# Multilayer Perceptron, Fuzzy Sets, and Classification

Sankar K. Pal, *Senior Member, IEEE*, and Sushmita Mitra, *Student Member, IEEE*

***Abstract*— A fuzzy neural network model based on the multilayer perceptron, using the back-propagation algorithm, and capable of fuzzy classification of patterns is described. The input vector consists of membership values to linguistic properties while the output vector is defined in terms of fuzzy class membership values. This allows efficient modeling of fuzzy or uncertain patterns with appropriate weights being assigned to the back-propagated errors depending upon the membership values at the corresponding outputs. During training, the learning rate is gradually decreased in discrete steps until the network converges to a minimum error solution. The effectiveness of the algorithm is demonstrated on a speech recognition problem. The results are compared with those of the conventional MLP, the Bayes classifier, and the other related models.**

***Index Terms*— Artificial neural networks; multilayer perceptron; fuzzy sets; speech recognition; linguistic variable; fuzzy classification.**

## I. Introduction

ARTIFICIAL neural networks [1]–[4] are massively parallel interconnections of simple *neurons* that function as a collective system. It has been observed that many problems in pattern recognition are solved more easily by humans than by computers, perhaps because of the basic architecture and functioning mechanism of their brains. Neural nets are designed in an attempt to mimic the human brain in order to emulate human performance and thereby function *intelligently*. These networks may be broadly categorized into two types:

- those that learn by adaptively, updating their connection weights during training;
- those whose parameters are time-invariant, i.e., whose weights are fixed initially and no eventual updating occurs.

In this work we consider a network of the first kind. These networks can be trained by examples (as is often required in real life) and sometimes generalize well for unknown test cases. The worthiness of a network lies in its inferencing or generalization capabilities over such test sets. Connectionist learning procedures are suitable in domains with several graded features that collectively contribute to the solution of a problem. In the process of learning, a network may discover important underlying regularities in the task domain.

Manuscript received May 1, 1991; revised October 30, 1991. This work was supported by CSIR through a fellowship to the second author.

The authors are with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta, 700035 India. Part of this work was performed while S. K. Pal was with the Johnson Space Center, Houston, TX, as the holder of an NRC-NASA Senior Research Award.

IEEE Log Number 9201466.

An advantage of neural nets [2], [5], [6] lies in the high computation rate provided by their massive parallelism, so that real-time processing of huge data sets becomes feasible with proper hardware. Information is encoded among the various connection weights in a distributed manner.

It may be mentioned that human reasoning is somewhat fuzzy in nature. The utility of fuzzy sets [7], [8] lies in their ability to model the *uncertain* or ambiguous data so often encountered in real life. Hence, to enable a system to tackle real-life situations in a manner more like humans, one may incorporate the concept of fuzzy sets into the neural network. It is to be noted that although fuzzy logic is a natural mechanism for propagating uncertainty, it may involve in some cases an increase in the amount of computation required (compared with a system using classical binary logic). This can be suitably offset by using fuzzy neural network models having the potential for parallel computation with high flexibility. Fuzzy concepts have already been incorporated into neural nets in control problems [9], in modeling output possibility distributions [10], in learning and extrapolating complex relationships between antecedents and consequents of rules [11], and in fuzzy reasoning [12].

The present work attempts to build a fuzzy version of the multilayer perceptron using the gradient-descent-based back-propagation algorithm [2], [13] by incorporating concepts from fuzzy sets at various stages. Unlike the conventional system, the proposed model is also capable of handling input available in linguistic form. Besides, conventional two-state neural net models generally deal with the ideal condition, where an input feature is either present or absent and each pattern belongs to either one class or another. They do not consider cases where an input feature may possess a property with a certain degree of confidence, or where a pattern may belong to more than one class with a finite degree of "belongingness." The proposed model incorporates these concepts and is capable of classification of fuzzy patterns.

Broadly, the network passes through two phases, viz., training and testing. During the training phase supervised learning is used to assign output membership values lying in the range [0, 1] to the training vectors. Hence each output class (node in the output layer) may be assigned a nonzero membership instead of choosing the single class (node) with the highest activation. This allows modelling of fuzzy data when the feature space involves overlapping pattern classes such that a pattern point may belong to more than one class with nonzero membership.

During training, each error in membership assignment is fed back and the connection weights of the network are appropriately updated. The back-propagated error is computed with respect to each desired output, which is a membership value denoting the degree of belongingness of the input vector to that class. Hence the error (which is back-propagated for weight updating) has inherently more weight in case of nodes with higher membership values. The contribution of ambiguous or uncertain vectors to the weight correction is automatically reduced. This is natural as vectors that are more *typical* of their class should have more influence in determining the position and shape of the decision surface.

The utility of the approach proposed here for the modeling of output values may be further appreciated by considering a point lying in a region of overlapping classes in the feature space. In such cases its membership in each of these classes may be nearly equal. Then there is no reason why we should follow the *crisp* approach of classifying this pattern as belonging to the class corresponding to that output neuron with a slightly higher activation, and thereby neglect the smaller yet significant response(s) obtained for the other overlapping class(es).

After a number of cycles the neural net may converge to a minimum error solution. The network now encodes the input space information in its connection weights. In the second phase, a part of the same fuzzy data (kept aside for testing during random selection of the training set) is applied as input and the network performance verified at the output. Note that the output is obtained in terms of fuzzy membership values to the various pattern classes. A confusion matrix is generated to evaluate the classification efficiency of the neural network. Here each test datum contributes a count at a particular position in this matrix, its row corresponding to the class to which it belongs (as determined from the *hard* labels attached to the input data set) and its column indicating the class corresponding to the neuronal output providing the *best match.*

The proposed fuzzy neural network model is capable of handling input features presented in quantitative and/or linguistic form. The components of the input vector consist of the membership values to the overlapping partitions of linguistic properties *low, medium,* and *high* corresponding to each input feature. This provides scope for incorporating linguistic information in both the training and the testing phases of the said model and increases its robustness in tackling imprecise or uncertain input specifications.

During training, the learning rate and the damping coefficient are gradually decreased until the network hopefully converges to a minimum error solution. This heuristic helps to avoid spurious local minima and usually prevents oscillations of the mean square error in the weight space. In the process the network undergoes a maximal number of sweeps through the training set.

The effectiveness of the proposed model is demonstrated on a speech recognition problem where the classes have ill-defined, fuzzy boundaries. A comparison is made with the standard Bayes classifier and the conventional multilayer perceptron, and the performance of the proposed model is found to be better.

## II. Multilayer Perceptron Using Backpropagation of Error

The multilayer perceptron (MLP) [2], [13], [14] consists of multiple layers of simple, two-state, sigmoid processing elements (nodes) or neurons that interact using weighted connections. After a lowermost input layer there are usually any number of intermediate, or hidden, layers followed by an output layer at the top. There exist no interconnections within a layer while all neurons in a layer are fully connected to neurons in adjacent layers. Weights measure the degree of correlation between the activity levels of neurons that they connect.

An external input vector is supplied to the network by clamping it at the nodes in the input layer. For conventional classification problems, during training, the appropriate output node is clamped to state 1 while the others are clamped to state 0. This is the desired output supplied by the *teacher.*

Consider the network given in Fig. 1. The total input, $x_j^{h+1}$, received by neuron $j$ in layer $h+1$ is defined as

$$x_j^{h+1} = \sum_i y_i^h w_{ji}^h - \theta_j^{h+1}, \tag{1}$$

where $y_i^h$ is the state of the $i$th neuron in the preceding $h$th layer, $w_{ji}^h$ is the weight of the connection from the $i$th neuron in layer $h$ to the $j$th neuron in layer $h+1$, and $\theta_j^{h+1}$ is the threshold of the $j$th neuron in layer $h+1$. Threshold $\theta_j^{h+1}$ may be eliminated by giving the unit $j$ in layer $h+1$ an extra input line with a fixed activity level of 1 and a weight of $-\theta_j^{h+1}$.

The output of a neuron in any layer other than the input layer ($h > 0$) is a monotonic nonlinear function of its total input and is given as

$$y_j^h = \frac{1}{1+e^{-x_j^h}}. \tag{2}$$

For nodes in the input layer,

$$y_j^0 = x_j^0, \tag{3}$$

where $x_j^0$ is the $j$th component of the input vector clamped at the input layer. All neurons within a layer, other than the input layer, have their states set by (1) and (2) in parallel while different layers have their states set sequentially in a *bottom-up* manner until the states of the neurons in the output layer $H$ are determined. The learning procedure has to determine the internal parameters of the hidden units based on its knowledge of the inputs and desired outputs. Hence learning consists of searching a very large parameter space and therefore is usually rather slow.

The least mean square (LMS) error in output vectors, for a given network weight vector $\boldsymbol{w}$ is defined as

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{j,c}\left(y_{j,c}^H(\boldsymbol{w}) - d_{j,c}\right)^2, \tag{4}$$

where $y_{j,c}^H(\boldsymbol{w})$ is the state obtained for output node $j$ in layer $H$ in input–output case $c$, and $d_{j,c}$ is its desired state specified by the teacher. One method for minimization of $E(\boldsymbol{w})$ is to apply the method of gradient descent by starting with any set

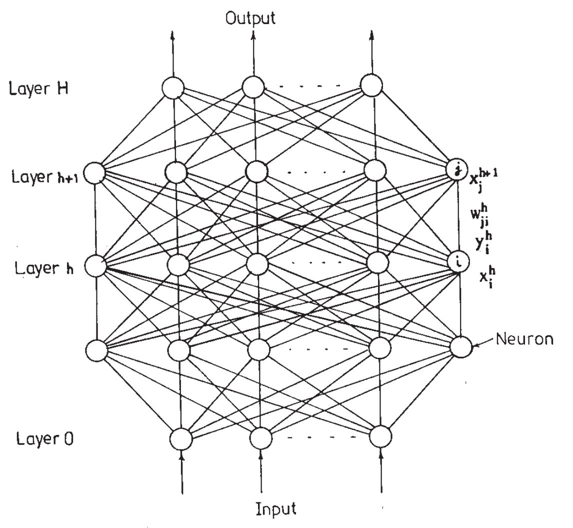


Fig. 1. A neural network with three hidden layers. Neurons connected to each other via variable connection weights.

of weights and repeatedly updating each weight by an amount

$$\Delta w_{ji}^h(t) = -\epsilon \frac{\partial E}{\partial w_{ji}} + \alpha \Delta w_{ji}^h(t-1) - hdec \cdot w_{ji}^h(t-1), \quad (5)$$

where the positive constant $\epsilon$ controls the descent, $0 \le \alpha \le 1$ is the damping coefficient or momentum, *hdec* is the % decay coefficient, and $t$ denotes the number of the iteration currently in progress. Using a decay factor $0.01 > hdec > 0$ enables only those weights doing *useful* work in reducing the error to survive and hence improves the generalization capabilities of the network [13, p. 8].

From (1), (2) and (4) we have

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial y_j} \frac{dy_j}{dx_j} \frac{\partial x_j}{\partial w_{ji}} \\ &= \frac{\partial E}{\partial y_j} y_j^h (1 - y_j^h) y_i^{h-1}. \end{aligned} \quad (6)$$

For the output layer ($h = H$), we substitute in (6)

$$\frac{\partial E}{\partial y_j} = y_j^H - d_j. \quad (7)$$

This assigns credit proportionately to those weights most responsible for the error, thus implementing gradient steepest descent in the weight space. The central idea is to first use a forward pass for each input–output case $c$, starting at the input neurons, to compute the activity levels of all the neurons in the network. Then a backward pass, starting at the output neurons, is used to compute the error derivative $\partial E / \partial y_j$ and back-propagate to allow weight updating until the input layer is reached. For the other layers, using (1), we substitute in (6)

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \frac{dy_k}{dx_k} \frac{\partial x_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \frac{dy_k}{dx_k} w_{kj}^h, \quad (8)$$

where units $j$ and $k$ lie in layers $h$ and $h+1$ respectively.

Consider a multidimensional weight space with an axis for each weight and one extra axis corresponding to the error measure. This weight space typically contains *ravines* with steep sides and a shallow gradient along the ravine. The acceleration methods used in (5) force the gradient to change the velocity of the current point in the weight space, help to speed convergence, and rarely get stuck at poor local minima. The roles of $\epsilon$ and $\alpha$ in (5) have natural interpretations in terms of physical movement along this error surface, composed of *hills, valleys, ravines, ridges, plateaus,* and *saddle points,* in the weight space.

During training, each pattern of the training set is used in succession to clamp the input and output layers of the network. A sequence of forward and backward passes using (1)–(8) constitutes a *cycle* and such a cycle through the entire training set is termed a *sweep.* After a number of sweeps through the training data, the error $E(\boldsymbol{w})$ in (4) may be minimized. At this stage the network is supposed to have discovered (learned) the relationship between the input and output vectors in the training samples.

In the testing phase the neural net is expected to be able to utilize the information encoded in its connection weights to assign the correct output labels for the test vectors that are now clamped only at the input layer. It should be noted that the optimal number of hidden layers and the number of units in each such layer are mostly empirical in nature. This is demonstrated in Section VI. The number of units in layer $H$ corresponds to the number of output classes.

MLP models using back-propagation have been applied in the exclusive OR problem [2] and in recognizing familiar shapes in novel positions [13], discovering semantic features [15], recognizing written text [16], recognizing speech, [14], playing backgammon [17], predicting sunspots [18] and identifying sonar targets [19].

## III. Pattern Representation in Linguistic Form

In conventional statistical designs, the input patterns are quantitatively exact to within the resolution of the sensors used to collect them. However, real processes also may possess imprecise or incomplete input features. In such cases it may become convenient to use linguistic variables and hedges [8] such as *low, medium, high, very,* and *more or less* to augment or even replace numerical input feature information.

The proposed model is capable of handling exact (numerical) and/or inexact (linguistic) forms of input data. Any input feature value can be described through a combination of membership values in the linguistic property sets *low, medium* and *high.*

### A. Fuzzy Sets

In traditional classifiers, an element $r$ either belongs or does not belong to a given class $A$. Hence the characteristic function of $A$ is expressed as

$$\mu_A(r) = \begin{cases} 1 & \text{if } r \in A \\ 0 & \text{otherwise.} \end{cases}$$

But in real-life problems the classes are often ill defined, overlapping, or fuzzy. A pattern may belong to more than one class with a nonzero degree of membership. Using fuzzy set-theoretic techniques [7], [8], [20], [21] a pattern point $r$, belonging to the universe $R$, may be assigned a grade of membership with the membership function $\mu_A(r)$ to a fuzzy set $A$. This is defined as

$$A = \{(\mu_A(r), r)\}, \qquad r \in R, \qquad \mu_A(r) \in [0, 1]. \tag{9}$$

### B. $\pi$ Membership Function

The $\pi$ function, with range [0, 1] and $\boldsymbol{r} \in \mathbb{R}^n$, is defined as [22]

$$\pi(\boldsymbol{r}; \boldsymbol{c}, \lambda) = \begin{cases} 2\left(1 - \frac{\|\boldsymbol{r}-\boldsymbol{c}\|}{\lambda}\right)^2 & \text{for } \frac{\lambda}{2} \le \|\boldsymbol{r} - \boldsymbol{c}\| \le \lambda \\ 1 - 2\left(\frac{\|\boldsymbol{r}-\boldsymbol{c}\|}{\lambda}\right)^2 & \text{for } 0 \le \|\boldsymbol{r} - \boldsymbol{c}\| \le \frac{\lambda}{2} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $\lambda > 0$ is the radius of the $\pi$ function with $\boldsymbol{c}$ as the central point, and $\|\cdot\|$ denotes the Euclidean norm. This is shown in Fig. 2 for $\boldsymbol{r} \in \mathbb{R}^2$. Note that when the pattern $\boldsymbol{r}$ lies at the central point $\boldsymbol{c}$ of a class, then $\|\boldsymbol{r} - \boldsymbol{c}\| = 0$ and its membership value is maximum, i.e., $\pi(\boldsymbol{c}; \boldsymbol{c}, \lambda) = 1$. The membership value of a point decreases as its distance from the central point $\boldsymbol{c}$, i.e., $\|\boldsymbol{r} - \boldsymbol{c}\|$, increases. When $\|\boldsymbol{r} - \boldsymbol{c}\| = \lambda/2$, the membership value of $\boldsymbol{r}$ is 0.5 and this is called a crossover point.

A fuzzy set with membership function $\pi(\boldsymbol{r}; \boldsymbol{c}, \lambda)$ therefore represents a set of points clustered around $\boldsymbol{c}$. In the proposed model we use the $\pi$ function (in the one-dimensional form) to assign membership values for the input features.

### C. Incorporation of the Linguistic Concept

Each input feature $F_j$ (in quantitative and/or linguistic form) can be expressed in terms of membership values to each of the three linguistic properties *low, medium,* and *high.* Therefore an $n$-dimensional pattern $\vec{\boldsymbol{F}}_i = [F_{i1}, F_{i2}, \cdots, F_{in}]$ may be represented as a $3n$-dimensional vector [23]

$$\vec{\boldsymbol{F}}_i = \left[\mu_{\text{low}(F_{i1})}\left(\vec{\boldsymbol{F}}_i\right), \mu_{\text{medium}(F_{i1})}\left(\vec{\boldsymbol{F}}_i\right), \mu_{\text{high}(F_{i1})}\left(\vec{\boldsymbol{F}}_i\right), \cdots, \mu_{\text{high}(F_{in})}\left(\vec{\boldsymbol{F}}_i\right)\right] \tag{11}$$

When the input feature $F_j$ is linguistic, its membership values for the $\pi$ sets *low, medium,* and *high* are quantified as

$$\begin{aligned} low &\equiv \left\{\frac{0.95}{L}, \frac{0.6}{M}, \frac{0.02}{H}\right\} \\ medium &\equiv \left\{\frac{0.7}{L}, \frac{0.95}{M}, \frac{0.7}{H}\right\} \\ high &\equiv \left\{\frac{0.02}{L}, \frac{0.6}{M}, \frac{0.95}{H}\right\}. \end{aligned} \tag{12}$$

When $F_j$ is numerical, we use the $\pi$ fuzzy set of (10) with appropriate $\boldsymbol{c}$ and $\lambda$ chosen as explained in subsection III-D.

The processing at the input layer is summarized in the block diagram in Fig. 3. Depending upon the numerical or linguistic nature of the input feature $F_j$, we use (10) or (12) to convert $F_j$ to its three-dimensional form given by (11). Note that for the training set the numerical $F_j$'s are kept aside to also compute the mean and standard deviation vectors of (16). However in case of the test set, once the membership values have been computed, the actual numerical values are no longer needed in future computations.

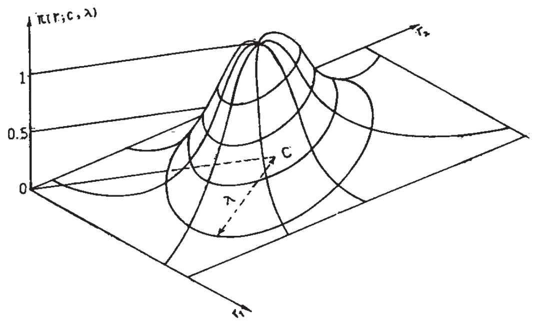


Fig. 2. $\pi$ function when $r \in \mathbf{R}^2$.

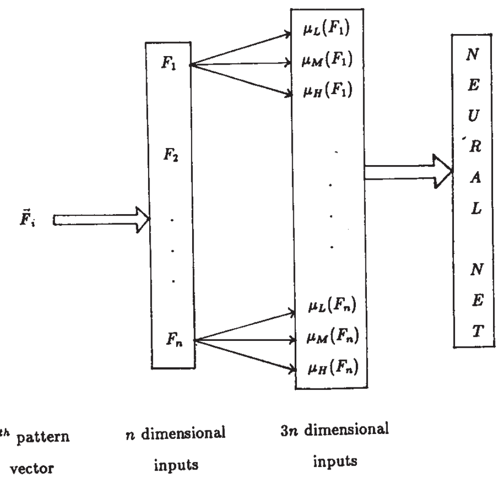


Fig. 3. Block diagram of the input phase of the proposed model.

Hence in trying to express an input $\vec{\boldsymbol{F}}_i$ with linguistic properties we are effectively dividing the dynamic range of each feature into three overlapping partitions. The sets *low, medium,* and *high* for each feature are represented by $\pi$ functions. Fig. 4 shows the overlapping structure of the three $\pi$ functions for a particular input feature $F_j$.

It should be noted in this context that triangular functions for the linguistic terms, *negative big, negative medium, negative small, zero, positive small, positive medium,* and *positive big,* have been used in [9] as input features in the design of neural-net-based control systems. Triangular functions having properties of (10) can also be used in the proposed model.

### D. Choice of Parameters of $\pi$ functions for Numerical Features

Let $F_{j\,\max}$ and $F_{j\,\min}$ denote the upper and lower bounds of the observed range of feature $F_j$ in all $L$ pattern points, considering numerical values only. Then for the three linguistic

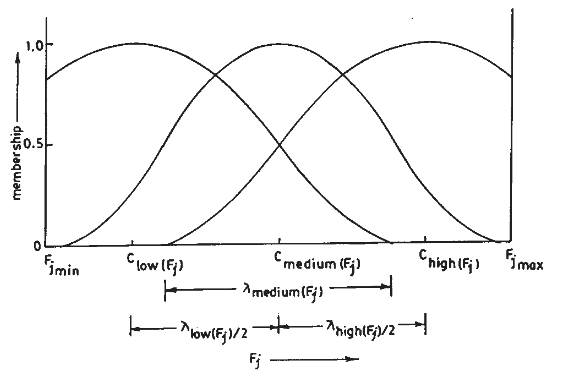


Fig. 4. Overlapping structure of the $\pi$ functions for the linguistic properties *low, medium,* and *high.*

property sets we define

$$\lambda_{\text{medium}(F_j)} = \frac{1}{2}(F_{j\max} - F_{j\min})$$

$$c_{\text{medium}}(F_j) = F_{j\min} + \lambda_{\text{medium}(F_j)} \tag{13}$$

$$\lambda_{\text{low}(F_j)} = \frac{1}{fdenom}\left(c_{\text{medium}(F_j)} - F_{j\min}\right)$$

$$c_{\text{low}(F_j)} = c_{\text{medium}(F_j)} - 0.5 \cdot \lambda_{\text{low}(F_j)} \tag{14}$$

$$\lambda_{\text{high}(F_j)} = \frac{1}{fdenom}\left(F_{j\max} - c_{\text{medium}(F_j)}\right)$$

$$c_{\text{high}(F_j)} = c_{\text{medium}(F_j)} + 0.5 \cdot \lambda_{\text{high}(F_j)} \tag{15}$$

where *fdenom* is a parameter controlling the extent of overlapping.

This combination of choices for the $\lambda$'s and $c$'s automatically ensures that each quantitative input feature value $r_j$ along the $j$th axis for pattern $\vec{F}_i$ is assigned membership value combinations in the corresponding three-dimensional linguistic space of (11) in such a way that at least one of $\mu_{\text{low}(F_{ij})}(\vec{F}_i)$, $\mu_{\text{medium}(F_{ij})}(\vec{F}_i)$, or $\mu_{\text{high}(F_{ij})}(\vec{F}_i)$ is greater than 0.5. This allows a pattern $\vec{F}_i$ to have strong membership to at least one of the properties *low, medium,* and *high.*

## IV. Class Memberships as Output Vectors

In the conventional MLP, used for pattern classification, the number of output nodes corresponds to the number of pattern classes present. During training, the output node corresponding to the class of a pattern vector is kept clamped at state 1 while the others are clamped to state 0. Hence the components of the desired output vector take on *crisp* two-state values. During testing, a *winner-take-all* mechanism causes the test pattern to be classified as belonging to that class corresponding to the output node with the highest activation.

In real-life problems, the data are generally ill defined, with overlapping or fuzzy class boundaries. Each pattern used in training may possess nonzero belongingness to more than one class. To model such data we clamp the desired membership values, lying in the range $[0,1]$, at the output nodes during training. Then the network back-propagates the error(s) with respect to the desired membership value(s) at the output(s). In the process, the network may become able to detect regularities in the input–output membership relation of the training set. Then, when a separate set of test patterns is presented at the input layer, the output nodes automatically generate the class membership values of the patterns to the corresponding classes. This procedure of assigning *fuzzy* output membership values, instead of the more conventional binary output values, enables the proposed neural net to more efficiently classify fuzzy data with overlapping class boundaries.

### A. Membership Functions

Consider an $l$-class problem domain such that we have $l$ nodes in the output layer. Let the $n$-dimensional vectors $\boldsymbol{O}_k$ and $\boldsymbol{V}_k$ denote the mean and standard deviation respectively of the numerical training data for the $k$th class. The weighted distance of the training pattern $\vec{F}_i$ from the $k$th class is defined as

$$z_{ik} = \sqrt{\sum_{j=1}^{n}\left[\frac{F_{ij} - o_{kj}}{v_{kj}}\right]^2} \quad \text{for } k = 1, \cdots, l, \tag{16}$$

where $F_{ij}$ is the value of the $j$th component of the $i$th pattern point, and $C_k$ is the $k$th class. The weight $1/v_{kj}$ is used to take care of the variance of the classes so that a feature with higher variance has less weight (significance) in characterizing a class. Note that when all the feature values of a class are the same, then the standard deviation will be zero. In that case, we consider $v_{kj} = 1$ such that the weighting coefficient becomes 1. This is obvious because any feature occurring with identical magnitudes in all members of a training set is certainly an *important* feature of the set, hence its contribution to the membership function should not be reduced [8], [24].

The membership [8] of the $i$th pattern to class $C_k$ is defined as follows:

$$\mu_k\left(\vec{F}_i\right) = \frac{1}{1 + \left(\frac{z_{ik}}{f_d}\right)^{f_e}}, \tag{17}$$

where $z_{ik}$ is the weighted distance from (16) and the positive constants $f_d$ and $f_e$ are the denominational and exponential fuzzy generators controlling the amount of fuzziness in this class-membership set (i.e., in the distance set). Obviously $\mu_k(\vec{F}_i)$ lies in the interval $[0,1]$. Here (17) is such that the higher the distance of a pattern from a class, the lower its membership value to that class. It is to be noted that when the distance is 0, the membership value is 1 (maximum) and when the distance is infinite, the membership value is 0 (minimum).

It is to be mentioned that as the training data have fuzzy class separation, a pattern point $\vec{F}_i$ may correspond to one or more classes in the input feature space. So a pattern point belonging to two classes (say, $C_{k_1}$ and $C_{k_2}$) corresponds to two *hard* labels in the training data, with $\vec{F}_i$ tagged to classes $C_{k_1}$ and $C_{k_2}$ respectively. In other words, there are two or more occurrences of point $\vec{F}_i$ in the training set such that sometimes it is tagged to class $C_{k_1}$ and sometimes to class $C_{k_2}$. In this case $\vec{F}_i$ is used in computing $\boldsymbol{O}_{k_1}$, $\boldsymbol{O}_{k_2}$, $\boldsymbol{V}_{k_1}$, and $\boldsymbol{V}_{k_2}$ only. Here the $l$-dimensional vector $\boldsymbol{Z}_i$ has only two nonzero components, viz., $z_{ik_1}$ and $z_{ik_2}$. However in the *hard* case $\vec{F}_i$ corresponds to only one *hard* label in the training data, say $C_{k_1}$, such that $\vec{F}_i$ is used in computing $O_{k_1}$ and $V_{k_1}$ only. Note that $Z_i$ has $l$ nonzero components in the *fuzziest* case and only one nonzero component in the hard case.

In the *fuzziest* case, we may use the fuzzy modifier INT to enhance contrast [8] in class membership. We have

$$\mu_{\text{INT}(k)}\left(\vec{F}_i\right) = \begin{cases} 2\left[\mu_k\left(\vec{F}_i\right)\right]^2 & \text{for } 0 \le \mu_k\left(\vec{F}_i\right) \le 0.5 \\ 1 - 2\left[1 - \mu_k\left(\vec{F}_i\right)\right]^2 & \text{otherwise.} \end{cases} \tag{18}$$

This is needed to increase the contrast within class membership values, i.e., to decrease the ambiguity in taking a decision.

When the training input is linguistic, we use

$$z_{ik} = \sqrt{\sum_{j=1}^{n}\left[\sum_{p=1}^{3}\frac{1}{3}\left(\mu_p(F_{ij}) - \mu_p(o_{kj})\right)^2\right]} \quad \text{for } k = 1, \cdots, l, \tag{19}$$

where $\mu_1(F_{ij})$, $\mu_2(F_{ij})$, and $\mu_3(F_{ij})$ correspond to the membership values given by (12) and $\mu_p(o_{kj})$ correspond to the $3n$-dimensional representation by (11) and (13)–(15) of the class mean $O_k$ computed using the numerical input values. Note that we assume that corresponding to each feature axis there must exist some pattern points in each class (in the training set) having numerical values (kept aside after computing the input vector, as explained in subsection III-C). However this restriction does not exist for the test set as the desired membership values are not required here.

### B. Applying the Membership Concept

For the $i$th input pattern we define the desired output of the $j$th output node

$$d_j = \begin{cases} \mu_{\text{INT}(j)}\left(\vec{F}_i\right) & \text{in the } fuzziest \text{ case} \\ \mu_j\left(\vec{F}_i\right) & \text{otherwise,} \end{cases} \tag{20}$$

where $0 \le d_j \le 1$ for all $j$. The $l$ output neurons are clamped with the fuzzy output membership values of (20) during training and the error is back-propagated for appropriate weight updating by (5)–(8).

The block diagram given in Fig. 5 illustrates the various stages in computing the desired output vector to be clamped at the output nodes of the proposed model. Note that the enhancement phase (third stage in the figure) is required in the *fuzziest* case only.

During testing the output of the $j$th output neuron $y_j^H$ of (2), computed in a forward pass after clamping the $3n$-dimensional input vector from (11) for the test pattern $\vec{F}_t$ at the input layer, indicates the inferred membership value of the pattern $\vec{F}_t$ to the $j$th class. So the output is generated as a measure of finite belongingness to each class. This is unlike the conventional *crisp* concept of either belonging or not belonging to a class.

$$\vec{F}_i \xrightarrow{\text{weighted distance}} \begin{pmatrix} z_{i1} \\ \vdots \\ z_{il} \end{pmatrix} \xrightarrow{\text{membership}} \begin{pmatrix} \mu_1(\vec{F}_i) \\ \vdots \\ \mu_l(\vec{F}_i) \end{pmatrix} \xrightarrow{\text{enhancement}} \begin{pmatrix} \mu_{INT(1)}(\vec{F}_i) \\ \vdots \\ \mu_{INT(l)}(\vec{F}_i) \end{pmatrix} \xrightarrow{\text{desired vector}} \vec{d}$$

Fig. 5. Block diagram of the output phase of the proposed model.

## V. Fuzzy Extension to the MLP Model

Consider an $(H+1)$-layered MLP with $3n$ neurons in the input layer and $l$ neurons in the output layer, such that there are $H-1$ hidden layers. The input vector, with components $x_j^0$ of (3) represented as in (11), is clamped at the input layer while the desired $l$-dimensional output vector with components $d_j$ from (20) is clamped during training at the output layer. The outputs of the neurons $y_j^H$ in the various layers $h = 0, 1, \cdots, H$, of the network are computed using (1)–(3) to obtain the error $E$ at the output layer by (4). It is to be noted that the $y_j^H$'s in (2) and the $d_j$'s in (4) are expressed as continuous fuzzy membership values lying in the interval $[0,1]$. This is an extension to the conventional MLP model that deals with actual input values and *crisp* (binary) output values. The proposed model is found to classify fuzzy data with overlapping class boundaries in a more efficient manner.

### A. Weight Updating

Initially the connection weights $w_{ji}^h$ between each pair of neurons $i$ in layer $h$ and $j$ in layer $h+1$ are set to small random values lying in the range $[-0.5, 0.5]$. The error between the desired output $d_j$ and computed output $y_j^H$ in the output layer (for all output neurons) is evaluated using (4) and back-propagated for appropriate weight updating by (5)–(8) such that the weights are modified in proportion to their contribution to the error. The updating occurs after each cycle and is not accumulated over the entire sweep in order to offset the problem caused by the inherent higher redundancy of *natural* data or, in other words, to counter the repetitive occurrence of very similar training samples [25, p.7]. It should be mentioned that we adjust $\Delta w_{ji}^h(t)$ to ensure that $-0.1 \le \Delta w_{ji}^h(t) \le 0.1$ for each updating step, so that the possibility of *overshooting* of the weights may be minimized in the course of smoothly approaching a minimum error solution.

For a pattern point lying in a region of overlapping in the feature space, there may be more than one output node with (say) $d_j > 0.5$ indicating appreciable belongingness to more than one pattern class. In the conventional MLP for such a case only that node with a slightly higher activation is clamped to state 1 while the others are clamped to state 0. This may result in oscillations on the decision surface separating the pattern classes while training the network, because nearly identical patterns may be clamped to different classes. In such a case, terminating the algorithm at an arbitrary point may or may not yield a "good" weight vector.

In other words, the conventional *crisp* algorithm may not terminate or necessarily converge to a *useful* solution when the data are nonseparable with overlapping fuzzy classes. The pattern vectors that cause the classes to overlap are probably responsible for the oscillatory behavior of the algorithm because these "ambiguous" vectors, although relatively less characteristic of their classes, are given full weightage in one class while back-propagating the corresponding errors for weight updating.

The proposed model overcomes this problem by assigning the class membership values to the corresponding output nodes. Thus, the error to be back-propagated has more weightage in case of nodes with higher membership values and hence can induce greater weight corrections in favor of that class for input data that demand such an adjustment. This is desirable, as points that have a larger belongingness to a particular pattern class and possess less ambiguity should influence positively the positioning of the decision surface separating the pattern classes. The contribution of ambiguous or uncertain vectors (i.e., those with output membership close to 0.5) to the weight correction in favor of any class is automatically reduced. As the actual output vectors are modified to approximate the corresponding *ambiguous* desired output vectors (unlike the *crisp* vectors of the conventional model based on highest activation), it helps to prevent oscillation on the decision surface in such cases and thus enables the model to function efficiently in environments with fuzzy class separation. Note the possible reverse but significant impact of pattern vectors in inducing weight updating along interconnections corresponding to classes for which they have lower output membership values (closer to 0). This is because such low values signify the degree of *not belonging* to a class and hence involve less ambiguity.

### B. Learning Rate, Mean Square Error, and Cross Entropy

In "typical" conventional MLP the two-state output is either 0.8 (true) or 0.2 (false) [13], [15] and the learning rate $\epsilon$ and momentum $\alpha$ in (5) of the back-propagation algorithm are constant throughout training. In the proposed model we introduce output membership values distributed over the range $[0,1]$, depending on the degree of belongingness of a pattern vector to a class. These output values are actually distributed over a suitable subinterval, say $[0.2, 0.8]$, as attainment of 0 or 1 by $y_j^H$ in (2) requires an infinite input. We usually encounter a large number of intermediate output values during the training phase. To distinguish finer intricacies in the desired outputs, $\epsilon$ is gradually decreased in discrete steps, taking values from the chosen set $\{2, 1, 0.5, 0.3, 0.1, 0.05, 0.01, 0.005, 0.001\}$. Further, $\alpha$ is also decreased from an initial value of 0.9 for $\epsilon = 2$ to a final value of 0.5 for all other values of $\epsilon$.

In the course of training, at each sweep we use the mean square error ($mse$) and the cross entropy ($s$) as performance measures of learning. We define

$$mse = \left[\sum_{\vec{F}\in\text{trainset}} \sum_{j=1}^{l} \left(d_j - y_j^H\right)^2\right] \Big/ (l * |\text{trainset}|) \quad (21)$$

and

$$s = \left[\sum_{\vec{F}\in\text{trainset}} \sum_{j=1}^{l} \left\{-d_j \ln y_j^H - (1-d_j)\ln\left(1-y_j^H\right)\right\}\right] \div (\ln 2 * l * |\text{trainset}|). \quad (22)$$

Here |trainset| refers to the number of input pattern vectors in the training set, and $d_j$ corresponds to the $j$th desired output component. It is to be noted that both $mse$ and $s$ decrease as $y_j^H$ approaches $d_j$ for all $j$.

In our experiments, both $mse$ and $s$ are found to steadily decrease and reach a local minimum, for each value of $\epsilon$, after several sweeps (say, 20 to 300). This is the appropriate time to decrease $\epsilon$, as otherwise oscillation generally sets in. The necessity of modeling a number of intermediate output values in the range $[0,1]$ seems to cause the weight space to have ravines with a corresponding error surface of greater complexity, so that there exist a large number of local minima. One typically starts a long way from the minimum error solution, and spends a lot of time oscillating across ravines in the weight space before numerical convergence is attained. A larger $\epsilon$ is initially necessary to cause weight updating along the gradient to occur fast in roughly the right direction without getting stuck at poor local minima. The momentum term with $\alpha$ helps maintain movement in a stable direction and hence should be high in the early stages to prevent the weights from accidentally attaining large values caused by the initially high gradient that may often be in incorrect directions. As the weight vectors move closer to the correct orientation both the $mse$ and $s$ may decrease. After several sweeps, (varying from 20 to 300, say), when the weight vectors have settled somewhat, $\epsilon$ appears to be too large for the $\Delta w_{ji}^h$'s to change by the small amount now required to attain the necessary minimum along the more shallow gradient encountered here. The model may oscillate between local minima, as the weights tend to *overshoot* the minimum error solution; at this point $\epsilon$ needs to be decreased. At this time it is preferable to decrease $\alpha$ to speed progress in the most effective direction. This allows more weight to the gradient (now with lower $\epsilon$) to model finer intermediate output values while also maintaining some damping (via $\alpha$) to prevent oscillation. The same procedure is repeated for the discrete set of $\epsilon$'s until it stops at the lowest chosen value of $\epsilon$. This scheme of decreasing $\epsilon$ and $\alpha$ is somewhat different from the approach used in conventional MLP, but is somewhat analogous to the method reported in [26]. A comparison in performance of the proposed model with the latter (termed variation O) is given in Table III. Keeping $\epsilon$ and $\alpha$ constant, as in the *crisp* model, or increasing it gradually [27] seems to lead to oscillation in our model. This is because the fuzzy version of MLP apparently is incapable of modeling intermediate output values encountered in the fuzzy data space, so that the weights tend to overshoot.

We use two measures of *percent* correct classification for the training set. The output, after a number of updating, is considered a *perfect match* if the value of each output neuron $y_j^H$ is within a margin of 0.1 of the desired membership value $d_j$. This is a *stricter* criterion than the *best match,* where we test whether the $j$th neuron output $y_j^H$ (for a particular training pattern) has the highest (or maximum) activation when the $j$th component $d_j$ of the desired output vector also has the highest value, provided $y_j^H > 0.5$. Each pattern in the training set is supposed to have $y_j^H > 0.5$ for some $j$, i.e., to possess finite positive belongingness to at least one class. As training proceeds, the percent perfect match $p$ and the percent best match $b$ gradually increase as the neural net moves towards a minimum error solution. Note that for $b$ we consider only that class to which the pattern belongs *best*.

Let the various values of $\epsilon$ be indicated by $\epsilon_0 = 2$, $\epsilon_1 = 1, \cdots, \epsilon_q = 0.001$ such that $\epsilon_i$ indicates the $(i+1)$th value of $\epsilon$. Let $\alpha_0 = 0.9$ and $\alpha_1 = \alpha_2 = \cdots = \alpha_q = 0.5$. Note that $\alpha$ close to zero is avoided because small values of $\alpha$ are unable to prevent unwanted oscillations. We use

$$i = \begin{cases} i+1 \text{ if } mse(nt-kn) - mse(nt) < \delta \\ \quad \text{or } s(nt-kn) - s(nt) < \delta \\ i \quad \text{otherwise} \end{cases} \tag{23}$$

where $i = 0$ initially, $|\epsilon| = q+1$, and $0 < \delta \le 10^{-2}$. Here $mse(nt)$ and $s(nt)$ denote the mean square error by (21) and cross entropy by (22) respectively at the end of the $nt$th sweep through the training set; $kn$ is a positive integer such that $mse$ and $s$ are sampled at intervals of $kn$ sweeps. The process is terminated when $i > q$ and $\epsilon_q = 0.001$. At this stage the network is said to have converged to a good minimum error solution if $90 \le b \le 100$. Obviously $p \le b$ always. The corresponding value of $nt$ indicates the maximal number of sweeps required in the process. It is to be mentioned that the relations $\Delta mse_j < \Delta mse_i$ and $\Delta s_j < \Delta s_i$ hold when $\epsilon_j < e_i$.

### C. Testing Phase

After training, the proposed model has presumably encoded in its weights information regarding class discrimination in the feature space. This is the desired fuzzy classifier. In the second stage, a separate set of test patterns (initially kept aside from the same fuzzy data) is supplied as input to the neural network model and its performance is evaluated.

During this phase a labeled input test vector $\vec{F}$ in the $3n$-dimensional linguistic space of (11) is applied to the network. The output neurons yield values by (1)–(3) indicating fuzzy memberships of the test pattern in the corresponding output classes. The normalized membership values in the range $[0.2, 0.8]$ are reconverted to the actual range $[0, 1]$.

Let the $j_1$th and $j_2$th neurons in the output layer $H$ generate the highest and second highest outputs $y_{j_1}^H$ and $y_{j_2}^H$, respectively, for test pattern $\vec{F}_t$. These represent, respectively, membership values to class $j_1$ using the best choice and to class $j_2$ using the second choice. It may be that $y_{j_1}^H \simeq y_{j_2}^H$ for a pattern $\vec{F}_t$ lying in overlapping regions in the input feature space, so that there is some ambiguity of decision in such cases. A confusion matrix is generated for the test set to evaluate the network performance. To determine the row of an entry in the said confusion matrix, we use the class $j$ of the particular test data as obtained from the maximum of the desired outputs $d_j$ by (16)–(20). Note that the nonzero components of $\boldsymbol{Z}_t$ are determined (as explained above) from the *hard* labels attached to the test pattern $\vec{F}_t$ as well as the training patterns having the same feature values. The column corresponds to the neuronal output class providing the *best match* using (1)–(3) and (11). The generalization capability of the proposed net in correctly classifying unseen patterns is then verified.

The mean square error for the test patterns, $mse_t$, is defined from (21) as

$$mse_t = \left[ \sum_{\vec{F} \in \text{testset}} \sum_j \left(d_j - y_j^H\right)^2 \right] \Big/ \left(l * |\text{testset}|\right) \tag{24}$$

where |testset| corresponds to the number of pattern vectors used during testing, and the $l$ components of $y_j^H$ in the output layer are computed from the $3n$ components $x_j^0$ of pattern $\vec{F}$ using (1)–(3). The $l$ components of the desired output $d_j$ are generated by using (20). This $mse_t$ is a measure of the amount of misclassification for the set of test patterns. Analogously, the cross entropy for the test patterns, $s_t$, is defined from (22) as

$$s_t = \left[ \sum_{\vec{F} \in \text{testset}} \sum_j \left\{ -d_j \ln y_j^H - (1-d_j) \ln\left(1 - y_j^H\right) \right\} \right] \div \left(\ln 2 * l * |\text{testset}|\right) \tag{25}$$

The closer $y_j^H$ is able to approach $d_j$, the lower the value of $s_t$. Hence this is also a measure of the amount of misclassification for the test set.

## VI. Implementation and Results

The above-mentioned algorithm was tested on a set of 871 Indian Telugu vowel sounds collected by trained personnel [28]. These were uttered in a consonant–vowel–consonant context by three male speakers in the age group of 30 to 35 years. The simulation was in C on a VAX-8650 computer. The data set has three features, $F_1$, $F_2$ and $F_3$, corresponding to the first, second, and third vowel format frequencies obtained through spectrum analysis of the speech data. Thus the dimension of the input vector in (11) for the proposed model is 9. Note that the boundaries of the classes in the given data set are seen to be ill defined (fuzzy). Fig. 6 shows a 2-D projection of the 3-D feature space of the six vowel classes $(\partial, a, i, u, e, o)$ in the $F_1$–$F_2$ plane (for ease of depiction).

It may be mentioned that the conventional *crisp* MLP has also been used in the vowel classification problem [29] to generate hard partitions in the formant frequency space of English vowel data. Besides, a radically different approach has also been used in [30], where context-dependent information is used to generate *hard* decision boundaries in an attempt to separate a different set of overlapping English vowel data in the formant frequency space.

On the other hand, the proposed neural network model, incorporating fuzzy concepts at various levels, has been used here to demonstrate its effectiveness in classifying the fuzzy Telugu vowel data. In this case overlapping fuzzy decision regions are generated in the output space.

The model under consideration has been tested for different numbers of hidden layers $(H-1)$ (such as 1, 2, 3), and with different numbers of neurons $m$ (such as 10, 15, 20), in each such layer. During learning, various sizes of training sets have been used by randomly choosing different percentages *perc* of samples (such as 10%, 50%), from each representative vowel

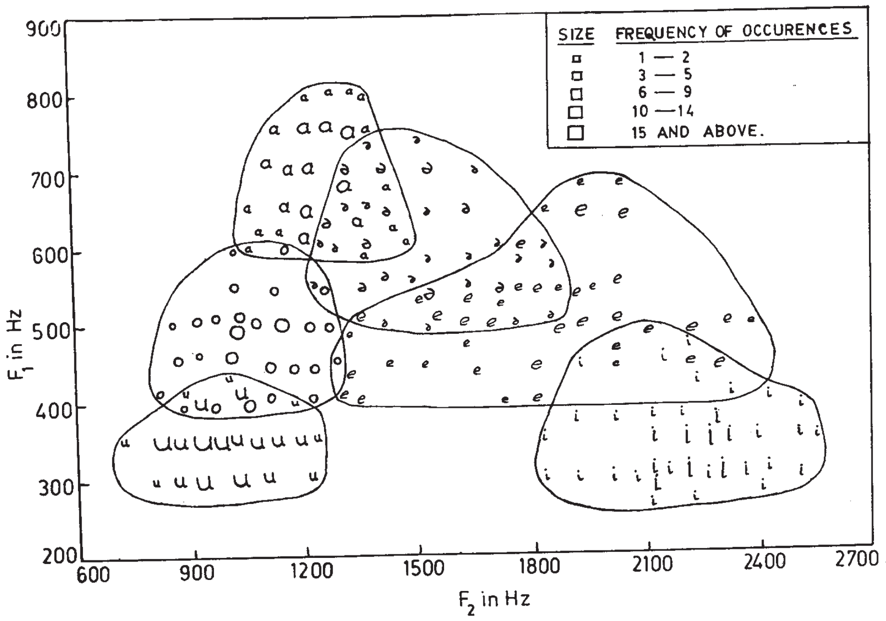


Fig. 6. Vowel diagram in the $F_1 - F_2$ plane.

class. In each case the remaining percentage (100-*perc*) of data was used as the test set. The parameters *fdenom*, $f_d$ $f_e$, $kn$, and $\delta$ in (14), (15), (17), and (23) were chosen as *fdenom* = 0.8, $f_d = 5$, $f_e = 1$, $kn = 10$, and $\delta = 0.0001$ after several experiments. Further, we observed that with $hdec > 0$ in (5), $\epsilon_q = 0.1$ at (23) gives best results whereas with $hdec = 0$ we require $\epsilon = 0.001$.

In Figs. 7 and 8, part (a) plots the percent correct classification using the best match criterion, while part (b) shows the variation of $mse$ of (21), $s$ of (22), and $mes_t$ of (24). In part (a), the class number $j$ (1 for $\partial$, 2 for $a$, 3 for $i$, 4 for $u$, 5 for $e$, 6 for $o$) indicate the classwise correct classification of the test set. The variables $b$ and $p$ correspond to the *best* match and *perfect* match criteria respectively obtained for the training set, while $t$ indicates the percent *best* match performance over the entire test set as a whole. In part (b) of the figures, the variation of the cross entropy $s$ is shown by the dotted curve while the mean square error for the training ($mse$) and test ($mse_t$) sets are plotted using solid curves. Note that the left vertical axis corresponds to the mean square error while the right vertical axis indicates the cross entropy.

Fig. 7 is drawn to illustrate the effect of varying the number of neurons in the hidden layers of the proposed model. We considered three hidden layers with a distribution $m : m : m$ of neurons such that $m = 10, 15$, and 20. It was observed that $m = 10$ gives the best results. A larger size is seen to provide better performance over the training set (higher $b, p$ in part (a) and lower mean square error $mse$ and cross entropy $s$ in part (b)), perhaps because of overlearning. This is probably due to "memorization," with poor generalization, which leads to poorer performance over the test set (lower $t$ in part (a) and higher mean square error $mse_t$ in part (b)). A smaller size of $m$ was incapable of handling all the information required, perhaps owing to insufficient capacity.

In Fig. 8 we observe the effect of introducing different numbers of hidden layers ($H-1$) with $m = 10$ neurons in each layer. A single hidden layer is obviously incapable of capturing the intricacies of the feature spece. Two hidden layers lead to a much better performance, as expected. However three hidden layers seem to give an overall best performance over both the training and test sets. The network fails to converge with four hidden layers, perhaps because of the resulting very large number of complicated interconnections and the comparatively limited set of data available for training the net.

Fig. 9 demonstrates the effects of $\epsilon$ and $\alpha$ of (5) on the performance of the model. A fixed low learning rate of $\epsilon = 0.1$ and a high momentum of $\alpha = 0.9$, as in the conventional MLP, result in oscillation and the worst classification rates (worst recognition for class 1, i.e., $\partial$, and nil *perfect* match). Varying $\epsilon$ from higher to lower values in discrete steps leads to notably better overall performance. Keeping $\alpha$ constantly low at 0.5 for this $\epsilon$ does not provide enough damping and results in a comparatively poor performance as oscillations still cannot be avoided. Clamping $\alpha$ high at 0.9 with more damping, while $\epsilon$ varies as proposed, improves the situation. However changing $\alpha$ from 0.5 to 0.9 shows some improvement in the recognition rate for class 1 ($\partial$) at the expense of the neighboring overlapping class 5 (e). The best overall performance is obtained by decreasing $\alpha$ from 0.9 to 0.5, while simultaneously also decreasing $\epsilon$ in steps. This has already been explained, in Section V. Increasing $\epsilon$ from lower to higher values leads to instability such that the network becomes incapable of classifying the fuzzy patterns. This is in sharp contrast of some models [27].

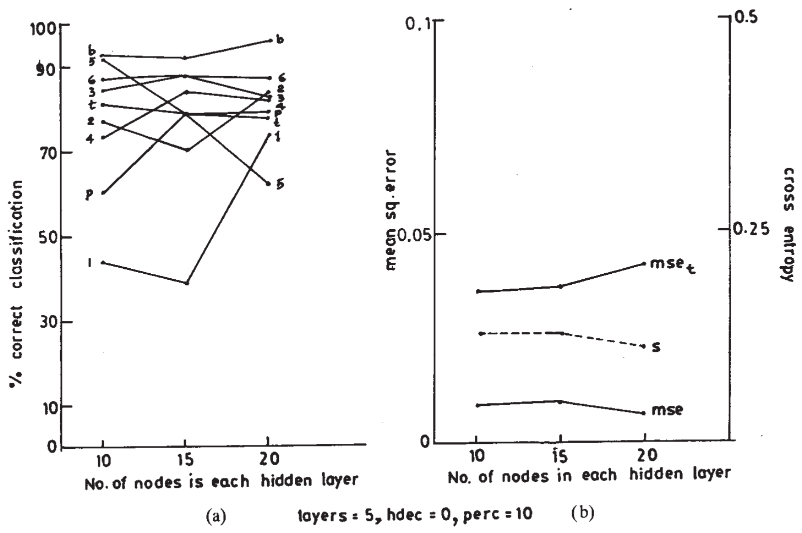


Fig. 7. Five-layered net with $perc = 10$ and $hdec = 0$. (a) Correct classification (%) *using best match* versus size of hidden layers. (b) Mean square error and cross entropy versus size of hidden layers.

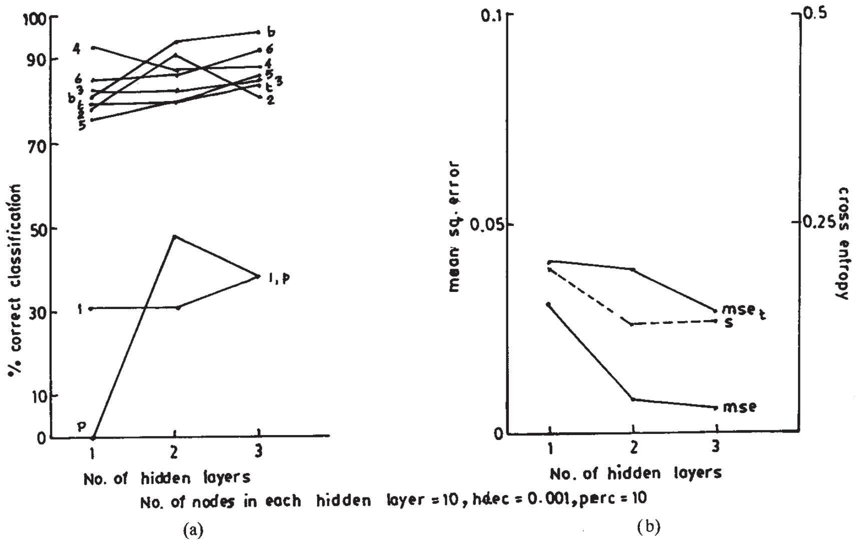


Fig. 8. Layered neural net with $perc = 10$, $hdec = 0.001$, and $m = 10$ nodes in each hidden layer. (a) Correct classification (%) *using best match* versus number of hidden layers. (b) Mean square error and cross entropy versus number of hidden layers.

Fig. 10 is drawn to illustrate the (crisp) output partitioning capability of the proposed network with three hidden layers using *alpha cuts.* An alpha cut of a fuzzy set $A$ is defined as $A_{\alpha'} = \{r|\mu_A(r) \geq \alpha'\}$, $1 \geq \alpha' \geq 0$. A pattern with output membership value $y_j^H > \alpha'(= 0.5)$ is plotted as a member of class $j$. The $l = 6$ vowel classes are labeled by their corresponding class numbers. Fig. 10(a) depicts the resultant output map (class boundaries) over the entire pattern set (both training and testing data) in the two-dimensional formant frequency space showing the fuzzy overlapping, as expected. In Fig. 10(b), the training samples for each class are superimposed on the generated partitions to demonstrate their generalization capability and impact on the classification performance of the model. Note that the topological ordering of the vowel classes with respect to each other and the amount of overlapping between them bear much similarity to the actual partitioning illustrated in Fig. 6. This shows that the proposed fuzzy model helps to satisfactorily preserve the structure of ambiguous (fuzzy) classes. (It is to be noted from Fig. 10(b) that sometimes distortion and overshadowing of the symbols occur because of the overlapping of the training points from the different pattern classes. This is especially true in the case

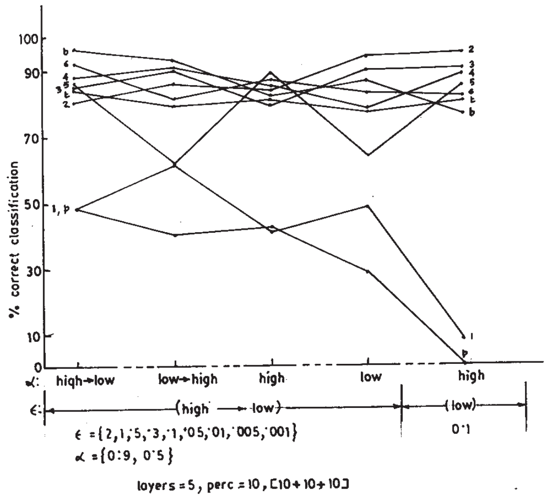
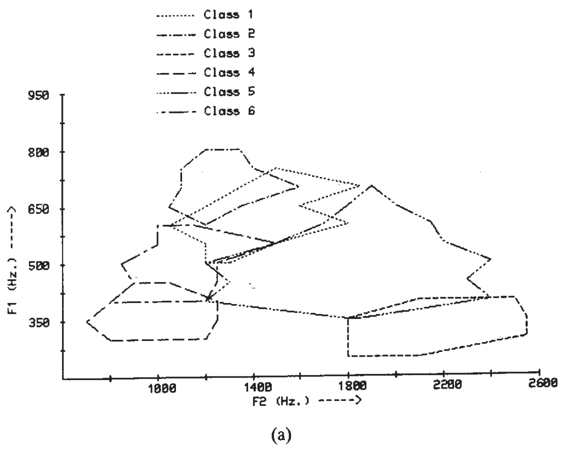


Fig. 9. Effect of various combinations of $\epsilon$ and $\alpha$ on the correct classification (%) *using best match* of a five-layered net with $perc$ = 10 and $m$ = 10 nodes in each hidden layer.



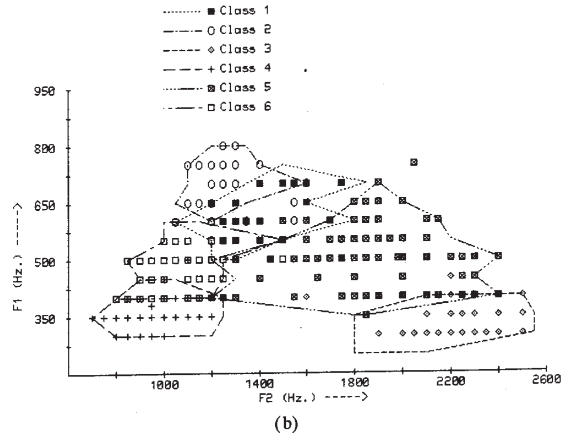


Fig. 10. Output partitioning of pattern space generated by a five-layered net with $perc$ = 50, $hdec$ = 0, $\alpha'$ = 0.5, and $m$ = 10 nodes in each hidden layer. (a) Class boundaries, with (b) the superimposition of the training samples on the different partitions.

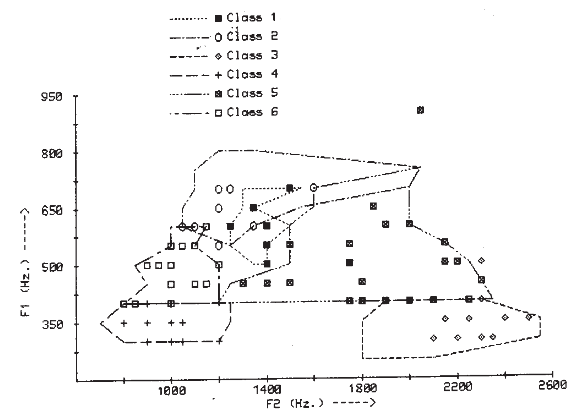


Fig. 11. Superimposition of training samples on the output partitioning of pattern space generated by a four-layered net with $perc$ = 10, $hdec$ = 0.001, $\alpha'$ = 0.5, and $m$ = 10 nodes in each hidden layer.

of the co-occurrence of points from classes 1 and 6 in the $F_1$–$F_2$ plane, owing to the nature of their symbols, as is evident from the figure).

Fig. 11 depicts the output partitioning obtained over the entire data set with two hidden layers. Here also the training samples are shown superimposed on the generated partitions. This network is seen to be incapable (compared with the one with three hidden layers) of properly classifying all pattern points, classes 1($\partial$) and 2($a$) suffering the most. This may also be verified from Fig. 8.

In Fig. 12 we illustrate the variation of the best match $b$ and perfect match $p$ performance (denoted by solid curves) and the mean square error $mse$ (plotted using a dotted curve) with the number of sweeps over the training set. The solid points along the curves mark the sweep number at which $\epsilon_i$ is changed to $\epsilon_{i+1}$ by (23). It is observed that initially high $\epsilon$ and $\alpha$ are necessary to cause fast weight updating in roughly the right direction without getting stuck at undesirable local minima. As the weight vectors approach a "correct" orientation, improvement in $b$, $p$, and $mse$ becomes slower, as indicated by the flatter nature of the curves for lower $\epsilon$. The need to model finer intermediate output values requires $\epsilon$ to be gradually decreased, as explained in Section V, until a suitably good solution is obtained. Note that the curves are neither smooth nor monotonically increasing/decreasing. This is due to the *local* nature of the weight updating process, which ultimately enables the neural net model to arrive at a *good global* solution by avoiding spurious local minima.

Table I is used to provide examples of input vectors $\vec{\boldsymbol{F}}_i$, desired output vectors $\boldsymbol{d}$, and the actual output vectors $\boldsymbol{y}^H$ (both in the range $[0, 1]$) for a set of sample patterns. Here the first three patterns correspond to training sets while the remaining three patterns belong to the test set. Note that although the desired output vector has a *hard* label for the fifth entry (test pattern), indicating belongingness to class $e$ alone, the proposed model generalizes to yield a fuzzy actual output vector in keeping with the vowel diagram of Fig. 6.

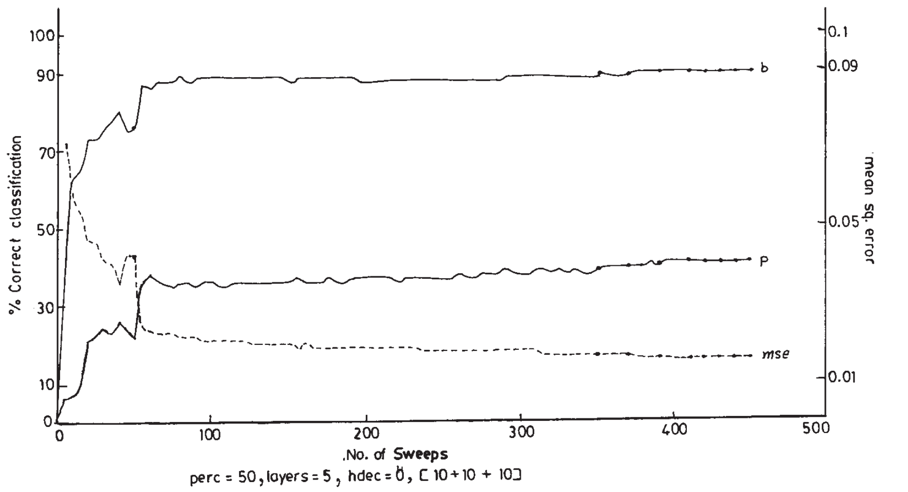


Fig. 12. Variation of perfect match (%), best match (%), and mean square error with number of *sweeps* through the training set, as $\epsilon$ is decreased in discrete steps from $\varepsilon_0 = 2$ to $\epsilon_q = 0.001$, using a five-layered neural net with $perc = 50$, *hdec* $= 0$, and $m = 10$ nodes in each hidden layer.

TABLE I
INPUT, DESIRED OUTPUT, AND ACTUAL OUTPUT VECTORS FOR A SET OF SAMPLE PATTERNS PRESENTED TO THE FIVE-LAYER NEURAL NETWORK

| Input Features $F_1, F_2, F_3$ | Input Vector | | | | | | | | | Output Vector | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | Desired | | | | | | Actual | | | | | |
| 700, 1500, 2600 | .07 | .70 | .93 | .69 | .96 | .31 | .30 | .96 | .70 | .84 | 0. | 0. | 0. | 0. | 0. | .87 | .02 | 0. | 0. | 0. | 0. |
| 750, 1150, 2600 | .01 | .43 | .99 | .98 | .47 | .02 | .30 | .96 | .70 | 0. | .82 | 0. | 0. | 0. | 0. | .03 | .87 | 0. | 0. | 0. | 0. |
| 300, 2200, 2700 | .94 | .05 | 0. | 0. | .29 | .99 | .15 | .84 | .85 | 0. | 0. | .91 | 0. | 0. | 0. | 0. | 0. | .96 | 0. | 0. | 0. |
| 350, 700, 2630 | .99 | .19 | 0. | .82 | 0. | 0. | .25 | .93 | .75 | 0. | 0. | 0. | .76 | 0. | 0. | .01 | 0. | .03 | .89 | 0. | 0. |
| 550, 1500, 2500 | .62 | .99 | .38 | .69 | .96 | .31 | .5 | 1. | .5 | 0. | 0. | 0. | 0. | .88 | 0. | .58 | .06 | 0. | .04 | .18 | 0. |
| 450, 1200, 2300 | .93 | .70 | .07 | .96 | .58 | .04 | .85 | .84 | .15 | 0. | 0. | 0. | 0. | 0. | .86 | .14 | .02 | .06 | .07 | .07 | .75 |

Table II compares the average percent correct recognition score (on the test set using the *best match* criterion, both classwise and overall) of the proposed fuzzy neural net model to that of a conventional MLP, its hard version, and the standard Bayes classifier. The training and test sets each constitute 50% of the pattern data. A comparison of the overall performance on the training set between the various neural net model variations is also provided. For this both the *best match b* and *perfect match p* criteria are used. We have used the Bayes classifier for multivariate normal patterns with the *a priori* probabilities $p_i = |C_i|/N$, where $|C_i|$ indicates the number of patterns in the $i$th class and $N$ is the total number of pattern points. The covariance matrices are different for each pattern class. The choice of normal densities for the vowel data has been found to be justified [31]. The MLP model and its variations all use three hidden layers with $m$ hidden nodes in each such layer. We have used $m = 10$ and 20 nodes in the current experiment. The first two MLP versions in Table II use *crisp* binary representation at the output. However in (a) the actual input features in the $n$-dimensional feature space are normalized to the range $[0, 1]$ and this is termed the conventional model. In (b) $3n$ linguistic input features with *crisp* values are used such that corresponding to a pattern $\vec{F}_i$, along the $j$th axis, we clamp the highest of $\mu_{\text{low}(F_{ij})}(\vec{F}_i)$, $\mu_{\text{medium}(F_{ij})}\vec{F}_i$, and $\mu_{\text{high}(F_{ij})}(\vec{F}_i)$ of (11) to 1 while the remaining two are kept clamped at 0. This is referred to as the MLP model with hard linguistic input features. In (c) we use the proposed fuzzy version of the MLP.

Although model (a) is found to generate somewhat high recognition scores (%) using *best match* we have observed that the outputs in most such *winning* cases were less than 0.5. This is perhaps an aftereffect of the *crisp* output labeling used for the training data. Hence the output partitioning of the feature space using alpha cut value $\alpha' = 0.5$ (as explained earlier) fails to generate the boundaries for classes $\partial$ and $o$ (using $m = 20$) and for class $o$ (using $m = 10$). Besides, the other class boundaries are also observed to be not very appropriate. This behavior of models (a) and (b) corresponds to the poorer recognition score (%) on the training set using the *perfect* match criterion $p$. Model (b) is found to have the worst efficiency. It is found to be able to generate all the output partitions, even though distorted in several cases. The *crisp* output labellings in (a) and (b) are perhaps therefore responsible for the resultant hindrance to *proper* convergence

TABLE II
COMPARISON OF PERCENT CORRECT RECOGNITION SCORE USING *best match* ON THE TEST SET BETWEEN THE BAYES CLASSIFIER AND VARIOUS NEURAL NET MODELS

| Model | | Bayes Classifier | Neural Net Models (a) Conventional | | (b) Hard Linguistic Input | | (c) Proposed Fuzzy Version | |
|---|---|---|---|---|---|---|---|---|
| | | | $m = 10$ | $m = 20$ | $m = 10$ | $m = 20$ | $m = 10$ | $m = 20$ |
| T | $\partial$ | 41.6 | 55.5 | 44.4 | 24.4 | 34.1 | 51.2 | 55.5 |
| e | a | 91.1 | 86.6 | 88.8 | 64.4 | 60.0 | 84.4 | 80.0 |
| s | i | 93.0 | 81.4 | 81.4 | 88.8 | 90.9 | 81.9 | 91.5 |
| t | u | 94.7 | 88.1 | 90.7 | 76.2 | 55.7 | 86.8 | 86.6 |
| s | e | 71.1 | 92.3 | 86.5 | 69.7 | 65.6 | 92.4 | 85.9 |
| e | o | 71.1 | 86.6 | 85.5 | 85.9 | 91.4 | 89.5 | 81.3 |
| t | Overall | 79.2 | 84.6 | 82.8 | 72.5 | 70.2 | 84.2 | 83.6 |
| *perfect p* | | — | 49.6 | 43.6 | 35.3 | 6.3 | 55.6 | 58.1 |
| *best b* | | — | 86.0 | 87.6 | 77.7 | 71.5 | 92.2 | 92.2 |

The neural network has three hidden layers, with $m$ hidden nodes in each layer, and $hdec = 0$. The *perfect match p* and *best match b* over the training set are also compared among the various neural net models.

leading to *poor* output partitioning. However the proposed fuzzy version is found to provide a very satisfactory overall performance.

It is to be noted that a good statistical (Bayes) classifier requires a great deal of sequential computation and a large number of reference vectors, whereas a neural network is massively parallel and can generalize appreciably well. Incorporation of fuzzy concepts in the neural net further enhances its capability in handling the impreciseness in input patterns and the uncertainty arising from overlapping/ill-defined regions.

In Table III we demonstrate a comparison in performance of the proposed model with that of the model in [26] (referred to as Model $O$ here). In model $O$, the learning rate $\epsilon$ is modified dynamically by an additive increase when the error measure decreases and a multiplicative decrease otherwise. We use fuzzy input and output representations (as explained earlier) for both models, to clearly demonstrate the usefulness of our mode of variation of $\epsilon$. The perfect match $p$ and the best match $b$ are provided for the training set. It is observed that the performance of the proposed model is generally better in terms of the recognition scores (%). The output partitioning has also been found to be more appropriate in our model. Note that the total number of sweeps required is larger in the case of model $O$.

TABLE III
COMPARISON OF PERCENT CORRECT RECOGNITION SCORE USING *best match* ON THE TEST SET AND BOTH *perfect match p* AND *best match b* ON THE TRAINING SET, BETWEEN THE PROPOSED MODEL AND THE SCHEME in [26] (HERE REFERRED TO AS MODEL $O$) WITH FIVE-LAYER NEURAL NETWORK HAVING TEN NODES IN EACH HIDDEN LAYER, $hdec = 0$, AND $perc = 50$

| Model | | Proposed | Variation O |
|---|---|---|---|
| T | $\partial$ | 51.2 | 53.6 |
| e | a | 84.4 | 88.8 |
| s | i | 81.9 | 83.1 |
| t | u | 86.8 | 89.4 |
| s | e | 92.4 | 91.5 |
| e | o | 89.5 | 77.9 |
| t | Overall | 84.2 | 83.1 |
| *best b* | | 92.2 | 91.7 |
| *perfect p* | | 55.6 | 69.6 |
| No. of sweeps | | 460 | 598 |

The input and output vector representations are fuzzy (in both cases) to aid better comparison between the two modes of varying the learning rate $\epsilon$.

## VII. CONCLUSIONS AND DISCUSSION

A neural network model based on the multilayer perceptron using the back-propagation algorithm which is capable of fuzzy classification of patterns has been defined and tested. The model converts numerical and linguistic inputs to linguistic terms, and provides output decisions in terms of class membership values. Back-propagated errors are assigned appropriate weightage for weight updating depending upon the membership values at corresponding outputs. The learning rate $\epsilon$ and damping coefficient $\alpha$ are gradually decreased to prevent oscillations as the neural network converges to a minimum error solution in a maximal number of sweeps through the training set. The problem of vowel recognition in the consonant–vowel–consonant context has been considered to demonstrate the effectiveness of the proposed model at every stage.

It may be noted that the parameters $\epsilon$ and $\alpha$ traverse a range of values in the course of the computations and, as such, no particular choices need to be made. However, in general, one may choose initially $0.6 < \alpha < 1.0$, $1.0 < \epsilon < 2.5$ and finally $0.2 < \alpha < 0.6$, $0.0001 < \epsilon < 0.1$ for good results. This is because initially high $\epsilon$ and $\alpha$, and finally very low $\epsilon$ but not so low $\alpha$ are needed, as explained in subsection V-B. Further, usually $0 < hdec < 0.01$.

It is worth mentioning that the fuzzy MLP in [11] used trapezoidal possibility distributions to represent each linguistic term, sampled at a fixed number of values over their respective domains of discourse, for fuzzy inferencing. Hence the sampling frequency had a direct bearing on the faithfulness of the representation of the linguistic terms as well as the cost of calculation required. On the other hand, we considered the $\pi$ function to model the inputs in terms of the linguistic variables. This resulted in a more cost-effective representation with fewer inputs dedicated to a particular input feature. The concept of using class membership functions for fuzzy modeling of multiclass problems was also basically different from the two-class single-layer-perceptron based fuzzy pattern classification algorithm [32].

Note that only three linguistic terms, *low, medium,* and *high,* have been used in the input stage of the proposed model. Incorporation of fuzzy hedges such as *more or less, very,* and *nearly* as additional properties might enhance performance. We have used *contrast enhancement* in the output stage of the proposed model.

In this connection, it may also be mentioned that the MLP has been used to approximate continuous functions for prediction in time series [33]. This approach of input–output representation was basically different from our proposed model, although in both cases the range of allowable values was $[0, 1]$. We used fuzzy membership concepts for modeling both the input and output vectors. While the input was generated in linguistic terms, the output was given as class membership values.

In the algorithm reported in [12], the smallest number of iterations was evaluated using a different approach. There an attempt was made to generate the "best" solution, and this was rather expensive, as it required computations involving results from various runs over several ratios of training and test set combinations. The proposed model, on the other hand, applied a heuristic in an attempt to obtain a *good* solution for any chosen ratio of training-test set combination in a single *run* (one sequence of *sweeps*) over this set.

Neural net performance for fuzzy classification of speech data was found to compare favorably with that of the Bayes classifier trained on the same data. In the proposed model we used many parallel interconnection links with simple processing elements. Therefore with appropriate parallel hardware the proposed model should be able to perform much faster and hence more efficiently than serial techniques. The incorporation of fuzzy concepts at various levels of the proposed model seemed to enhance its performance.

Regarding the relative merits of the different numbers of hidden layers and/or the total number of neurons, it might be noted that the approach in [34] used an algorithm making weight updates only once per sweep. The proposed model, on the other hand, updated its weights after each pattern presentation, as explained in subsection V-A. [25]. In this work we demonstrated a few experimental observations in this direction. For this we used $m$ nodes in each hidden layer to bring uniformity in the design of the neural network. We did not consider different numbers of nodes in the various hidden layers in order to simplify the building procedure. Incorporation of pruning and further addition of layers as in [34], to improve the generalizing capabilities of the network, seemed nontrivial in the current context and might be an interesting topic for future research.

## Acknowledgment

The authors express their gratitude to Prof. D. Dutta Majumder for his interest in this work and to S. Chakraborty for drawing the diagrams.

## References

[1] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Acoust., Speech, Signal Processing Mag.,* vol. 61, pp. 4–22, 1987.

[2] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing,* vol. 1, Cambridge, MA: MIT Press, 1986.

[3] T. Kohonen, "An introduction to neural computing," *Neural Networks,* vol. 1, pp. 3–16, 1988.

[4] T. Kohonen, *Self-Organization and Associative Memory.* Berlin: Springer-Verlag, 1989.

[5] S. E. Fahlman and G. E. Hinton, "Connectionist architectures for artificial intelligence," *IEEE Computer,* pp. 100–109, 1987.

[6] G. E. Hinton, "Connectionist learning procedures," *Artificial Intelligence,* vol. 40, pp. 185–234, 1989.

[7] G. J. Klir and T. Folger, *Fuzzy Sets, Uncertainty and Information.* Reading, MA: Addison Wesley, 1989.

[8] S. K. Pal and D. Dutta Majumder, *Fuzzy Mathematical Approach to Pattern Recognition.* New York: Wiley (Halsted Press), 1986.

[9] C. C. Lee, "Intelligent control based on fuzzy logic and neural net theory," in *Proc. 1990 Int. Conf. Fuzzy Logic and Neural Networks,* (Iizuka, Japan), pp. 759–764.

[10] H. Ishibuchi and H. Tanaka, "Identificaiton of real-valued and intervalued membership functions by neural networks," in *Proc. 1990 Int. Conf. Fuzzy Logic and Neural Networks,* (Iizuka, Japan), pp. 179–182.

[11] J. M. Keller and H. Tahani, "Backpropagation neural networks for fuzzy logic," *Inform. Sci.,* to be published.

[12] H. Takagi and I. Hayashi, "Artificial neural network driven fuzzy reasoning," *Int. J. Approximate Reasoning,* vol. 5, pp. 191–212, 1991.

[13] G. E. Hinton, "Learning translation invariant recognition in a massively parallel network," in *PARLE: Parallel Architectures and Languages Europe,* (G. Goos and J. Hartmanis, Eds.) Berlin: Springer-Verlag, 1987, pp. 1–13.

[14] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Comlex Systems,* vol. 1, pp. 145–168, 1987.

[15] G. E. Hinton, "Learning distributed representation of concepts," in *Proc. 8th Annual Conf. Cognitive Science Society,* (Amherst, MA), 1986, pp. 1–12.

[16] D. J. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans. Acoust., Speech, Signal Process.,* vol. 36, pp. 1162–1168, 1988.

[17] G. Tesauro and T. J. Sejnowski, "A parallel network that learns to play backgammon," *Artificial Intelligence,* vol. 39, pp. 357–390, 1989.

[18] J. Villarreal and P. Beffes, "Sunspot prediction using neural networks," in *Proc. 3rd Annual Workshop on Space Operations, Automation and Robotics (SOAR'89), (Johnson Space Center, Houston, TX, NASA Conf. Publication 3059,* 1989, pp. 525–535.

[19] R. P. Gorman and T. J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Trans. Acoust., Speech Signal Process.,* vol. 36, pp. 1135–1140, 1988.

[20] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* New York: Plenum Press, 1981.

[21] E. H. Ruspini, "A new approach to clustering," *Inform. Cont.,* vol. 15, pp. 22–32, 1969.

[22] S. K. Pal and P. K. Pramanik, "Fuzzy measures in determining seed points in clustering," *Pattern Recognition Lett.,* vol. 4, pp. 159–164, 1986.

[23] S. K. Pal and D. P. Mandal, "Linguistic recognition system based on approximate reasoning," *Inform. Sci.,* vol. 61, no. 2, pp. 135–161, 1992.

[24] G. S. Sebestyen, *Decision Making Processes in Pattern Recognition.* New York: Macmillan, 1962.

[25] S. Becker and Y. le Cun, "Improving the convergence of back-propagation learning with second order methods," Tech. Rep. CRG-TR-88-5, Connectionist Research Group, University of Toronto, Canada, 1988.

[26] R. H. Silverman and A. S. Noetzel, "Image processing and pattern recognition in ultrasonograms by backpropagation," *Neural Networks,* vol. 3, pp. 593–603, 1990.

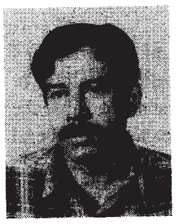
[27] D. C. Plaut and G. E. Hinton, "Learning sets of filters using backpropagation," *Computer Speech and Language,* vol. 2, pp. 35–61, 1987.

[28] S. K. Pal and D. Dutta Majumder, "Fuzzy sets and decision making approaches in vowel and speaker recognition," *IEEE Trans. Syst. Man, Cybern.,* vol. SMC-7, pp. 625–629, 1977.

[29] W. Y. Huang and R. P. Lippmann, "Neural net and traditional classifiers," in *Neural Information Processing Systems,* (D. Z. Anderson, Ed.), New York: American Institute of Physics, 1988, pp. 387–396.

[30] R. Watrous, "Context-modulated discrimination of similar vowels using second-order connectionist networks," Tech. Rep. CRG-TR-89-5, Connectionist Research Group, University of Toronto, Canada, 1989.

[31] S. K. Pal, "Studies on the application of fuzzy set-theoretic approach in some problems of pattern recognition and man–machine communication by voice," Ph.D. Thesis, University of Calcutta, Calcutta, India, 1978.
[32] J. K. Keller and D. J. Hunt, "Incorporating fuzzy membership functions into the perceptron algorithm," *IEEE Trans. Pattern Anal. Machine Intell.* vol. PAMI-7, pp. 693–699, 1985.
[33] A. Lapedes and R. Farber, "How neural nets work," in *Neural Information Processing Systems,* D. Anderson, Ed. New York: American Institute of Physics, 1988, pp. 442–456.
[34] J. Sietsma and R. Dow, "Creating artificial neural networks that generalize," *Neural Networks,* vol. 4, pp. 67–79, 1991.



**Sankar K. Pal** (M'81–SM'84) obtained B.Sc. (Hons) in physics and B. Tech., M. Tech., and Ph.D. degrees in radiophysics and electronics in 1972, 1974, and 1979, respectively from the University of Calcutta, India. In 1982 he received another Ph.D. in electrical engineering along with DIC from Imperial College, University of London, England. He received a Commonwealth Scholarship in 1979 and an MRC (U.K.) postdoctoral award in 1981 to work at Imperial College, London. In 1986 he was awarded a Fulbright Post-doctoral Visiting Fellowship to work at the University of California, Berkeley, and the University of Maryland, College Park.

He is a Professor in the Electronics and Communication Science Unit at the Indian Statistical Institute Calcutta. In 1989 he received an NRC-NASA Senior Research Award to work at the NASA Johnson Space Center, Houston, TX. He received the 1990 Shanti Swarup Bhatnagar Prize in Engineering Sciences for his contribution in pattern recognition.

At present he is working as a Guest Investigator in the Software Technology Branch, NASA Johnson Space Center. He served as a Profesor-in-Charge of the Physical and Earth Sciences Division, Indian Statistical Institute, during 1988–90. He was also a Guest Lecturer (1983–86) in Computer Science, Calcutta University. His research interests mainly include pattern recognition, image processing, artificial intelligence, neural nets, and fuzzy sets and systems. He has coauthored the books *Fuzzy Mathematical Approach to Pattern Recognition,* at John Wiley & Sons (Halsted, NY 1986), which received the Best Production Award at the 7th World Book Fair, New Delhi, and *Fuzzy Models for Pattern Recognition* (IEEE Press, NY, 1992). He has about one hundred and fifty research papers including ten in edited books and more than eighty in international journals to his credit. He has also lectured at different U.S. and Japan universities and laboratories on his research work.

Dr. Pal is a Reviewer of the Mathematical Reviews (American Mathematical Society), an Associate Editor of the Int. J. Approximate Reasoning, a fellow of the IETE, a life member of the Indian Statistical Institute, and Treasurer of the Indian Society for Fuzzy Mathematics and Information Processing (ISFUMIP). He is also a permanent member of the INDO-US Forum for Cooperative Research and Technology Transfer (IFCRTT) and a organizing/program committee member of various international conferences and meetings.

**Sushmita Mitra** (S'91) completed her B.Sc. (Hons.) in physics at Calcutta University in 1984. Then she obtained B.Tech. and M.Tech degrees in computer science from Calcutta University in 1987 and 1989 respectively. She was a Senior Research Fellow of the Council for Scientific and Industrial Research from 1989 to 1991, working for her Ph.D. degree.

She is a programmer in the Electronics and Communication Sciences Unit of the Indian Statistical Institute, Calcutta. Her research interests include pattern recognition, fuzzy sets, artificial intelligence, and neural networks. She is a student member of the INNS.