Fundamenta Informaticae 37 (1999) 177–199
IOS Press

Relation Between *VGA-classifier* and MLP : Determination of Network Architecture

Sanghamitra Bandyopadhyay, Sankar K. Pal*
Machine Intelligence Unit
Indian Statistical Institute
203 B.T. Road
Calcutta 700 035, India
Email: res9407@isical.ac.in, sankar@isical.ac.in

Abstract. An analogy between a genetic algorithm based pattern classification scheme (where hyperplanes are used to approximate the class boundaries through searching) and multilayer perceptron (MLP) based classifier is established. Based on this, a method for determining the MLP architecture automatically is described. It is shown that the architecture would need atmost two hidden layers, the neurons of which are responsible for generating hyperplanes and regions. The neurons in the second hidden and output layers perform the AND & OR functions respectively. The methodology also includes a post processing step which automatically removes any redundant neuron in the hidden/output layer. An extensive comparative study of the performance of the MLP, thus derived using the proposed method, with those of several other conventional MLPs is presented for different data sets.

Keywords: hyperplane fitting, boundary approximation, hard limiting neuron, network architecture design, variable string length genetic algorithm.

1. Introduction

Genetic Algorithms (GAs) [1, 2] are randomized search and optimization techniques guided by the principles of evolution and natural genetics. They are efficient, adaptive and robust search processes, producing near optimal solutions and have a large amount of implicit parallelism. GAs

*Address for correspondence: Machine Intelligence Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India

deal with individuals, called *chromosomes*, (usually binary strings) which encode the parameters of the problem space and represent a potential solution. An *objective function* of a string provides a mapping from the chromosomal space to the solution space. A *fitness function* is also associated with a string which indicates the degree of 'goodness' of the solution represented by the string. A set of chromosomes constitute a *population* which is initially created randomly. Biologically inspired operators like *selection, crossover* and *mutation* are applied on the population over a number of generations till a termination criterion is achieved. The best string obtained at this point (or obtained so far) represents the solution of the problem.

In pattern recognition, there are many tasks involved in the process of analyzing/identifying a pattern which need appropriate parameter selection and efficient search in complex spaces in order to obtain optimum solutions. Therefore, the application of GAs for solving certain problems of pattern recognition (which need optimization of computation requirements, and robust, fast and close approximate solution) appears to be appropriate and natural [3, 4]. Such an attempt for pattern classification in $\mathbb{R}^N$ has been made in [5] to develop a *GA-classifier*, where the class boundaries are approximated by a number of hyperplanes. The characteristics of GAs are exploited for search and placement of a fixed number, H, of hyperplanes in the feature space, such that the number of misclassified points is minimized.

Since an *a priori* knowledge of H is difficult to obtain, a conservative (or overestimated) value is usually assumed for its operation. This first of all leads to the problem of an overdependence of the algorithm on the training data, especially for small sample size. In other words, since a large number of hyperplanes can readily and closely fit the classes, this may provide good performance during training but poor generalization capability. Secondly, a large value of H unnecessarily increases the computational effort, and may lead to the presence of redundant hyperplanes in the final decision boundary. (A hyperplane is termed redundant if its removal has no effect on the classification capability of the *GA-classifier*.)

Subsequently, a method has been described in [6] to automatically evolve the value of H as a parameter of the problem. For this purpose, the concept of variable length strings in GAs (VGA) has been adopted. Unlike the conventional GAs, here the length of a string is not fixed. Crossover and mutation operators are accordingly defined. A factor has been incorporated into the fitness function that rewards a string with smaller number of misclassified samples as well as smaller number of hyperplanes. It has been theoretically shown in [6] that for infinitely large number of iterations, the number of misclassified training data points for the said classifier (called *VGA-classifier*) will be minimum. At the same time, the number of hyperplanes required for modeling the class boundaries in order to provide the minimum number of misclassified points will also be minimum.

It is known that the Multilayer Perceptron (MLP) [7, 8, 9, 10], with the neurons executing hard limiting non linearities, can also approximate class boundaries using piecewise linear segments. Thus, a clear analogy exists between the two methodologies, viz. classifiers based on MLP and *VGA*. If the parameters of the hyperplanes provided by the *VGA-classifier* are encoded in the connection weights and threshold values of MLP, then the performance provided by

VGA-classifier and MLP will be the same. The architecture along with the connection weights of the MLP can thus be determined from the output results of the *VGA-classifier*.

Based on this realization, we describe, in this article, a methodology where the architecture along with the weights of MLP, with each neuron executing the hard limiting function, is determined automatically using the principle of pattern classification with *VGA* [6]. It is guaranteed that the number of hidden layers (excluding the input and output layers) in the resulting MLP will be atmost two. The neurons of the first hidden layer are responsible for generation of the equations of hyperplanes. The neurons in the second hidden layers are responsible for generating the regions by performing the AND function, whereas those in the output layer are responsible for producing a combination of different regions by performing the OR function. The algorithm also includes a post processing step which removes the redundant neurons, if there are any, in the hidden/output layers. The performance of the MLP derived from this methodology is compared with those of its conventional version and some more using different architectures, for two types of artificial data, Iris data, a speech data and a cancer data.

In this context may be mentioned that there are several approaches for determining the MLP architecture and connection weights [11, 12, 13]. In [11], the connection weights for a given MLP architecture are determined using GAs, where the weights are encoded in the chromosomes. The weighted error is taken as the fitness of a string. This method, therefore, totally eliminates the necessity of using back propagation (BP) algorithm for training. In [12], parallel genetic algorithm is used for evolving the topology and weights of feedforward artificial neural networks. Here both the connectivity and the weights are encoded in the chromosomes. Additionally, the granularity i.e., the number of bits used for encoding the weights is also encoded as a parameter of the problem. This method, thus, utilizes variable string lengths for topology and weight determination. Another method based on the construction of Voronoi diagrams over the set of training patterns is described in [13], where the number of layers, number of neurons in each layer and the connection weights are automatically determined. Pruning a network of a large size is another approach towards determination of proper network architecture. A detailed survey can be found in [14].

2. Brief Description of the *VGA-classifier*

The *VGA-classifier* is based on the principle of modeling the class boundaries of a given training data set using variable number of hyperplanes. Genetic algorithm is used for search and placement of an appropriate number of hyperplanes for modeling the class boundaries such that the number of points misclassified by the generated boundary is minimum. The classifier is described here in brief.

Hyperplane Representation

From elementary geometry, the equation of a hyperplane in N dimensional space ($X_1 - X_2 - \cdots - X_N$) is given by

$$x_N \cos\alpha_{N-1} + \beta_{N-1} \sin\alpha_{N-1} = d \tag{1}$$

where $\beta_{N-1} = x_{N-1}\cos\alpha_{N-2} + \beta_{N-2}\sin\alpha_{N-2}$

$\beta_{N-2} = x_{N-2}\cos\alpha_{N-3} + \beta_{N-3}\sin\alpha_{N-3}$

$\vdots$

$\beta_1 = x_1\cos\alpha_0 + \beta_0\sin\alpha_0.$

The various parameters are as follows :

X_i : the ith feature of the training points.

$(x_1, x_2, \ldots, x_N)$: a point on the hyperplane

α_{N-1} : the angle that the unit normal to the hyperplane makes with the X_N axis.

α_{N-2} : the angle that the projection of the normal in the $(X_1 - X_2 - \cdots - X_{N-1})$ space makes with the X_{N-1} axis.

$\vdots$

α_1 : the angle that the projection of the normal in the $(X_1 - X_2)$ plane makes with the X_2 axis.

α_0 : the angle that the projection of the normal in the (X_1) plane makes with the X_1 axis $= 0$. Hence, $\beta_0 \sin\alpha_0 = 0$.

d : the perpendicular distance of the hyperplane from the origin.

Thus the N tuple $< \alpha_1, \alpha_2, \ldots, \alpha_{N-1}, d >$ specifies a hyperplane in N dimensional space.

Each angle α_j, $j = 1, 2, \ldots, N-1$ is allowed to vary in the range of 0 to 2π. If b_1 bits are used to represent an angle, then the possible values of α_j are

$$0, \delta * 2\pi, 2\delta * 2\pi, 3\delta * 2\pi, \ldots, (2^{b_1} - 1)\delta * 2\pi$$

where $\delta = \frac{1}{2^{b_1}}$. Consequently, if the b_1 bits contain a binary string having the decimal value v_1, then the angle is given by $v_1 * \delta * 2\pi$.

Once the angles are fixed, the orientation of the hyperplane becomes fixed. Now only d must be specified in order to specify the hyperplane. For this purpose the hyper rectangle enclosing the training points is considered. Let x_i^{min} and x_i^{max} be the minimum and maximum values of feature X_i as obtained from the training points. Then the vertices of the enclosing hyper rectangle are given by

$$(x_1^{ch_1}, x_2^{ch_2}, \ldots, x_N^{ch_N})$$

where each ch_i, $i = 1, 2, \ldots, N$ can be either max or min. (Note that there will be 2^N vertices.) Let $diag$ be the length of the diagonal of this hyper rectangle given by

$$diag = \sqrt{(x_1^{max} - x_1^{min})^2 + (x_2^{max} - x_2^{min})^2 + \ldots + (x_N^{max} - x_N^{min})^2}.$$

A hyperplane is designated as the *base hyperplane* with respect to a given orientation (i.e., for some $\alpha_1, \alpha_2, \ldots, \alpha_{N-1}$) if

i : it has the same orientation

ii : it passes through one of the vertices of the enclosing rectangle

iii : its perpendicular distance from the origin is minimum (among the hyperplanes passing through the other vertices). Let this distance be d_{min}.

If b_2 bits are used to represent d, then a value of v_2 in these bits represents a hyperplane with the given orientation and for which d is given by $d_{min} + \frac{diag}{2^{b_2}} * v_2$.

Population Initialization

The chromosomes are represented by strings of 1, 0 and # (don't care), encoding the parameters of variable number of hyperplanes. As mentioned before, in $\mathbb{R}^N$, N parameters are required for representing one hyperplane. These are $N-1$ angle variables, $angle_1^i, \ldots, angle_{N-1}^i$, indicating the orientation of hyperplane i ($i = 1, 2, \ldots, H$ when H hyperplanes are encoded in the chromosome), and one perpendicular distance variable, d^i indicating its perpendicular distance from the origin. Let H_{max} represent the maximum number of hyperplanes that may be required to model the decision boundary of a given data set. It is specified *a priori*.

Initial population is created in such a way that the first and the second strings encode the parameters of H_{max} and 1 hyperplanes respectively to ensure sufficient diversity in the population. For the remaining strings, the number of hyperplanes is generated randomly in the range [1, H_{max}], and the corresponding bits are initialized randomly to 1s and 0s.

Fitness Computation

For each string i encoding H_i hyperplanes, the number of misclassified points $miss_i$, is found as in [6]. If n is the size of the training data, then the fitness fit_i is defined as

$$fit_i = (n - miss_i) - \alpha H_i, \qquad 1 \leq H_i \leq H_{max} \tag{2}$$

$$= 0 \qquad otherwise, \tag{3}$$

where $\alpha = \frac{1}{H_{max}}$. This definition of the fitness function ensures maximization of it primarily minimizes the number of misclassified points and then the number of hyperplanes.

Genetic Operators

Among the operations of selection, crossover and mutation, the selection operator used here may be one of those used in conventional GA [1, 15], while crossover and mutation need to be newly defined for VGA.

Crossover : Two strings, i and j, having lengths l_i and l_j respectively are selected from the mating pool. Let $l_i \leq l_j$. Then string i is padded with #s so as to make the two lengths equal. Conventional crossover like single point crossover, two point crossover [1] is now performed over these two strings with probability μ_c. The following two cases may now arise :

- All the hyperplanes in the offspring are complete. (A hyperplane in a string is called *complete* if all the bits corresponding to it are either defined (i.e., 0s and 1s) or #s. Otherwise it is incomplete.)
- Some hyperplanes are incomplete.

In the second case let u = number of defined bits (either 0 or 1) and t = total number of bits per hyperplane. Then, for each incomplete hyperplane, all the #s are set to defined bits (either 0 or 1 randomly) with probability $\frac{u}{t}$. In case this is not permitted, all the defined bits are set to #. Thus each hyperplane in the string becomes complete. Subsequently, the string is rearranged so that all the #s are pushed to the end, or in other words all the hyperplanes are transposed to

the beginning of the strings. The information about the number of hyperplanes in the strings is updated accordingly.

Mutation : In order to introduce greater flexibility in the method, the mutation operator is defined in such a way that it can both increase and decrease the string length. For this, the strings are padded with #s such that the resultant length becomes equal to l_{max}. Now for each defined bit position, it is determined whether conventional mutation [1] can be applied or not with probability μ_m. Otherwise, the position is set to # with probability μ_{m_1}. Each undefined position is set to a defined bit (randomly chosen) according to another mutation probability μ_{m_2}. These are described in Fig. 1.

Note that mutation may result in some incomplete hyperplanes, and these are handled in a manner, as done for crossover operation. For example, the operation on the defined bits, i.e., when $k \leq l_i$ in Fig. 1, may result in a decrease in the string length, while the operation on #s, i.e., when $k > l_i$ in the figure, may result in an increase in the string length. Also, mutation may yield strings having all #s indicating that no hyperplanes are encoded in it. Consequently, this string will have fitness = 0 and will be automatically eliminated during selection. ♣

```
Begin
    l_i = length of string i
    Pad string i with # so that its length becomes l_max
    for k = 1 to l_max do
        Generate rnd, rnd1 and rnd2 randomly in [0,1]
        if k ≤ l_i do /* for defined bits */
            if rnd < μ_m do /* conventional mutation */
                Flip bit k of string i
            else /* try changing to # */
                if rnd1 < μ_m1 do
                    Set bit k of string i to #
                endif
            endif
        else /* k > l_i i.e., for #s */
            if rnd2 < μ_m2 do /* set to defined */
                Position k of string i is set to 0 or 1 randomly
            endif
        endif
    endfor
End
```

Figure 1. Mutation operation for string i

The operations of selection, crossover and mutation are performed over a number of generations till a user specified termination condition is attained. In the elitist model of GAs the best string of the current generations is preserved during each generation. The best string seen upto

the last generation, alongwith its associated labelling of regions provides the solution to the problem. During testing, for each point with unknown classification, the task of the classifier is to check the region in which it lies, and to put the label (or classify) accordingly.

3. Multilayer Perceptron

A Multilayer Perceptron (MLP) consists of several layers of simple neurons with full connectivity existing between neurons of adjacent layers. Fig. 2 shows an example of a four layer MLP which consists of an input layer (*layer 0*), two hidden layers (*layers 1* and *2*) and an output layer (*layer 3*).

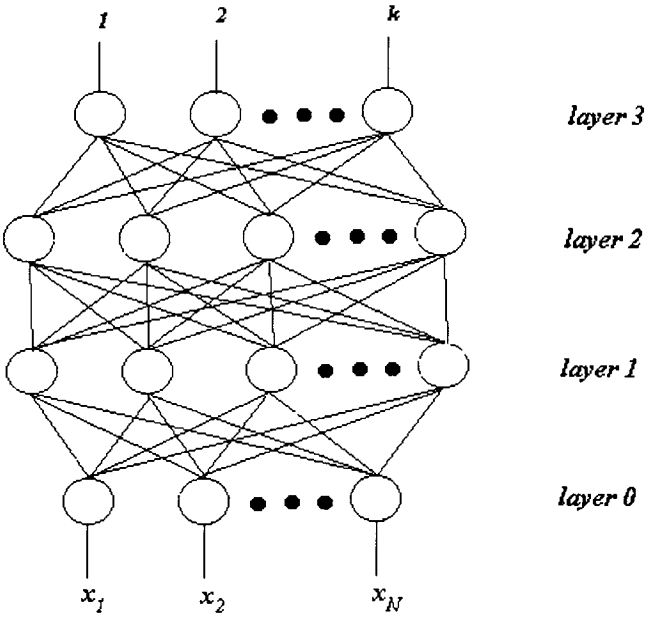

Figure 2. Multilayer perceptron

The neurons in the input layer serve the purpose of fanning out the input values to the neurons of *layer 1*. Let $w_{ji}^{(l)}$, $l = 1, 2, 3$ represent the connections weight on the link from the ith neuron in layer $l-1$ to the jth neuron in layer l. Let $\theta_j^{(l)}$ represent the threshold of the jth neuron in layer l. The total input, $x_j^{(l)}$, received by the jth neuron in layer l is given by

$$x_j^{(l)} = \sum_i y_i^{(l-1)} \, w_{ji}^{(l)} + \theta_j^{(l)} \tag{4}$$

where $y_i^{(l-1)}$ is the output of the ith neuron in layer $l-1$. For the input layer

$$y_i^{(0)} = x_i \tag{5}$$

where x_i is the ith component of the input vector. For the other layers

$$y_i^{(l)} = f(x_i^{(l)}) \qquad l = 1,\ 2,\ 3. \tag{6}$$

Several functional forms like threshold logic, hard limiter, sigmoid etc. can be used for $f(.)$.

There are several algorithms for training the network in order to learn the connection weights and the thresholds from a given training data set. Backpropagation (BP) is one such learning algorithm, where the least mean square error of the network output is computed, and this is propagated in a top down manner (i.e., from the output side) in order to update the weights. The error is computed as the difference between the actual and the desired output when a known input pattern is presented to the network. A gradient descent method along the error surface is used in BP.

4. Analogy between Multilayer Perceptron and *VGA-classifier*

It is known in the literature [8] that Multilayered Perceptron (MLP) with hard limiting non linearities approximates the decision boundaries by piecewise linear surfaces. The parameters of these surfaces are encoded in the connection weights and threshold biases of the network. Similarly, the *VGA-classifier* also generates decision boundaries by appropriately fitting a number of hyperplanes in the feature space. The parameters are encoded in the chromosomes. Thus a clear analogy exists between these two models.

Both the methods start from an initial randomly generated state (the set of initial random weights in MLP). Both of them iterate over a number of generations while attempting to decrease the classification error in the process.

The obvious advantage of the GA based method over that of the MLP is that the *GA-classifier* performs concurrent search for a number of sets of hyperplanes, each representing a different classification in the feature space. On the other hand, the MLP deals with only one such set. Thus it has a greater chance of getting stuck at a local optimum, which the *GA-classifier* can overcome. Moreover, *VGA-classifier* does not assume any fixed value of the number of hyperplanes, while MLP assumes a fixed number of hidden nodes and layers. This results in the problem of over fitting with an associated loss of generalization capability for MLP. In this context one must note that since the *VGA-classifier* has to be terminated after finitely many iterations, and the size of the data set is also finite, it may not always end up with the optimal number of hyperplanes. Consequently, the problem of overfitting exists for *VGA-classifier* also, although it is comparatively reduced.

5. Deriving the MLP architecture

In this section we describe how the principle of fitting a number of hyperplanes using GA, for approximating the class boundaries, can be exploited in determining the appropriate architecture of MLP. Since our aim is to model the equation of hyperplanes, we use the hard

limiting function in the neurons of the MLP, defined as

$$f(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0. \end{cases}$$

5.1. Terminology

Let us assume that the *VGA-classifier* provides
H_{VGA} hyperplanes, designated by

$$\{Hyp_1, Hyp_2, \ldots, Hyp_{H_{VGA}}\},$$

r regions, designated by

$$\{R_1, R_2, \ldots, R_r\},$$

and k be classes, designated by

$$\{C_1, C_2, \ldots, C_k\}.$$

Note that more than one region may be labelled with a particular class, indicating that $r \geq k$.

Let R^1 be the region representing class C_1, and let it be a union of r_1 regions given by

$$R^1 = R_{j_1^1} \bigcup R_{j_2^1} \bigcup \ldots R_{j_{r_1}^1}, \qquad 1 \leq j_1^1, j_2^1, \ldots, j_{r_1}^1 \leq r.$$

Generalizing the above, let R^i $(i = 1, 2, \ldots, k)$ be the region representing class C_i, and let it be a union of r_i regions given by

$$R^i = R_{j_1^i} \bigcup R_{j_2^i} \bigcup \ldots \bigcup R_{j_{r_i}^i}, \qquad 1 \leq j_1^i, j_2^i, \ldots, j_{r_i}^i \leq r.$$

Note that each R^i is disjoint, i.e.,

$$R^i \bigcap R^j = \phi, \qquad i \neq j, \quad i, j = 1, 2, \ldots, k.$$

5.2. Network Construction Algorithm

The network construction algorithm (NCA) is a four step process where the number of neurons, their connection weights and the threshold values are determined. It guarantees that the total number of hidden layers (excluding the input and output layers) will be atmost two. (In this context, Kolmogorov's Mapping Neural Network Existence Theorem may be mentioned. The theorem states that any continuous function can be implemented exactly by a three layer, including input and output layers, feedforward neural network. The proof can be found in [16]. However, nothing has been stated about the selection of connection weights and the neuronal functions.)

The output of the *VGA-classifier* is the parameters of the H_{VGA} hyperplanes. These are obtained as follows :

$$\begin{array}{llll} \alpha_1^1, & \alpha_2^1, \ldots, & \alpha_{N-1}^1, & d^1 \\ \alpha_1^2, & \alpha_2^2, \ldots, & \alpha_{N-1}^2, & d^2 \\ \vdots & & & \\ \alpha_1^{H_{VGA}}, & \alpha_2^{H_{VGA}}, \ldots, & \alpha_{N-1}^{H_{VGA}}, & d^{H_{VGA}} \end{array}$$

Step 1 : Allocate N neurons in the input layer, *layer 0*, where N is the dimensionality of the input vector. The neurons in this layer simply transmit the value in the input links to all the output links.

Step 2 : Allocate H_{VGA} neurons in *layer 1*. Each neuron is connected to the N neurons of *layer 0*. Let the equation of the ith hyperplane ($i = 1, 2, \ldots, H_{VGA}$) be

$$c_1^i x_1 + c_2^i x_2 + \ldots + c_N^i x_N - d = 0$$

where from Eqn. 1 we may write

$$\begin{array}{lcl} c_N^i & = & \cos\alpha_{N-1}^i \\ c_{N-1}^i & = & \cos\alpha_{N-2}^i \sin\alpha_{N-1}^i \\ c_{N-2}^i & = & \cos\alpha_{N-3}^i \sin\alpha_{N-2}^i \sin\alpha_{N-1}^i \\ \vdots & & \\ c_1^i & = & \cos\alpha_0^i \sin\alpha_1^i \ldots \sin\alpha_{N-1}^i \\ & = & \sin\alpha_1^i \ldots \sin\alpha_{N-1}^i \end{array}$$

since $\alpha_0^i = 0$.

Then the corresponding weights on the links to the ith neuron in *layer 1* from those in *layer 0* are

$$w_{ij}^1 = c_j^i, \qquad j = 1, 2, \ldots, N,$$

and

$$\theta_i^1 = -d^i,$$

since the bias term is added to the weighted sum of the inputs to the neurons.

Step 3 : Allocate r neurons in *layer 2* corresponding to the r regions. If the ith region R_i ($i = 1, 2, \ldots, r$) lies on the positive side of the jth hyperplane Hyp_j ($j = 1, 2, \ldots, H_{VGA}$), then

$$w_{ij}^2 = +1.$$

Otherwise

$$w_{ij}^2 = -1,$$

and

$$\theta_i^2 = -(H_{VGA} - 0.5).$$

Note that the neurons in this layer effectively serve the AND function, such that the output is high (+1) if and only if all the inputs are high (+1). Otherwise, the output is low (-1).

Step 4 : Allocate k neurons in *layer 3* (output layer), corresponding to the k classes. The task of these neurons is to combine all the distinct regions that actually correspond to a single class. Let the ith class ($i = 1, 2, \ldots, k$) be a combination of r_i regions. That is,

$$R^i = R_{j_1^i} \bigcup R_{j_2^i} \bigcup \ldots \bigcup R_{j_{r_i}^i}.$$

Then the ith neuron of *layer 3*, ($i = 1, 2, \ldots, k$), is connected to neurons $j_1^i, j_2^i \ldots j_{r_i}^i$ of *layer 2* and,

$$w_{ij}^3 = 1, \qquad j \in \{j_1^i, j_2^i \ldots j_{r_i}^i\}$$

whereas

$$w_{ij}^3 = 0, \qquad j \notin \{j_1^i, j_2^i \ldots j_{r_i}^i\}$$

and

$$\theta_i^3 = r_i - 0.5.$$

Note that the neurons in this layer effectively serve the OR function, such that the output is high (+1) if at least one of the inputs is high (+1). Otherwise, the output is low (-1). For any given point, atmost one output neuron, corresponding to its class, will be high. Also, none of the output neurons will be high if an unknown pattern, lying in a region with unknown classification (i.e., there were no training points in the region) becomes an input to the network.

5.3. An Example

In order to demonstrate the functioning of the algorithm, let us consider the following problem in Fig. 3 for $N = 2$ (two dimensional).

Let L_1 be the line resulting from the application of *VGA-classifier* for partitioning the two classes shown. The corresponding angle (considering angles from the normal to the X_2 axis in the anticlockwise direction) and perpendicular distance values are

$$L_1 \rightarrow \alpha_1^1 = 315^o \text{ and } d^1 = 0.$$

In other words, the equation of L_1 is given by

$$x_2 \cos(315) + x_1 \sin(315) = 0$$

$$\frac{x_2}{\sqrt{2}} - \frac{x_1}{\sqrt{2}} = 0.$$

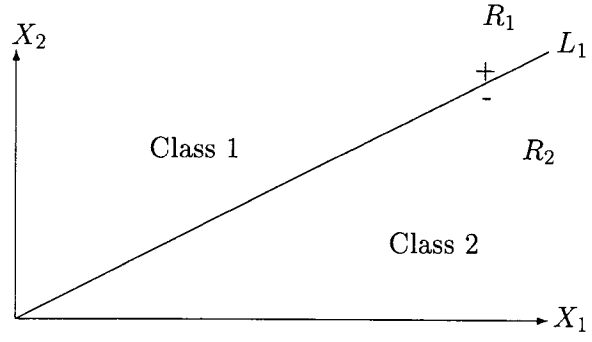

Figure 3. Problem for demonstrating the network construction algorithm

As can be seen from Fig. 3, there are 2 distinct regions viz. R_1, and R_2, of which R_1 represents the region for class 1 and R_2 represents the region for class 2. Also,

$$\begin{aligned} R_1 &\rightarrow \text{+ve side of } L_1 \\ R_2 &\rightarrow \text{-ve side of } L_1 \end{aligned}$$

Applying NCA, we obtain the following :

Step 1 : 2 neurons in the input layer, since N =2.

Step 2 : 1 neuron in *layer 1*, since $H_{VGA} = 1$. The connection weights and the threshold are as follows :

$$\begin{aligned} w_{11}^1 &= \cos\alpha_0^1 \times \sin\alpha_1^1 \\ &= -\tfrac{1}{\sqrt{2}} \\ w_{12}^1 &= \cos\alpha_1^1 \\ &= \tfrac{1}{\sqrt{2}} \\ \theta_1^1 &= -d^1 \\ &= 0.0 \end{aligned}$$

Step 3 : 2 neurons in *layer 2*, since there are two distinct regions, $r = 2$. The connection weights and the thresholds are as follows :

$$\begin{aligned} w_{11}^2 &= 1 \\ \theta_1^2 &= -0.5 \\ w_{21}^2 &= -1 \\ \theta_2^2 &= -0.5 \end{aligned}$$

Step 4 : 2 neurons in the output layer, *layer 3*, since there are two classes, $k = 2$. The

connection weights and the thresholds are as follows :

$$\begin{aligned} w_{11}^3 &= 1 \\ w_{12}^3 &= 0 \\ \theta_1^2 &= 0.5 \\ w_{21}^3 &= 0 \\ w_{22}^3 &= 1 \\ \theta_2^2 &= 0.5 \end{aligned}$$

Note that the zero weights effectively mean that the corresponding connections do not exist. The resulting network is shown in Fig. 4.

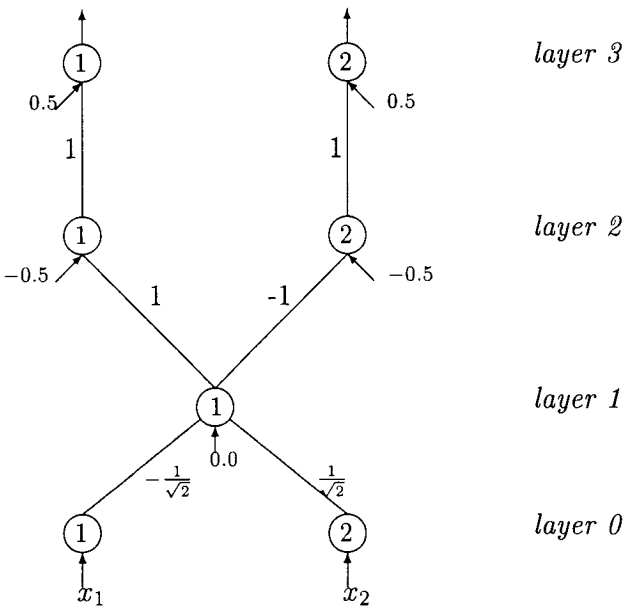

Figure 4. Network for the problem in Fig. 2

5.4. Post Processing Step

The network obtained from the application of NCA may be further optimized in terms of the links and neurons in the output layer. A neuron in *layer 3* that has an input connection from only one neuron in *layer 2* may be eliminated completely. Mathematically, let for some i, $1 \leq i \leq k$,

$$w_{ij}^3 = 1 \quad \text{if } j = j'$$
$$= 0 \quad \text{otherwise,}$$

then neuron i of *layer 3* is eliminated and is replaced by neuron j' of *layer 2*. Its output then becomes the output of the network. Note that this step produces a network where a neuron in layer i is connected to a neuron in layer $i + 2$.

In the extreme case, when all the neurons in the output layer (*layer 3*) get their inputs from exactly one neuron in *layer 2*, the output layer can be totally eliminated, and *layer 2* becomes the output layer. This reduces the number of layers from three to two. This will be the case when $r = k$, i.e., a class is associated with exactly one region formed by the H_{VGA} hyperplanes.

Applying the post processing step to the network obtained in Fig. 4, we find that neurons 1 and 2 of *layer 3* have links only from neurons 1 and 2 of *layer 2* respectively. Consequently, one entire layer may be removed and this results in a network as shown in Fig. 5.

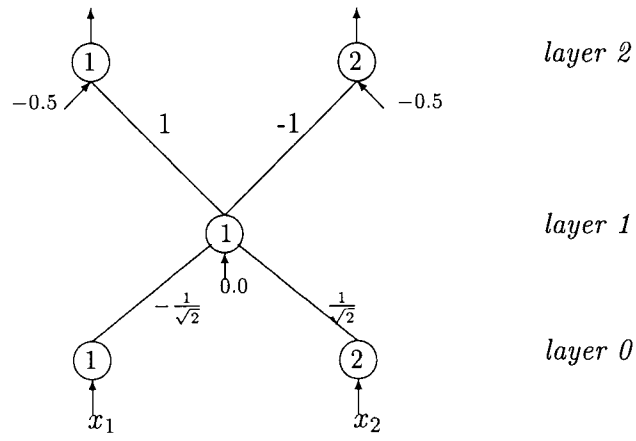

Figure 5. Modified network after post processing

6. Implementation

The effectiveness of the network construction algorithm (NCA) is demonstrated here on a number of real life and artificial data sets.

6.1. Data Sets

ADS 1 and ADS 2 : These are two dimensional, two class, artificial data sets shown in Figs. 6 and 7 respectively. The first one consists of 557 data points while the second consists of 417

data points. The boundaries for both the data sets are seen to be highly non-linear, although the classes are separable.

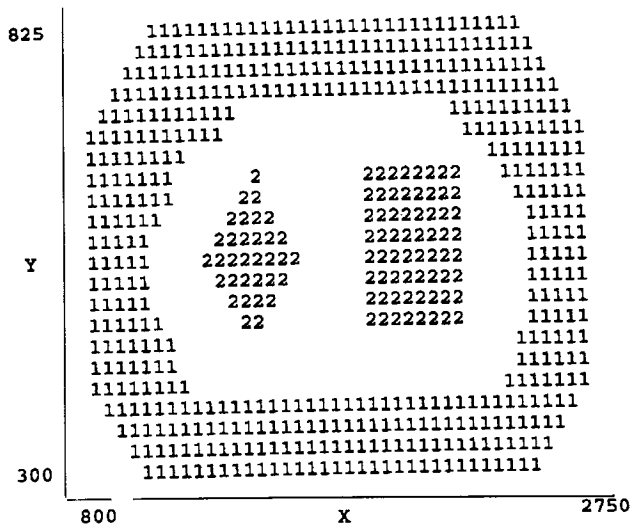

Figure 6. ADS 1

Vowel Data : This data consists of 871 Indian Telugu vowel sounds [17]. These were uttered in a consonant-vowel-consonant context by three male speakers in the age group of 30-35 years. The data set has three features F_1, F_2 and F_3, corresponding to the first, second and third vowel formant frequencies, and six classes $\{\delta, a, i, u, e, o\}$. Fig. 8 shows the distribution of the six classes in the $F_1 - F_2$ plane. (It is known [17] that these two features are more important in characterizing the classes than F_3.) Note that the boundaries of the classes are very ill-defined and overlapping.

Iris Data : This data represents different categories of irises. The four feature values per sample represent the sepal length, sepal width, petal length and the petal width in centimeters [18]. It has three classes with 50 samples per class.

Cancer Data : This breast cancer database, obtained from the University of Wisconsin Hospital, Madison [19], is used for the purpose of demonstrating the effectiveness of the classifier in classifying high dimensional patterns. It has 683 samples belonging to two classes *Benign* (class 1) and *Malignant* (class 2), and nine features corresponding to *clump thickness, cell size uniformity, cell shape uniformity, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli* and *mitoses.*

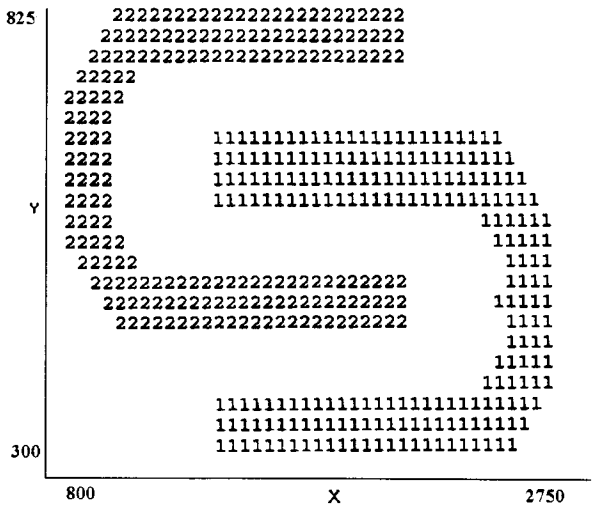

Figure 7. ADS 2

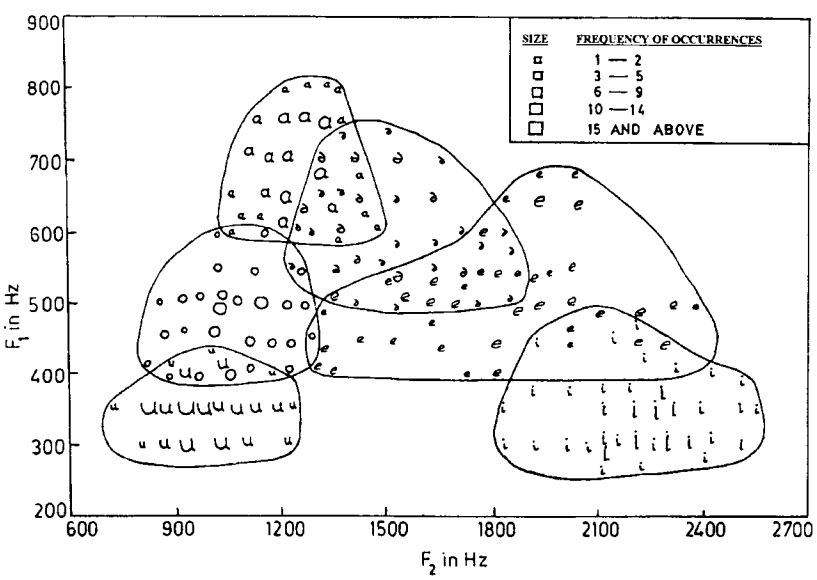

Figure 8. Vowel data in the $F_1 - F_2$ plane

6.2. Results

For the *VGA-classifier*, a fixed population size of 20 is chosen. *Roulette wheel strategy* is used to implement proportional selection. *Single point crossover* is applied with a fixed crossover probability of 0.8. A variable value of mutation probability μ_m is selected from the range [0.015, 0.333]. 200 iterations are performed with each mutation probability value. The values of μ_{m_1} and μ_{m_2} are set to 0.1. The process is executed for a maximum of 3000 iterations. *Elitism* is incorporated in the process. The recognition scores provided here are the average values obtained over five different runs of the algorithm. H_{max} is set to 10, so $\alpha = 0.1$.

The MLP is executed using both hard limiters and the sigmoid function in the neurons. The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}.$$

The learning rate and momentum factor are fixed at 0.8 and 0.2 respectively. Online weight updation, i.e., updation after each training data input, is performed for a maximum of 3000 iterations.

The performance of *VGA-classifier* and consequently that of the MLP derived using NCA (i.e., where the architecture and the connection weights have been determined using NCA) are compared with that of a conventional MLP having the same architecture as provided by NCA, but trained using the back propagation (BP) algorithm with the neurons executing the sigmoid function. For the purpose of comparison, we have also considered here three more typical architectures for the conventional MLP having two hidden layers with 5, 10 and 20 nodes in each layer respectively. Tables 1-5 summarize the results obtained. The MLP architecture is denoted by Arch. in the tables.

The number of hyperplanes (H_{VGA}) and regions (r) obtained by the *VGA-classifier* starting from $H_{max} = 10$ are mentioned in columns 2-3. These are used to select the MLP architectures as shown in columns 10-11 and 12-13.

From Table 1 corresponding to *ADS 1*, it is found that the MLP trained using BP does not succeed in learning the boundaries of class 2, for all the architectures (columns 4-11).

Table 1. Classwise and overall recognition scores (%) for *ADS 1* during training and testing

Class	*VGA-classifier* $H_{VGA} = 3, r = 5$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=2:5:5:2		Arch.=2:10:10:2		Arch.=2:20:20:2		Arch.=2:3:5:2		Arch.=2:3:5:2	
	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.
1	100.00	95.89	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	95.89
2	100.00	94.31	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.00	94.31
Overall	100.00	95.62	83.63	82.47	83.63	82.47	83.63	82.47	83.63	82.47	100.00	95.62

Table 2. Classwise and overall recognition scores (%) for *ADS 2* during training and testing

Class	*VGA-classifier* $H_{VGA} = 6, r = 12$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=2:5:5:2		Arch.=2:10:10:2		Arch.=2:20:20:2		Arch.=2:6:12:2		Arch.=2:6:12:2	
	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.
1	100.00	93.52	100.00	95.01	100.00	100.00	100.00	100.00	100.00	100.00	100.00	93.52
2	84.21	78.85	100.00	88.62	84.21	76.57	73.68	68.10	89.47	86.85	84.21	78.85
Overall	92.68	88.16	100.00	91.92	92.68	89.09	87.80	85.10	95.12	93.88	92.68	88.16

Table 3. Classwise and overall recognition scores (%) for *Vowel* during training and testing

Class	*VGA-classifier* $H_{VGA} = 6, r = 7$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=3:5:5:6		Arch.=3:10:10:6		Arch.=3:20:20:6		Arch.=3:6:7:6		Arch.=3:6:7:6	
	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.
δ	10.30	8.21	85.71	12.30	87.51	47.69	100.00	47.23	85.71	6.15	10.30	8.21
a	100.00	91.35	75.00	43.20	100.00	16.04	100.00	27.62	87.50	13.58	100.00	91.35
i	94.11	84.51	100.00	69.67	100.00	79.35	100.00	80.58	100.00	78.70	94.11	84.51
u	73.33	66.91	86.67	80.14	100.00	77.94	100.00	83.29	100.00	91.91	73.33	66.91
e	89.99	85.56	80.00	68.44	94.98	90.37	100.00	87.30	80.00	59.35	89.99	85.56
o	83.33	75.92	88.89	77.16	94.44	54.93	94.44	51.70	77.78	43.21	83.33	75.92
Overall	80.00	73.66	87.05	65.26	95.82	67.55	98.82	68.48	88.23	56.36	80.00	73.66

Table 4. Classwise and overall recognition scores (%) for *Iris* during training and testing

Class	*VGA-classifier* $H_{VGA} = 2, r = 3$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=4:5:5:3		Arch.=4:10:10:3		Arch.=4:20:20:3		Arch.=4:2:3:3		Arch.=4:2:3:3	
	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.
1	100.00	100.00	100.00	100.00	100.00	98.81	100.00	66.67	100.00	100.00	100.00	100.00
2	100.00	93.33	100.00	66.67	100.00	64.77	100.00	71.11	100.00	77.78	100.00	93.33
3	100.00	93.33	100.00	77.78	100.00	80.51	100.00	97.78	100.00	66.67	100.00	93.33
overall	100.00	95.56	100.00	81.48	100.00	81.36	100.00	78.51	100.00	81.48	100.00	95.56

Table 5. Classwise and overall recognition scores (%) for *Cancer* during training and testing

Class	*VGA-classifier* $H_{VGA} = 2, r = 4$		MLP									
			SIGMOID								Hard Limiter	
			Arch.=9:5:5:2		Arch.=9:10:10:2		Arch.=9:20:20:2		Arch.=9:2:4:2		Arch.=9:2:4:2	
	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.	Train.	Test.
1	97.72	98.50	100.00	97.25	100.00	97.25	100.00	97.25	97.65	96.75	97.72	98.50
2	100.00	83.79	100.00	90.74	100.00	90.74	100.00	90.74	100.00	93.75	100.00	83.79
overall	95.50	93.34	100.00	94.97	100.00	94.97	100.00	94.97	98.51	95.70	95.50	93.34

In fact, as seen from Fig. 6, class 2 is totally surrounded by class 1. The *VGA-classifier*, on the other hand, is able to place the lines appropriately, thereby yielding a significantly better score both during training and testing (columns 2-3). Consequently, the network derived using NCA (which has the performance same as that of the *VGA-classifier*) also provides a significantly better score (columns 12-13).

Similar is the case for *Vowel* and *Iris* data (Tables 3 and 4 respectively) where the *VGA-classifier*, and consequently the MLP derived using NCA provide a superior performance than the MLPs trained with BP. For *ADS 2* (Table 2) and *Cancer* data (Table 5), the situation is different where MLPs trained with BP provide superior performance (except one case for *ADS 2*). The overall recognition score during testing for *Vowel* is found to increase with the increase in the number of nodes in the hidden layers (columns 5, 7 and 9) since the classes are highly overlapping. For *Iris* data, the reverse is true, indicating a case of overfitting the classes.

Note that the Arch. values of the MLPs mentioned in columns 10-11 and 12-13 of the tables are the ones obtained without the application of the post processing step. These values are put in order to clearly represent the mapping from *VGA-classifier* to MLP, in terms of the number of hyperplanes and regions, although the post processing task could have reduced the size of the network while keeping the performance same. For example in the case of *Iris* data, the number of hyperplanes and regions are 2 and 3 (columns 2-3) respectively. Keeping analogy with this, the Arch. value in column 12-13 are mentioned to be 4:2:3:3. In practice, after post processing, the said values became 4:2:3. Similarly, for *ADS 1*, *Vowel* and *Cancer* data, the values after post processing were found to be 2:3:4:2, 3:6:2:6 and 9:2:3:2 respectively. In the case of *ADS 2*, there was no change before and after post processing.

7. Discussion and Conclusions

A method for automatic determination of MLP architecture and the associated connection weights is described, based on its analogy with the *VGA-classifier* in terms of placement capability of hyperplanes for approximating the class boundaries. The method guarantees that the architecture will involve atmost two layers (excluding the input and output layers), with the neurons in the first and second hidden layers being responsible for hyperplane and region generation, and those in the output providing a combination of regions for the classes.

This investigation may also be considered as an application of the *VGA-classifier*. It not only finds a relation of *VGA-classifier* with the MLP, but also provides a way of determining an appropriate archtecture and connection weights for MLP. Moreover, the said analogy will augment the application domain of the *VGA-classifier* to those areas where MLP has widespread use.

Since the principle of *VGA-classifier* is used for developing NCA, it becomes mandatory to consider hard limiting neurons in the derived MLP. Although this makes the network rigid and susceptible to noise and corruption in the data, one may use NCA for providing a possible appropriate structure of conventional MLPs.

Acknowledgment : This work was carried out when Ms. Sanghamitra Bandyopadhyay held a Dr. K. S. Krishnan fellowship awarded by the Department of Atomic Energy, Govt. of India.

References

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

[2] L. Davis, *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

[3] E. S. Gelsema, ed., *Pattern Recognition Letters*, vol. 16, no. 8. Elsevier, August 1995. Special Issue on Genetic Algorithms.

[4] S. K. Pal and P. P. Wang, eds., *Genetic Algorithms for Pattern Recognition.* Boca Raton: CRC Press, 1996.

[5] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Pattern classification using genetic algorithms," *Patt. Recog. Lett.*, vol. 16, no. 8, pp. 801–808, 1995.

[6] S. Bandyopadhyay, C. A. Murthy, and S. K. Pal, "Pattern classification using genetic algorithms : Determination of H." *Pattern Recog. Lett.* (to appear).

[7] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation.* New York: Addison Wesley, 1991.

[8] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4–22, April 1987.

[9] D. E. Rumelhart, J. McClelland, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986.

[10] D. E. Rumelhart, J. McClelland, *et al.*, *Parallel Distributed Processing*, vol. 2. Cambridge, MA: MIT Press, 1986.

[11] S. K. Pal and D. Bhandari, "Selection of optimum set of weights in a layered network using genetic algorithms," *Inform. Sci.*, vol. 80, pp. 213–234, 1994.

[12] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 39–53, 1994.

[13] N. K. Bose and A. K. Garga, "Neural network design using voronoi diagrams," *IEEE Transactions on Neural Networks*, vol. 4, pp. 778–787, September 1993.

[14] R. Reed, "Pruning algorithms - a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[15] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* New York: Springer Verlag, 1992.

[16] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *1st IEEE Intl. Conf. on Neural Networks*, vol. 3, (San Diego, California), pp. 11–14, 1987.

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

[18] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 3, pp. 179–188, 1936.

[19] O. L. Mangasarin, R. Setiono, and W. H. Wolberg, "Pattern recognition via linear programming: Theory and application to medical diagnosis," in *Large-scale Numerical Optimization* (T. F. Coleman and Y. Li, eds.), pp. 22–30, Philadelphia: SIAM Publications, 1990.