ELSEVIER

Neurocomputing 36 (2001) 45–66

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Evolutionary modular design of rough knowledge-based network using fuzzy attributes[☆]

Sushmita Mitra, Pabitra Mitra, Sankar K. Pal*

Machine Intelligence Unit, Indian Statistical Institute, 203, Barrackpore Trunk Road, Calcutta 700035, India

Received 23 March 1999; accepted 29 August 2000

Abstract

This article describes a way of integrating rough set theory with a fuzzy MLP using a modular evolutionary algorithm, for classification and rule generation in soft computing paradigm. The novelty of the method lies in applying rough set theory for extracting dependency rules directly from a real-valued attribute table consisting of fuzzy membership values. This helps in preserving all the class representative points in the dependency rules by adaptively applying a threshold that automatically takes care of the shape of membership functions. An l-class classification problem is split into l two-class problems. Crude subnetwork modules are initially encoded from the dependency rules. These subnetworks are then combined and the final network is evolved using a GA with restricted mutation operator which utilizes the knowledge of the modular structure already generated, for faster convergence. The GA tunes the fuzzification parameters, and network weight and structure simultaneously, by optimising a single fitness function. This methodology helps in imposing a structure on the weights, which results in a network more suitable for rule generation. Performance of the algorithm is compared with related techniques. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Soft computing; Fuzzy MLP; Rough sets; Knowledge-based network; Genetic algorithms; Modular neural network

[☆]This work was partly supported by GTZ Project 93.2254.6-02.196/9602, Germany and CSIR Grant 25(0093)/97/EMR-II, India.

* Corresponding author. Tel.: + 91-33-577-8085x3100; fax: + 91-33-556-6680/6925.

E-mail addresses: sushmita@isical.ac.in (S. Mitra), pabitra_r@isical.ac.in (P. Mitra), sankar@isical.ac.in (S.K. Pal).

1. Introduction

Soft computing is a consortium of methodologies which works synergetically (not competitively) and provides, in one form or another, flexible information processing capability for handling real-life ambiguous situations [27]. Its aim is to exploit the tolerance for imprecision, uncertainty, approximate reasoning and partial truth in order to achieve tractability, robustness, low-cost solutions and close resemblance with human-like decision making. The guiding principle is to devise methods of computation which lead to an acceptable solution at low cost by seeking for an approximate solution to an imprecisely/precisely formulated problem. There are ongoing efforts to integrate artificial neural networks (ANNs), fuzzy set theory, genetic algorithms (GAs), rough set theory and other methodologies in the *soft computing* paradigm [20]. Hybridization [16,17] exploiting the characteristics of these theories include *neuro-fuzzy*, *rough-fuzzy*, *neuro-genetic*, *fuzzy-genetic*, *neuro-rough*, *rough-neuro-fuzzy* approaches. Among these *neuro-fuzzy* computing is most visible.

Generally, ANNs consider a fixed topology of neurons connected by links in a pre-definend manner. These connection weights are usually initialized by small random values. Recently, there have been some attempts in improving the efficiency of neural computation by using knowledge-based nets. This helps in reducing the searching space and time while the network traces the optimal solution. Knowledge-based networks [3,22] constitute a special class of ANNs that consider crude domain knowledge to generate the initial network architecture, which is later refined in the presence of training data. Such a model has the capability of outperforming a standard MLP as well as other related algorithms including symbolic and numerical ones [3,22]. Recently, the theory of rough sets has been used to generate knowledge-based networks.

A recent trend in neural network design for large-scale problems is to split the original task into simpler subtasks, and use a subnetwork module for each of the subtasks [6]. The popular methods available for decomposition include the local model network (LMN) [10] and the CALM model [6]. It has been shown that by combining the output of several subnetworks in an ensemble, one can improve the generalization ability over that of a single large network [5].

Many researchers have combined genetic algorithm with neural network for building more powerful adaptive systems. Here one simultaneously optimizes the weight values and thresholds along with the network topology and learning parameters [9,23–25]. Note that the search space of all possible network topologies is extremely large. Other hybridizations involving GAs include the *genetic-fuzzy* [8,18] and *genetic-neuro-fuzzy* [7,12,28] schemes. Another promising approach is the use of GAs for theory refinement in connectionist systems. Among such successful schemes are the REGENT, TOPGEN [11] and other such algorithms.

In the present article an evolutionary strategy is suggested for designing a connectionist system, integrating fuzzy sets and rough sets. The basic building block is a rough fuzzy MLP. Unlike the existing approaches [1,26], the proposed algorithm can directly extract dependency rules from a real-valued attribute table consisting of fuzzy membership values. The evolutionary training algorithm generates the weight values for a parsimonious network and simultaneously tunes the fuzzification parameters by optimizing a single fitness function. Rough set theory is used to obtain a set of probable knowledge-based subnetworks from the decision rules. These form the initial population of the GA. The modules are then integrated and evolved with a *restricted* mutation operator that helps preserve extracted localized rule structures as potential solutions. This type of 'divide and conquer' strategy accelerates the training significantly as compared to the training of the entire network using a single GA. A restricted mutation operator is implemented, which utilizes the knowledge of modular structure evolved to achieve faster convergence. The aforesaid integration of four different soft computing tools is found to significantly enhance the performance of the system.

2. Fuzzy MLP

The fuzzy MLP model [15] incorporates fuzziness at the input and output levels of the MLP, and is capable of handling exact (numerical) and/or inexact (linguistic) forms of input data. Any input feature value is described in terms of some combination of membership values in the linguistic property sets *low* (L), *medium* (M) and *high* (H). Class membership values (μ) of patterns are represented at the output layer of the fuzzy MLP. During training, the weights are updated by backpropagating errors with respect to these membership values such that the contribution of uncertain vectors is automatically reduced. A schematic diagram depicting the whole procedure is provided in Fig. 1. The various phases of the algorithm are described below.

A three-layered feedforward MLP is used. The output of a neuron in any layer (h) other than the input layer ($h = 0$) is given as

$$y_j^h = \frac{1}{1 + \exp(-\sum_i y_i^{h-1} w_{ji}^{h-1})}, \tag{1}$$

where y_i^{h-1} is the output of the ith neuron in the preceding $(h-1)$th layer w_{ji}^{h-1} is the weight of the connection from the ith neuron in layer $h-1$ to the jth neuron in layer h. For nodes in the input layer, y_j^0 corresponds to the jth component of the input vector. Note that $x_j^h = \sum_i y_i^{h-1} w_{ji}^{h-1}$.

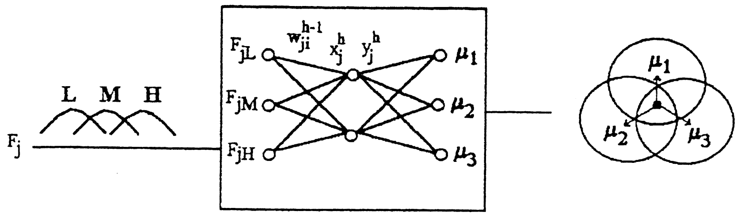


Fig. 1. Block diagram of fuzzy MLP.

An n-dimensional pattern $\boldsymbol{F}_i = [F_{i1}, F_{i2}, \dots, F_{in}]$ is represented as a $3n$-dimensional vector

$$\begin{aligned}\boldsymbol{F}_i &= [\mu_{\text{low}(F_{i1})}(\boldsymbol{F}_i),\ \mu_{\text{medium}(F_{i1})}(\boldsymbol{F}_i),\ \mu_{\text{high}(F_{i1})}(\boldsymbol{F}_i),\ \mu_{\text{low}(F_{i2})}(\boldsymbol{F}_i),\ \dots,\ \mu_{\text{high}(F_{in})}(\boldsymbol{F}_i)]\\ &= [y_1^0, y_2^0, \dots, y_{3n}^0], \end{aligned}\tag{2}$$

where the μ values indicate the membership functions of the corresponding linguistic π sets *low, medium* and *high* along each feature axis and $y_1^0, \dots, y_{3n}^0$ refer to the activations of the $3n$ neurons in the input layer.

When the pattern element is numerical, we use the π-fuzzy sets (in the one-dimensional form), with range [0,1], represented as [15]

$$\pi(F_j; c, \lambda) = \begin{cases} 2\left(1 - \dfrac{\|F_j - c\|}{\lambda}\right)^2 & \text{for} \dfrac{\lambda}{2} \le \|F_j - c\|; \le \lambda, \\ 1 - 2\left(\dfrac{\|F_j - c\|}{\lambda}\right)^2 & \text{for} \dfrac{\lambda}{2} \le \|F_j - c\|; \le \dfrac{\lambda}{2}, \\ 0 & \text{otherwise}, \end{cases}\tag{3}$$

where $\lambda(>0)$ is the radius of the π–function with c as the central point. Note that features in linguistic and set forms can also be handled in this framework [15].

Let m_j be the mean of the pattern points along the jth-axis. Then m_{jl} and m_{jh} are defined as the mean (along the jth-axis) of the pattern points having coordinate values in the range $[F_{j\min}, m_j)$ and $(m_j, F_{j\max}]$, respectively, where $F_{j\max}$, and $F_{j\min}$ denote the upper and lower bounds of the dynamic range of feature F_j (for the training set) considering numerical values only. For the three linguistic property sets along the jth-axis, the centers and the corresponding radii of the corresponding π–functions are defined as

$$\begin{aligned} &c_{\text{low}(F_j)} = m_{jl}, \quad c_{\text{medium}(F_j)} = m_j, \quad c_{\text{high}(F_j)} = m_{jh},\\ &\lambda_{\text{low}(F_j)} = c_{\text{medium}(F_j)} - c_{\text{low}(F_j)},\\ &\lambda_{\text{high}(F_j)} = c_{\text{high}(F_j)} - c_{\text{medium}(F_j)},\\ &\lambda_{\text{medium}(F_j)} = 0.5(c_{\text{high}(F_j)} - c_{\text{low}(F_j)}), \end{aligned}\tag{4}$$

respectively. Here we take into account the distribution of the pattern points along each feature axis while choosing the corresponding centers and radii of the linguistic properties. The nature of the membership function is illustrated in Fig. 2.

Consider an l-class problem domain such that we have l nodes in the output layer. Let the n-dimensional vectors $\boldsymbol{o}_k = [o_{k1}, \dots, o_{kl}]$ and $\boldsymbol{v}_k = [v_{k1}, \dots, v_{kl}]$ denote the mean and standard deviation, respectively, of the numerical training data for the kth class c_k. The weighted distance of the training pattern $\boldsymbol{F}_i$ from kth class c_k is defined as

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

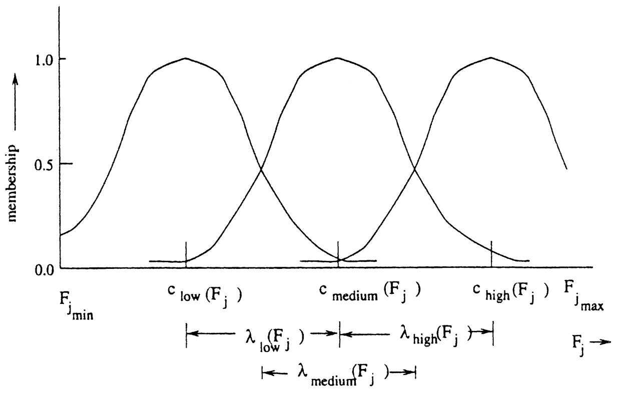


Fig. 2. Overlapping membership functions along feature F_j.

where F_{ij} is the value of the jth component of the ith pattern point. The memership of the ith pattern in class k, lying in the range [0, 1] is defined as [14]

$$\mu_k(\boldsymbol{F}_i) = \frac{1}{1 + (z_{ik}/f_d)^{f_e}}, \tag{5}$$

where positive constants f_d and f_e are the denominational and exponential fuzzy generators controlling the amount of fuzziness in the class membership set.

3. Rough fuzzy MLP

The formulation of a rough fuzzy MLP is described in this section. This is an extension of the model described in [1]. The extracted crude domain knowledge is encoded among the connection weights. This helps one to automatically generate an appropriate network architecture in terms of hidden nodes and links. The method models arbitrary decision regions with multiple object representatives. This knowledge encoding algorithms is radically different from existing models [3,22]. A three-layered fuzzy MLP (Section 2) is considered where the feature space gives the condition attributes and the output classes the decision attributes, so as to result in a decision table. This table may be transformed, keeping the complexity of the network to be constructed in mind. Rules are then generated from the (transformed) table by computing relative reducts. The dependency factors of these rules are used to encode the initial connection weights of the resultant knowledge-based network.

3.1. Rule generation

Let $S = \langle U, A \rangle$ be a decision table, with C and $D = \{d_1, \ldots, d_l\}$ its sets of condition and decision attributes, respectively. Divide the decision table $S = \langle U, A \rangle$ into l tables

$S_i = \langle U_i, A_i \rangle$, $i = 1, \dots, l$, corresponding to the l decision attributes $d_1, \dots, d_l$, where

$$U = U_1 \cup \dots \cup U_l \quad \text{and} \quad A_i = C \cup \{d_i\}.$$

The objective is to generate dependency rules seperately for each of the l classes. Let $\{x_{i1}, \dots, x_{ip}\}$ be the set of those object of U_i that occur in $S_i, i = 1, \dots, l$.

Now for each d_i-reduct $B = \{b_1, \dots, b_k\}$ (say), a discernibility matrix (denoted $\boldsymbol{M}_{d_i}(B)$) from the d_i-discernibility matrix is defined as follow [21]:

$$c_{ij} = \{a \in B: a(x_i) \neq a(x_j)\} \tag{6}$$

for $i,j = 1, \dots, n$.

For each object $x_j \in x_{i_1}, \dots, x_{i_p}$, the discernibility function $f_{d_i}^{x_j}$ is defined as

$$f_{d_i}^{x_j} = \bigwedge \{ \bigvee (c_{ij}) : 1 \leq i, j \leq n, j < i, c_{ij} \neq \emptyset \}, \tag{7}$$

where $\bigvee(c_{ij})$ is the disjunction of all members of c_{ij}. Then $f_{d_i}^{x_j}$ is brought to its conjunctive normal form (c.n.f). One thus obtains a dependency rule r_i, viz., $P_i \leftarrow d_i$, where P_i is the disjunctive normal form (d.n.f) of $f_{d_i}^{x_j}, j \in i_1, \dots, i_p$.

The dependency factor $\mathrm{d}f_i$ for r_i is given by

$$\mathrm{d}f_i = \frac{card(POS_i(d_i))}{card(U_i)}, \tag{8}$$

where $POS_i(d_i) = U_{X \in Id_i} l_i(X)$, and $l_i(X)$ is the lower approximation of X with respect to I_i. In this case $\mathrm{d}f_i = 1$ [1]. This is used while initializing the connection weights of the network as explained in the following section.

3.2. Knowledge encoding

Consider the case of feature F_j for class c_k in the l-class problem domain. The inputs for the ith representative sample $\boldsymbol{F}_i$ are mapped to the corresponding three-dimensional feature space of $\mu_{\text{low}(F_{ij})}(\boldsymbol{F}_i)$, $\mu_{\text{medium}(F_{ij})}(\boldsymbol{F}_i)$ and $\mu_{\text{high}(F_{ij})}(\boldsymbol{F}_i)$, by Eq. (2). Let these be represented by L_j, M_j and H_j, respectively. As the method considers multiple objects in a class a separate $n_k \times 3n$-dimensional attribute-value decision table is generated for each class c_k (where n_k indicates the number of objects in c_k).

The absolute distance between each pair of objects is computed along each attribute L_j, M_j, H_j for all j. We modify Eq. (6) to directly handle a real-valued attribute table consisting of fuzzy membership values. We define

$$c_{ij} = \{a \in B: |a(x_i) - a(x_j)| > Th\} \tag{9}$$

for $i, j = 1, \dots, n_k$, where Th is an adaptive threshold.

Note that the adaptivity of this threshold is in-built, depending on the inherent shape of the membership function. Consider Fig. 3. Let a_1, a_2 correspond to two membership functions (attributes) with a_2 being steeper as compared to a_1. It is

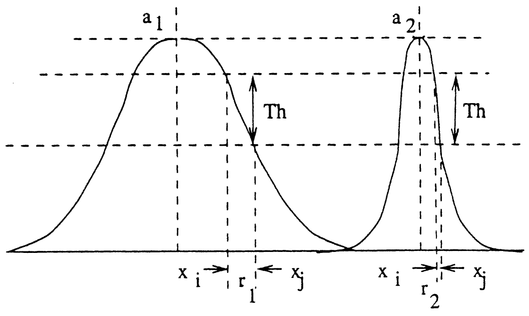


Fig. 3. Illustration of adaptive thresholding of membership functions.

observed that $r_1 > r_2$. This results in an implicit adaptivity of *Th* while computing c_{ij} in the discernibility matrix directly from the real-valued attributes. Here lies the novelty of the proposed method. Moreover, this type of thresholding also enables the discernibility matrix to contain all the representative points/clusters present in a class. This is particularly useful in modeling multi-modal class distributions. A related concept which links fuzzy sets and rough sets is *shadowed sets* by Pedrycz [19].

While designing the initial structure of the rough fuzzy MLP, the union of the rules of the l classes is considered. The input layer consists of $3n$ attribute values while the output layer is represented by l classes. The hidden layer nodes model the first-level (innermost) operator in the antecedent part of a rule, which can be either a conjunct or a disjunct. The output layer nodes model the outer level operands, which can again be either a conjunct or a disjunct. For each inner level operator, corresponding to one output class (one dependency rule), one hidden node is dedicated. Only those input attributes that appear in this conjunct/disjunct are connected to the appropriate hidden node, which in turn is connected to the corresponding output node. Each outer level operator is modeled at the output layer by joining the corresponding hidden nodes. Note that a single attribute (involving no inner level operators) is directly connected to the appropriate output node via a hidden node, to maintain uniformity in rule mapping.

Let the dependency factor for a particular dependency rule for class c_k be $\mathrm{d}f = \alpha = 1$ by Eq. (8). The weight w_{ki}^1 between a hidden node i and output node k is set at $\alpha/fac + \varepsilon$, where *fac* refers to the number of outer level operands in the antecedent of the rule and ε is a small random number taken to destroy any symmetry among the weights. Note that $fac \geq 1$ and each hidden node is connected to only one output node. Let the initial weight so clamped at a hidden node be denoted as β. The weight $w_{ia_j}^0$ between an attribute a_j (where a corresponding to *low* (L), *medium* (M) or *high* (H)) and hidden node i is set to $\beta/facd + \varepsilon$, such that *facd* is the number of attributes connected by the corresponding inner level operator. Again $facd \geq 1$. Thus, for an l-class problem domain there are at least l hidden nodes. This encoding is explained for an example one-class problem in Fig. 4. All other possible connections in the resulting fuzzy MLP are set as small random numbers. It is to be mentioned that the number of hidden nodes is determined from the dependency rules.

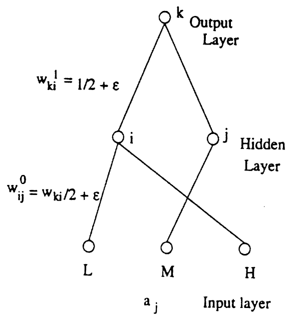

k Output Layer
$w_{ki}^{1} = 1/2 + \varepsilon$
i
j Hidden Layer
$w_{ij}^{0} = w_{ki}/2 + \varepsilon$
L
M
H
a_j
Input layer

Fig. 4. Example of an encoded network.

4. Modular knowledge-based network

It is believed that the use of modular neural network (MNN) enables a wider use of ANNs for large-scale systems. Embedding modularity (i.e. to perform local and encapsulated computation) into neural networks leads to many advantages compared to the use of a single network. For instance, constraining the network connectivity increases its learning capacity and permits its application to large-scale problems [6]. It is easier to encode a priori knowledge in modular neural networks. In addition, the number of network parameters can be reduced by using modularity. This feature speeds computation and can improve the generalization capability of the system [2]. Modular networks are not affected by the interference problem that affect global networks like MLP.

We use two phases. First an l-class classification problem is split into l two-class problems. Let there be l sets of subnetworks, with $3n$ inputs and one output node each. Rough set theoretic concepts are used to encode domain knowledge into each of the subnetworks, using Eqs. (7)–(9). The number of hidden nodes and connectivity of the knowledge-based subnetworks is automatically determined. A two-class problem leads to the generation of one or more crude subnetworks, each encoding a particular decision rule. Let each of these constitute a pool. So we obtain $m \geq l$ pools of knowledge-based modules. Each pool k is perturbed to generate a total of n_k subnetworks, such that $n_1 = \cdots = n_k = \cdots = n_m$. These pools constitute the initial population of subnetworks, which are then evolved independently using genetic algorithms.

At the end of network training using GA, the modules/subnetworks corresponding to each two-class problem are concatenated to form an initial network for the second phase. The inter module links are initialized to small random values as depicted in Fig. 5. A set of such concatenated networks forms the initial population of the GA. Note that the individual modules cooperate, rather than compete, with each other while evolving towards the final solution. The mutation probability for the inter-module links is now set to a high value, while that of intra-module links is set to a relatively lower value. This sort of *restricted* mutation helps preserve some of the localized rule structures, already extracted and evolved, as potential solutions. The initial population for the GA of the entire network is formed from all possible

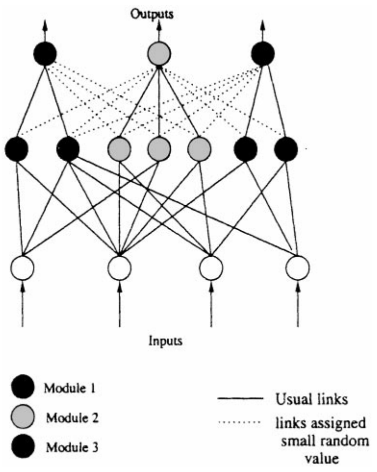


Fig. 5. Intra and Inter module links.

combinations of these individual network modules and random perturbations about them. This ensures that for complex multi-modal pattern distributions all the different representative points remain in the population. The algorithm then searches through the reduced space of possible network topologies. The block-diagram of the entire process is provided in Fig. 6.

Use of the above scheme for generating modular knowledge-based networks has several advantages:

(a) Sufficient reduction in training time is obtained, as the above approach parallelizes the GA to an extent. The search string for the GA for subnetworks being smaller, more than linear decrease in searching time is obtained. Also, very small number of training cycles are required in the refinement phase, as the network is already very close to the solution. Note that the modular aspect of our algorithm is similar to the co-evolutionary algorithm (CEA) used for solving large-scale problems with EAs [29]. However, there exist no guidelines for the decomposition of network modules in [29]. Here arbitrary subnetworks are assigned to each of the classes. Use of networks with the same number of hidden nodes for all classes leads to overlearning in the case of simple classes and poor learning in complex classes.

(b) The use of rough sets for knowledge encoding provides an established mathematical framework for network decomposition. The search space is reduced, leading to shorter training time. The initial network topology is also automatically determined and provides good *building blocks* for the GA.

(c) The algorithm indirectly constrains the solution in such a manner that a structure is imposed on the connection weights. This is helpful for subsequent rule-extraction from the weights, as the resultant network obtained has sparse but strong interconnection among the nodes. Although in the above process some amount of optimality is sacrificed, and often for many-class problems the number of nodes required may be higher than optimal, yet the network is less redundant. However the

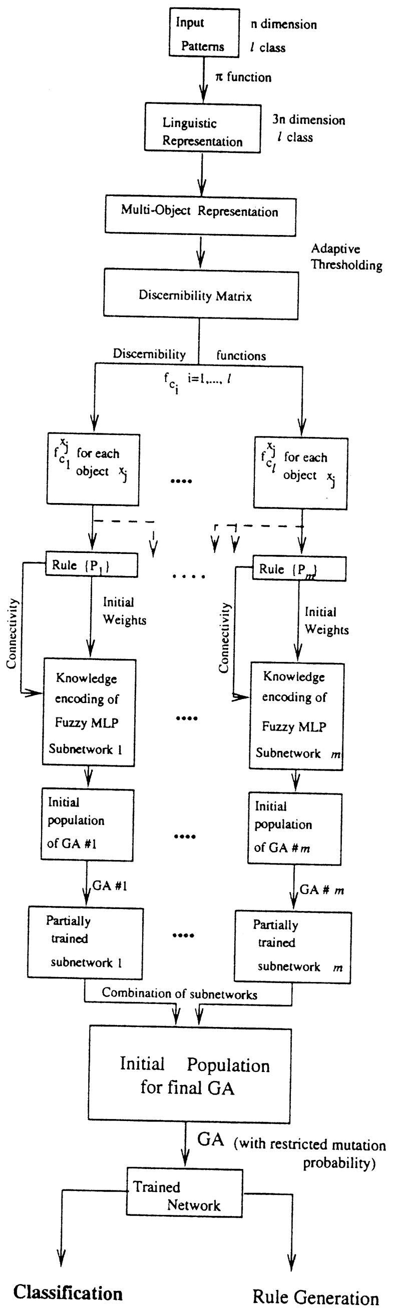

Input Patterns
n dimension
l class
π function
Linguistic Representation
3n dimension
l class
Multi-Object Representation
Adaptive Thresholding
Discernibility Matrix
Discernibility functions
f_{c_i} i=1,..., l
$f_{c_1}^{x_j}$ for each object x_j
$f_{c_l}^{x_j}$ for each object x_j
Rule {P_1}
Rule {P_m}
Connectivity
Initial Weights
Knowledge encoding of Fuzzy MLP Subnetwork 1
Knowledge encoding of Fuzzy MLP Subnetwork m
Initial population of GA #1
Initial population of GA #m
GA #1
GA # m
Partially trained subnetwork 1
Partially trained subnetwork m
Combination of subnetworks
Initial Population for final GA
GA (with restricted mutation probability)
Trained Network
Classification
Rule Generation

Fig. 6. Block diagram.

above choice of objective function and training algorithm enables sufficient amount of link pruning and the total number of links are found to be significantly less.

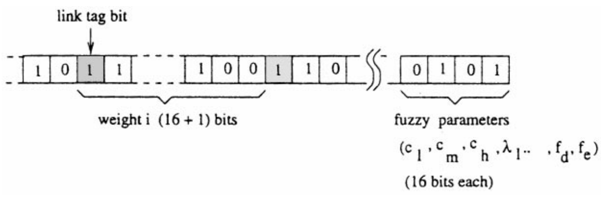
5. Evolutionary design

Genetic algorithms are highly parallel and adaptive search processes based on the principles of natural selection [4]. Here we use GAs for evolving the weight values as well as the structure of the fuzzy MLP used in the framework of modular neural networks. The input and output fuzzification parameters are also tuned. Unlike other theory refinement systems which train only the *best* network approximation obtained from the domain theories, the initial population here consists of all possible networks generated from rough set theoretic rules. This is an advantage because potentially valuable information may be wasted by discarding the contribution of less successful networks at the initial level itself.

Genetic algorithms involve three basic procedures — encoding of the problem parameters in the form of binary strings, application of genetic operators like crossover and mutation, selection of individuals based on some objective function to create a new population. Each of these aspects is discussed below with relevance to our algorithm.

5.1. Chromosomal representation

The problem variables consists of the weight values and the input/output fuzzification parameters. Each of the weights is encoded into a binary word of 16 bit length, where [000 ... 0] decodes to -128 and [111 ... 1] decodes to 128. An additional bit is assigned to each weight to indicate the presence or absence of the link. If this bit is 0 the remaining bits are unrepresented in the phenotype. The total number of bits in the string is therefore dynamic [11]. Thus a total of 17 bits are assigned for each weight. The fuzzificationn parameters tuned are the centers (c) and radius (λ) for each of the linguistic attributes *low, medium* and *high* of each feature (Eq. (3)), and the output fuzzifiers f_d and f_e (Eq. (5)). These are also coded as 16 bit strings in the range [0,2]. For the input parameters, [000 ... 0] decodes to 0 and [111 ... 1] decodes to 1.2 times the maximum value attained by the corresponding feature in the training set. This assumes that the maximum value of a feature in the test set would not exceed 1.2 times that in the training set. The chromosome is obtained by concatenating all the above strings. Sample values of the string length are around 2000 bits for reasonably sized networks.



Initial population is generated by coding the networks obtained by rough set-based knowledge encoding, and by random perturbations about them. A population size of 64 was considered.

5.2. Genetic operators

5.2.1. Crossover

It is obvious that due to the large string length, single point crossover would have little effectiveness. Multiple point crossover is adopted, with the distance between two crossover points being a random variable between 8 and 24 bits. This is done to ensure a high probability for only one crossover point occurring within a word encoding a single weight. The crossover probability is fixed at 0.7.

5.2.2. Mutation

The mutation operator has a profound influence on the search dynamics because of large string size [24]. Each of the bits in the string is chosen to have some mutation probability (*pmut*). This mutation probability however has a spatio-temporal variation. The variation of *pmut* with iterations is shown in Fig. 7. The maximum value of *pmut* is chosen to be 0.4 and the minimum value as 0.01. The mutation probabilities also vary along the encoded string as shown in Fig. 8, with the bits corresponding to inter-module links being assigned a probability *pmut* (i.e., the value of *pmut* at that iteration) and intra-module links assigned a probability *pmut*/10. This is done to ensure least alterations in the structure of the individual modules already evolved. Hence, the mutation operator indirectly incorporates the domain knowledge extracted through rough set theory.

5.3. Choice of fitness function

In GAs the fitness function is the final arbiter for string creation, and the nature of the solution obtained depends on the objective function. An objective function of the form described below is chosen:

$$F = \alpha_1 f_1 + \alpha_2 f_2, \tag{10}$$

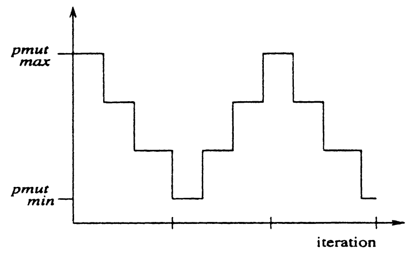


Fig. 7. Variation of mutation probability with iterations.

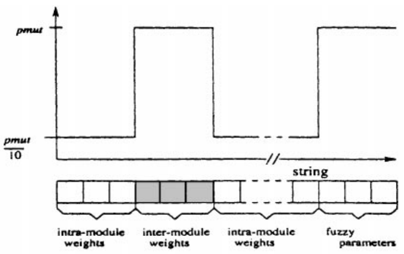


Fig. 8. Variation of mutation probability along the encoded string.

where

$$f_1 = \frac{\textit{No. of correctly classified sample in training set}}{\textit{Total no. of samples in training set}},$$

$$f_2 = 1 - \frac{\textit{No. of links present}}{\textit{Total no. of links possible}}.$$

Here α_1 and α_2 determine the relative importance of each of the factors. α_1 is taken to be 0.9 and α_2 is taken as 0.1, to give more importance to the classification score compared to the network size in terms of number of links. Note that we optimize the network connectivity, weights and input/output fuzzification parameters simultanesouly.

5.4. Selection

Selection is done by the *roulette wheel* method. The probabilities are calculated on the basis of ranking of the individuals in terms of the objective function, instead of the objective function itself. Fitness ranking overcomes two of the biggest problems inherited from traditional fitness scaling: *over compression* and *under expansion*. *Elitism* is incorporated in the selection process to prevent oscillation of the fitness function with generation. The fitness of the best individual of a new generation is compared with that of the current generation. If the latter has a higher value — the corresponding individual replaces a randomly selected individual in the new population.

6. Implementation and results

The genetic-rough-neuro-fuzzy algorithm has been implemented on both real-life (speech, medical) and artificially generated data. The data sets are available at http://www.isical.ac.in/~sushmita/pattern. Let the proposed methodology be termed Model S. Other models compared include:

Model O: An ordinary MLP trained using backpropagation (BP) with weight decay. The rule employed for weight decay was $w_{ij}^{new} = (1 - \varepsilon_{ij})w_{ij}^{old}$. The term

$$\varepsilon_{ij} = \frac{\gamma}{(1 + w_{ij}^2)^2},$$

where γ is the learning rate, insures that those weights which are not changing much gradually decay to zero. The learning rate γ is adaptive and decays in the final phase.

Model F: A fuzzy multilayer perceptron trained using BP [15] (with weight decay).

Model R: A fuzzy multilayer perception trained using BP (with weight decay), with initial knowledge encoding using rough sets (using Eq. (9) for handling fuzzy attributes, as modification to [1]).

Model FM: A modular fuzzy multilayer perceptron trained with GAs along with tuning of the fuzzification parameters. Here the term modular refers to the use of subnetworks corresponding to each class, that are later concatenated using GAs.

The dependency rules generated via rough set theory and used in the encoding scheme are fist provided. Recognition scores obtained for each of the data by the proposed soft modular network (Model S) are then presented and compared with other related MLP-based classification methods using different levels of hybridizations described above. In all cases, 10% of the samples are used as training set and the network is tested on the remaining samples.

6.1. Speech data

The speech data **Vowel** deals with 871 Indian Telegu vowel sounds. These were uttered in a consonant–vowel–consonant context by three male speakers in the age group of 30 to 35 years. The data set has three features: F_1, F_2 and F_3 corresponding to the first, second and third vowel formant frequencies obtained by trained personnel through spectrum analysis of the speech data [13]. Fig. 9 depicts the projection in the F_1- F_2 plane, of the six vowel classes δ, a, i, u, e, o. These overlapping classes will be denoted by $c_1, c_2, \ldots, c_6$.

The rough set theoretic technique [Eqs. (7)–(9)] is applied on the vowel data to extract some knowledge which is initially encoded among the connection weights of the subnetworks. The data is first transformed into a nine-dimensional linguistic space by Eq. (2).

The dependency rules obtained by using the methodology, described in Section 3.1, are provided below. Note that L_i, M_i, H_i correspond to the linguistic labels *low, medium, high* of the *i*th feature [Eq. (2)]:

$$c_1 \leftarrow M_1 \vee L_3,$$

$$c_1 \leftarrow M_1 \vee M_2,$$

$$c_2 \leftarrow M_2 \vee M_3 \vee (H_1 \wedge M_2),$$

$$c_2 \leftarrow M_2 \vee H_3,$$

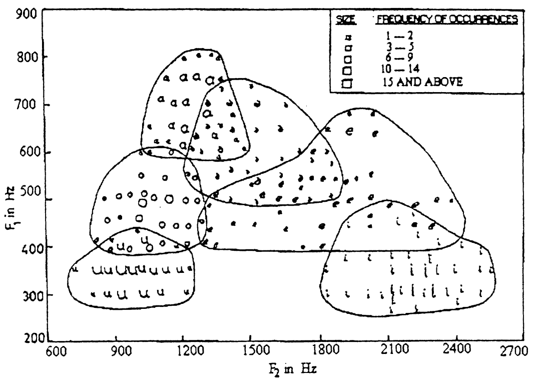


Fig. 9. Projection in F_1–F_2 plane of the **Vowel** data.

$$c_3 \leftarrow (L_1 \wedge H_2) \vee (M_1 \wedge H_2),$$

$$c_3 \leftarrow (L_1 \wedge H_2) \vee (L_1 \wedge M_3),$$

$$c_4 \leftarrow (L_1 \wedge L_2) \vee (L_1 \wedge L_3) \vee (L_2 \wedge M_3) \vee (L_1 \wedge M_3),$$

$$c_5 \leftarrow (H_1 \wedge M_2) \vee (M_1 \wedge M_3) \vee (M_1 \wedge M_2) \vee (M_2 \wedge L_1),$$

$$c_5 \leftarrow (H_1 \wedge M_2) \vee (M_1 \wedge M_2) \vee (H_1 \wedge H_3) \vee (H_2 \wedge L_1),$$

$$c_5 \leftarrow (L_2 \wedge L_1) \vee (H_3 \wedge M_3) \vee M_1,$$

$$c_6 \leftarrow L_1 \vee M_3 \vee L_2,$$

$$c_6 \leftarrow M_1 \vee H_3,$$

$$c_6 \leftarrow L_1 \vee H_3,$$

$$c_6 \leftarrow M_1 \vee M_3 \vee L_2.$$

The above rules are used to get initial subnetwork modules using the scheme outlined in Section 3.2. The integrated network contains a single hidden layer with 18 nodes. In all, 96 such networks are obtained. The initial population is formed by randomly selecting 64 networks from them.

The performance of Model S along with its comparison with other models using the same number of hidden nodes is presented in Table 1. In the first phase of the GA (for models FM and S), each of the subnetworks are partially trained for 10 sweeps each. It is observed that Model S performs the best with the least network size after being trained for only 90 sweeps in the final phase. Comparing Models F and R, we observe that the incorporation of domain knowledge in the latter through rough sets boosts its performance. Similarly, using the modular approach with GA in Model FM improves its efficiency over that of Model F. Note that Model S encompasses both models R and FM. Hence it results in the least redundant yet effective model. This is corroborated from Fig. 10 which demonstrates the rise in correct classification (%)

Table 1
Comparative performance of different models for **Vowel** data

Models	Model O		Model F		Model R		Model FM		Model S	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
c_1(%)	11.2	8.1	15.7	14.2	44.1	42.4	42.4	32.5	62.0	58.4
c_2(%)	75.7	76.4	82.5	88.4	88.8	87.5	95.0	88.8	100	88.8
c_3(%)	80.0	85.5	90.9	92.4	88.4	88.7	90.9	89.5	94.2	92.4
c_4(%)	71.4	65.2	93.2	87.2	88.2	87.4	90.9	90.0	90.2	90.2
c_5(%)	68.5	59.1	80.0	78.5	94.2	93.4	82.2	80.42	85.8	82.4
c_6(%)	76.4	71.1	96.2	93.9	94.4	94.2	100	100	95.1	94.9
Net(%)	65.2	64.2	84.3	81.8	86.8	85.8	85.4	82.4	87.2	85.8
# Links	131		210		152		124		84	
Sweeps	5600		5600		2000		200		90	

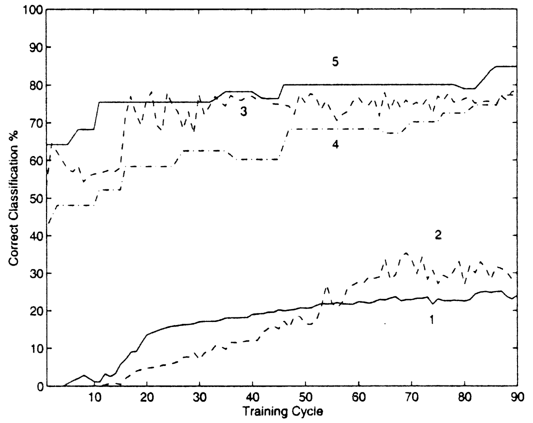


Fig. 10. Correct classification percentage with number of sweeps for **Vowel** data: (1) MLP trained with BP (Model O). (2) Fuzzy MLP trained with BP (Model F). (3) Rough Fuzzy MLP trained with BP (Model R). (4) Fuzzy MLP trained with GA (Model FM). (5) Modular rough fuzzy MLP trained with GA (Model S).

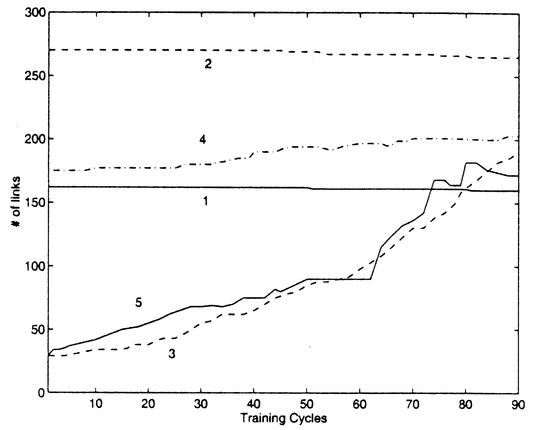


Fig. 11. Evolution of the number of links with iterations for different training schemes on **Vowel** data. (The curve numbering is the same as that in Fig. 10.)

with number of training sweeps. It is observed that curve 5 (Model S) performs the best. Here all curves are plotted up to 90 sweep, for each of comparison.

Fig. 11 depicts the evolution of the number of links with training. As models O and F (curves 1 and 2) involve backpropagation, there is no significant change. Model R (curve 3) incorporates domain knowledge obtained from rough set rules. Hence, the initial network is small but more number of links grow with training. In Model FM (curve 4) the initial network is also sparse as the inter-module links are initialized to zero, but links grow as the network is refined. In the case of Model S initial knowledge encoding, modular training and link pruning all are present. Hence a network with

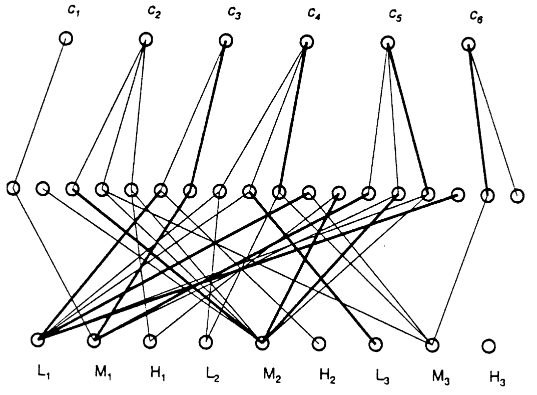


Fig. 12. Connectivity of the network obtained for the **Vowel** data, using Model S.

least number of nodes is obtained. It may be noted that the training algorithm suggested is successful in imposing a structure among the connection weights.

Consider a simple heuristic for rule extraction. Let us define the following quantities: $Thres_1$ = *mean of the weights* > 0, $Thres_2$ = *mean of the weights* > $Thres_1$, $Thres_3$ = *mean of the weights* > $Thres_2$. We consider weights having value greater than $Thres_3$ as strong connections (plotted as thick lines in Fig. 12), weights having value between $Thres_2$ and $Thres_3$ as moderate links (plotted as normal lines in Fig. 12), and weights having value between $Thres_1$ and $Thres_2$ as weak links (plotted as faint lines in Fig. 12). We obtained $Thres_1 = 20.76$, $Thres_2 = 74.14$ and $Thres_3 = 92.65$. If the same set of threshold values are applied to Model F, only a few strong links are obtained. Hence, it is not possible to extract any path from output to input nodes, leading to an absence of *certain* rules. On the other hand, the network obtained using the proposed Model S contains a number of strong links which can be used in extracting meaningful rules.

A sample set of rules extracted from the network, considering only the strong and moderate links, is presented below:

$$c_1 \leftarrow M_1,$$

$$c_2 \leftarrow (M_2 \wedge M_3) \vee M_2 \vee (H_1 \wedge M_2),$$

$$c_3 \leftarrow (L_1 \wedge H_2) \vee M_1,$$

$$c_4 \leftarrow (L_1 \wedge L_2) \vee (L_1 \wedge L_3) \vee (L_2 \wedge M_3),$$

$$c_5 \leftarrow (M_1 \wedge M_2) \vee M_1 \vee (L_1 \wedge M_2),$$

$$c_6 \leftarrow M_3.$$

6.2. Synthetic data

The Synthetic data **Pat** consists of 880 pattern points in the two-dimensional space F_1–F_2 as depicted in Fig. 13. There are three linearly non-separable pattern classes.

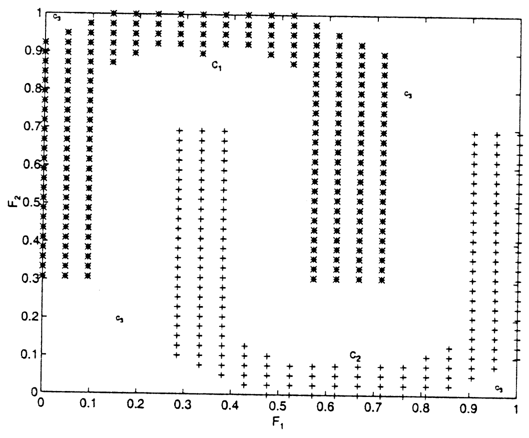


Fig. 13. Artificially generated lineraly nonseperable pattern **Pat**.

The figure is marked with classes 1(c_1) and 2 (c_2), while class 3 (c_3) corresponds to the background region.

The features are mapped to six-dimensional linguistic space. The following dependency rules are generated:

$$c_1 \leftarrow (L_1 \wedge H_2) \vee (M_1 \wedge H_2) \vee (M_1 \wedge M_2),$$

$$c_2 \leftarrow (H_1 \wedge M_2) \vee (H_1 \wedge L_2),$$

$$c_2 \leftarrow (M_1 \wedge M_2) \vee (H_1 \wedge M_2) \vee (M_1 \wedge L_2) \vee (H_1 \wedge L_2),$$

$$c_2 \leftarrow (M_1 \wedge L_2) \vee (H_1 \wedge L_2) \vee L_1,$$

$$\begin{aligned} c_3 \leftarrow & (L_1 \wedge H_2) \vee (H_1 \wedge L_2) \vee (M_1 \wedge L_2) \vee (H_1 \wedge M_2) \\ & \vee (H_1 \wedge H_2) \vee (M_1 \wedge H_1) \vee (L_2 \wedge L_1) \vee (L_1 \wedge M_2) \\ & \vee (M_1 \wedge H_2) \vee (L_1 \wedge M_2). \end{aligned}$$

Note that use of Eq. (9) resulted in generation of terms like $(L_1 \wedge L_2)$, $(H_1 \wedge H_2)$, $(H_1 \wedge L_2)$, $(L_1 \wedge H_2)$ to account for clusters (marked in Fig. 13) corresponding to class c_3. This is not possible when one uses Eq. (8) as in [3]. The resultant subnetworks when connected, produces a network with a single hidden layer having 15 nodes. Three such networks are obtained, considering all possible combinations. The remaining 61 networks are obtained by random perturbations about them. The performance is presented in Table 2. Here also Model S outperforms the other models significantly, considering recognition scores, number of links and learning time. The network corresponding to Model S is shown in Fig. 14.

A sample set of rules extracted from the network is presented below. Here we obtained $Thres_1 = 33.98$, $Thres_2 = 83.41$ and $Thres_3 = 95.06$. Since no strong links

Table 2
Comparative classification scores of different models for **Pat** data

Models	Model O		Model F		Model R		Model FM		Model S	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
c_1(%)	71.0	71.0	73.4	72.2	74.5	73.0	73.4	73.0	78.2	77.4
c_2(%)	68.0	68.0	71.8	70.2	76.7	75.2	74.5	74.2	78.2	77.2
c_3(%)	24.2	22.4	60.4	58.4	68.2	67.1	64.4	62.8	70.2	69.8
Net (%)	55.2	54.8	68.8	68.0	73.2	71.1	70.0	69.8	75.5	74.8
# Links	62		105		82		84		72	
Sweeps	2000		2000		1500		150		90	

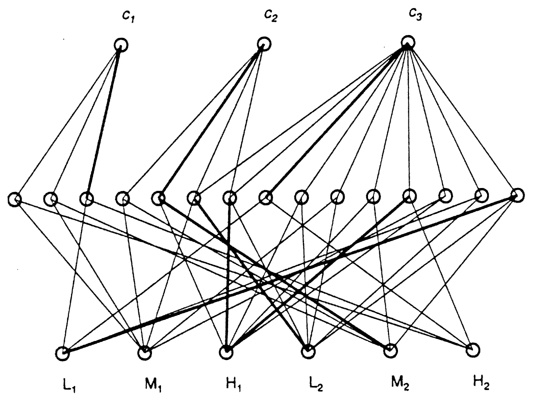


Fig. 14. Connectivity of the network obtained for the **Pat** data, using Model S.

could be obtained from Model F, therefore no meaningful logical rules are extracted:

$$c_1 \leftarrow (M_1 \wedge H_2) \vee (M_1 \wedge M_2) \vee (L_1 \wedge H_2),$$

$$c_2 \leftarrow (M_1 \wedge M_2) \vee (H_1 \wedge M_2) \vee (L_1 \wedge M_2) \vee (H_1 \wedge L_2),$$

$$c_3 \leftarrow (M_1 \wedge L_2) \vee (H_1 \wedge L_2) \vee (L_1 \wedge H_2) \vee (H_1 \wedge M_2)$$
$$\vee (H_1 \wedge H_2) \vee (M_1 \wedge H_1) \vee (L_1 \wedge L_2) \vee (L_1 \wedge L_2 \wedge M_2).$$

7. Conclusions and discussion

A methodology for integrating four soft computing tools, *viz.*, artificial neural networks, fuzzy sets, genetic algorithms and rough sets for designing a knowledge-based network for pattern classification and rule generation is presented. The proposed algorithm involves synthesis of several fuzzy MLP modules, each encoding the rough set rules for a particular class. These knowledge-based modules are refined using a GA. The genetic operators are implemented in such a way that they help preserve the modular structure already evolved. It is seen that this methodology results in superior performance in terms of classification score, training time, and network sparseness (thereby enabling easier extraction of rules) as compared to earlier hybridizations. There is scope for future investigation regarding the exact nature of influence of the mutation probability on the search dynamics. GAs with an adaptive objective function can also be studied.

Unlike other approaches [1, 26], this algorithm directly extracts dependency rules from a fuzzy attribute table consisting of membership values. An adaptive threshold depending on the shape of the membership function, is used in the process. This helps in preserving all representative points/clusters present in a multi-modal class distribution.

The role of fuzzy sets here is to handle uncertainty with input-output membership functions, while rough sets exploit the granularity in information to obtain dependency rules used for knowledge encoding in ANN. The initial rough set rules represent crude domain knowledge. The final rules represent refined knowledge extracted from the trained network and are more meaningful and compact.

References

[1] M. Banerjee, S. Mitra, S.K. Pal, Rough fuzzy MLP: knowledge encoding and classification, IEEE Trans. Neural Network 9 (6) (1998) 1203–1216.

[2] E.B. Baum, D. Haussler, What size nets give valid generalization? Neural Comput. 1 (1989) 151-160.

[3] L.M. Fu, Knowledge-based connectionism for revising domain theories, IEEE Trans. Systems Man Cybernet. 23 (1993) 173–182.

[4] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addision-Wesley, Reading, MA, 1989.

[5] L. Hansen, P. Salamon, Neural network ensembles, IEEE Trans. Pattern Anal. Mach. Intelligence 12 (1990) 993–1001.

[6] B.M. Happel, J.J. Murre, Design and evolution of modular neural network architectures, Neural Networks 7 (1994) 985–1004.

[7] H. Ishigami, T. Fukuda, T. Shibata, F. Arai, Structure optimisation of fuzzy neural network by genetic algorithm, Fuzzy Sets and Systems 71 (1995) 257–264.

[8] H. Ishibuchi, K. Nozaki, N. Yamamoto, H. Tanaka, Selecting fuzzy If-Then rules for classification problems using genetic algorithms, IEEE Trans. Fuzzy System 3 (1995) 260–270.

[9] V. Maniezzo, Genetic evolution of the topology and weight distribution of neural networks, IEEE Trans. Neural Networks 5 (1994) 39–53.

[10] R. Murray-Smith, A Local Model Network approach to non-linear modelling, Ph.D. Thesis, Department of Computer Science, University of Strathclyde, Glasgow, UK, 1994.

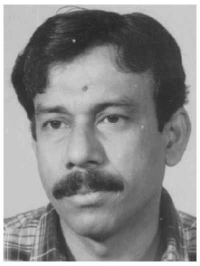
[11] D.W. Opitz, J.W. Shavlik, Connectionist theory refinement: genetically searching the space of network topologies, J. Artifical Intelligence Res. 6 (1997) 177–209.

[12] S.K. Pal, D. Bhandari, Genetic algorithms with fuzzy fitness function for object extraction using cellular neural networks, Fuzzy Sets and Systems 65 (1994) 129–139.

[13] S.K. Pal, D. Dutta Majumder, Fuzzy sets and decision making approaches in vowel and speaker recognition, IEEE Trans. Systems Man Cybernet. 7 (1977) 625–629.

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

[15] S.K. Pal, S. Mitra, Multi-layer perceptron, fuzzy sets and classification, IEEE Trans. Neural Networks 3 (1992) 683–697.
[16] S.K. Pal, S. Mitra, Neuro-Fuzzy Pattern Recognition: Methods in soft Computing, Wiley, New York, 1999.
[17] S.K. Pal, A. Skowron (Eds.), Rough Fuzzy Hybridization: New Trends in Decision Making, Springer, Singapore, 1999.
[18] W. Pedrycz (Ed.), Fuzzy Evolutionary Computation, Kluwer Academic, Boston, 1997.
[19] W. Pedrycz, Shadowed sets: representing and processing fuzzy sets, IEEE Trans. Systems Man Cybernet. 28 (1998) 103–109.
[20] T.Y. Lin (Ed.), Proceedings of Third Workshop on Rough Sets and Soft Computing (RSSC'94), San José, USA, November 1994.
[21] A. Skowron, C. Rauszer, The discernibility matrices and functions in information systems, in: R. Slowiński (Ed.), Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory, Kluwer Academic, Dordrecht, 1993, pp. 331-362.
[22] G.G. Towell, J.W. Shavlik, Knowledge-based artifical neural networks, Artificial Intelligence 70 (1994) 119–165.
[23] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, Parallel Comput. 14 (1990) 347–361.
[24] X. Yao, A review of evolutionary artifical neural networks, Internat. J. Intelligent Sytems 8 (1993) 539–567.
[25] X. Yao, Y. Liu, A new evolutionary system for evolving artifical neural networks, IEEE Trans. Neural Networks 8 (3) (1997) 694–713.
[26] R. Yasdi, Combining rough sets learning and neural learning method to deal with uncertain and imprecise information, Neurocomputing 7 (1995) 61–84.
[27] L.A. Zadeh, Fuzzy logic, neural networks, and soft computing, Comm. ACM 37 (1994) 77–84.
[28] Y.-Q. Zhang, A. Kandel, Compensatory Genetic Fuzzy Neural Networks and their Applications, World Scientific, Singapore, 1998.
[29] Q. Zhao, A co-evolutionary algorithm for neural network learning, Proceedings of the IEEE International Conference on Neural Networks, Houston, 1997, pp. 432–437.



Sankar K. Pal is a *Distinguished Scientist*, and *Founding Head* of Machine Intelligence Unit, at the Indian Statistical Institute, Calcutta. He received the M.Tech and Ph.D. degrees in Radio physics and Electronics in 1974 and 1979, respectively, from the University of Calcutta. In 1982 he received another Ph.D. in Electrical Engineering along with DIC from Imperial College, University of London. He worked at the University of California, Berkeley and the University of Maryland, College Park during 1986–1987 as a *Fulbright Post-doctoral Visiting Fellow*; at the NASA Johnson Space Center, Houston, Texas during 1990–1992 and 1994 as a *Guest Investigator* under the *NRC-NASA Senior Research Associateship program*; and at the Hong Kong Polytechnic University, Hong Kong in 1999 as a *Visiting Professor*. He served as a *Distinguished Visitor of IEEE Computer Society* (*USA*) for the *Asia-Pacific Region* during 1997–1999.

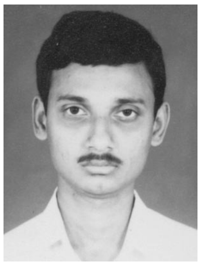
Prof. Pal is a *Fellow* of the IEEE, USA, Third World Academy of Sciences, Italy, and all the four National Academies for Science/Engineering in India. His research interests includes Pattern Recognition, Image Processing, Soft Computing, Neural Nets, Genetic Algorithms, and Fuzzy Systems. He is a co-author of six books including *Fuzzy Mathematical Approach* to *Pattern Recognition*, John Willey (Halsted), N Y, 1986, and *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing*, John Wiley, NY 1999.

He has received the 1990 S. S. Bhatnagar Prize (which is the most coveted award for a scientist in India), 1993 Jawaharlal Nehru Fellowship, 1993 Vikram Sarabhai Research Award, 1993 NASA Tech Brief Award, 1994 IEEE Trans. Neural Networks Outstanding Paper Award, 1995 NASA Patent Application Award, 1997 IETE-Ram Lal Wadhwa Gold Medal, 1998 Om Bhasin Foundation Award, and the 1999 G.D. Birla Award for Scientific Research.

Prof. Pal is an *Associate Editor,* IEEE Trans. Neural Networks (1994–98), Pattern Recognition Letters, Neurocomputing, Applied Intelligence, Information Sciences, Fuzzy Sets and Systems, and Fundamenta Informaticae; a *Member, Executive Advisory Editorial Board*, IEEE Trans. Fuzzy Systems and Int. Journal of Approximate Reasoning, and a *Guest Editor* of many journals including the IEEE Computer.

Sushmita Mitra obtained her B.Sc. (Hons.) in Physics and B. Tech and M. Tech. in Computer Science from the University of Calcutta in 1984, 1987 and 1989 respectively, and Ph.D. in Computer Science from Indian Statistical Institute, Calcutta in 1995. During 1992 to 1994 she was with the European Laboratory for Intelligent Techniques Engineering, Aachen, as a *German Academic Exchange Service (DAAD)* Fellow. Since 1995, she is an Associate Professor of the Indian Statistical Institute, Calcutta, where she joined in 1989.

She was a recipient of the *National Talent Search Scholarship* (1978–1983) from the National Council for Educational Research and Training, India, the *IEEE TNN Outstanding Paper Award* in 1994 and CIMPA-INRIA-UNESCO Fellowship in 1996. She has co-authored a book *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing Paradigm* published by John Wiley, NY 1999. She was a *Visiting Professor* at Meiji University, Japan in 1999. Her research interests include pattern recognition, fuzzy sets, artifical intelligence, neural networks and soft computing.



Pabitra Mitra obtained his B. Tech in Electrical Engg. in 1996, from Indian Institute of Technology, Kharagpur. He worked as a Scientist at Centre for Artifical Intelligence and Robotics, Bangalore and currently is a Junior Research Fellow at the Machine Intelligence Unit, Indian Statistical Institute. His research interests are in the fields of data mining, knowledge discovery, medical expert systems, learning.