, 1991. 17-30.

[3] Kautz, H. E., Lerch, B. A. "Preliminary Investigation of Acousto-Ultrasonic Evaluation of Metal-Matrix Composite Specimens". Materials Evaluation, Vol. 49, No. 5, 1990, pp. 607-612.

[4] Thomsen, J. J., and Lund, K. "Quality Control of Composite Materials by Neural Network Analysis of Ultrasonic Power Spectra". Materials Evaluation, Vol. 49, No. 5, 1990, pp. 564-900.

[5] Vary, A. "Acousto-ultrasonics". Non-destructive Testing of Fiber-Reinforced Plastics Composites, Vol. 2, Elsevier Science, 1990, pp. 1-54.

[6] Kautz, H. E. "Ultrasonic Evaluation of Mechanical Properties Thick, Multilayered, Filament-Wound Composites". Materials Evaluation, Vol. 45, No. 12, 1987, pp. 242-258.

[7] Cios, K. J., Vary, A., Berke, L., and Kautz, H. E. "Application of Neural Networks to Prediction of Advanced Composite Structures Mechanical Response and Behavior". Computing Systems in Engineering, Vol. 3, Nos. 1-4, pp. 539-544.

[8] Fahlman, S.E. and Lebiere, C. "The Cascade-Correlation Learning Architecture." Tech Report CMU-CS-90-100, Carnegie Mellon University, School of Computer Science.