*Int. J. Engng Ed.* Vol. 21, No. 4, pp. 649–667, 2005
Printed in Great Britain.

0949-149X/91 $3.00+0.00
© 2005 TEMPUS Publications.

# Teaching Automated Diagnostic Systems for Doppler Ultrasound Blood Flow Signals to Biomedical Engineering Students using MATLAB*

ELIF DERYA ÜBEYLI
*Department of Electrical and Electronics Engineering, Faculty of Engineering, TOBB Ekonomi ve Teknoloji Üniversitesi, Sögütözü, Ankara, Turkey*

INAN GÜLER
*Department of Electronics and Computer Education, Faculty of Technical Education, Gazi University, 06500 Teknikokullar, Ankara, Turkey. E-mail: iguler@gazi.edu.tr*

*This paper presents an initiative to teach the concept of automated diagnostic systems for Doppler ultrasound blood flow signals to biomedical engineering students. The approach was based on illustrative applications that highlight the performance of multilayer perceptron neural networks (MLPNN) and adaptive neuro-fuzzy inference system (ANFIS). Following a brief description of the artificial neural networks (ANNs) and ANFIS, applications of the models to the Doppler signals obtained from ophthalmic artery and internal carotid artery were done by means of a series of MATLAB functions. The functions involved in the neural network and fuzzy logic toolboxes of MATLAB can be used to develop automated diagnostic systems for the signal under study. The authors suggest that the use of MATLAB exercises will assist the students in gaining a better understanding of the various automated diagnostic systems in blood flow signals.*

## INTRODUCTION

MEDICAL DIAGNOSTIC decision-support systems have become an established component of medical technology. The main concept of the medical technology is an inductive engine that learns the decision characteristics of the diseases and can then be used to diagnose future patients with uncertain disease states. A number of quantitative models including linear discriminant analysis, logistic regression, *k* nearest neighbor, kernel density, recursive partitioning, and neural networks are being used in medical diagnostic support systems to assist human decision-makers in disease diagnosis. The simplest methods, linear discriminant analysis and logistic regression, are based on parametric models with a minimum number of parameters to fit. To capture more complex features of data, a non-parametric method such as *k* nearest neighbor or kernel density is frequently used. If there is a need to model complex nonlinear features of the data, the developer resorts to neural networks, such as the multilayer perceptron (MLP), radial basis function (RBF), self-organizing map (SOM), mixture-of-experts neural architecture. Neural networks have been used in a great number of medical diagnostic decision-support system applications because of the belief that they have greater predictive power. Unfortunately, there is no theory available to guide an intelligent choice of models based on the complexity of the diagnostic task. In most situations, developers are simply picking a single model that yields satisfactory results, or they are benchmarking a small subset of models with cross-validation estimates on test sets [1–3].

Spectral analysis of the Doppler signals produces information concerning the blood flow in the arteries [4–7]. However, artificial neural networks (ANNs) may offer a potentially superior method of Doppler signal analysis to the spectral analysis methods. In contrast to the conventional spectral analysis methods, ANNs not only model the signal, but also make a decision as to the class of signal [8–15]. Furthermore, fuzzy set theory plays an important role in dealing with uncertainty when making decisions in medical applications. Therefore, fuzzy sets have attracted the growing attention and interest in modern information technology, production techniques, decision making, pattern recognition, diagnostics, data analysis, etc [16–18].

Neuro-fuzzy systems are fuzzy systems which use ANNs theory in order to determine their properties (fuzzy sets and fuzzy rules) by processing data samples. Neuro-fuzzy systems harness the power of the two paradigms: fuzzy logic and ANNs, by utilizing the mathematical properties

* Accepted 2 April 2005.

of ANNs in tuning rule-based fuzzy systems that approximate the way people process information. A specific approach in neuro-fuzzy development is the adaptive neuro-fuzzy inference system (ANFIS), which has shown significant results in modeling nonlinear functions. In ANFIS, the membership function parameters are extracted from a data set that describes the system behavior. The ANFIS learns features in the data set and adjusts the system parameters according to a given error criterion [19, 20]. Successful implementations of ANFIS in biomedical engineering have been reported, for classification [15, 21, 22] and data analysis [23]. Two selected illustrative applications among our related studies, which were developed as automated diagnostic systems for Doppler ultrasound blood flow signals, are presented in this paper [12, 15].

In recent years, biomedical engineers have developed many algorithms and processing techniques in order to help doctors in the examination of many different biosignals, and to develop automated diagnostic systems [1–3]. In this respect, automated diagnostic systems for Doppler ultrasound blood flow signals could be introduced both in the graduate and undergraduate biomedical engineering programs. Most engineering students are introduced to MATLAB and the various toolboxes at an early stage of their careers in the general areas of pattern recognition, data classification, simulation, nonlinear system identification, and control. MATLAB is the basic 'engine' with add-on components called toolboxes. MATLAB and its toolboxes allow students to investigate the characteristics of the algorithm and easily design their algorithm with its vast assortment of graphical, signal modeling, and simulation functions. The data or signals generated by any component of the system (software or hardware) can be displayed and/or saved for subsequent use, such as pattern recognition or offline signal modeling.

In this paper, a MATLAB-based approach was proposed and implemented to demonstrate the concept of automated diagnostic systems for Doppler ultrasound blood flow signals, one of the important topics in biomedical engineering. This approach is the topic performed in the biomedical instrumentation course in the Electronics and Computer Education Department, Gazi University. Before delving into the details of how MATLAB is used as a learning tool, it is necessary to understand automated diagnostic systems. Then how MATLAB can be used to reinforce these concepts are discussed. The results obtained from the illustrative applications are presented. Educational contribution of the presented approach is explained.

## THEORETICAL BASIS

### *Artificial neural networks (ANNs)*

ANNs are computational modeling tools that have recently emerged and found extensive acceptance in many disciplines for modeling complex real-world problems. ANNs may be defined as structures comprised of densely interconnected adaptive simple processing elements (neurons) that are capable of performing massively parallel computations for data processing and knowledge representation. The attractiveness of ANNs comes from the remarkable information processing characteristics of the biological system such as nonlinearity, high parallelism, robustness, fault and failure tolerance, learning, ability to handle imprecise and fuzzy information, and their capability to generalize. Artificial models possessing such characteristics are desirable because:

- nonlinearity allows better fit to the data;
- noise-insensitivity provides accurate prediction in the presence of uncertain data and measurement errors;
- high parallelism implies fast processing and hardware failure-tolerance;
- learning and adaptivity allow the system to update (modify) its internal structure in response to changing environment;
- generalization enables application of the model to unlearned data.

ANNs have been utilized in a variety of applications ranging from modeling, classification, pattern recognition and multivariate data analysis [3, 24].

ANNs may be classified in many different ways according to one or more of their relevant features. Generally, classification of ANNs may be used on:

1. The function that the ANN is designed to serve (e.g., pattern association, clustering).
2. The degree (partial/full) of connectivity of the neurons in the network.
3. The direction of flow of information within the network (recurrent and nonrecurrent), with recurrent networks being dynamic systems in which the state at any given time is dependent on previous states.
4. The type of learning algorithm, which represents a set of systematic equations that utilize the outputs obtained from the network along with an arbitrary performance measure to update the internal structure of the ANN.
5. The learning rule (the driving engine of the learning algorithm).
6. The degree of learning supervision needed for ANN training.

Supervised learning involves training of an ANN with the correct answers (i.e., target outputs) being given for every example, and using the deviation (error) of the ANN solution from corresponding target values to determine the required amount by which each weight should be adjusted (Fig. 1). Reinforcement learning is supervised, however the ANN is provided with a critique on correctness of output rather than the correct answer itself.

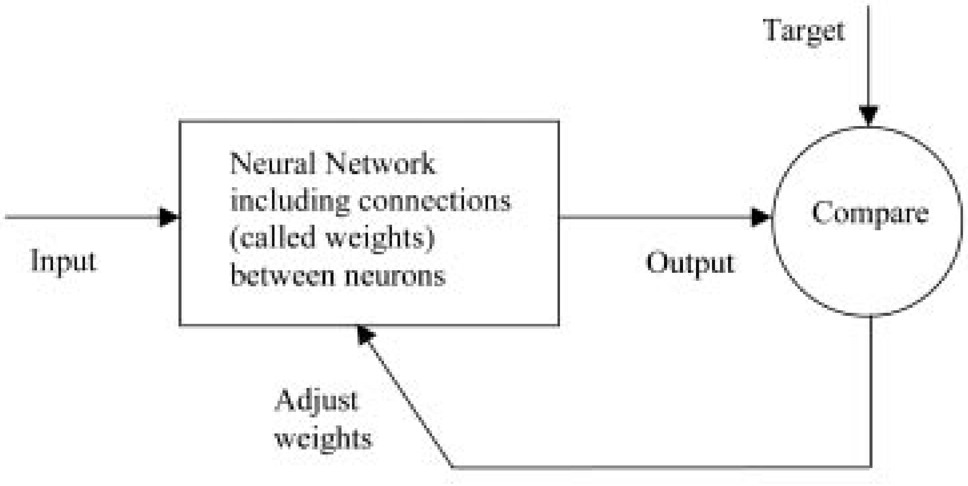

Target
Neural Network including connections (called weights) between neurons
Input
Output
Compare
Adjust weights

Fig. 1. Supervised learning.

Unsupervised learning does not require a correct answer for the training examples, however the network, through exploring the underlying structure in the data and the correlation between the various examples, organizes the examples into clusters (categories) based on their similarity or dissimilarity (e.g., Kohonen networks). Finally, the hybrid learning procedure combines supervised and unsupervised learning [24].

Feedforward neural networks are a basic type of neural networks capable of approximating generic classes of functions, including continuous and integrable ones. An important class of feedforward neural networks is multilayer perceptron neural networks (MLPNNs). In the present study, more emphasis will be given to the MLPNNs as being the most popular and versatile type of networks. The MLPNN, which has features such as the ability to learn and generalize, smaller training set requirements, fast operation, ease of implementation and therefore most commonly used neural network arhitectures, is shown in Fig. 2. As shown in Fig. 2, a MLPNN consists of:

- an input layer with neurons representing input variables to the problem;
- an output layer with neurons representing the dependent variables (what is being modeled);
- one or more hidden layers containing neurons to help capture the nonlinearity in the data.

The MLPNN is a nonparametric technique for performing a wide variety of detection and estimation tasks [10–12, 24, 25]. In the MLPNN, each neuron $j$ in the hidden layer sums its input signals $x_i$ after multiplying them by the strengths of the respective connection weights $w_{ji}$ and computes its output $y_j$ as a function of the sum:

$$y_j = f\left(\sum w_{ji} x_i\right) \tag{1}$$

where $f$ is transfer function that is necessary to transform the weighted sum of all signals impinging onto a neuron. The transfer function $(f)$ can be a simple threshold function, or a sigmoidal, hyperbolic tangent, or radial basis function.

The sum of squared differences between the desired and actual values of the output neurons $E$ is defined as:

$$E = \frac{1}{2}\sum_j (y_{dj} - y_j)^2 \tag{2}$$

where $y_{dj}$ is the desired value of output neuron $j$ and $y_j$ is the actual output of that neuron. Each weight $w_{ji}$ is adjusted to reduce $E$ as rapidly as possible. How $w_{ji}$ is adjusted depends on the training algorithm adopted [10–12, 24, 25].

Training algorithms are an integral part of ANN model development. An appropriate topology may still fail to give a better model, unless trained by a suitable training algorithm. A good training algorithm will shorten the training time, while achieving a better accuracy. Therefore, training process is an important characteristic of the ANNs, whereby representative examples of the knowledge are iteratively presented to the network, so that it can integrate this knowledge within its structure. There are a number of training algorithms used to train a MLPNN and a frequently used one is called the backpropagation training algorithm. The backpropagation algorithm, which is based on searching an error surface using gradient descent for points with minimum error, is relatively easy to implement. However, the backpropagation has some problems for many applications. The algorithm is not guaranteed to find the global minimum of the error function since gradient descent may get stuck in local minima, where it may remain indefinitely. In addition to this, long training sessions are often required in order to find an acceptable weight solution because of the well known difficulties inherent in gradient descent optimization. Therefore, a lot of variations to improve the convergence of the backpropagation were proposed such as delta-bar-delta, extended delta-bar-delta and quick propagation [26–29].

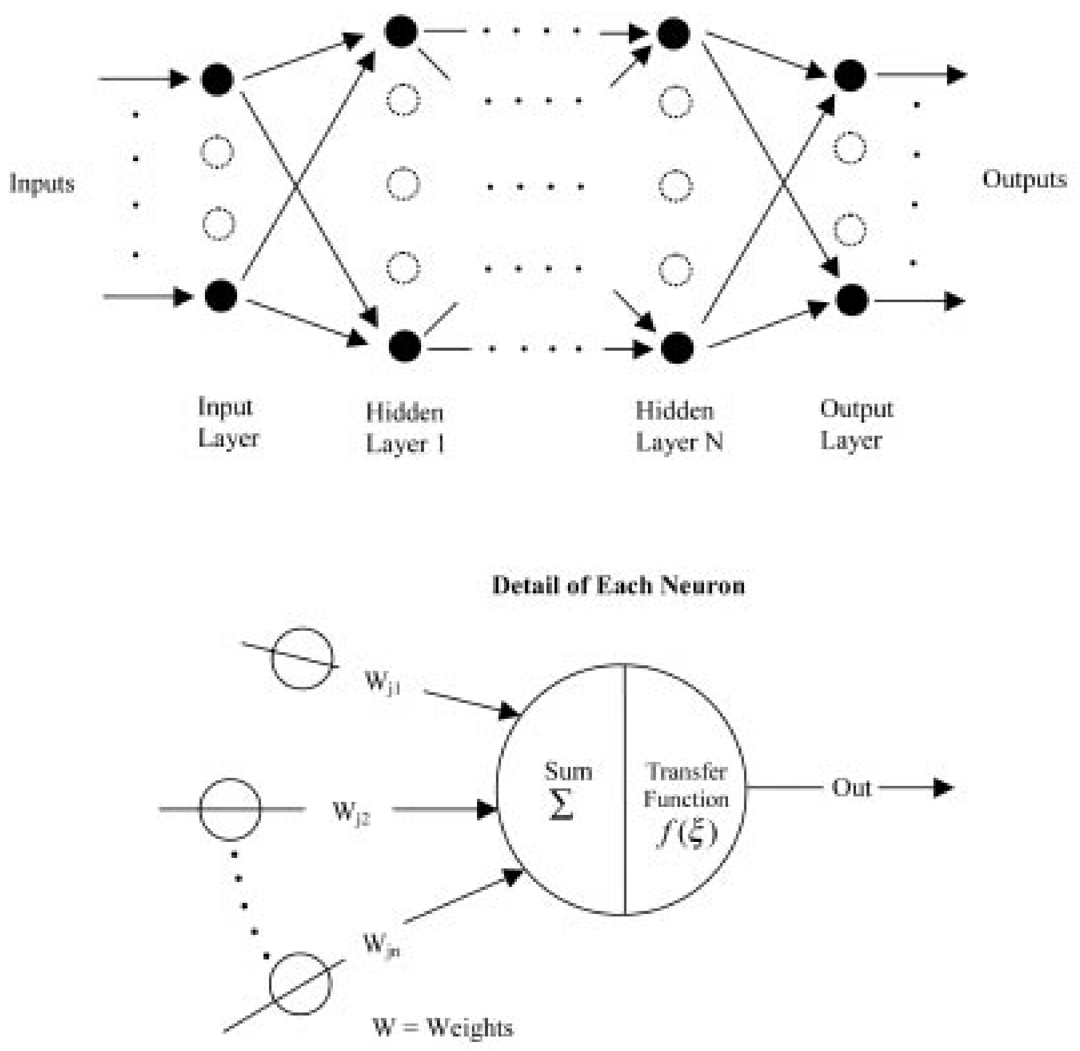


Fig. 2. Multilayer peceptron neural network topology.

Optimization methods such as second-order methods (conjugate gradient, quasi-Newton, Levenberg-Marquardt) have also been used for ANN training in recent years. The Levenberg-Marquardt algorithm combines the best features of the Gauss-Newton technique and the steepest-descent algorithm, but avoids many of their limitations. In particular, it generally does not suffer from the problem of slow convergence [30, 31]. A number of researchers have carried out comparative studies of MLPNN training algorithms [32–34]. The results of the studies have illustrated that the relative performance of algorithms depends on the problem being used. Therefore, in the applications the MLPNNs are trained with different algorithms and the algorithm which gives the best accuracy is selected.

ANN architectures are derived by trial and error and the complexity of the neural network is characterized by the number of hidden layers. There is no general rule for selection of appropriate number of hidden layers. A neural network with a small number of neurons may not be sufficiently powerful to model a complex function. On the other hand, a neural network with too many neurons may lead to overfitting the training sets and lose its ability to generalize which is the main desired characteristic of a neural network. The most popular approach to finding the optimal number of hidden layers is by trial and error. In some applications, after several trials the network architecture which achieved the task in high accuracy can be determined [10–12].

### *Adaptive neuro-fuzzy inference system (ANFIS)*

The ANFIS is a fuzzy Sugeno model put in the framework of adaptive systems to facilitate learning and adaptation [19, 20]. Such a framework makes the ANFIS modeling more systematic and less reliant on expert knowledge. To present the ANFIS architecture, two fuzzy if-then rules based on a first-order Sugeno model are considered:

Rule 1: If ($x$ is $A_1$) and ($y$ is $B_1$) then
($1f_1 = p_1x + q_1y + r_1$)
Rule 2: If ($x$ is $A_2$) and ($y$ is $B_2$) then
($f_2 = p_2x + q_2y + r_2$)

where $x$ and $y$ are the inputs, $A_i$ and $B_i$ are the

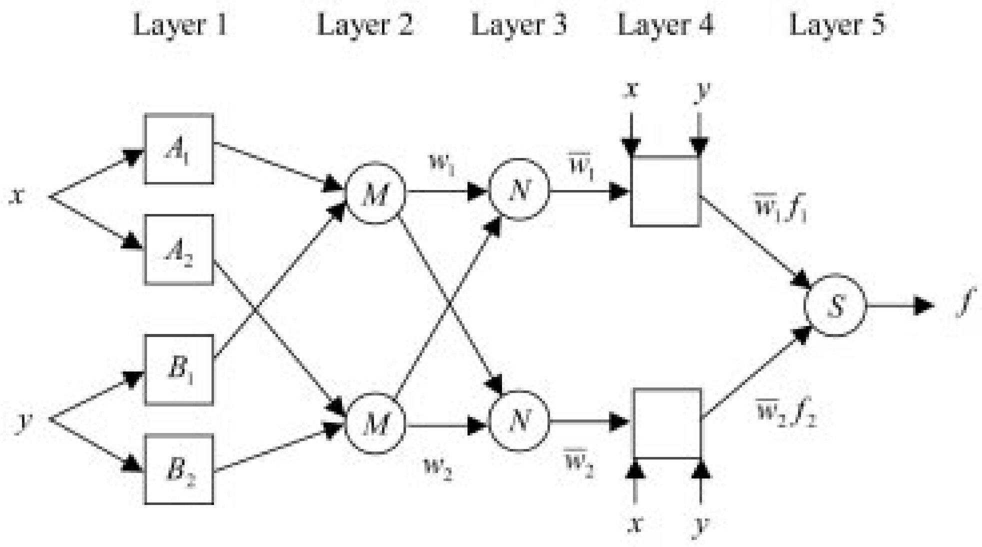


Fig. 3. ANFIS architecture.

fuzzy sets, $f_i$ are the outputs within the fuzzy region specified by the fuzzy rule, $p_i$, $q_i$ and $r_i$ are the design parameters that are determined during the training process. The ANFIS architecture to implement these two rules is shown in Fig. 3, in which a circle indicates a fixed node, whereas a square indicates an adaptive node.

In the first layer, all the nodes are adaptive nodes. The outputs of layer 1 are the fuzzy membership grade of the inputs, which are given by:

$$O_i^1 = \mu_{A_i}(x) \quad i = 1,2 \tag{3}$$

$$O_i^1 = \mu_{B_{i-2}}(y) \quad i = 3,4 \tag{4}$$

where $\mu_{A_i}(x)$, $\mu_{B_{i-2}}(y)$ can adopt any fuzzy membership function. For example, if the bell shaped membership function is employed, $\mu_{A_i}(x)$ is given by:

$$\mu_{A_i}(x) = \frac{1}{1 + \left\{\left(\frac{x - c_i}{a_i}\right)^2\right\}^{b_i}} \tag{5}$$

where $a_i$, $b_i$ and $c_i$ are the parameters of the membership function, governing the bell shaped functions accordingly.

In the second layer, the nodes are fixed nodes. They are labeled with $M$, indicating that they perform as a simple multiplier. The outputs of this layer can be represented as:

$$O_i^2 = w_i = \mu_{A_i}(x)\mu_{B_i}(y) \quad i = 1,2 \tag{6}$$

which are the so-called firing strengths of the rules.

In the third layer, the nodes are also fixed nodes. They are labeled with $N$, indicating that they play a normalization role to the firing strengths from the previous layer.

The outputs of this layer can be represented as:

$$O_i^3 = \overline{w}_i = \frac{w_i}{w_1 + w_2} \quad i = 1,2 \tag{7}$$

which are the so-called normalized firing strengths.

In the fourth layer, the nodes are adaptive nodes. The output of each node in this layer is simply the product of the normalized firing strength and a first-order polynomial (for a first-order Sugeno model). Thus, the outputs of this layer are given by:

$$O_i^4 = \overline{w}_i f_i = \overline{w}_i(p_i x + q_i y + r_i) \quad i = 1,2 \tag{8}$$

In the fifth layer, there is only one single fixed node labeled with $S$. This node performs the summation of all incoming signals. Hence, the overall output of the model is given by:

$$O_i^5 = \sum_{i=1}^{2} \overline{w}_i f_i = \frac{\left(\sum_{i=1}^{2} w_i f_i\right)}{w_1 + w_2} \tag{9}$$

It can be observed that there are two adaptive layers in this ANFIS architecture, namely the first layer and the fourth layer. In the first layer, there are three modifiable parameters $\{a_i, b_i, c_i\}$, which are related to the input membership functions. These parameters are the so-called premise parameters. In the fourth layer, there are also three modifiable parameters $\{p_i, q_i, r_i\}$, pertaining to the first-order polynomial. These parameters are so-called consequent parameters [19, 20].

The task of the learning algorithm for this architecture is to tune all the modifiable parameters, namely $\{a_i, b_i, c_i\}$ and $\{p_i, q_i, r_i\}$, to make the ANFIS output match the training data. When the premise parameters $a_i$, $b_i$ and $c_i$ of the membership function are fixed, the output of the ANFIS model can be written as:

$$f = \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2 \tag{10}$$

Substituting equation (7) into equation (10) yields:

$$f = \overline{w}_1 f_1 + \overline{w}_2 f_2 \tag{11}$$

Substituting the fuzzy if-then rules into equation (11), it becomes:

$$f = \overline{w}_1(p_1 x + q_1 y + r_1) + \overline{w}_2(p_2 x + q_2 y + r_2) \tag{12}$$

After rearrangement, the output can be expressed as:

$$\begin{aligned} f &= (\overline{w}_1 x)p_1 + (\overline{w}_1 y)q_1 + (\overline{w}_1)r_1 \\ &\quad + (\overline{w}_2 x)p_2 + (\overline{w}_2 y)q_2 + (\overline{w}_2)r_2 \end{aligned} \tag{13}$$

which is a linear combination of the modifiable consequent parameters $p_1$, $q_1$, $r_1$, $p_2$, $q_2$ and $r_2$. The least squares method can be used to identify the optimal values of these parameters easily. When the premise parameters are not fixed, the search space becomes larger and the convergence of the training becomes slower. A hybrid algorithm combining the least squares method and the gradient descent method is adopted to solve this problem. The hybrid algorithm is composed of a forward pass and a backward pass. The least squares method (forward pass) is used to optimize the consequent parameters with the premise parameters fixed. Once the optimal consequent parameters are found, the backward pass starts immediately. The gradient descent method (backward pass) is used to adjust optimally the premise parameters corresponding to the fuzzy sets in the input domain. The output of the ANFIS is calculated by employing the consequent parameters found in the forward pass. The output error is used to adapt the premise parameters by means of a standard backpropagation algorithm. It has been proven that this hybrid algorithm is highly efficient in training the ANFIS [19, 20].

## ILLUSTRATIVE APPLICATIONS

The data acquisition system used in the illustrative applications had five components as shown in Fig. 4. These are ultrasonic transducer (5 MHz for internal carotid artery, 10 MHz for ophthalmic artery), analog Doppler unit, analog tape recorder (Sony), analog/digital interface board (Sound Blaster Pro-16 bit), a personal computer with a printer. The analog Doppler unit was equipped with imaging facility that made it possible to focus the sample volume at a desired location. The beam of ultrasound transfixed the vessel axis at angle of around 60°. The output of the analog Doppler unit was transferred to a PC via a 16-bit sound card on an analog/digital interface board.

### *Classification results of automated diagnostic systems*

Performance indicators of automated diagnostic systems:

- *Measuring error:* Given a random set of initial weights, the outputs of the network will be very different from the desired classifications. As the network is trained, the weights of the system are continually adjusted to reduce the difference between the output of the system and the desired response. The difference is referred to as the error and can be measured in different ways. The most common measurement is the mean square error (MSE). The MSE is the average of the squares of the difference between each output and the desired output. In addition to MSE, normalized mean squared error (NMSE), mean absolute error (MAE), minimum absolute error and maximum absolute error can be used for the measuring error of the neural network [10–12].
- *Cross validation:* Cross validation is a highly recommended criterion for stopping the training of a network. During performance analysis of network, cross validation can be used for determining the final training. In general, it is known that a network with enough weights will always learn the training set better as the number of iterations is increased. However, neural network researchers have found that this decrease in the training set error was not always coupled to better performance in the test. When the network is trained too much, the network memorizes the training patterns and does not generalize well. The training holds the key to an accurate solution, so the criterion to stop training must be very well described. The aim of the stop criterion is to maximize the network's generalization [10–12].
- *Classification and regression:* Neural networks are used for both classification and regression.

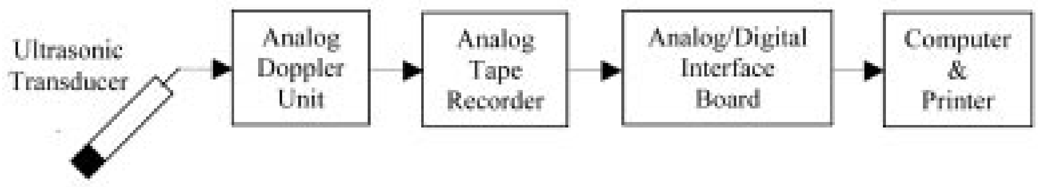


Fig. 4. Block diagram of the measurement system.

In classification, the aim is to assign the input patterns to one of several classes, usually represented by outputs restricted to lie in the range from 0 to 1, so that they represent the probability of class membership. While the classification is carried out, a specific pattern is assigned to a specific class according to the characteristic features selected for it. In regression, desired output and actual network output results can be shown on the same graph and performance of network can be evaluated in this way [10–12].

- *Confusion matrix:* A confusion matrix is a simple methodology for displaying the classification results of a network. The confusion matrix is defined by labeling the desired classification on the rows and the actual network outputs on the columns. For each exemplar, a 1 is added to the cell entry defined by desired classification and the actual network outputs. Since the actual network outputs and the desired classification wanted to be the same, the ideal situation is to have all the exemplars end up on the diagonal cells of the matrix (the diagonal that connects the upper-left corner to the lower right) [10–12].
- *Specificity, sensitivity, and receiver operating characteristic curve analysis:* The simplest classification problem is that of separating one-dimensional feature vectors into two groups. In this situation the only choice that needs to be made is where to locate the decision threshold. If there is no overlap between the magnitudes of the vectors obtained from patients belonging to the two classes, the threshold can simply be chosen to separate the classes completely. In general, the results from the two classes do overlap and so depending on where the threshold is placed some signals from normal subjects will be adjudged abnormal and/or some signals from abnormals will be adjudged normal. The best choice of threshold will then depend on a number of factors. There are three important measures of the performance of a diagnostic test; specificity, sensitivity, and total classification accuracy which are defined as:
- *Specificity:* number of true negative decisions/ number of actually negative cases
- *Sensitivity:* number of true positive decisions/ number of actually positive cases
- *Total classification accuracy:* number of correct decisions/total number of cases. A true positive decision occurs when the positive detection of the network coincided with a positive detection of the physician. A true negative decision occurs when both the network and the physician suggested the absence of a positive detection. These measures are dependent since they are both affected by the position of the decision threshold and as the threshold is moved to increase sensitivity, so specificity decreases. Receiver operating characteristic (ROC) curve is one of the best method of evaluating the performance of a test and defining an appropriate decision threshold. The best choice of threshold will then depend on a number of factors including the consequences of making both types of false classification (false positive and false negative) and the prevalence of disease in the target population [10–12].
- *Correlation coefficient:* The size of MSE can be used to determine how well the network output fits the desired output, but it may not reflect whether the two sets of data move in the same direction. The correlation coefficient ($r$) solves this problem. The correlation coefficient is limited with the range [–1, 1]. When $r - 1$ there is a perfect positive linear correlation between network output and desired output, which means that they vary by the same amount. When $r = -1$ there is a perfectly linear negative correlation between network output and desired output, that means they vary in opposite ways (when network output increases, desired output decreases by the same amount). When $r = 0$ there is no correlation between network output

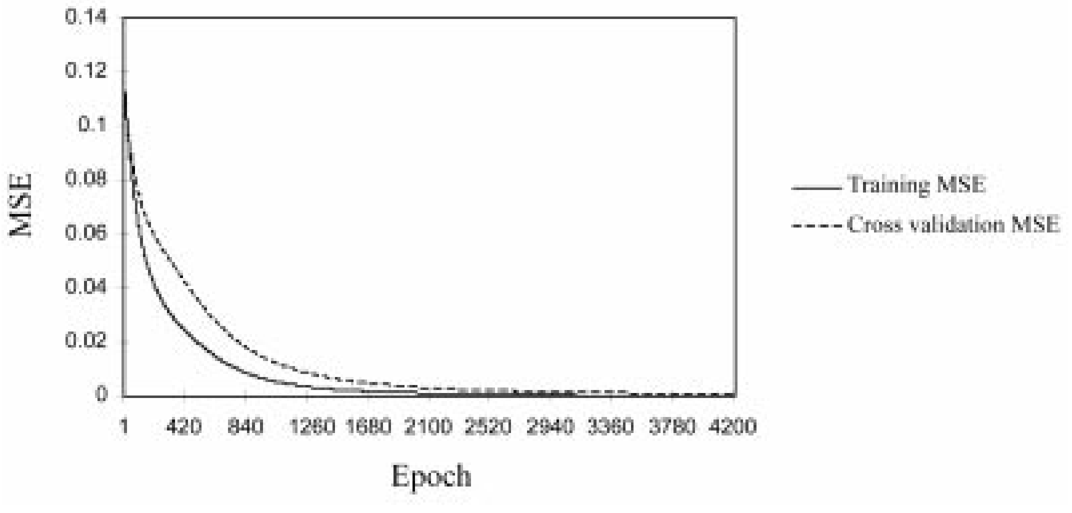


Fig. 5. Training and cross-validation MSE curves of the MLPNN.

Table 1. The values of minimum and final MSE during training and cross-validation

| Best networks | Training | Cross validation |
|---|---|---|
| Epoch number | 4200 | 4200 |
| Minimum MSE | 0.000256223 | 0.000360517 |
| Final MSE | 0.000256223 | 0.000360517 |

and desired output (the variables are called uncorrelated). Intermediate values describe partial correlations [10–12].

*MLPNN for detection of ophthalmic arterial doppler signals with Behcet disease.*

In the first illustrative application, the MLPNN employing delta-bar-delta training algorithm was used for the interpretation of the ophthalmic artery Doppler waveforms [12]. The MLPNN for detection the presence of ocular Behcet disease was implemented by using the Neural Network Toolbox of MATLAB. In order to determine the MLPNN inputs spectral analysis of the ophthalmic arterial Doppler signals was performed using the least squares autoregressive method. The MLPNN employing delta-bar-delta was trained with the training set, cross-validated with the cross-validation set and checked with the test set. In Fig. 5, the error in training set and the cross-validation set is shown on the same graph. The values of minimum MSE and final MSE during training and cross-validation are given in Table 1. As it is seen from Table 1, training was done in 4200 epochs since the cross-validation error began to rise at 4200 epochs. Training of the MLPNN was determined as successful owing to MSE (Fig. 5) converged to a small constant approximately zero in 4200 epochs.

Table 2. The values of statistical parameters of the MLPNN

| Statistical parameters | Values |
|---|---|
| Specificity | 96.43% |
| Sensitivity | 93.75% |
| Total classification accuracy | 95.00% |

Classification results of the MLPNN were displayed by a confusion matrix. According to confusion matrix, 1 healthy subject was classified incorrectly by the MLPNN as a subject suffering from ocular Behcet disease and 2 subjects suffering from ocular Behcet disease were classified as healthy subjects. Confusion matrix:

| | Output result | |
|---|---|---|
| Desired result | Healthy | Behcet disease |
| Healthy | 27 | 2 |
| Behcet disease | 1 | 30 |

The values of the specificity, sensitivity, and total classification accuracy are given in Table 2. The MLPNN classified healthy subjects and subjects suffering from ocular Behcet disease with an accuracy of 96.43% and 93.75%, respectively. The healthy subjects and subjects suffering from ocular Behcet disease were classified with an accuracy of 95.00%. The performance of a test can be evaluated by plotting a ROC curve for the test. A good test (curve in Fig. 6) is one for which

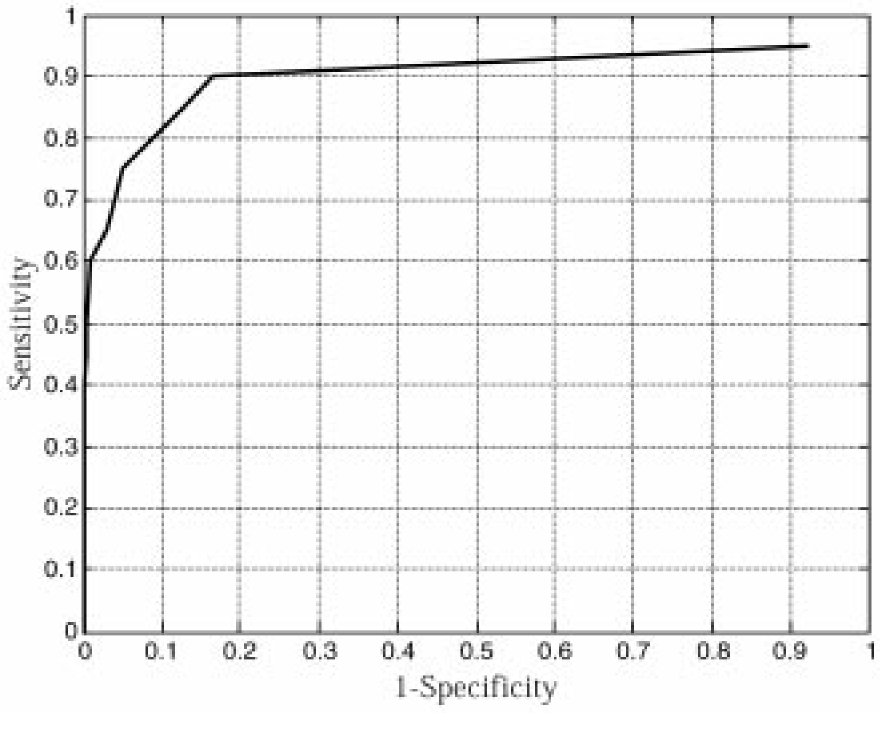


Fig. 6. ROC curve.

Table 3. The values of performance evaluation parameters

| Performance | Result (healthy) | Result (ocular Behcet disease) |
|---|---|---|
| MSE | 0.003565628 | 0.003710403 |
| NMSE | 0.315698741 | 0.372141745 |
| MAE | 0.095475511 | 0.099590421 |
| Minimum absolute error | 0.007361279 | 0.007595658 |
| Maximum absolute error | 0.918734692 | 0.921097517 |
| r | 0.946251742 | 0.924021453 |
| Percent correct | 96.42857361 | 93.75011523 |

sensitivity rises rapidly and 1-specificity hardly increases at all until sensitivity becomes high. ROC curve which is shown in Fig. 6 represents the MLPNN performance on the test file. The values of performance evaluation parameters of the presented MLPNN are given for healthy subjects and subjects suffering from ocular Behcet disease in Table 3. The results of performance evaluation and statistical measures, showed that the MLPNN employing delta-bar-delta training algorithm was effective to detect the ophthalmic arterial Doppler signals with Behcet disease [12].

### *ANFIS for analysis of internal carotid arterial doppler signals*

In the second illustrative application, a new approach based on ANFIS was presented for the detection of internal carotid artery stenosis and occlusion [15]. Three ANFIS classifiers were used to detect internal carotid artery conditions (normal, stenosis and occlusion) when two features, resistivity and pulsatility indices (RI and PI), defining changes of internal carotid arterial Doppler waveforms were used as inputs. The ANFIS classifiers were implemented by using the Fuzzy Logic Toolbox of MATLAB. We trained the three ANFIS classifiers since there were three possible outcomes of the diagnosis of internal carotid artery conditions (normal, stenosis and occlusion). Each of the ANFIS classifier was trained so that they are likely to be more accurate for one type of internal carotid artery condition than the other conditions.

As we have mentioned in our previous study [11], it is difficult to separate the normal, stenosis and occlusion subjects using the values of the RI and PI. Since there is a considerable overlap in the RI and PI values of normal, stenosis and occlusion groups, 129 points of the logarithm of the internal carotid artery Doppler spectrum were used as the MLPNN inputs in our previous study [11]. However, fuzzy set theory plays an important role in dealing with uncertainty when making decisions in medical applications. Therefore, we chose fuzzy logic in this application due to the uncertainty in internal carotid arterial Doppler signals classification, which is a result of imprecise boundaries between RI and PI values of normal, stenosis and occlusion groups. Using fuzzy logic enabled us to use the uncertainty in the classifier design and consequently to increase the credibility of the system output. We trained the fourth ANFIS classifier to combine the predictions of the three ANFIS classifiers. The outputs of the three ANFIS classifiers were used as the inputs of the fourth ANFIS classifier. Since the number of inputs used in the ANFIS model was much less than the number of inputs used in the MLPNN presented in our previous study [11], the response time of ANFIS model was much less than that of the MLPNN.

Figure 7 shows the fuzzy rule architecture of each ANFIS using a generalized bell-shaped membership function defined in Equation (5). There are a total of 27 fuzzy rules in the architecture. Figure 8 shows the fuzzy rule architecture of the combining ANFIS classifier (the fourth ANFIS classifier) using a generalized bell-shaped membership function. There are a total of 30 fuzzy rules in the architecture. Membership function of each input parameter was divided into three regions, namely, small, medium, and large. Matlab function, `gbellmf` (generalized bell curve membership function), was used to generate membership functions. The examination of initial and final membership functions indicates that there are considerable changes in the final membership functions of RI (input 1) and PI (input 2). Figure 9 shows the initial and final membership functions of the RI values using the generalized bell-shaped membership function. This

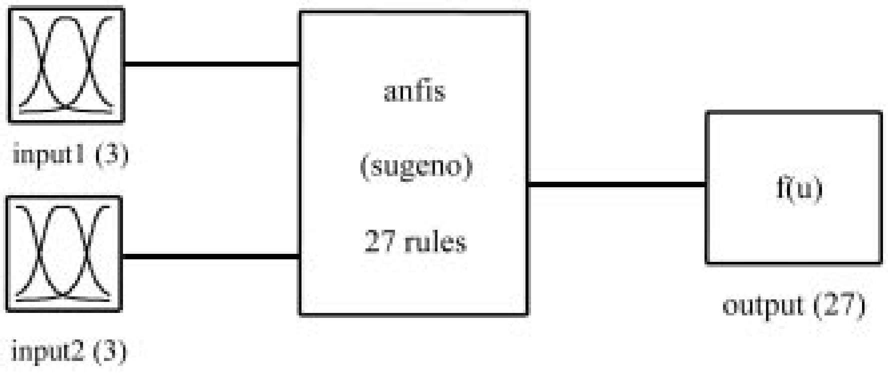


Fig. 7. Fuzzy rule architecture of the three ANFIS classifiers. System ANFIS: 2 inputs, 1 output, 27 rules.

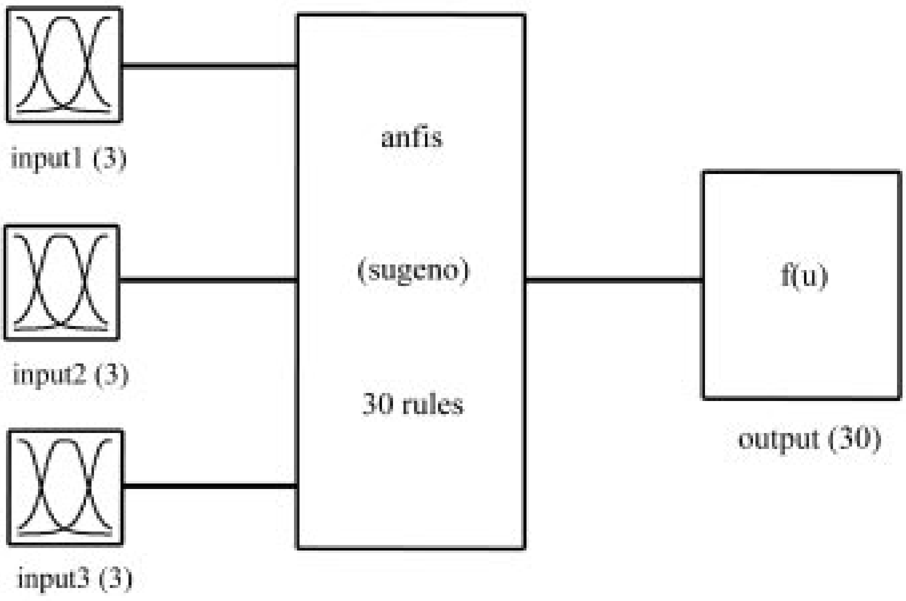


Fig. 8. Fuzzy rule architecture of the combining ANFIS (the fourth ANFIS classifier). System ANFIS: 3 inputs, 1 output, 30 rules.

analysis was done since the amount of changes in the final membership functions of inputs indicates the impact of inputs on the detection of output. Based on the analysis of membership functions, it can be mentioned that both RI and PI values have considerable impact on the detection of internal carotid artery stenosis and occlusion.

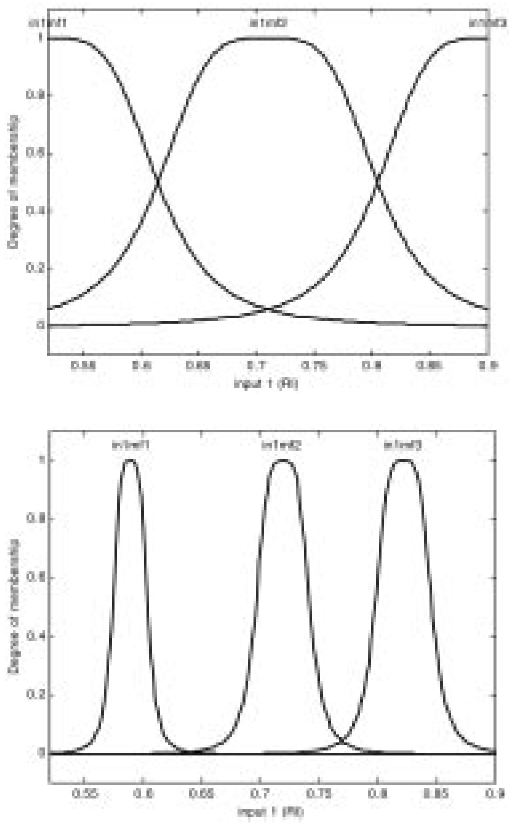


Fig. 9. (a) Initial and (b) final generalized bell shaped membership function of input 1 (RI).

Classification results of the ANFIS model were displayed by a confusion matrix. The confusion matrix showing the classification results of the ANFIS model is given below. According to the confusion matrix, 1 normal subject was classified incorrectly by the ANFIS model as a subject suffering from stenosis, 1 subject suffering from stenosis was classified as a normal subject, 1 subject suffering from occlusion was classified as a subject suffering from stenosis. The test performance of the ANFIS model was determined by the computation of the statistical parameters. The values of the statistical parameters are given in Table 4. As it is seen from Table 4, the ANFIS model classified normal subjects and subjects suffering from stenosis and occlusion with an accuracy of 96.43%, 96.77%, 96.55%, respectively. The normal subjects, subjects suffering from stenosis and occlusion, were classified with an accuracy of 96.59%. The correct classification rates of the standalone neural network (MLPNN) presented in our previous study [11] were 95.24% for normal subjects, 91.30% for subjects suffering from stenosis and 91.67% for subjects suffering from occlusion. The total classification accuracy of the standalone neural network was 92.65%. Thus, the accuracy rates of the ANFIS model presented for this application were found to be higher than that of the standalone neural network model.

Table 4. The values of statistical parameters of the ANFIS model

| Statistical parameters | Values |
|---|---|
| Specificity | 96.43% |
| Sensitivity (stenosis) | 96.77% |
| Sensitivity (occlusion) | 96.55% |
| Total classification accuracy | 96.59% |

These results indicate that the proposed ANFIS model has some potential in detecting internal carotid artery stenosis and occlusion [15]. Confusion matrix:

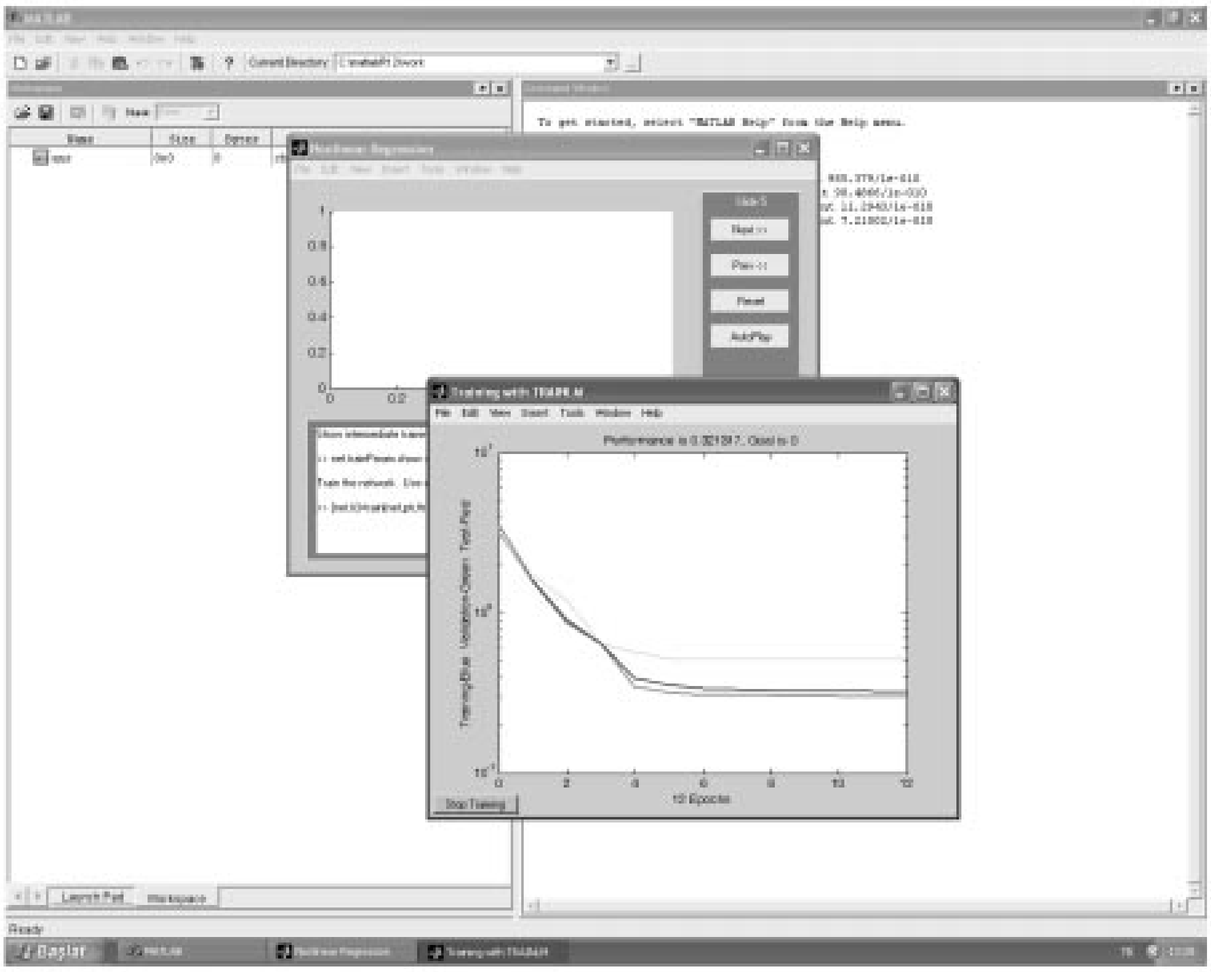
| | Output result | | |
|---|---|---|---|
| Desired result | Normal | Stenosis | Occlusion |
| Normal | 27 | 1 | 0 |
| Stenosis | 1 | 30 | 1 |
| Occlusion | 0 | 0 | 28 |

## NEURAL NETWORK TOOLBOX OF MATLAB

We introduced the students to the basic functions of the Neural Network Toolbox of MATLAB. Command window of MATLAB taken from students' projects related with neural network analysis of Doppler ultrasound blood flow signals is presented in Fig. 10. The Neural Network Toolbox extends the MATLAB computing environment to provide tools for the design, implementation, visualization and simulation of neural networks [35]. Neural networks are uniquely powerful tools in applications where formal analysis would be difficult or impossible, such as pattern recognition and nonlinear system identification and control. The Neural Network Toolbox provides comprehensive support for many proven network paradigms, as well as a graphical user interface (GUI) that allows the students to design and manage their networks. The toolbox's modular, open, and extensible design simplifies the creation of customized functions and networks. Because neural networks require intensive matrix computations, MATLAB provides a natural framework for rapidly implementing neural networks and for studying their behavior and application.

*Neural network toolbox GUI*

This tool lets the students import potentially large and complex data sets. The GUI also allows the students to create, initialize, train, simulate and manage their networks. Simple graphical representations allow the students to visualize and understand network architecture.



```
To get started, select "MATLAB Help" from the Help menu.
```

Fig. 10. Command window of MATLAB taken from students' projects related with neural network analysis of Doppler ultrasound blood flow signals

*Supervised networks*

Supervised neural networks are trained to produce desired outputs in response to example inputs, making them particularly well suited for modeling and controlling dynamic systems, classifying noisy data and predicting future events. The Neural Network Toolbox supports the following supervised networks:

- *Feed-forward networks* have one-way connections from input to output layers. They are commonly used for prediction, pattern recognition, and nonlinear function fitting. Supported feed-forward networks include feed-forward backpropagation, cascade-forward backpropagation, feed-forward input-delay backpropagation, linear, and perceptron networks.
- *Radial basis networks* provide an alternative fast method for designing nonlinear feed-forward networks. Supported variations include generalized regression and probabilistic neural networks.
- *Recurrent networks* use feedback to recognize both spatial and temporal patterns. Supported recurrent networks include Elman and Hopfield.
- *Learning vector quantization (LVQ)* is a powerful method for classifying patterns that are not linearly separable. LVQ allows the students to specify class boundaries and the granularity of classification.

In this section, we want to present the description of the MLPNN (Fig. 2) which is the most commonly used neural network with the backpropagation algorithm. Each input is weighted with an appropriate $w$. The sum of the weighted inputs forms the input to the transfer function $f$. Neurons may use any differentiable transfer function $f$ to generate their output. MLPNNs often use the log-sigmoid transfer function `logsig`, which is shown in Fig. 11(a). The function `logsig` generates outputs between 0 and 1 as the neuron's net input goes from negative to positive infinity.

Alternatively, MLPNNs may use the tan-sigmoid transfer function `tansig`, which is shown in Fig. 11(b). Occasionally, the linear transfer function `purelin` is used in backpropagation networks (Fig. 11(c)). If the last layer of a MLPNN has sigmoid neurons, then the outputs of the network are limited to a small range. If linear output neurons are used the network outputs can take on any value. In backpropagation it is important to be able to calculate the derivatives of any transfer functions used. Each of the transfer functions above, `tansig`, `logsig`, and `purelin`, have a corresponding derivative function: `dtansig`, `dlogsig` and `dpurelin`.

*Unsupervised networks*

Unsupervised neural networks are trained by letting the network continually adjust itself to new inputs. They find relationship within data as it is presented and can automatically define classification schemes. The Neural Network Toolbox supports two types of self-organizing unsupervised networks:

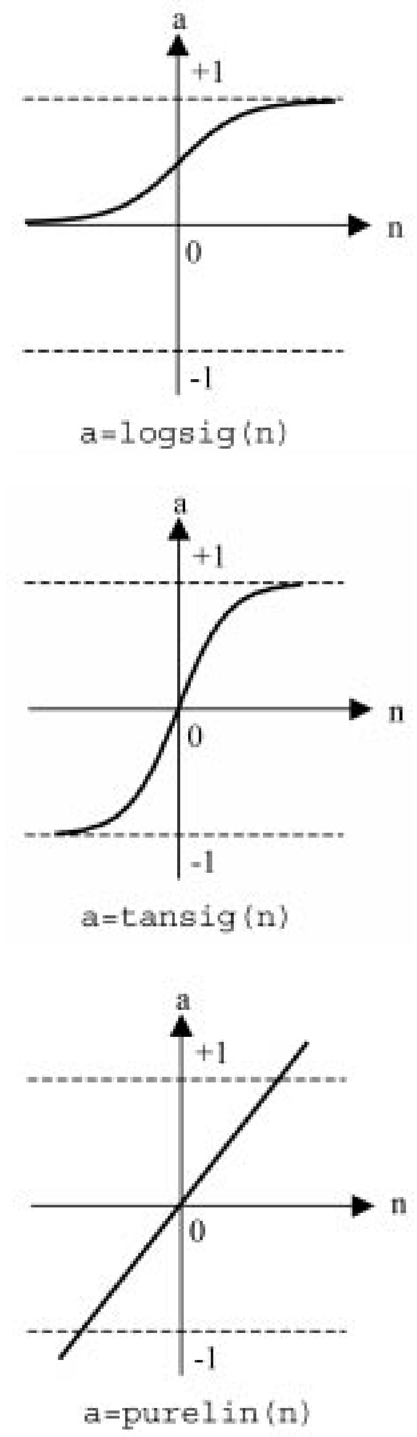


Fig. 11. Transfer functions: (a) log-sigmoid transfer function, (b) tan-sigmoid transfer function, (c) linear transfer function.

1. *Competitive layers* recognize and group similar input vectors. By using these groups, the network automatically sorts the input categories.
2. *Self-organizing maps* learn to classify input vectors according to similarity. Unlike competitive layers, they also preserve the topology of the input vectors, assigning nearby inputs to nearby categories.

Table 5. Supported training functions

| Function | Purpose |
|---|---|
| `trainb` | Batch training with weight and bias learning rules |
| `trainbfg` | BFGS quasi-Newton backpropagation |
| `trainbr` | Bayesian regularization |
| `trainc` | Cyclical order incremental update |
| `traincgb` | Powell-Beale conjugate gradient backpropagation |
| `traincgf` | Fletcher-Powell conjugate gradient backpropagation |
| `traincgp` | Polak-Ribiere conjugate gradient backpropagation |
| `traingd` | Gradient descent backpropagation |
| `traingda` | Gradient descent with adaptive learning rate (lr) backpropagation |
| `traingdm` | Gradient descent with momentum backpropagation |
| `traingdx` | Gradient descent with momentum & adaptive lr backpropagation |
| `trainlm` | Levenberg-Marquardt backpropagation |
| `trainoss` | One step secant backpropagation |
| `trainr` | Random order incremental update |
| `trainrp` | Resilient backpropagation (Rprop) |
| `trains` | Sequential order incremental update |
| `trainscg` | Scaled conjugate gradient backpropagation |

## SUPPORTED TRAINING AND LEARNING FUNCTIONS

Training and learning functions are mathematical procedures used to automatically adjust the network's weights and biases. The training function dictates a global algorithm that affects all the weights and biases of a given network. The learning function can be applied to individual weights and biases within a network. Supported training and learning functions are presented in Tables 5 and 6, respectively.

### *Creating a network*

The first step in training a feedforward network is to create the network object. The function `newff` creates a trainable feedforward network. It requires four inputs and returns the network object. The first input is an R × 2 matrix of minimum and maximum values for each of the R elements of the input vector. The second input is an array containing the sizes of each layer. The third input is a cell array containing the names of the transfer functions to be used in each layer. The final input contains the name of the training function to be used. For example, the following command will create a two-layer network. There will be one input vector with two elements, three neurons in the first layer and one neuron in the second (output) layer. The transfer function in the first layer will be tan-sigmoid, and the output layer transfer function will be linear. The values for the first element of the input vector will range between –1 and 2, the values of the second element of the input vector will range between 0 and 5, and the training function will be `traingd`:

```
net=newff([-1 2; 0 5], [3, 1],
  {'tansig','purelin'},'traingd');
```

Table 6. Supported learning functions

| Function | Purpose |
|---|---|
| `learncon` | Conscience bias learning function |
| `learngd` | Gradient descent weight/bias learning function |
| `learngdm` | Gradient descent with momentum weight/bias learning function |
| `learnh` | Hebb weight learning function |
| `learnhd` | Hebb with decay weight learning rule |
| `learnis` | Instar weight learning function |
| `learnk` | Kohonen weight learning function |
| `learnlv1` | LVQ1 weight learning function |
| `learnlv2` | LVQ2 weight learning function |
| `learnos` | Outstar weight learning function |
| `learnp` | Perceptron weight and bias learning function |
| `learnpn` | Normalized perceptron weight and bias learning function |
| `learnsom` | Self-organizing map weight learning function |
| `learnwh` | Widrow-Hoff weight and bias learning rule |

This command creates the network object and also initializes the weights and biases of the network; therefore the network is ready for training. There are times when the students may wish to re-initialize the weights, or to perform a custom initialization. The details of the initialization process is explained below.

### *Initializing weights*

Before training a feedforward network, the weights and biases must be initialized. The initial weights and biases are created with the command `init`. This function takes a network object as input and returns a network object with all weights and biases initialized. Here is how a network is initialized:

```
net=init(net);
```

The specific technique which is used to initialize a given network will depend on how the network parameters `net.initFcn` and `net.layers{i}. initFcn` are set. The parameter `net.initFcn` is used to determine the overall initialization function for the network. The default initialization function for the feedforward network is `initlay`, which allows each layer to use its own initialization function. With this setting for `net.initFcn`, the parameters `net.layers{i}.initFcn` are used to determine the initialization method for each layer. For feedforward networks there are two different layer initialization methods which are normally used: `initwb` and `initnw`. The `initwb` function causes the initialization to revert to the individual initialization parameters for each weight matrix (`net.inputWeights{i,j}.initFcn`) and bias. For the feedforward networks the weight initialization is usually set to `rands`, which sets weights to random values between –1 and 1. It is normally used when the layer transfer function is linear. The function `initnw` is normally used for layers of feedforward networks where the transfer function is sigmoid. It generates initial weight and bias values for a layer so that the active regions of the layer's neurons will be distributed roughly evenly over the input space. It has several advantages over purely random weights and biases: (1) few neurons are wasted (since the active regions of all the neurons are in the input space), (2) training works faster (since each area of the input space has active neuron regions).

The initialization function `init` is called by `newff`, therefore the network is automatically initialized with the default parameters when it is created, and `init` does not have to be called separately. However, the user may want to re-initialize the weights and biases, or to use a specific method of initialization. For example, in the network that we just created, using `newff`, the default initialization for the first layer would be `initnw`. If we wanted to re-initialize the weights and biases in the first layer using the `rands` function, we would issue the following commands:

```
net.layers{1}.initFcn=
  'initwb';
net.inputWeights{1,1}.initFcn='-
  rands';
net.biases{1,1}.initFcn='rands';
net.biases{2,1}.initFcn='rands';
net=init(net);
```

*Simulink support*

Once a network has been created and trained, it can be easily incorporated into Simulink models. A simple command (`gensim`) automatically generates network simulation blocks for use with Simulink. This feature also makes it possible for the students to view their networks graphically [35]. Here is how a Simulink block for neural network simulation is generated:

```
gensim(net,st)
```

This command creates a Simulink system containing a block which simulates neural network `net`. The command takes these inputs: `net` = Neural network; `st` = Sample time (default = 1) and creates a Simulink system containing a block which simulates neural network `net` with a sampling time of `st`. If `net` has no input or layer delays, `net.numInputDelays` and `net.numLayerDelays` are both 0; then you can use –1 for `st` to get a continuously sampling network. An example is given below:

```
net=newff( [0 1], [5 1] );
gensim(net)
```

*Pre- and post-processing functions*

Pre-processing the network inputs and targets improves the efficiency of neural network training. Post-processing enables detailed analysis of network performance. The Neural Network Toolbox provides the following pre- and post-processing functions:

- Principal component analysis reduces the dimensions of the input vectors
- *Post-training analysis* performs a regression analysis between the network response and the corresponding targets.
- *Scale minimum and maximum* scales inputs and targets so that they fall in the range [–1,1].
- *Scale mean and standard deviation* normalizes the mean and standard deviation of the training set.

*Improving generalization*

Improving the network's ability to generalize helps prevent overfitting, a common problem in neural network design. Overfitting occurs when a network has memorized the training set but has not learned to generalize to new inputs. Overfitting produces a relatively small error on the training set but will produce a much larger error when new data is presented to the network. The Neural Network Toolbox provides two solutions to improve generalization:

- *Regularization* modifies the network's performance function, the measure of error that the training process minimizes. By changing it to include the size of the weights and biases, training produces a network that not only performs well with the training data, but produces smoother behavior when presented with new data.
- *Early stopping* is a technique that uses two different data sets: the training set, which is used to update the weights and biases, and the validation set, which is used to stop training when the network begins to overfit the data [35].

## FUZZY LOGIC TOOLBOX OF MATLAB

Fuzzy logic has two different meanings. In a narrow sense, fuzzy logic is a logical system, which is an extension of multivalued logic. But in a wider sense—which is in predominant use today—fuzzy logic is almost synonymous with the theory of fuzzy sets, a theory which relates to classes of objects with unsharp boundaries in which membership is a matter of degree. The Fuzzy Logic Toolbox is a collection of functions built on the MATLAB numeric computing environment [36]. The toolbox provides three categories of tools:

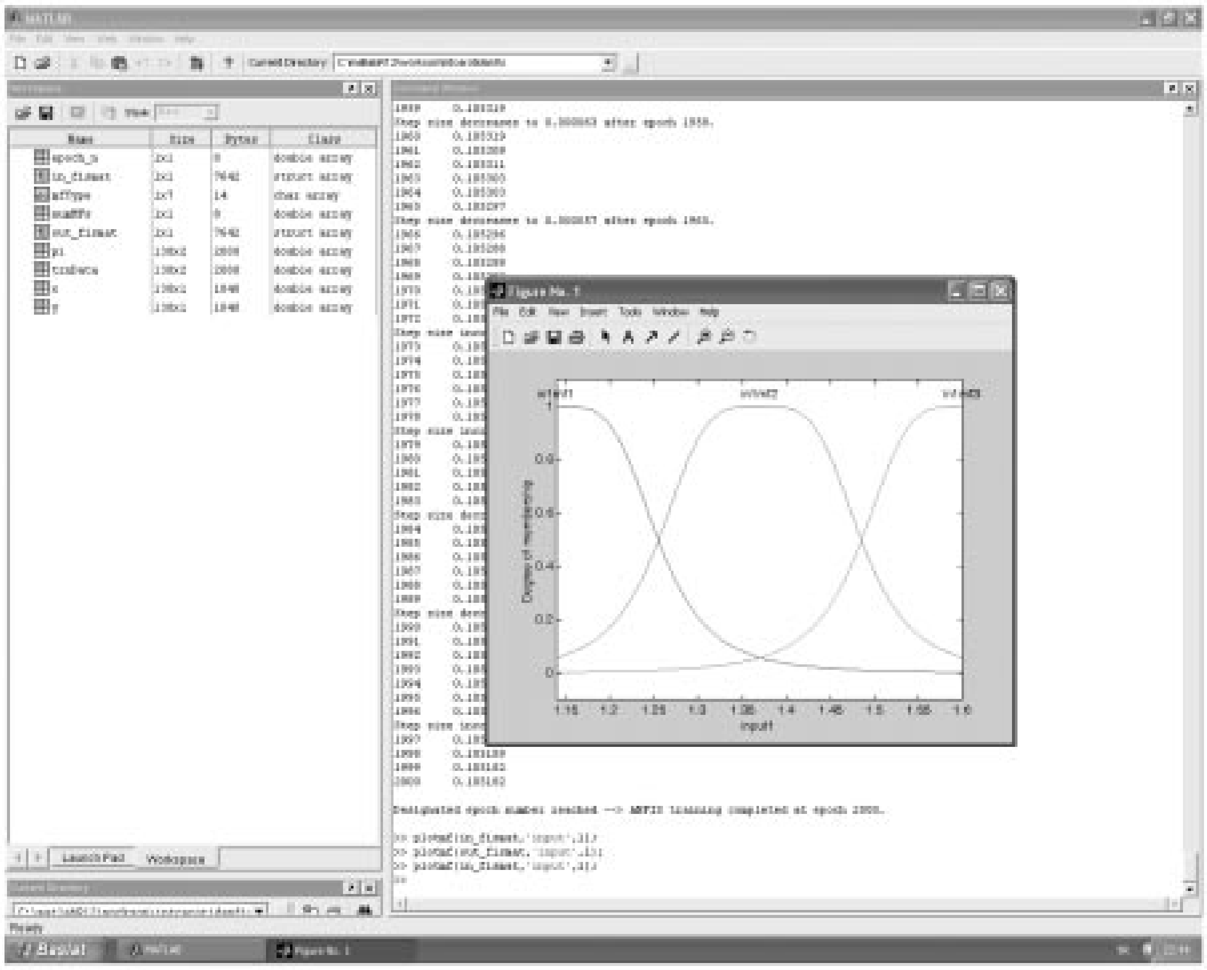
- command line functions;
- graphical interactive tools;
- Simulink blocks and examples.

The first category of tools is made up of functions that you can call from the command line or from your own applications. Secondly, the toolbox provides a number of interactive tools that let you access many of the functions through a GUI. The brief descriptions of the GUI tools, membership functions and advanced techniques are presented in Tables 7–9, respectively. The third category of tools is a set of blocks for use with the Simulink simulation software. These are specifically designed for high speed fuzzy logic inference in the Simulink environment. In order to build your own fuzzy simulink models you can use the Fuzzy Logic Controller block in the Fuzzy Logic Toolbox library, which you can open either by selecting Fuzzy Logic Toolbox in the Simulink Library Browser, or by typing `fuzblock` at the MATLAB prompt. This command opens a Simulink library that contains two Simulink blocks you can use:

- The Fuzzy Logic Controller
- The Fuzzy Logic Controller With Rule Viewer. This block forces the Rule Viewer to pop open during a Simulink simulation [36].

A trend which is growing in visibility relates to the use of fuzzy logic in combination with neurocomputing. More generally, fuzzy logic and neurocomputing may be viewed as the principal constituents of what might be called soft computing. Unlike the traditional, hard computing, soft computing is aimed at an accommodation with the pervasive imprecision of the real world. The guiding principle of soft computing is: exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost. Among various combinations of methodologies in soft computing, the one which has highest visibility at this juncture is that of fuzzy logic and neurocomputing, leading to so-called neuro-fuzzy systems. Within fuzzy logic, such systems play a particularly important role in the introduction of rules from observations. An effective method developed by Jang [19, 20] for this purpose is called ANFIS. This method is an important component of the Fuzzy Logic Toolbox [36]. Command window of MATLAB taken from students' projects related with ANFIS implemented for Doppler ultrasound blood flow signals is presented in Fig. 12.

In this section, we discuss the use of the function `anfis` and the ANFIS Editor GUI in the Fuzzy Logic Toolbox. These tools apply fuzzy inference techniques to data modeling. The basic idea behind these neuro-adaptive learning techniques is very simple. These techniques provide a method for the fuzzy modeling procedure to learn information about a data set, in order to compute the membership function parameters that best allow the associated fuzzy inference system (FIS) to track the



| Name | Size | Bytes | Class |
|---|---|---|---|
| epoch_n | 1x1 | 8 | double array |
| in_fismat | 1x1 | 7642 | struct array |
| mfType | 1x7 | 14 | char array |
| numMFs | 1x1 | 8 | double array |
| out_fismat | 1x1 | 7642 | struct array |
| p1 | 130x2 | 2080 | double array |
| trnData | 130x2 | 2080 | double array |
| x | 130x1 | 1040 | double array |
| y | 130x1 | 1040 | double array |

```
1959      0.185319
Step size decreases to 0.000043 after epoch 1958.
1960      0.185319
1961      0.185308
1962      0.185311
1963      0.185303
1964      0.185303
1965      0.185297
Step size decreases to 0.000037 after epoch 1965.
1966      0.185296
1967      0.185289
1968      0.185289
[illegible]
1998      0.185139
1999      0.185183
2000      0.185183

Designated epoch number reached --> ANFIS training completed at epoch 2000.

>> plotmf(in_fismat,'input',1);
>> plotmf(out_fismat,'input',1);
>> plotmf(in_fismat,'input',1);
>>
```

Fig. 12. Command window of MATLAB taken from students' projects related with ANFIS implemented for Doppler ultrasound blood flow signals.

given input/output data. This learning method works similarly to that of neural networks. The parameters associated with the membership functions will change through the learning process. The computation of these parameters (or their adjustment) is facilitated by a gradient vector, which provides a measure of how well the FIS is modeling the input/output data for a given set of parameters. Once the gradient vector is obtained, any of several optimization routines could be applied in order to adjust the parameters so as to reduce some error measure (usually defined by the sum of the squared difference between actual and desired outputs). The `anfis` function uses either backpropagation or a combination of least squares estimation and backpropagation for membership function parameter estimation. The Fuzzy Logic Toolbox function that accomplishes this membership function parameter adjustment is called `anfis`. This function can be accessed either from the command line, or through the ANFIS Editor GUI. Since the functionality of the command line function `anfis` and the ANFIS Editor GUI are similar, they are used somewhat interchangeably in this discussion. The ANFIS Editor GUI is shown in Fig. 13. From this GUI the students can:

- Load data (training, testing, and checking) by selecting appropriate radio buttons in the **Load data** portion of the GUI and then selecting **Load Data**. The loaded data is plotted on the plot region.
- Generate an initial FIS model or load an initial FIS model using the options in the **Generate FIS** portion of the GUI.
- View the FIS model structure once an initial FIS has been generated or loaded by selecting the **Structure** button.
- Choose the FIS model parameter optimization method: backpropagation or a mixture of backpropagation and least squires (hybrid method).
- Choose the number of training epochs and the training error tolerance.
- Train the FIS model by selecting the **Train Now** button. This training adjusts the membership function parameters and plots the training (and/or checking data) error plot(s) in the plot region.
- View the FIS model output versus the training, checking, or testing data output by selecting the **Test Now** button. This function plots the test data against the FIS output in the plot region [36].

## EDUCATIONAL CONTRIBUTION

The main goal of the tool was the development of an educational platform allowing students to improve their knowledge about new trends in neural networks and ANFIS. The key features of

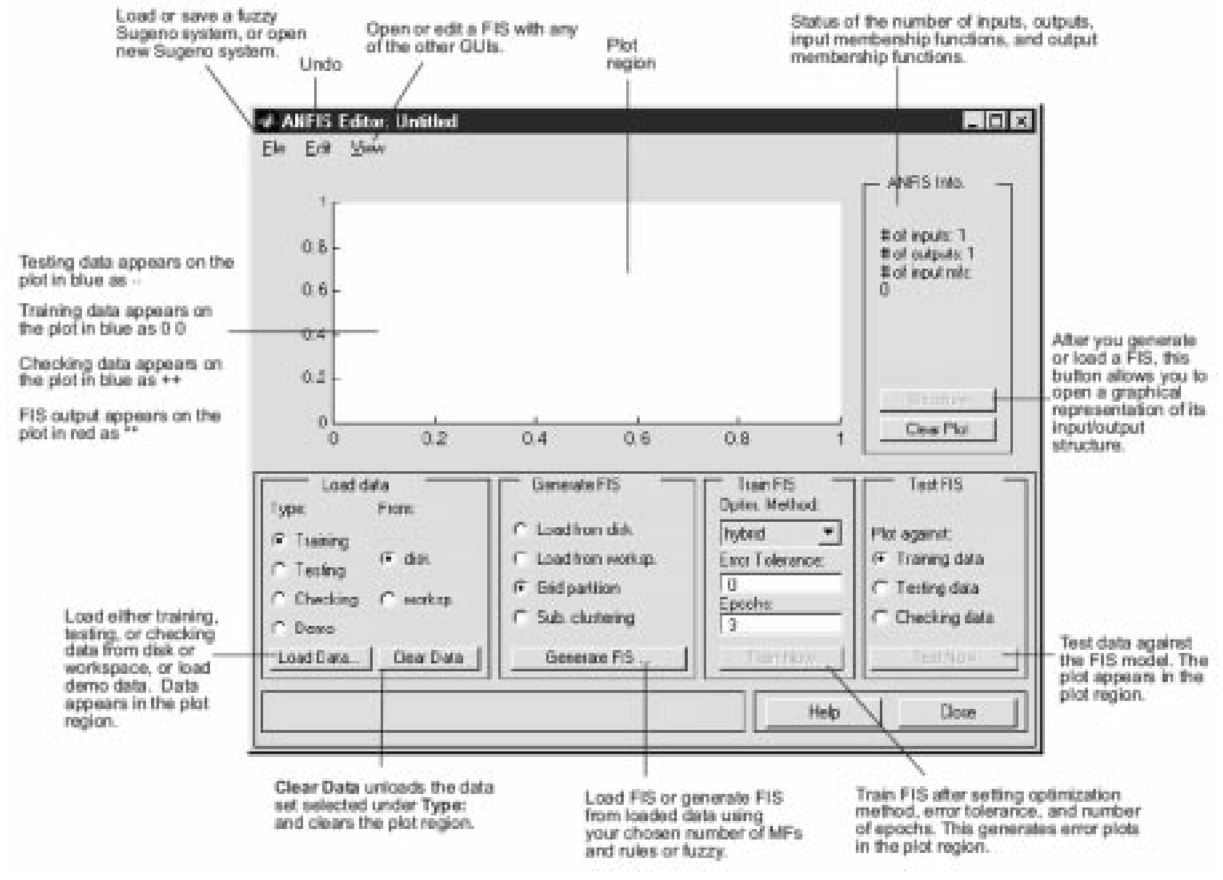

Load or save a fuzzy Sugeno system, or open new Sugeno system.
Undo
Open or edit a FIS with any of the other GUIs.
Plot region
Status of the number of inputs, outputs, input membership functions, and output membership functions.
ANFIS Editor: Untitled
File Edit View
ANFIS Info.
# of inputs: 1
# of outputs: 1
# of input mfs: 0
Testing data appears on the plot in blue as ··
Training data appears on the plot in blue as 0 0
Checking data appears on the plot in blue as ++
FIS output appears on the plot in red as **
After you generate or load a FIS, this button allows you to open a graphical representation of its input/output structure.
Structure
Clear Plot
Load data
Type: From:
Training
Testing
Checking
Demo
disk
worksp.
Load Data...
Clear Data
Generate FIS
Load from disk
Load from worksp.
Grid partition
Sub. clustering
Generate FIS ...
Train FIS
Optim. Method:
hybrid
Error Tolerance:
0
Epochs:
3
Train Now
Test FIS
Plot against:
Training data
Testing data
Checking data
Test Now
Help
Close
Load either training, testing, or checking data from disk or workspace, or load demo data. Data appears in the plot region.
Clear Data unloads the data set selected under Type: and clears the plot region.
Load FIS or generate FIS from loaded data using your chosen number of MFs and rules or fuzzy.
Train FIS after setting optimization method, error tolerance, and number of epochs. This generates error plots in the plot region.
Test data against the FIS model. The plot appears in the plot region.

Fig. 13. ANFIS Editor GUI.

Table 7. GUI Tools

| Function | Purpose |
|---|---|
| anfisedit | ANFIS editor GUI |
| fuzzy | Basic fuzzy inference system (FIS) editor |
| mfedit | Membership function editor |
| ruleedit | Rule editor and parser |
| ruleview | Rule viewer and fuzzy inference diagram |
| surfview | Output surface viewer |

Table 8. Membership functions

| Function | Purpose |
|---|---|
| dsigmf | Difference of two sigmoid membership functions |
| gauss2mf | Two-sided Gaussian curve membership function |
| gaussmf | Gaussian curve membership function |
| gbellmf | Generalized bell curve membership function |
| pimf | Pi-shaped curve membership function |
| psigmf | Product of two sigmoidal membership functions |
| smf | S-shaped curve membership function |
| sigmf | Sigmoid curve membership function |
| trapmf | Trapezoidal membership function |
| trimf | Triangular membership function |
| zmf | Z-shaped curve membership function |

Table 9. Advanced techniques

| Function | Purpose |
|---|---|
| anfis | Training routine for a Sugeno-type FIS |
| fcm | Find clusters with fuzzy c-means (FCM) clustering |
| genfis1 | Generate FIS matrix using grid method |
| genfis2 | Generate FIS matrix using subtractive clustering |
| subclust | Find cluster centers with subtractive clustering |

the Neural Network and Fuzzy Logic Toolboxes of MATLAB are listed below:

- GUI for creating, training, and simulating their neural networks.
- Support for the most commonly used supervised and unsupervised network architectures.
- A comprehensive set of training and learning functions.
- A suite of Simulink blocks, as well as documentation and demonstrations of control-system applications.
- Automatic generation of Simulink models from neural network objects.
- Modular network representation, allowing an unlimited number of input sets, layers, and network interconnections.
- Pre- and post-processing functions for improving network training and assessing network performance.
- Routines for improving generalization.
- Visualization functions for viewing network performance.
- Fuzzy logic is tolerant of imprecise data. The students can create a fuzzy system to match any set of input-output data. This process is made particularly easy by adaptive techniques like ANFIS, which are available in the Fuzzy Logic Toolbox.

During the last week of the semester, students were asked to comment on the problems they faced in this course. A number of suggestions were made. Several features have been taken into account:

- The students are asked to use the Neural Network and Fuzzy Logic Toolboxes of MATLAB. Students not only save time by calling these M-files but also have time to explore other practical issues. By analyzing the functions of the toolboxes, students are given a good platform on which they can learn how to write their own optimized functions. From the feedback, students benefited most by working with these efficient programming templates.
- The system allows students to apply fundamental techniques for Doppler ultrasound blood flow signals. The main goal is to let students gain confidence before attempting to develop automated diagnostic systems for the signal under study. These projects are also open-ended, allowing students to broaden the assignment and good students will attempt these parts and gain extra bonus marks.
- Evidence from the feedback demonstrates that most students expressed their strong support and keenness in working on these projects. Some students who had a good project involvement experience stay on for their graduate study.

## CONCLUSION

The objective of this paper is to provide a preliminary understanding of ANNs, ANFIS classifiers and answer the why and when these computational tools are needed, the motivation behind their development, the various learning rules and ANN types, computations involved, design considerations, application to real-world problems, advantages and limitations. Such understanding is essential for making efficient use of their features. A theoretical basis of developing automated diagnostic systems was presented starting with the basic equations and performing applications of these methods to the arterial Doppler signals. The MLPNNs and ANFIS classifiers were used to classify the arterial Doppler signals which were given as illustrative examples. The intent of classification of the Doppler ultrasound blood flow signals was to serve as an introduction to the use of the neural networks and ANFIS and thus motivate their teaching with the help of MATLAB in the classroom. We emphasized the performance characteristics of the neural network models and ANFIS classifiers for applications to the arterial Doppler signals. The drawn results can be helpful to the students. However, modeling their own signals with various neural network architectures and ANFIS classifiers will be better because the performance characteristics of the classifiers can vary according to the signal under study. MATLAB was introduced in this concept because of its ease in building mathematical functions and its powerful graphical user interface for displaying the results. The students can undertake the projects related with developing automated diagnostic systems as a part of their homework assignments, making it easy to visualize the intricacies and understand the relationship between the different parameters involved in the models. Our experience has shown that this teaching platform and methodology have clearly heightened students' interest in the learning of neural network models, ANFIS and their applications.

## REFERENCES

1. D. West and V. West, Model selection for a medical diagnostic decision-support system: a breast cancer detection case, *Artificial Intelligence in Medicine*, **20**(3), 2000, pp. 183–204.
2. D. West and V. West, Improving diagnostic accuracy using a hierarchical neural network to model decision subtasks, *Int. J. Medical Informatics*, **57**(1), 2000, pp. 41–55.
3. H. Kordylewski, D. Graupe and K. Liu, A novel large-memory neural network as an aid in medical diagnosis applications, *IEEE Trans. Information Technology in Biomedicine*, **5**(3), 2001, pp. 202–209.
4. I. Güler, F. Hardalaç and E. D. Übeyli, Determination of Behcet disease with the application of FFT and AR methods, *Computers in Biology and Medicine*, **32**(6), 2002, pp. 419–434.
5. I. Güler and E. D. Übeyli, Application of classical and model-based spectral methods to ophthalmic arterial Doppler signals with uveitis disease, *Computers in Biology and Medicine*, **33**(6), 2003, pp. 455–471.
6. E. D. Übeyli and I. Güler, Comparison of eigenvector methods with classical and model-based methods in analysis of internal carotid arterial Doppler signals, *Computers in Biology and Medicine*, **33**(6), 2003, pp. 473–493.
7. E. D. Übeyli and I. Güler, Spectral analysis of internal carotid arterial Doppler signals using FFT, AR, MA, and ARMA methods, *Computers in Biology and Medicine*, **34**(4), 2004, pp. 293–306.
8. I. A. Wright, N. A. J. Gough, F. Rakebrandt, M. Wahab and J. P. Woodcock, Neural network analysis of Doppler ultrasound blood flow signals: a pilot study, *Ultrasound in Medicine & Biology*, **23**(5), 1997, pp. 683–690.
9. I. A. Wright and N. A. J. Gough, Artificial neural network analysis of common femoral artery Doppler shift signals: classification of proximal disease, *Ultrasound in Medicine & Biology*, **24**(5), 1999, pp. 735–743.
10. I. Güler and E .D. Übeyli, Detection of ophthalmic artery stenosis by least-mean squares backpropagation neural network, *Computers in Biology and Medicine*, **33**(4), 2003, pp. 333–343.
11. E. D. Übeyli and I. Güler, Neural network analysis of internal carotid arterial Doppler signals: predictions of stenosis and occlusion, *Expert Systems with Applications*, **25**(1), 2003, pp. 1–13.
12. I. Güler and E. D. Übeyli, Detection of ophthalmic arterial Doppler signals with Behcet disease using multilayer perceptron neural network, *Computers in Biology and Medicine*, 2004 (in press).
13. E. D. Übeyli and I. Güler, Improving medical diagnostic accuracy of ultrasound Doppler signals by combining neural network models, *Computers in Biology and Medicine*, **35**(2), 2005, pp. 121–132.
14. I. Güler and E. D. Übeyli, A mixture of experts network structure for modelling Doppler ultrasound blood flow signals, *Computers in Biology and Medicine*, **35**(6), 2005, pp. 533–554.
15. E. D. Übeyli and I. Güler, Adaptive neuro-fuzzy inference systems for analysis of internal carotid arterial Doppler signals, *Computers in Biology and Medicine*, **35**(7), 2005, pp. 565–582.
16. D. Dubois and H. Prade, An introduction to fuzzy systems, *Clinica Chimica Acta*, **270**, 1998, pp. 3–29.
17. L. I. Kuncheva and F. Steimann, Fuzzy diagnosis, *Artificial Intelligence in Medicine*, **16**, 1999, pp. 121–128.
18. D. Nauck and R. Kruse, Obtaining interpretable fuzzy classification rules from medical data, *Artificial Intelligence in Medicine*, **16**, 1999, pp. 149–169.
19. J.-S. R. Jang, ANFIS: Adaptive-network-based fuzzy inference system, *IEEE Trans. Systems, Man and Cybernetics*, **23**(3), 1993, pp. 665–685.
20. J.-S. R. Jang, Self-learning fuzzy controllers based on temporal backpropagation, *IEEE Trans. Neural Networks*, **3**(5), 1992, pp. 714–723.
21. J. Usher, D. Campbell, J. Vohra and J. Cameron, A fuzzy logic-controlled classifier for use in implantable cardioverter defibrillators, *Pace-Pacing and Clinical Electrophysiology*, **22**, 1999, pp. 183–186.
22. S. Y. Belal, A. F. G. Taktak, A. J. Nevill, S. A. Spencer, D. Roden and S. Bevan, Automatic detection of distorted plethysmogram pulses in neonates and paediatric patients using an adaptive-network-based fuzzy inference system, *Artificial Intelligence in Medicine*, **24**, 2002, pp. 149–165.
23. I. Virant-Klun and J. Virant, Fuzzy logic alternative for analysis in the biomedical sciences, *Computers and Biomedical Research*, **32**, 1999, pp. 305–321.
24. I. A. Basheer and M. Hajmeer, Artificial neural networks: fundamentals, computing, design, and application, *J. Microbiological Methods*, **43**(1), 2000, pp. 3–31.
25. B. B. Chaudhuri and U. Bhattacharya, Efficient training and improved performance of multilayer perceptron in pattern classification, *Neurocomputing*, **34**, 2000, pp. 11–27.
26. D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Nature*, **323**, 1986, pp. 533–536.

27. R. A. Jacobs, Increased rate of convergence through learning rate adaptation, *Neural Networks*, **1**, 1988, pp. 295–307.
28. A. A. Minai and R. D. Williams, Back-propagation heuristics: a study of the extended delta-bar-delta algorithm, *Proc. Int. Joint Conf. Neural Networks*, **1**, 17–21 June 1990, San Diego, California, pp. 595–600.
29. S. E. Fahlman, An empirical study of learning speed in backpropagation networks, *Computer Science Technical Report*, CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, 1988.
30. M. T. Hagan and M. B. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Transactions on Neural Networks*, **5**(6), 1994, pp. 989–993.
31. R. Battiti, First- and second-order methods for learning: between steepest descent and Newton's method, *Neural Computation*, **4**, 1992, pp. 141–166.
32. L-W. Chan, Efficacy of different learning algorithms of the back propagation network, *IEEE Region 10 Conference on Computer and Communication Systems*, **1**, 24–27 September 1990, Hong Kong, pp. 23–27.
33. A. Sidani and T. Sidani, A comprehensive study of the backpropagation algorithm and modifications, *IEEE Conference Record*, (29–31 March 1994) Orlando FL, USA, pp. 80–84.
34. J. M. Hannan and J. M. Bishop, A comparison of fast training algorithms over two real problems, *IEE Fifth Int. Conf. Artificial Neural Networks*, Conference Publication No. **440**, 7–9 July 1997, Cambridge, UK, pp. 1–6.
35. *Neural Network Toolbox User's Guide,* Mathworks, Inc., Natick, MA, 1998.
36. *Fuzzy Logic Toolbox User's Guide*, Mathworks, Inc., Natick, MA, 2000.

**Inan Güler** graduated from Erciyes University in 1981. He took his MS degree from Middle East Technical University in 1985, and his Ph.D. degree from Istanbul Technical University in 1990, all in Electronic Engineering. He is a professor at Gazi University where he is Head of Department. His interest areas include biomedical systems, biomedical signal processing, biomedical instrumentation, electronic circuit design, neural network, and artificial intelligence. He has written more than 100 articles on biomedical engineering.

**Elif Derya Übeyli** graduated from Çukurova University in 1996. She took her MS degree in 1998, all in electronic engineering. She took her Ph.D. degree from Gazi University, electronics and computer technology. She is an instructor at the Department of Electrical and Electronics Engineering at TOBB Economics and Technology University. Her interest areas are biomedical signal processing, neural networks, and artificial intelligence.