ON SOME SELF-ORGANIZING MODELS AND THEIR APPLICATIONS

AMITAVA DATTA

COMPUTER AND STATISTICAL SERVICE CENTRE
INDIAN STATISTICAL INSTITUTE
203, B. T. ROAD
CALCUTTA-700 035
INDIA

A THESIS SUBMITTED TO THE INDIAN STATISTICAL INSTITUTE IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

1999

ON SOME SELF-ORGANIZING MODELS AND THEIR APPLICATIONS

AMITAVA DATTA

COMPUTER AND STATISTICAL SERVICE CENTRE
INDIAN STATISTICAL INSTITUTE
203 B. T. ROAD
CALCUTTA-700 035
INDIA

A THESIS
SUBMITTED TO THE INDIAN STATISTICAL INSTITUTE
IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Acknowledgments

I thank my supervisor Dr. S. K. Parui for his valuable guidance. I also thank Prof. B. B. Chaudhuri, Head, Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, Calcutta, for his constant encouragement and for letting me use the resources of his Unit. I would also like to thank Prof. D. Dutta Majumder, Emeritus Professor, for his inspiration. Thanks are also due to my colleagues for their help.

I also thank my family members. I dedicate this thesis towards the memory of my late father Shri Kalipada Dutta.

Date: August 16, 1999 AMITAVA DATTA

Contents

1 INTRODUCTION 1

1.1 Artificial neural network 1

1.2 Hamming network 4

1.3 Maxnet 5

1.4 Simple competitive learning 6

1.5 Kohonen's SONN model 9

1.6 Characteristics of Kohonen's SONN model 13

1.7 Convergence 17

1.8 Extensions of the SONN model 18

1.9 Scopes of the thesis 19

1.9.1 Shape extraction 20

1.9.2 Convex hull 25

1.9.3 Minimum spanning circle 26

1.10 Conclusions 26

2 A DYNAMIC SELF-ORGANIZING NEURAL NETWORK FOR SHAPE EXTRACTION 29

2.1 Introduction 29
2.2 The DySONN model 32
2.2.1 Arc patterns 33
2.2.2 Tree patterns 35
2.2.3 Loop patterns 39
2.3 Choice of parameters 42
2.3.1 Choice of δ 42
2.3.2 Choice of θ 45
2.4 Discussions and remarks 46
2.5 Conclusions 49

3 A TOPOLOGY ADAPTIVE SELF-ORGANIZING NEURAL NETWORK FOR SHAPE EXTRACTION 51
3.1 Introduction 51
3.2 The background 53
3.3 Shortcomings of NGN/GNGN model 55
3.4 The TASONN model 60
3.5 Shape extraction by TASONN model 65
3.6 Discussions and remarks 68
3.7 Conclusions 71

4 COMPARISONS OF NEURAL NETWORK BASED AND CONVENTIONAL SHAPE EXTRACTION TECHNIQUES 73
4.1 Introduction 73
4.2 Medial axis representation 75

4.3 Robustness 76

4.3.1 Boundary noise 77

4.3.2 Object noise 78

4.4 Rotation invariance 82

4.5 Data reduction efficiency 83

4.6 Extendability to dot pattern and gray-level pattern 84

4.6.1 Dot pattern 84

4.6.2 Gray-level pattern 86

4.7 Computational aspects 88

4.8 Conclusions 89

5 A SELF-ORGANIZING MODEL TO COMPUTE CONVEX HULL 91

5.1 Introduction 91

5.2 The proposed model 94

5.3 The architecture of the model 98

5.3.1 Computational aspects 104

5.4 Results and conclusions 105

6 MINIMUM SPANNING CIRCLE BY SELF-ORGANIZATION 107

6.1 Introduction 107

6.2 The MSC problem 109

6.3 The proposed model for the MSC problem 110

6.4 Convergence 113

6.5 Discussions 122

6.6 Conclusions 124

CONCLUSIONS **126**

7.1 Contributions of the thesis 126

7.2 Future scopes 128

7.2.1 Shapes in higher dimensions 128

7.2.2 Extraction of outer shape 129

7.2.3 Extension of minimum spanning circle model 130

List of Figures

1.1 Architecture of Hamming Network. 5

1.2 Maxnet architecture. 6

1.3 Simple competitive learning network. 7

1.4 Network architecture for SONN model (a) one-dimensional network and (b) two-dimensional (rectangular grid) network. 10

1.5 Varying neighbourhood over time $0 < t_1 < t_2 < t_3$ for the processor π_i in (a) one-dimensional (b) two-dimensional network. 12

1.6 The weight vectors are represented by circles. (a) not ordered topographically, (b) ordered topographically. (c) not ordered topographically, (d) ordered topographically. 14

1.7 The cluster centroids (weight vectors) are shown by circles. (a) not ordered topographically, (b)-(c) ordered topographically. 15

1.8 The orientation of the output network automatically changes to the direction of the higher variance. 16

1.9 The output networks with different number of processors for the same input distribution. 16

1.10 The 4- and 8-neighbourhood of a digitized image. 20

1.11 (a) Binary image (b) raster skeleton (c) vector skeleton (d) connection graph. 24

2.1 The output network by the SONN model for a pattern having a circular shape. 30

2.2 The problems of the SONN model due to its fixed network topology. (a) circular shape (b) fork shape. 31

2.3 Different stages of the network for an arc-pattern 'S' ($\delta = 12$). (a) Initial network. Intermediate stages (b) after 20 sweeps (c) after 40 sweeps. (d) Final network giving the vector skeleton after 60 sweeps and (e) the raster skeleton. 36

2.4 (a), (c), (e) : the network locally forms a spike to indicate one of the branchings, (b), (d), (f) : a processor U with higher degree is inserted to accommodate the branching. 36

2.5 Different stages of the network for the pattern 'T' ($\delta = 12$). Intermediate stages (a) after 20 sweeps (b) after 40 sweeps (c) after 50 sweeps. (d) Final vector skeleton after 70 sweeps and (e) the raster skeleton. . 38

2.6 Different stages of the network for the pattern 'A' ($\delta = 12$). Intermediate stages (a) after 20 sweeps (b) after 40 sweeps (c) after 70 sweeps. (d) Final vector skeleton after 100 sweeps and (e) the raster skeleton. 40

2.7 The role of δ on the output skeleton. (a) δ=5 (b) δ=7 (c) δ=9 (d) δ=11 (e) δ=13 (f) δ=15. 42

2.8 The activation regions at different time $t = t_1,\ t_2,\ t_3$ $(t_1 < t_2 < t_3)$. (a) at time t_1 (b) at time t_2 (c) at time t_3. 43

2.9 The raster skeletons obtained (a),(b) without activation level and (c),(d) with activation level (δ=5). 44

2.10 The role of parameter θ. 45

2.11 Output networks for a A-shaped pattern when (a) δ=30, (b) δ=25, (c) δ=20, (d) δ=15 (e) δ=10, (f) δ=5. 49

2.12 Raster skeletons produced by DySONN model. 50

3.1 A clustering application 56

3.2 Stability-plasticity dilemma. (a) Kohonen's SONN model, (b) NGN model, (c) the proposed TASONN model. 58

3.3 For uniform input over O-shaped pattern, Kohonen's output network with (a) linear topology (b) rectangular topology, (c) output network of NGN/GNGN models, (d) output network of TASONN (only links with nonzero β are shown). 59

3.4 For uniform input over Y-shaped pattern, Kohonen's output network with (a) linear topology (b) rectangular topology, (c) output network of NGN/GNGN models, (d) output network of TASONN (only links with nonzero β are shown). 59

3.5 For L-shaped pattern, the output network achieved by TASONN after convergence (a) δ=25, (b) δ=50. The nonzero β values are shown against each link. 63

3.6 Link establishments in (a) arc pattern (b) fork pattern. 67

3.7 Output networks for an A-shaped pattern when (a) δ=30, (b) δ=25, (c) δ=20, (d) δ=15 (e) δ=10, (f) δ=5 without activation level. 69

3.8 Output networks for an A-shaped pattern when (a) δ=10, (b) δ=7, (c) δ=5, (d) δ=3 with activation level. 70

3.9 Output raster skeletons by TASONN model for different patterns. . . 71

4.1 Robustness to boundary noise, an illustration. '0' represents object and '*' represents skeletal pixel. (a) A line segment with four boundary noise pixels, (b) result of a conventional iterative thinning algorithm, (c)-(d) results of our neural algorithms with the same noise and higher noise. 79

4.2 Boundary noise immunity. 80

4.3 (a) Input with two white noise pixels inside the object; output of conventional thinning algorithms (b) without noise; (c) with noise; (d) vector skeleton by the two proposed neural algorithms with the same noise as in (b); (e) output raster skeleton for the same. 81

4.4 The final skeletons obtained for the pattern 'A' with object noise. (a) SNR= 2 (b) SNR= 1.5 (c) SNR= 1.1. 82

4.5 Effect of rotation by angles (a) 45° (b) 22° and (c) 11°. 83

4.6 Results on a B-shaped dot pattern (a)-(c) by DySONN and (d)-(f) by TASONN model. (c) and (f) are the final vector skeletons and others are intermediate results. 86

4.7 Output skeletons of gray-level characters and chromosome pattern by DySONN model ($\delta = 5$). 87

4.8 Output skeletons of gray-level characters and chromosome pattern by TASONN model ($\delta = 5$). 88

5.1 A point set in $2D$ and its convex-hull. 92

5.2 Point-processors and hull-processors (circled). The labels in parentheses denote the processor types. 96

5.3 The subnetwork architecture for a given type of hull-processor. Solid dots indicate point-processors and the circled dots represent hull-processors. All the links represented by lines are bidirectional. 99

5.4 The complete network architecture for the proposed model. 101

5.5 The intermediate and final results on a planar set. (a)-(c) The results after iterations 1, 2 and 3 respectively; (d) the final result after 4 iterations. 105

6.1 The minimum spanning circle determined by two points. '+' represents the initial position of the weight vector and 'x' represents the final centre of the circle ($t = 50$). 112

6.2 The minimum spanning circle determined by three points ($t = 70$). . 113

6.3 Farthest point Voronoi diagrams for two planar sets. MSC is determined by (a) three points, (b) two points. 114

6.4 The trajectories of $W(t)$. (a) Case 2, (b) Case 3(a), (c) Case 3(b). The FPVD is denoted by solid lines and the triangle formed by the input points is denoted by dashed lines. 120

6.5 The architecture of the MSC model. Solid tiny circles represent point processors. 123

List of Tables

4.1 Index of medial axis representation 76

Chapter 1

INTRODUCTION

Abstract: *Self-organizing neural network models constitute the main theme of this thesis. Some well-known self-organizing models are surveyed and their properties are discussed. The application areas on which the thesis focuses are briefly described.*

This thesis deals with *Artificial Neural Network* models, in particular, *Self-organizing (unsupervised)* models. We develop here a few self-organizing neural network models to solve certain problems which are well studied in the areas of *Image Processing* and *Computational Geometry* and have wide applications in *shape extraction* and *optimization.*

1.1 Artificial neural network

The study of *Biological Neural Networks* originally comes under biological sciences. They deal with the brain functions in living organisms. A lot of research work in this area have established that human brains are composed of a huge number of *neurons* (elementary processing elements) with massive parallel interconnections (forming a network). Such networks have learning capabilities from external stimuli (input signals) and they can store, in some manner, what they have learned. *Artificial Neural Networks* are man-made (simplified) models of the biological neural networks. Such

artificial models have been of great interest for some years in various areas like optimization, pattern recognition, computer vision, image processing, robotics, classification, industries etc. [10, 15, 42, 43, 45, 69, 76, 86, 96, 99, 101]. Artificial neural network models or simply neural networks are massively parallel inter-connections of simple computational elements (*processors* — also called *nodes* or *units*) that work as a collective system (we shall use these terms interchangeably). Instead of performing operations sequentially, neural network models (henceforth, by neural network models we shall mean artificial neural network models) are capable of doing the same simultaneously using massively parallel networks composed of a number of processors connected by links, and thus provide high computational rates for real-time processing. This is why neural network models are also called *connectionist* models and *parallel distributed processing* (PDP) models.

The ability of parallel computation is one of the major motivations of using neural network as a tool for solving various problems in the areas mentioned above. Neural networks are also popular due to their adaptive nature. They provide a new form of parallel computing, called *neurocomputing* as against *conventional computing*, in an adaptive manner as observed in living beings. Starting from an initial set of weights (usually random) neural network models can adapt or adjust the initial weights to improve performance. The adaptation is a major focus of neural network research. With the help of a number of processors, serving as the neurons in biological systems, connected by links, the neural network can artificially work in a similar way as the brain does. Moreover, in neural network technology, a complex problem can be solved by a number of processors, working collectively, while each processor needs to do much simpler computations only. Thus simple processors can work collectively and can solve a much complex problem. Neural network models, by its nature, provide robustness to some extent. Damage to a few processors or links may not affect the overall performance and thus such models have sometimes higher fault tolerance. Moreover, in some cases, neurocomputing is found to an efficient tool where conventional computing poses problems or performs poorly.

Several neural network models have been proposed so far (see [43, 44, 69, 79, 86, 96, 100, 101]). Neural network models are specified by their network topology, node

(processor) characteristics and training/learning rules. Training is accomplished by sequentially applying external input signals (usually vector valued). The rules specify how the weight vectors are updated against the presentation of each input vector. After a finite number of updates the weight vectors converge and then the network is said to be trained/converged. A subsequent input, presented to the trained network, would then produce the desired output. Based on the training/learning technique, the neural network models can be broadly classified into two categories: (1) *supervised* and (2) *unsupervised.*

In supervised models, a pair of vectors (training pair), the input vector and the desired output vector (target vector), is presented to the network for training purpose. An input vector is presented to the network and the network calculates the output vector on the basis of the current values of the weight vectors. The calculated output vector is then compared with the desired output vector and the difference (error) is fed back to the network and the weight vectors are updated according to some rule which tends to minimize the error. The training pairs are applied sequentially and every time the weight vectors are updated. This process repeats until the error is significantly low. Examples of supervised model are *Perceptrons* and *Multi-layer perceptrons* (see [69, 79, 96]). In unsupervised models, the learning is accomplished without any supervision. Only the input vectors are presented and the network learns without any target vector. A set of learning rules lead the network to a desired goal. The process continues until some stopping criteria are met. Among the most well-known unsupervised neural network models one can mention Kohonen's feature map [60]-[65] and adaptive resonance theory (ART) [14].

The present thesis deals with unsupervised neural network models on a few well-known problems. **These models are *self-organizing* in that the learning takes place automatically without any supervision**. The problems that are included here are studied very often in the literature of image processing and computational geometry. Both the areas have had tremendous growth in last few decades. In this chapter, we shall first describe the existing self-organizing models. Next, the application problems will be discussed. In the rest of the thesis, we shall propose new self-organizing models for applications on these problems. Before going to discuss

the existing models, some basic models need to be described as they are required by the models studied in the thesis. Certain unsupervised learning models using very simple methods are essential for more complex models. These models work as the basic functions of the complex ones. We shall describe them now.

1.2 Hamming network

In unsupervised learning, distance calculations and comparisons are essential. The Hamming network computes a particular distance measure called the *Hamming distance* that gives the number of differing bits in two binary vectors. The network contains two layers of nodes (neurons or processors) – an input layer and an output layer (Fig.1.1). The connection weights between the layers are the components of the stored binary input vector say, $X_j = (x_{j1}, x_{j2}, \ldots, x_{jm})$ of dimensionality m where $x_{ji} \in \{-1, 1\}$, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$. That is, if w_{ji} is the connection weight from the ith node of the input layer to the jth node of the output layer then the weight matrix W can be written as

$$W = \frac{1}{2}\begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{pmatrix} \tag{1.1}$$

Given a threshold vector

$$\Theta = \begin{pmatrix} -\frac{m}{2} \\ -\frac{m}{2} \\ \vdots \\ -\frac{m}{2} \end{pmatrix} \tag{1.2}$$

when a new input vector $X = (x_1, x_2, \ldots, x_m)$ is presented to the network, the output nodes generate the output as given by

$$O = WX + \Theta = \frac{1}{2}\begin{pmatrix} X_1.X - m \\ X_2.X - m \\ \vdots \\ X_n.X - m \end{pmatrix} \tag{1.3}$$

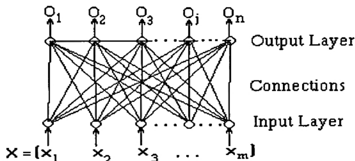


Figure 1.1: Architecture of Hamming Network.

where the dot product

$$X_j.X = \sum_{k=1}^{m} x_{jk}x_k$$

Given W and Θ, each output value O_j $(j = 1, 2, \ldots, n)$ represents the negative of the Hamming distance between a stored vector X_j and a new input vector X. After computing the Hamming distance, one can determine which stored vector is the nearest to the presented input vector by simply taking the maximum of the outputs of the nodes in the output layer. This may be accomplished by a *maxnet* as described below. This 'maxnet' is attached on the top of the output layer of the Hamming network.

1.3 Maxnet

The 'maxnet' is a recurrent network. It has only one layer of n nodes (Fig.1.2) which compete to determine the node with the highest initial value. The network performs an iterative process where each node receives inhibitory input from other nodes through *lateral* (intra-layer) connections.

All the nodes update their outputs simultaneously (in parallel). The single node

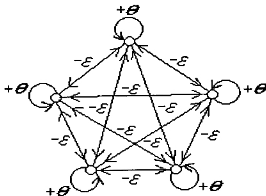


Figure 1.2: Maxnet architecture.

whose value is initially the maximum eventually prevails as the "winner" node, and the activation of all other nodes subsides to zero. The *activation function* (by which the outputs are computed) of a node is given by

$$O_j = \max(0, \sum_{i=1}^{n} w_{ji} x_i) \tag{1.4}$$

where x_i's $(i = 1, 2, \ldots, n)$ are the input to the ith node. The output of each node at the current iteration is fed as the input at the next iteration to compute the output of the next iteration. The self-excitation weight $w_{jj} = \theta = 1$ and the inhibition weights w_{ji} $(i \neq j)$ are $\varepsilon \leq \frac{1}{n}$ (see Fig.1.2).

Thus the 'maxnet' allows the parallel computation of the maximum value from a given set of values where every computation is local to each node rather than being controlled by a central processor. It can be seen that the number of iterations to select the winner node does not depend on the input size n.

1.4 Simple competitive learning

The above networks, Hamming network and the 'maxnet', assist other networks in computing distances and determining the nearest weight vector to the presented input vector. The Hamming network assumes binary input and output vectors, which can

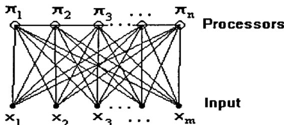


Figure 1.3: Simple competitive learning network.

be generalized to real-valued vectors. We now describe a *simple competitive network* model, consisting of an input layer with m nodes and an output layer with n nodes, which works as follows. Each node in the input layer is connected to each node in the output layer (Fig.1.3). Let the connection weight from the lth input node to the ith output node be w_{il}, $l = 1, 2, \ldots, m$; $i = 1, 2, \ldots, n$. The nodes $\pi_1, \pi_2, \ldots, \pi_n$ in the output layer are interconnected with inhibitory connections as described in the 'maxnet'. m dimensional input vectors $X_j = (x_{j1}, x_{j2}, \ldots, x_{jm})$, $(j = 1, 2, \ldots, N)$ are presented to the input layer of the network. The connection weight vectors $W_i = (w_{i1}, w_{i2}, \ldots, w_{im})$, $i = 1, 2, \ldots, n$; from the input layer to the output layer are initially set at random. All the nodes in the output layer compute the Euclidean distances from their respective weight vectors to the presented input vector in parallel. A competition occurs to find the "winner" among the nodes in the output layer whose weight vector is the nearest from the input vector.

Suppose that the weight vectors are normalized. Then $\sum_{l=1}^{m} w_{il}^2 = 1$, for $i = 1, 2, \ldots, n$. If $d(.)$ stands for the Euclidean distance then

$$\begin{aligned} d^2(W_i, X_j) &= \sum_{l=1}^{m} (w_{il} - x_{jl})^2 \\ &= 1 + \sum_{l=1}^{m} x_{jl}^2 - 2\sum_{l=1}^{m} w_{il} x_{jl} \\ &= 1 + \sum_{l=1}^{m} x_{jl}^2 - 2W_i.X_j \end{aligned} \tag{1.5}$$

This means, for a given X_j, the Euclidean distance $d(W_i, X_j)$ will be minimized when

the dot product $W_i.X_j$ is maximized. Thus a 'maxnet' may be used to minimize the Euclidean distance.

After finding the winner node (say, it is the kth node) we now update its weight as follows so that it moves towards the input vector X_j, while all other weight vectors are unchanged. This is known as the *winner-take-all* phase. Since the nodes compete for being the one to fire (or, to become the winner), these nodes are often called winner-take-all nodes.

$$W_k^{(new)} = W_k^{(old)} + \alpha(X_j - W_k^{(old)}) \tag{1.6}$$

where the gain coefficient α ($0 < \alpha < 1$) satisfies some conditions that will be dealt with in detail later on. It can be shown that in the limiting case, each weight vector W_i converges to the average of all the inputs X_j for which W_i is the winner (see [79]).

It does not matter much how the winner processor is selected [51]. In simulations simply the maximum value can be computed. In a network, a set of winner-take-all nodes with lateral inhibitions and self-excitatory connections similar to the 'maxnet', for example, can do the job. The lateral weights and the activation function must be chosen correctly to ensure that only one output is chosen. With these preliminaries on basic models, we now proceed to our main theme of discussion, that is, the self-organizing neural networks.

Among the unsupervised neural network models, Kohonen's *Self-Organizing Neural Network (SONN) model* , also called *self-organizing feature maps* [64], is well-known. **The term 'self-organization' refers to the ability to adapt or learn from the input without having any prior supervising information and is a major issue of the present thesis.** It is argued that self-organization works as a basic principle of sensory paths of the human audio and visual systems. In this case, self-organization means a meaningful ordering of the neurons (processors) of the brain, where the ordering does not mean moving of neurons physically from one place to another. It is the set of their internal parameters (weights) which defines the ordering and is made to change. The maps formed by the self-organizing system are

able to describe topological [1] relations of input patterns using a 1- or 2-dimensional medium of representation.

1.5 Kohonen's SONN model

Kohonen's SONN model combines a *competitive learning* principle with *topological structuring* [2] of the processors. Competitive learning, as seen earlier, is a process of learning where a competition is conducted among the processors (also see [66]) after each input vector is presented to the network to decide the winner processor that will take part in the learning process for the given input. The SONN model uses a network that consists of a single layer of processors as shown in Fig.1.4. The processors are extensively interconnected with many local connections that define their topological relations, and may be arranged in a linear or a planar array. Apart from the interconnections among the processors, every processor is connected to every input line (via a variable connection weight) through which external signals are presented to the network.

Suppose the input signal $X = (x_1, x_2, \ldots, x_m)$ comes from m-dimensional space R^m. Suppose the set of processors is represented by $\{\pi_1, \pi_2, \ldots, \pi_n\}$. Every processor π_i has a weight vector $W_i = (w_{i1}, w_{i2}, \ldots, w_{im})$ where w_{il} represents the connection weight between ith processor and lth component of the input vector. These weights are initially assigned with random values. If the weights are normalized then the following dot product can be a measure of similarity between the input and the weight vectors. To train the network, an input vector X is presented and the dot products

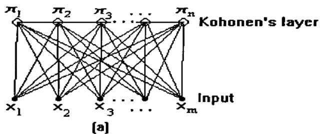
$$W_i.X = \sum_{l=1}^{m} w_{il}x_l, \qquad i = 1, 2, \ldots, n \tag{1.7}$$

are calculated. Alternatively, the Euclidean distance $d(X, W_i)$ can be used as the measure of similarity

$$d^2(X, W_i) = \sum_{l=1}^{m}(x_l - w_{il})^2 = ||X - W_i||^2 \tag{1.8}$$

[1] The topology of the input pattern is in terms of proximity in the pattern space

[2] The network topology, in the thesis, is specified in terms of a neighbourhood relation among processors



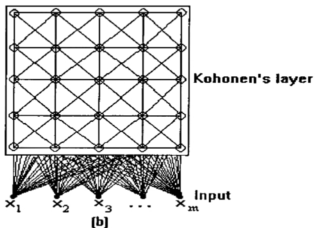


Figure 1.4: Network architecture for SONN model (a) one-dimensional network and (b) two-dimensional (rectangular grid) network.

The learning is accomplished by selecting the weight vector that is most similar (best match) to the presented input vector, and making it along with its neighbouring weight vectors still more similar to the input. Thus Kohonen's SONN model consists of two major steps (formulated for a discrete-time index $t = 0, 1, 2, \ldots$).

1. **Similarity matching:** Select $W_k(t)$ such that

$$||X(t) - W_k(t)|| = \min_i ||X(t) - W_i(t)|| \tag{1.9}$$

where $X(t)$ and $W_i(t)$ are the input and weight vectors respectively at time t. The processor π_k is called the *winner* processor since it wins the above competition.

2. **Updating:** Update the weight vectors $W_k(t)$ and its neighbouring ones as follows

$$W_i(t+1) = \begin{cases} W_i(t) + \alpha(t)[X(t) - W_i(t)] & \text{for } \pi_i \in N_k(t) \\ W_i(t) & \text{otherwise} \end{cases} \tag{1.10}$$

where $N_k(t)$ represents the topological neighbourhood of the winner processor π_k at time t. The term $\alpha(t)$ is the *gain coefficient* or *gain term* at time t which assumes a small value ($0 < \alpha(t) < 1$). It is easy to see that as a result of the above update the weight vector $W_k(t)$ and its neighbouring weight vectors are pulled towards the input vector $X(t)$. The learning continues until the network is stable or some stopping criteria are met. Kohonen's algorithm is described in Algorithm SONN.

Algorithm SONN

Step 1: Decide the network topology.

Step 2: Initialize $t = 0$.
Choose $N_i(t)$ for all the processors and
initialize the weight vectors $W_i(t)$, $i = 1, 2, \ldots, n$
with random values.
Set the initial value of $\alpha(t)$.

Step 3: At t-th iteration, for an input pattern $X(t)$

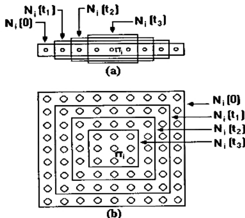


Figure 1.5: Varying neighbourhood over time $0 < t_1 < t_2 < t_3$ for the processor π_i in (a) one-dimensional (b) two-dimensional network.

(a) Select the winner processor by Eqn.(1.9).

(b) Update the weight vectors W_i's according to Eqn.(1.10).

Step 4: If the network is not stable, set $t = t + 1$ and go to Step 3.

Step 5: Stop.

By simulations it is observed that better results are obtained by the SONN model if the neighbourhood is fairly wide in the beginning and then it is shrunk over time (for example, as shown in Fig.1.5). The neighbourhoods $N_i(t_3)$, $N_i(t_2)$, $N_i(t_1)$ are called the *first, second, third order* neighbourhoods respectively and so on. The value of $\alpha(t)$ is also decreased to zero over time satisfying certain conditions similar to those imposed on stochastic approximation processes, that is:

$$\begin{array}{ll} (i) & \alpha(t) \rightarrow 0 \text{ as } t \rightarrow \infty \\ (ii) & \sum \alpha(t) = \infty \text{ and} \\ (iii) & \sum \alpha^2(t) < \infty \end{array} \qquad (1.11)$$

1.6 Characteristics of Kohonen's SONN model

The following characteristics of Kohonen's SONN model have been observed from a set of simulations. The characteristics, described below, are based on the generalization of some of the mathematical results available for very restricted cases and the supporting simulations.

(1) Even if the initial values of the weight vectors are arbitrary, the weight vectors eventually become ordered *topographically* in that the neighbouring processors are associated with weight vectors that are near each other in the input space (Fig.1.6). In other words, the weights are organized in such a fashion that the topologically close processors become sensitive to inputs that are physically close. Thus the SONN model maps input vectors from a high dimensional space onto the processors of the network (which is normally one- or two-dimensional only). For this reason, the SONN model is also called a *self-organizing feature map.* It is self-organizing in the sense that the weight vectors tend to approximate the input vectors, with the neighbourhood relation translating into proximity in Euclidean space although the initial neighbourhood relationships are chosen at random. This map which is *topologically correct* [3] has been observed in biological organisms. For example, auditory cortex of mammals receive vector signals, representing the frequency components, and they are topographically mapped onto different portions of the brain. The topographic ordering is explained in Fig.1.6. In Fig.1.6(a)-(b) both the input and the network are one-dimensional while in Fig.1.6(c)-(d) the input is from two dimensions and the network is one-dimensional.

(2) The SONN model performs one kind of *Vector Quantization* (VQ) of the input vectors. In vector quantization, the input space is to be divided into several regions where each region can be represented using a single vector called prototype vector or reference vector or codebook vector. It induces a partition $\{S_i\}$ on the input space S into n regions such that

$$S = \bigcup_{i=1}^{n} S_i \quad \text{and} \quad S_i \bigcap S_j = \phi \quad \text{for } i \neq j \tag{1.12}$$

[3] A mapping that preserves the topological order of the input

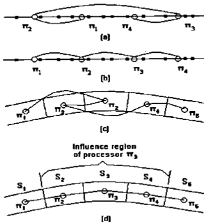


Figure 1.6: The weight vectors are represented by circles. (a) not ordered topographically, (b) ordered topographically. (c) not ordered topographically, (d) ordered topographically.

The set of prototype vectors is a compressed form of the information represented by the input data vectors since many different input vectors may be mapped onto the same prototype vector. It is found that the SONN model drives the weight vectors to the centroids of the respective regions S_i. Thus the weight vectors specify the prototype vectors. Unlike traditional vector quantization, however, the prototype vectors here are linked (Fig.1.7) according to the distance relationships among the regions of the input vectors that are represented by the weight vectors. In other words, the weight vectors are organized topographically.

(3) The formation of the above map is such that the probability distribution of the input vectors imprints on the final network. That is, the regions of the input space corresponding to frequent occurrences are mapped onto larger areas of the network than regions corresponding to rarer input vectors. In other words, the number of processors with weight vectors in a given region in the input space is approximately proportional to the number of input vectors in that region.

(4) Although the input vectors may come from a high dimensional space, it

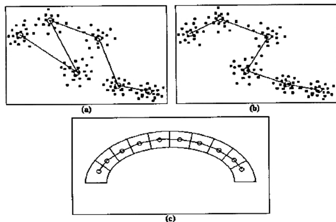


Figure 1.7: The cluster centroids (weight vectors) are shown by circles. (a) not ordered topographically, (b)-(c) ordered topographically.

is found that the feature mapping principle is able to pick up automatically two of the most important feature dimensions (when assuming a two-dimensional network), namely, those directions in which the density function of the input vectors has the highest variance. These two dimensions are used as the basis of the map. Kohonen conducted a simple experiment to demonstrate the phenomenon [64]. A linear (one-dimensional) network is taken and uniform distribution over a rectangular shape (two-dimensional) is considered for the input. The variances of the input vectors in the two directions are changed gradually — one is reduced while the other is increased. If the two variances significantly differ then the resulting network remains almost straight line oriented along the direction with the higher variance (Fig.1.8).

On the other hand, if the they are nearly equal then the output network takes a zigzag shape like a "Peano curve". Kohonen observed similar behaviour of the network when the number of processors in the network is quite high compared to the range of "lateral interactions" (Fig.1.9). The extent of zigzagness depends also on the ratio of the two variances. If the number of processors are increased, keeping the variances fixed, it is found that the asymptotic straight line form is achieved when the *influence regions* (see Fig.1.6(d)) are nearly square (excepting the end processors).

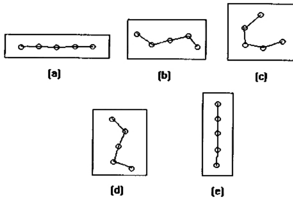


Figure 1.8: The orientation of the output network automatically changes to the direction of the higher variance.

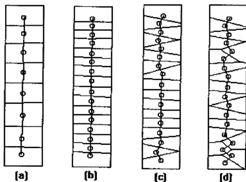


Figure 1.9: The output networks with different number of processors for the same input distribution.

1.7 Convergence

The above characteristics are not proved in general. That is, it is not established mathematically whether the above characteristics are correct if the input vectors are from two or more dimensions and the network has more than one dimension. The essential factor that leads to the above characteristics is achieving an ordered (topographically) mapping. It is not proved in general that the process will converge to such an ordering. However, for one-dimensional input and one-dimensional network, rigorous treatments are made by Kohonen [60], Ritter and Schulten [93], Erwin et al. [36]. Mathematical analysis of the SONN model, in general cases, seems to be very difficult and hence most of the authors working in this area have done it for the simplest case where both, the input and the network, are one-dimensional.

Kohonen [60], Ritter and Schulten [93], and others have shown that in one-dimensional space, if the neighbourhood relation satisfies certain properties, an ordered configuration is stationary and stable. Erwin et al. [36] have proved that this state is reached with probability 1 if the neighbourhood function is positive-valued, normalized and decreasing with distance, showing that there exists a sequence of input vectors that will lead the network into the ordered configuration. The authors consider the unit interval $[0, 1]$ and a one-dimensional network (linear chain) of processors and show that the learning process of the SONN model cannot be described by a gradient descent on a single energy function, but may be described by a set of potential functions, one for each processor, which are minimized independently by a stochastic gradient descent.

The execution of the SONN model can be viewed as a combination of two stages. The processors in the first stage are more "volatile" searching for niches to move into while in the second "sober" stage they settle into cluster centroids in the vicinity of the positions found in the first stage.

Despite the absence of rigorous theoretical proofs for higher dimensions, the SONN model has been a useful tool in several applications. In practice, it often converges to well ordered maps in high-dimensional problems yielding quite satisfactory results (see [79]).

1.8 Extensions of the SONN model

Several extensions of the SONN model have been proposed to achieve better solutions to various problems. The extensions proposed by different authors are outlined here.

Growing cell structures (GCS): Fritzke [39, 40] suggested *growing cell structures* in which the number of processors changes over time. A "signal counter" is associated with each processor. Apart from updating the weights, the counter is also adjusted for each processor. After updating the weights a fixed number of times, a new processor is added to the network to share the "load" of the processor with the largest counter value. On the other hand, processors whose counter values are too low are removed from the network. Thus in the GCS model, the network grows and shrinks repeatedly. The author demonstrated that the growing network is able to approximate probability distributions better than Kohonen's SONN model.

Selective multi-resolution (SMR): Sabourin and Mitiche [98] proposed this model which is based on Kohonen's SONN model but is "dynamic". It is not necessary to specify the number processors a priori. The SMR model attempts to overcome some of the shortcomings of the SONN model, namely, prior knowledge of the size of the network, data collisions (overloaded processors), large amount of processing time etc. SMR works in a hierarchical manner. More training efforts are given to that portion of the input in which data collisions occur. Moreover, the SMR model is claimed to be computationally more efficient than the SONN model.

Self-creating and organizing neural network (SCONN): Choi and Park [20] addressed a few defects of Kohonen's SONN model and proposed two models SCONN and SCONN2 to overcome them in a VQ problem. As addressed by Choi and Park, in the SONN model, if the input vectors have circular distribution, the processors near the centre may become dead. Second, if the input pattern have a complex structure, the network becomes unstable. Weight vectors in zero-density areas are affected by input vectors from all the surrounding parts of the nonzero distribution. As a result of residual effect from the rigid neighbourhoods, some weight vectors remain outliers. Third, the SONN model cannot overcome the "stability-plasticity" dilemma. It cannot adapt to new input regions. In SCONN model, a concept of

"activation level" for a processor is introduced. The level is decreased depending upon the time or the activation history. The model automatically decides whether to update the weights of the existing processors or to create a new processor. Initially the network has only one processor with small random weight and its activation level is set large enough to respond to any input. The network grows in size over time.

Neural gas network (NGN): Martinetz and Schulten introduced a topology-adaptive model called "neural gas network" (NGN) [72, 73, 77]. The model initially starts with a set of processors without any links. For each input vector, it ranks the weight vectors on the basis of their distances from the input vector and updates first few (say, r) of them. It updates r nearest weights where r has a large value initially which decreases over time according to a predefined schedule. For each input signal, the two nearest processors are joined by an edge. To delete an edge, an "edge aging" scheme is used. Fritzke [41] modified the NGN model and suggested how to insert processors in the network dynamically. Thus in the "growing neural gas network" (GNGN) model of Fritzke, the number of weight vectors to start with need not be known a priori. One starts with just two processors and then goes on growing the network by inserting new processors and by linking/delinking edges until certain stopping criteria occur. The models NGN and GNGN are found to be effective methods in topology learning. They have some advantages over Kohonen's SONN model because the SONN model assumes a predefined topology of the network that remains fixed during the learning. The major difference between the NGN and GNGN models is that the former starts with a given number of processors n, while the latter starts with only two processors and dynamically inserts new processors.

1.9 Scopes of the thesis

The problems, we consider in this thesis, are encountered in shape analysis and in optimization. These problems are quite well-known in their respective areas. The new self-organizing neural network models that will be proposed here are designed to solve these problems which will be described now.

p_8	p_1	p_5
p_4	p	p_2
p_7	p_3	p_6

Figure 1.10: The 4- and 8-neighbourhood of a digitized image.

1.9.1 Shape extraction

By *shape* we mean the shape of an *object* present in an *image* or stored in some other form. An image is a replica of an *object* or a *pattern* which can be created in two dimensions ($2D$) as in photographs. It can also be created in three dimensions using some sophisticated devices. But in the present discussions only two-dimensional images are considered. The light received from a scene by an optical system produces a two-dimensional image. This image can be converted into electrical signals by a sensor and subsequently can be stored in *digitized* form after proper *sampling* and *quantization* [95]. This digitization is essential for computational purposes. Hereafter by images we shall mean digitized images only. An image can be mathematically stated as a two-dimensional space mapped by a function $f(x, y)$ of two variables (co-ordinates in the image plane, corresponding to two orthogonal spatial directions). The $f(x, y)$ represents the brightness value of the picture at (x, y)th picture element (*pixel*, in short). In the black-and-white case, the values $f(x, y)$ are called *gray-levels* or *gray-scales*.

An image so formed is called *gray-level image* or *gray-scale image*. It has a range of gray-levels (usually 0–255) over the entire surface of the image. The gray-levels are non-negative since brightness cannot be negative; and are bounded since brightness cannot be arbitrarily high. They are zero outside a finite region so that the image is of finite size. We normally assume this region to be rectangular. Thus an image can be stored as a MxN matrix $I = (f(x, y) : x = 1, 2, \ldots, M; y = 1, 2, \ldots, N)$ where $x = 1, 2, \ldots, M$ and $y = 1, 2, \ldots, N$ represent the row and column indices of the matrix I.

After digitization, the problem is to segment [85] the image into its object and

p_8	p_1	p_5
p_4	p	p_2
p_7	p_3	p_6

Figure 1.10: The 4- and 8-neighbourhood of a digitized image.

1.9.1 Shape extraction

By *shape* we mean the shape of an *object* present in an *image* or stored in some other form. An image is a replica of an *object* or a *pattern* which can be created in two dimensions ($2D$) as in photographs. It can also be created in three dimensions using some sophisticated devices. But in the present discussions only two-dimensional images are considered. The light received from a scene by an optical system produces a two-dimensional image. This image can be converted into electrical signals by a sensor and subsequently can be stored in *digitized* form after proper *sampling* and *quantization* [95]. This digitization is essential for computational purposes. Hereafter by images we shall mean digitized images only. An image can be mathematically stated as a two-dimensional space mapped by a function $f(x, y)$ of two variables (co-ordinates in the image plane, corresponding to two orthogonal spatial directions). The $f(x, y)$ represents the brightness value of the picture at (x, y)th picture element (*pixel*, in short). In the black-and-white case, the values $f(x, y)$ are called *gray-levels* or *gray-scales.*

An image so formed is called *gray-level image* or *gray-scale image.* It has a range of gray-levels (usually 0—255) over the entire surface of the image. The gray-levels are non-negative since brightness cannot be negative; and are bounded since brightness cannot be arbitrarily high. They are zero outside a finite region so that the image is of finite size. We normally assume this region to be rectangular. Thus an image can be stored as a MxN matrix $I = (f(x, y) : x = 1, 2, \dots, M; y = 1, 2, \dots, N)$ where $x = 1, 2, \dots, M$ and $y = 1, 2, \dots, N$ represent the row and column indices of the matrix I.

After digitization, the problem is to segment [85] the image into its object and

non-object (background) parts. Many applications need the segmentation to be done into two levels only - one for the object and the other for the background. For example, in character recognition problems the character patterns are to be separated from the background; in fingerprint analysis only the fingerprint ridges are required. An image having only two levels are called *binary image* or *two-tone image* [17]. In our present discussion, these two levels are: '1' or 'black' for object and '0' or 'white' for background. An example of binary image is shown in Fig.1.11(a).

Definition 1.1. Within a 3x3 window of a pixel p (Fig.1.10), the pixels p_1, p_2, p_3, p_4 are called *4-neighbours* of p and $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8$ are called *8-neighbours* of p.

Shape of an object

In many computer vision and pattern recognition/classification applications the *shape* of the object in a given image is to be analyzed. Shape extraction of an object which plays an important role in such areas, has received considerable attention from research end. There can be two approaches to defining the shape of an object: *boundary-based* approach and *region-based* approach. The former defines shapes by extracting the features based on the object boundaries. For example, edge detection in an image [12, 28, 71]; B-spline curve fitting to an image [46]; fitting a polygon to the boundary points of a planar set for computation of convex-hull or alpha-hull [34, 90] etc. On the other hand, region-based approach defines the shape of an object based on its geometrical structure and topological relationships. By this approach, an object having some definite elongated shape can have a piecewise linear or curvilinear representation — known as *skeleton*.

In the latter approach, the objects in an image can be characterized satisfactorily by structures composed of lines or arc patterns. Common examples are hand-written or printed characters, fingerprint ridges, engineering drawings, chromosomes etc. The thickness of such patterns does not contribute to the recognition process and hence is redundant. A suitable transformation of the pattern is required to arrive at their skeletal structure that contains the essential topological and geometrical information of the input pattern. Apart from shape representation of a pattern for the

ease of recognition, a skeleton serves other purposes too. Skeletons provide a large reduction in volume of data i.e., data compression. Transforming patterns to thin-line representation can reduce the amount of data as well as further shape analysis can be done more easily. In an OCR application, to an extent, it provides a unification of character shapes by reducing the effects of varying type fonts and thus simplifies the feature selection and feature extraction tasks.

Both the above approaches to shape extraction, boundary-based and region-based, are useful for pattern representation and recognition. The *skeletal shape* is appropriate for applications where the input pattern is line-like or ribbon-like or tree-like (i.e., elongated type where the length is considerably higher than the width) and in particular where human perception of a shape is in terms of lines and strokes. On the other hand, for a square-like object where the length and the width are more or less same, a skeletal shape is not very meaningful and is hardly of any practical use. The boundary-based approach is appropriate for such objects to describe the shape.

In the next three chapters we are concerned with the region-based shape only. **Henceforth, by shape we shall mean the skeletal shape**. The process which extracts this shape from a digitized image is commonly known as *skeletonization* or *thinning*. Most of these techniques deal with binary images. The definitions, theories and algorithms are mainly based on binary images. The primary objective of this process is to find an approximation of *medial axis* that will be defined shortly.

Although the necessity of a skeleton initially arose from character recognition, in later days skeleton have been used to a great variety of patterns for different purposes. In biomedical field, skeletons have been used to count and quantify the constituent parts of white blood cells to classify abnormal cells [55, 91]. They have also been used in analysis of chromosomes [52] and X-ray images [92]. Skeletons have also been useful in fingerprint classification [81], measurement of soil cracking [84], visual inspection of industrial parts [82], printed circuit boards [114] and in several other application areas. Due to its wide applicability, skeletal shape extraction algorithms have been a very active research area [67, 107].

Medial axis transformation (MAT)

Blum [11] introduced the skeleton of a continuous binary image and called it medial axis. His procedure for finding the medial axis was to set up "grassfire" which can be described as follows. Assume that the object region is filled with dry grass and fire is set everywhere on the boundary simultaneously. As the fire burns the grass, the object becomes thinner. Suppose that the fire line propagates with constant speed from the boundary of the object towards its inside. Then all the points lying in positions where at least two wavefronts of the fire line meet during the propagation (quench points) will form the medial axis of the object. The formal definition of MAT, in continuous case, can be given by:

Definition 1.2. For each point X of the object, find its nearest boundary point. If X has more than one such (that is, of the same distance) boundary points then it is said to belong to the *medial axis* of the object.

An alternative definition of medial axis has been given as:

Definition 1.3. Consider the circles that totally lie within the object and touches the object boundary at two or more points. The centres of all such circles form the *medial axis* of the object.

The digital (discrete) implementation of the above transformation is a challenging problem. To illustrate this, let us consider Fig.1.11(a) which is a binary image of the alphabet "R".

The objective of skeletonization is to obtain one of the representations shown in Figs.1.11(b)-(d). In Fig.1.11(b) the pattern is still an image, where the width is reduced to just one everywhere. Such a skeleton is called a *raster skeleton.* Only the skeletal pixels, instead of all the object pixels, are stored to represent the object. Fig.1.11(c) gives a piecewise linear representation (or a *vector skeleton*) of the input pattern. Fig.1.11(d) lacks some more details and provides a graphical representation

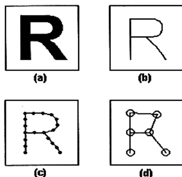


Figure 1.11: (a) Binary image (b) raster skeleton (c) vector skeleton (d) connection graph.

what can be called a *connection graph* of the input pattern. For Fig.1.11(c) and (d) all that are stored are the co-ordinates of the circled points and their linkage information. This can be done by a network data structure. Most of the thinning algorithms create outputs similar to Fig.1.11(b). Figs.1.11(c) and (d), if required, can be obtained subsequently from this. Some algorithms of skeletonization derive skeletons similar to what is shown in Figs.1.11(c)-(d). It can be seen that data reduction is more in Fig.1.11(c) than in Fig.1.11(b) and the data reduction is more in Fig.1.11(d) than in Fig.1.11(c).

Several articles have been published on various aspects of this problem since its inception. Classified surveys in this area can be found in [67] and [107]. Most of the articles are on binary images. The existing algorithms can be classified as sequential algorithms and parallel algorithms. Sequential algorithms (for example, [9, 94, 108]; also see [67]) operate on a single pixel at a time while parallel algorithms (for example, [18, 19, 21, 49, 53, 57, 112, 115]; also see [67]) operate on a set of pixels simultaneously. Parallel algorithms can either use multiple passes or only one pass in each iteration. A large number of thinning algorithms iteratively remove outside layer of pixels from the object to arrive at an one-pixel thick skeleton.

In our discussion, we primarily deal with binary images. For other types of input, it will be mentioned separately. **The shape extraction process here will**

mean the following. Decompose the object into several smaller regions or blocks. Then compute the centres of gravity of all these regions and build up links between the centres of adjacent regions. Similar concepts have been used by several researchers [37, 88, 108]. The difference between these techniques and our technique is that they explicitly compute these regions and subsequently join them by edges while in our approach these regions are formed adaptively and the final regions and the links describe the output shape of the object. In the existing literature, the terms "skeletonization" and "thinning" are almost synonymous. But in the present thesis, they are different. Determining the centres of several decomposed blocks and joining them by their adjacency to get the vector skeleton is skeletonization and reducing the object by removal of boundary pixels (outer layer) to a one-pixel thick raster skeleton is thinning. In Chapters 2-3, we have dealt with the design of self-organizing neural network models for the skeletonization problem. We have carried out a comprehensive comparative study between our neural network based methods and some of the existing well-cited thinning methods in Chapter 4.

1.9.2 Convex hull

Another form of an object or a pattern (apart from an image) we are concerned with is a finite set of points in the Euclidean plane. We also call this a *dot pattern* or a *planar set*. In this case, instead of storing the object in a matrix as in the case of an image, the planar co-ordinates of the object points are stored. Thus a planar set is : $S = \{(x, y) : x \text{ and } y \text{ are real numbers}\}$. In our discussion (without loss of generality) we assume x and y are positive. For example, a telescopic photograph of a galaxy where each star/planet appears as a dot forms a dot pattern. A dot pattern can also originate from a binary image when it is highly corrupted by noise.

One problem, well studied in computational geometry, is the computation of the *convex-hull* of a given planar set. The convex-hull of a given set of points is defined as the smallest convex polygon containing all the points in the set. The concept of convex-hull is quite natural and easy to understand. It can be imagined as a stretched rubber band surrounding the set of points and then being released to shrink.

The convex-hull has wide applications in pattern recognition, image processing, statistics, cluster analysis, operations research, robust estimation, computer graphics, robotics and several other areas. In Chapter 5, we formulate the convex-hull problem as a self-organizing neural network problem. We describe how a network orders itself iteratively on the basis of the input points and self-organizes accordingly to finally arrive at the convex-hull after a finite number of iterations.

1.9.3 Minimum spanning circle

Another computational geometry problem that has drawn considerable attention of many researchers is the computation of the circular hull of a planar set. While the convex-hull provides the smallest convex polygon enclosing the set of input points, the circular hull provides the smallest circle that encloses all the input points. The circular hull is also called the *minimum spanning circle* (MSC).

The application of the MSC lies in optimization, pattern recognition, image analysis, statistical estimation etc. It has applications in transmission and transportation problems, for example, in optimum location of a facility. We can find out where should a facility be located so as to minimize the maximum distance from the facility to a user.

In Chapter 6, we describe a self-organizing neural network model that computes the MSC of a given planar set . The problem is to find the radius and the centre of the smallest circle such that the circle encloses n given points in the plane. In location theory this is the unweighted Euclidean 1-centre problem in the plane.

1.10 Conclusions

Artificial neural networks (ANN) are man-made models of biological neural systems. They are composed of a number of processors which are interconnected by links. The links are associated with connection weights which are adapted through a learning process. The characteristics of the processors, the connections, the learning rules etc.

characterize an ANN model. ANN models, instead of performing operations sequentially, perform the same simultaneously using massively parallel networks composed of a number of processors connected by links, and thus provide high computational speeds. Neural networks are also popular due to their adaptive nature. They provide a new form of parallel computing in an adaptive manner as observed in living organisms. By using neural networks, a complex problem can be solved with a number of processors working collectively, while each processor performs much simpler computations only. Neural network models sometimes provide robustness to some extent (see [69, 100]). If a few processors or links are damaged, the overall performance is not affected much and thus such models sometimes have higher fault tolerance. Moreover, in some situations where conventional computing poses problems or performs poorly, neurocomputing provides an efficient way of solving them.

Depending on the learning type (supervised or unsupervised) the ANN models can be divided into two categories – supervised network models and unsupervised network models. The theme of the present thesis is unsupervised neural networks which are also called self-organizing networks. In unsupervised models, the training is performed without any target vector unlike in supervised models where for each input vector a target vector is also required to train the network.

Some interesting characteristics of Kohonen's self-organizing model are discussed. Quite a few extensions of this model are developed in the recent past which are described here. These models are found to be more efficient in some respects. An important property of these extensions is that they are dynamic in nature and has the capability to grow over time. As a result, these models need not know the network size a priori unlike in Kohonen's model.

The requirement of prior knowledge of the number of processors in the network, as has been pointed out by several authors, poses problems in applying the SONN model to different situations. These problems have been overcome by dynamically growing and shrinking the network by addition and deletion of processors in the network. The addition and deletion criteria are set depending upon the application in hand. In Chapters 2-3, we propose new self-organizing models in shape extraction problems where not only the network grows in size but also the local topology evolves

over time. The former property is required to automatically select the optimal number of processors in the network while the latter is required to adapt the structure of the input pattern. A predefined rigid topology of the network, as is used in the SONN model, cannot properly adapt the structure of the given pattern. In Chapter 4, we make a comparative study between our neural network based models with some of the existing well-cited conventional algorithms. In the next two chapters, we shall deal with two more problems that are frequently studied in computational geometry. Chapter 5 proposes a self-organizing neural network model to compute the convex-hull of a planar set. In Chapter 6, we propose a self-organizing model for finding the minimum spanning circle of a planar set.

Chapter 2

A DYNAMIC SELF-ORGANIZING NEURAL NETWORK FOR SHAPE EXTRACTION

Abstract: *A modified version of Kohonen's self-organizing model is proposed and is applied on shape extraction of an object in a binary image. The networks can grow itself to an optimum size. It can also adapt the local topology of the input pattern by certain suitably chosen criteria.*

2.1 Introduction

There are a few shortcomings of the SONN model, discussed in Chapter 1, in view of the shape extraction of a pattern. In Kohonen's SONN model, a network having either a *linear topology* or a *planar topology* is used. As mentioned in Chapter 1, such rigid neighbourhood topologies are found inadequate in some situations. In particular, they pose problems in shape extraction of a pattern. When the input pattern has a prominent shape such neighbourhood definitions are found unsuitable

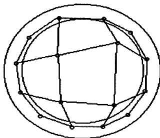


Figure 2.1: The output network by the SONN model for a pattern having a circular shape.

[59]. This is due to the fact that during the update process the weight vectors lying in zero-density areas are affected by input vectors from the surrounding parts of the nonzero distribution. As the neighbourhoods are shrunk the fluctuation vanishes and as a result some processors may remain outliers due to the residual effect from the rigid neighbourhood (Fig.2.1). On the contrary, in a shape extraction problem, the output network is not only required to be within the pattern but should also be satisfactorily close to the medial axis of the pattern. Figs.1.8(b)-(d) illustrate some situations when the output networks do not represent the skeletal shapes.

Another problem with the SONN model is that it assumes a predefined fixed topology of the network which is maintained throughout. But at different regions of the input pattern, we may require different topology of the network. Different regions of the pattern may have different structures namely, a simple arc, a fork, a crossing etc. This is illustrated in Fig.2.2 where a linear chain of processors fails to adapt the skeletal shape of circular and fork structures. For the circular pattern, the topology of the pattern is no longer preserved by the output network (the pattern is a closed loop while the output network is not). The fork structure cannot be adapted by the linear network structure. To adapt the fork structure, the network requires one processor to have three neighbours.

Due to the above shortcomings, Kohonen's SONN model is modified here to make it applicable for shape extraction. The size of the network (number of processors

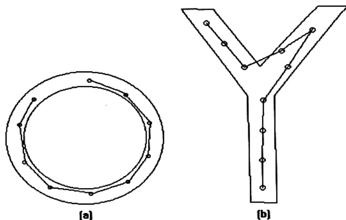


Figure 2.2: The problems of the SONN model due to its fixed network topology. (a) circular shape (b) fork shape.

in the network) is allowed to vary during learning unlike in the SONN model where it is set a priori and remains fixed throughout. The proposed model is called *Dynamic Self-Organizing Neural Network* (DySONN) [24] model. Apart from growing in size the DySONN model can be distinguished from the SONN model by the property that in the DySONN model the network can adapt the structure of the input pattern even when the input pattern has a nonlinear structure (for example, patterns having forks, crossings, loops etc.). As a result, the DySONN model overcomes the problems inherent in the SONN model while extracting the shape of a pattern.

In the neural network model discussed here, the connections and the update rules are similar to those in Kohonen's self-organizing feature map. But while the number of processors in Kohonen's feature map is fixed, the present network is dynamic in the sense that during the learning process new processors can be added to or old processors can be deleted from the network. A major advantage of such a dynamic model is that the number of processors need not be known a priori, but can be learnt. On the other hand, a model with a fixed number of processors can cause some processors to be overloaded and some processors to remain idle (see [98]). Dynamic neural networks have been used by several researchers in various applica-

tions, for example, in modelling probability distribution in the plane [39], in shape classification [98], in vector quantization [20], in convex-hull computation [22] and in shape approximation [87]. In the present chapter, a dynamic neural network model is proposed for extraction of the shape of a binary object, in the form of a skeleton.

In his model Kohonen has used the term array of processors to represent the network. Since in our model the processors are inserted/deleted during the learning process, we use the term "list of processors" to represent the network structure. In the list, each node represents a processor and the links represent the connections between processors. This makes the insertion/deletion of processors meaningful and we can follow the standard insertion/deletion algorithms available in the list processing literature. In our discussion, by network structure we mean a (linked) list of processors having linear or nonlinear structure.

We first describe our shape extraction method for patterns of simple structures and then go for structures with more complexity. In the next section we have three subsections. Subsection 2.2.1 describes the method for simple patterns like arcs. Subsection 2.2.2 explains how to take care of more complicated patterns having branching, forks and crossing (for convenience we call them together tree patterns). Subsection 2.2.3 considers patterns that contain loop structures. In all the cases the starting structures of the network is linear only. It is the dynamic nature of the network that enables it to learn the structure of the input pattern and to expand (topologically) accordingly.

2.2 The DySONN model

From Chapter 1 we learn that Kohonen's feature mapping network is an array of processors where each processor is connected to one or more surrounding processors and every processor is assigned a weight vector. The network of processors is normally represented in either one or two dimensions. The map (more specifically, the weight vectors) is adapted on the basis of a set of input feature vectors which can be of an arbitrary dimension. The dimension of the weight vectors is the same as that of the input vectors. Suppose the array of processors under consideration is represented

as $\{\pi_1, \pi_2, \ldots, \pi_n\}$. Denote the *neighbourhood* N_i of the processor π_i by $\{\pi_p : \pi_p$ is connected to $\pi_i\}$ which includes π_i. Let the weight vector for the processor π_i be $W_i(t) = (w_{i1}(t), w_{i2}(t), \ldots, w_{im}(t))$ at time instance t. The starting weight vectors $W_i(0)$ are chosen at random. Suppose the set of input vectors is $S = \{P_1, P_2, \ldots, P_N\}$ where the dimension of each P_j is also m. The update rules for the weight vectors then go as follows.

At time instance t, P_j is presented to the network. All the processors compete and let $W_k(t)$ be the nearest weight vector to P_j. That is,

$$||W_k(t) - P_j|| \leq ||W_i(t) - P_j|| \quad \text{for all } i \tag{2.1}$$

The weight vectors of the processor π_k and its neighbours are updated [1] as:

$$\begin{aligned} W_k(t+1) &= W_k(t) + \alpha_1(t)[P_j - W_k(t)] \\ W_p(t+1) &= W_p(t) + \alpha_2(t)[P_j - W_p(t)] \quad \text{for } \pi_p \in N_k - \{\pi_k\} \end{aligned} \tag{2.2}$$

where $\alpha_1(t)$ and $\alpha_2(t)$ ($\alpha_1(t) \geq \alpha_2(t)$ for all t) are the gain terms at time t satisfying the conditions (1.11) in Chapter 1.

We have classified the input binary patterns into three categories: 1) Arc patterns like character patterns 'C', 'L', 'S', 'M' etc. which have a linear structure; 2) Tree patterns like 'T', 'X', 'Y' etc. which have forks and branchings; 3) Loop patterns like 'A', 'B', 'O' etc. which contain a loop structure. To describe our model, we will deal with these categories separately. In the figures, the object pixels are coloured as black. The networks are represented by tiny circles and lines denoting processors (nodes) and links (edges) respectively. The circles and lines are black in white background; and white in black background.

2.2.1 Arc patterns

We first deal with input patterns having an arc shape, the structure of which can be represented by a linear structure. We start with a network having a linear structure

[1] Since the gain term may be different for the winner processor and its neighbours, we break Kohonen's update rule into two using $\alpha_1(t)$ and $\alpha_2(t)$

represented by a list of processors $[\pi_1, \pi_2, \ldots, \pi_n]$ where π_i is connected to exactly two processors π_{i-1} and π_{i+1} (the two end processors are connected to exactly one processor each). Here the input feature vectors are the co-ordinates (row and column indices) of the object pixels in an image and hence $m = 2$. $S = \{P_1, P_2, \ldots, P_N\}$ is the set of N object pixels where $P_j = (x_j, y_j)$. The weight vectors of the processors π_i are updated iteratively on the basis of the points in S. The initial weight vectors of π_i are, say, $(w_{i1}(0), w_{i2}(0))$. Suppose, the point P_j is presented at the t-th iteration. Let

$$||W_k(t) - P_j|| \leq ||W_i(t) - P_j|| \quad \text{for all } i \tag{2.3}$$

where $W_i(t)$ is the i-th weight vector at the t-th iteration. Then P_j updates the weight vectors in the following way.

$$\begin{aligned} W_p(t+1) &= W_p(t) + \alpha_1(t)[P_j - W_p(t)] \quad \text{for } p = k \\ W_p(t+1) &= W_p(t) + \alpha_2(t)[P_j - W_p(t)] \quad \text{for } p = k-1,\ k+1 \end{aligned} \tag{2.4}$$

If this update continues then the weights tend to approximate the distribution of input vectors in an orderly fashion. Note that the processors do not move physically during update. It is the weight vectors, representing the locations of the processors, that are made to change to define the ordering. One presentation each of all the points in S makes one *sweep* consisting of N iterations. After one sweep is completed, the iterative process for the next sweep starts again from P_1 through P_N. Several sweeps make one *phase*. One phase is completed when the weight vectors of the current set of processors converge (are stable), that is, when

$$||W_i(t) - W_i(t')|| < \varepsilon \quad \text{for all } i \tag{2.5}$$

where t and t' are the iteration numbers at the end of two consecutive sweeps and ε is a predetermined small positive quantity. Note that one phase here can be looked upon as one complete convergence of the SONN model. Only after a phase is completed, are processors inserted. Suppose, at the end of the s-th phase, the weight vectors of the processors are $W_1(t_s), \ldots, W_{n(s)}(t_s)$ where $n(s)$ is the number of processors during the s-th phase and t_s is the total number of iterations needed to reach the end of the s-th phase. If the weight vectors of two neighbouring processors are far apart, a processor is inserted between them.

Processor insertion

If

$$||W_l(t_s) - W_{l+1}(t_s)|| = \max_{i=1,\dots,n(s)-1} ||W_i(t_s) - W_{i+1}(t_s)|| > \delta \tag{2.6}$$

then one processor is inserted between π_l and π_{l+1} and the new processor has the weight vector as $\frac{1}{2}[W_l(t_s) + W_{l+1}(t_s)]$.

Note that δ is a predetermined positive quantity. After the insertion of a processor, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$||W_i(t_s) - W_{i+1}(t_s)|| \le \delta \quad \text{for} \quad i = 1, 2, \dots, n(s) - 1 \tag{2.7}$$

The condition (2.7) means that the weight vectors of no two neighbouring processors are too far apart. The processors (on the basis of their weight vectors) at this stage give an approximate global shape of the input pattern (Fig.2.3).

2.2.2 Tree patterns

Now we deal with patterns that have branchings, forks or crossings (we term these features as *junctures*). For example, consider character patterns 'T', 'X', 'Y'. Here it is required that a processor in the network has more than two neighbours. In the case of arc patterns the number of neighbours (call it *degree* of the processor) for each processor was known and was fixed during the learning process. But now a processor (representing a juncture in the pattern) can have a variable number of neighbours and the number is not known a priori. We shall discuss how to learn the required degree of the processors.

Let us consider a pattern with a fork. Since initially there is no topological information about the pattern, we start with a linear network (the same network as in Fig.2.3(a)). Adapting a fork, present in a pattern, is explained in Fig.2.4. After a number of iterations, some processor forms a significantly small acute angle (decided

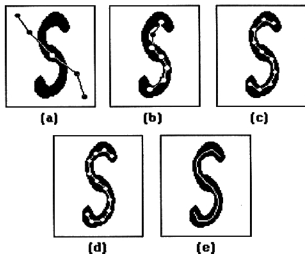


Figure 2.3: Different stages of the network for an arc-pattern 'S' ($\delta = 12$). (a) Initial network. Intermediate stages (b) after 20 sweeps (c) after 40 sweeps. (d) Final network giving the vector skeleton after 60 sweeps and (e) the raster skeleton.

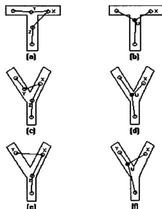


Figure 2.4: (a), (c), (e) : the network locally forms a spike to indicate one of the branchings, (b), (d), (f) : a processor U with higher degree is inserted to accommodate the branching.

on the basis of some threshold say, θ) with its two neighbours (Figs.2.4(a),(c),(e)) to indicate a fork in the pattern. Such a spike is formed because, by a property of Kohonen's feature map, the network tries to span the entire range of input pattern space and also, the topological relationship of the pattern is preserved in the network (as discussed in Chapter 1). Thus, formation of a spike suggests that a fork is to be created in the network. In Figs.2.4(a),(c),(e), processor X forms a spike with its neighbouring processors Y and Z indicating a fork lying between Y and Z. Therefore, the following procedure is executed to accommodate a fork in the network (Figs.2.4(b),(d),(f)) when a processor X forms a spike:

Procedure Create-fork

(a) Create a new processor say, U (denoted by a solid circle) halfway between Y and Z.

(b) Establish a link between U and X.
If Y has neighbour(s) other than X (Figs.2.4(a),(c)) then establish link(s) between U and these neighbour(s) of Y and delete processor Y (Figs.2.4(b),(d)); otherwise (Fig.2.4(e)) establish link between U and Y (Fig.2.4(f)). Take similar action for processor Z.
(deleting a processor here means all the links associated with it are also removed).

The above procedure can be justified as follows. Since there is no prior knowledge about the structure of the input pattern, we start with the simplest network structure (here linear). As the network tends to the input, at a point of time when we encounter a spike, we update our knowledge by creating a fork structure locally in the network. The same actions are taken for all the processors forming a spike. These actions are taken after a phase is complete and then the subsequent phases of learning are continued to enable the network to approach towards a closer approximation of the shape of the pattern. Similar principles used earlier in the case of arc patterns are followed for insertion of processors and for convergence of the algorithm. For this the conditions (2.6) and (2.7) are modified to accommodate all possible pairs

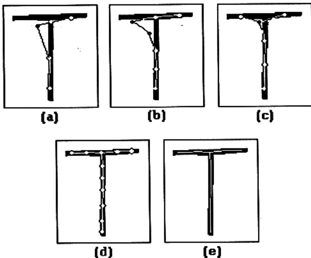


Figure 2.5: Different stages of the network for the pattern 'T' ($\delta = 12$). Intermediate stages (a) after 20 sweeps (b) after 40 sweeps (c) after 50 sweeps. (d) Final vector skeleton after 70 sweeps and (e) the raster skeleton.

of neighbouring processors as follows.

Processor insertion

If

$$||W_l(t_s) - W_{l'}(t_s)|| = \max_{i=1,\ldots,n(s)} \max_{\pi_{i'} \in N_i - \{\pi_i\}} ||W_i(t_s) - W_{i'}(t_s)|| > \delta \qquad (2.8)$$

then one processor is inserted between π_l and $\pi_{l'}$ and the new processor has the weight vector as $\frac{1}{2}[W_l(t_s) + W_{l'}(t_s)]$.

After the insertion, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$\text{for all } i, \qquad ||W_i(t_s) - W_{i'}(t_s)|| \leq \delta, \;\; \forall\, \pi_{i'} \in N_i - \{\pi_i\} \qquad (2.9)$$

Different stages for a 'T'-pattern are shown in Fig.2.5. The initial network was taken as in Fig.2.3(a).

2.2.3 Loop patterns

The techniques discussed above work for patterns excepting those containing loops. Consider the pattern 'A'. Our algorithm can generate, on the basis of the principles discussed in Sections 2.2.1 and 2.2.2, an incomplete skeleton as shown in Fig.2.6(c). It is now necessary to complete the loop by means of bridging the gap (between processors E and F.

As discussed in Chapter 1, the asymptotic values of the weight vectors constitute some kind of vector quantization [64]. In particular, the distance measure and the update rules as considered in our algorithm, induce a partition of the input pattern space specified as

$$S_i = \{P_j \in S : ||W_i - P_j|| \leq ||W_r - P_j|| \quad \forall r\} \tag{2.10}$$

The above partition is a Voronoi tessellation which, in the present situation, means partitioning of the input pattern space into regions within each of which all input vectors have the same weight vector as their nearest one. Therefore, each set S_i is associated with a single processor. Hence the input pattern vectors can easily be labelled according to the S_i to which it belongs. In other words, each input vector is given a label according to its nearest processor. Such input labeling has earlier been used by Sabourin and Mitiche [98].

Definition 2.1: When the input pattern is a binary image, two processors π_i and π_j $(i \neq j)$ are said to be *adjacent* if there exists at least one pair of object pixels $P \in S_i$ and $Q \in S_j$ such that P and Q are 8-neighbours (see Definition 1.1) to each other.

Procedure Loop-join

After convergence (let us call it initial convergence) as mentioned in Sections 2.2.1 and 2.2.2, label the input vectors as mentioned above and then check, for each processor, whether it is adjacent to any processor other than its neighbours. If it is, introduce a link between these two processors.

In Fig.2.6(c), processor E is adjacent to processor F and they are not neighbours

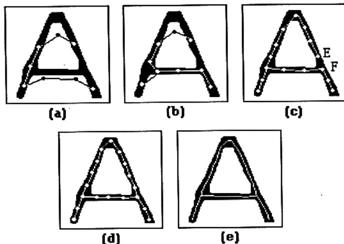


Figure 2.6: Different stages of the network for the pattern 'A' ($\delta = 12$). Intermediate stages (a) after 20 sweeps (b) after 40 sweeps (c) after 70 sweeps. (d) Final vector skeleton after 100 sweeps and (e) the raster skeleton.

to each other. So, they are made connected and become neighbours. After this we continue the algorithm until the final convergence is reached (Fig.2.6(d)), that is, when condition (2.9) is satisfied. The whole process can now be stated briefly by Algorithm DySONN.

Algorithm DySONN

Step 1: [Initialization]

Initialize t = 0; Initialize the weight vectors $W_i(t), (i = 1, 2, \ldots, n)$ with random values.

Step 2: [Sweep]

For all input patterns $P_j, j = 1, 2, \ldots, N$. Update weight vectors according to rules (2.2).

Step 3: [Phase]

If condition (2.5) is false then go to Step 2.

Step 4: [End of Phase]

If no processor forms a spike go to Step 6.

Step 5: [Fork Creation]

Create new processor U by the procedure *Create-fork*.

Go to Step 2.

Step 6: If condition (2.9) is true go to Step 8.

Step 7: [Processor Insertion]

Insert processor according to condition (2.8).

Go to Step 2.

Step 8: [Loop Join]

Label the input vectors as mentioned above. If no processor is adjacent to any processor other than its neighbour go to Step 10.

Step 9: Join the processor and the processor adjacent (by Definition 2.1) to it by the procedure *Loop-join*.

Go to Step 2.

Step 10: Stop.

The final network obtained by the above algorithm gives a vector skeleton for the given input pattern (see Figs.2.3(d), 2.5(d), 2.6(d)). The raster skeleton can easily be derived from the network as follows (see Figs.2.3(e), 2.5(e), 2.6(e)).

Procedure Raster-skel

For each link, the line segment connecting the weight vectors of the two corresponding processors, is considered. The set of all object pixels intersecting such a line segment gives the raster skeleton.

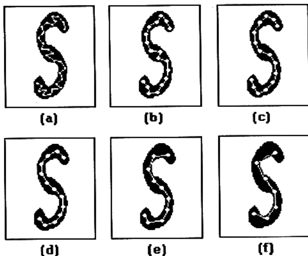


Figure 2.7: The role of δ on the output skeleton. (a) δ=5 (b) δ=7 (c) δ=9 (d) δ=11 (e) δ=13 (f) δ=15.

2.3 Choice of parameters

2.3.1 Choice of δ

It is easy to see that δ plays a role here. A very low value of δ might produce a zigzag network (Figs.2.7(a)-(b)) which does not represent the true shape of the pattern. (The same problem occurs in the SONN model which is discussed in Section 1.6). On the other hand, if δ is very high, the skeleton does not properly represent the medial axis (Figs.2.7(e)-(f)). In the latter case, a portion of the output skeleton may lie outside the object. In this example, it can be seen from the figure that the optimum values of δ are $9 \leq \delta \leq 11$ in view of the medial axis representation. But, in this case, the choice of the optimum value needs user intervention. We now suggest an adaptive mechanism so that a satisfactory medial axis representation can be achieved automatically. One such mechanism is to introduce *activation level* in the weight updating process which is described below.

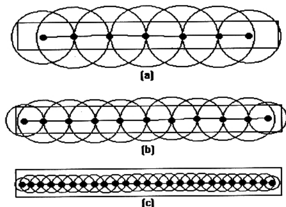


Figure 2.8: The activation regions at different time $t = t_1,\ t_2,\ t_3\ \ (t_1 < t_2 < t_3)$. (a) at time t_1 (b) at time t_2 (c) at time t_3.

We define an *activation region* of a processor so that if an input vector falls within the region then only it activates the processor. The activation region decreases over time. In the present problem it is defined as follows.

Definition 2.2: The *activation level* $a_i(s)$ of ith processor, for $i = 1, \ldots, n(s)$, at the end of the s-th phase, are defined as

$$a_i(s) = \frac{1}{c_i} \sum_{\pi_{i'} \in N_i - \{\pi_i\}} ||W_i(t_s) - W_{i'}(t_s)|| \tag{2.11}$$

where c_i is the number of neighbours of π_i excluding π_i.

Definition 2.3: The *activation region* of a processor is a circle with the corresponding weight vector as the centre and the activation level as its radius (see Fig.2.8).

A processor is called *active* if the presented input vector lies within its activation region. In other words, if an input vector lies outside all the activation regions, it is ignored in the competition and updating process (Eqns. (2.1), (2.2)). An input must activate a processor first before entering into the competition. Only active processors are qualified for competition, after which the winner processor is selected and the

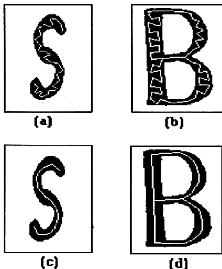


Figure 2.9: The raster skeletons obtained (a),(b) without activation level and (c),(d) with activation level (δ=5).

weight vectors are updated accordingly. Thus, the weak signals (here the object pixels near the boundary of the object) gradually become ineffective in the weight updating process. This is so because, as the processors become more and more close to each other, the region of input vectors that activate (influence) a processor is also shrunk and thereby the influence of the outer layers is symmetrically decreased. The object pixels near the medial axis acquire more control over the process (Fig.2.8(c)). There may be other ways to define the activation region. But the proposed method does not require any parameter setting and is data-driven.

The algorithm described earlier is modified to incorporate the activation level. Results of the modified algorithm are shown in Fig.2.9. It is found that the modified algorithm gives much better results. In the original algorithm if δ takes small value (say, δ=5) then the skeletons become zigzag (Figs.2.9(a),(b)). But in the modified algorithm the skeletons do not become so with the same δ (Figs.2.9(c),(d)). Thus we can always set δ very low and by introducing the activation level, a satisfactory

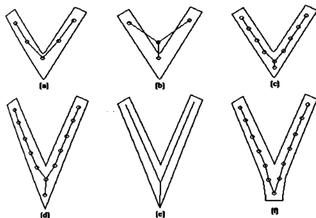


Figure 2.10: The role of parameter θ.

medial axis representation can be obtained.

Note that a smaller value of δ will take longer time to converge. However, in some applications, a crude approximation of the shape serves the purpose. In such situations, we can choose a high value of this parameter and get the output faster. Thus the user has an option to make the algorithm faster at the cost of accuracy. This issue is discussed in more detail later in Section 2.4.

2.3.2 Choice of θ

The procedure *Create-fork* uses a threshold angle θ to judge whether an angle is acute enough to create a fork in the network. The value of θ should not be too high or too low. If it is very high, a fork will be formed at an early stage which should not have been created at all (Fig.2.10(b)). It might cause a spurious tail in the output skeleton (Fig.2.10(c)). However, this tail may be removed by some simple post-processing if it is not long enough. This tail will be long only when the input pattern has a very sharp peak (Fig.2.10(d)). In such situations, existing conventional thinning algorithms also create a tail (Fig.2.10(e)).

On the other hand, if the value of θ is very small, the fork creation takes more time. Due to a very small θ, a small branch of the pattern may be missed in the output skeleton (Fig.2.10(f)). But if the the branch is long then in course of processor insertion the angle will become more acute and at one time it will be less than θ and hence a fork will be created.

Experimental results have shown that the proposed model works well and produces satisfactory results for a wide range of the parameter θ. The final result is not affected by the choice of θ within this range. We have tested on a good number of English character patterns keeping $30° < \theta < 80°$ and have observed hardly any effect of θ on the final skeleton.

2.4 Discussions and remarks

For arc patterns, it can easily be seen that the resulting network, after convergence, gives a skeletal shape of the pattern. Here the array of processors is linear and the inputs are from a two- dimensional distribution (see [64], p.153). In the present model, we start with a given number of processors and at the end of a phase, we obtain an output similar to the Kohonen's model. After each phase a processor is inserted according to condition (2.8). In the process of insertion, the existing global ordering of the processors is never disturbed. Note that each phase here can be looked upon as a full execution of Kohonen's algorithm because each phase starts with a given number of processors and converges to an output without changing this number. Thus the only difference between our model and the original model in terms of convergence, is that the former is a repetitive application of the latter, every time by increasing the size of the network without disturbing the global ordering. And from the fact that once the processors are ordered they remain so for all t (see [64], p.143), the output network in the proposed model will give the skeletal shape of the arc pattern as is given by Kohonen's model.

For tree-patterns, after a few phases (when almost all the weight vectors are positioned within the pattern or at least quite close to it), a spike in the network is replaced by a fork structure locally. The starting network being linear, a spike

here represents a junction in the local neighborhood of the pattern. The method is repeated after each phase. For a '+'-like junction two such replacements are required. For other parts of the pattern, the argument holds good since a tree-pattern is a union of several arc-patterns.

If the pattern has a loop, the algorithm (up to Step 7 of Algorithm DySONN) yields only a tree-structured network. Subsequently loops are formed depending on the Voronoi regions (Eqn.(2.10)). If two Voronoi regions are adjacent but the respective processors are not already linked then a link between the two processors is established. The above arguments justify that the resulting network achieved by the DySONN model will give an approximation of the medial axis of the given pattern in the form of a vector skeleton. From the vector skeleton one can easily obtain a raster skeleton by the procedure *Raster-skel*.

In Section 2.2, we have discussed processor insertion mechanism where a new processor is created after the end of a phase. The edge with the maximum length is found after each phase and a processor is created at the middle of it. Thus only one processor is created in each phase. Alternatively, one can insert more processors simultaneously at the end of each phase by creating processors at the middle of all the edges having length greater than δ. That is, at the end of s-th phase (after t_s iterations), if

$$R(t_s) = \{(W_i(t_s), W_{i'}(t_s)) \text{ is an edge } : ||W_i(t_s) - W_{i'}(t_s)|| > \delta\} \tag{2.12}$$

then processors are inserted at the middle of all the edges in the set $R(t_s)$. We have experimented both the mechanisms of insertion and found that the learning is slightly better in the former in terms of the quality of the output skeleton. In the latter mechanism, the size of the network grows very fast from the beginning as in the early stage of learning almost all the edges are expected to have length higher than δ. In the former mechanism, the growth of the network is slow and the learning is better. However, at the later stage, simultaneous insertions are more suitable to obtain the output faster. Another point to note is that the activation level should be kept very high or it should not be used at the beginning so that the network has the freedom to capture the entire input. Otherwise, some branchings of the input pattern may be lost in the output network in which case the overall topology of the pattern is not

preserved.

With the above observations, we propose that the DySONN model should be executed in two stages (a) finding an initial skeleton to get the overall topology of the pattern without activation level and (b) arriving at a more accurate final skeleton after incorporating the activation level. In the first stage, set δ high and get an initial skeleton that captures the overall topology of the input pattern (Figs.2.11) and gives a rough approximation of the required skeleton. In the second stage, to get a close medial axis approximation, assign a very small value to δ and continue the algorithm in an effort to position the weight vectors more accurately along the medial axis. Processor insertions, at this stage, are done according to Eqn.(2.12).

The question is: what value of δ should we choose in stage (a)? Experiments have shown that this choice can be made from a considerably wide range and hence one need not guess an exact value. For example, in Fig.2.11, for an 'A'-shaped pattern, the essential topology could be achieved for $15 \leq \delta \leq 30$. Within this range, the initial skeletons have the same topology. In many situations, the initial skeleton itself (for example, Fig.2.11(a)-(d)) serves the purpose. For example, skeletonization is applied as a preprocessing step in character recognition problems. In character recognition, a vector skeleton of the character pattern makes the recognition task much easier. This vector skeleton need not be very close to the medial axis. Crude vector skeletons similar to that shown in Fig.2.11(a)-(d) are good enough for structural analysis of the input pattern and its recognition.

Moreover, for printed and hand-printed character recognition (for a given font, fixed pen thickness and scanner resolution) we can learn, by trial and error method, an estimate of δ from a number of training characters so that the initial skeleton itself is satisfactory. This estimate can be subsequently used in stage (a) on the test patterns.

The algorithm described in this chapter has been tested on a number of character patterns and the results obtained are quite satisfactory. In addition to the examples used for explanation of the model, a few more results are given in Fig.2.12. The raster skeletons are shown for each pattern. To compute the raster skeletons,

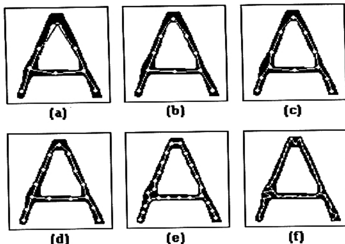


Figure 2.11: Output networks for a A-shaped pattern when (a) δ=30, (b) δ=25, (c) δ=20, (d) δ=15 (e) δ=10, (f) δ=5.

first a vector skeleton was computed with $\delta = 5$. In all the input patterns the initial network is taken to be the same as shown in Fig.2.3(a). The values of the gain terms $\alpha_1(s) = \frac{0.001}{[1+s/50]}$ and $\alpha_2(s) = \frac{0.001}{[1+s/20]}$ where s is the sweep number. Our learning algorithm has been tested with several choices of the starting weight vectors of the processors and the final output is found to be independent of the starting vectors. The input vectors are fed to the network in a random order. The value of ε is taken as 0.001.

2.5 Conclusions

A new shape extraction method for a binary object is proposed in this chapter. The method uses the basic characteristics like topology preservation and automatic dimension selection of Kohonen's self-organizing feature map. As this map is not directly applicable to compute the skeleton, we have suggested certain modifications on it. A dynamic version of the SONN model is developed so that no prior knowledge

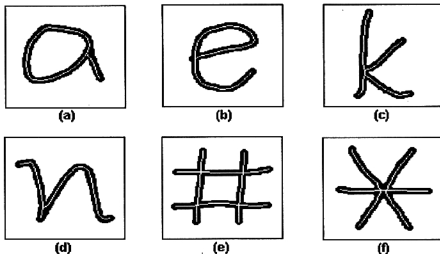


Figure 2.12: Raster skeletons produced by DySONN model.

of the required number of processors in the network is necessary. Combined with a few heuristics, like deciding a fork on the basis of a spike and joining a loop on the basis of adjacency of two Voronoi regions, the proposed DySONN model can produce the shape of a binary object.

The proposed model here is a dynamic variation of Kohonen's self-organizing model. It is dynamic in two respects: (1) the size of the network is initially small and then grows in course of time; (2) although the initial network is linear, the neighbourhoods of the processors may change over time to adapt the structure of the input pattern. While the former is done by certain processor insertion mechanism, the latter is based on some heuristics as mentioned in the *Create-fork* and the *Loop-join* procedures. In the next chapter, we shall discuss another self-organizing model for shape extraction which is free of these heuristics and provides a more efficient way to adapt the topology of the input pattern.

Chapter 3

A TOPOLOGY ADAPTIVE SELF-ORGANIZING NEURAL NETWORK FOR SHAPE EXTRACTION

Abstract: *An efficient topology-adaptive model for shape extraction is proposed. The model is a variation of the neural gas network model and has the capability to expand to an optimum size.*

3.1 Introduction

This chapter, like Chapter 2, deals with the vector skeleton (as mentioned in Chapter 1). That is, a piecewise linear approximation of the skeletal shape of binary patterns will be studied here. In Chapter 2, we have proposed certain modifications on Kohonen's self-organizing neural network model to get a suitable model (DySONN) for skeletal shape extraction of an object. In the DySONN model, the starting network can have a linear structure with a small number of processors. Such a linear structure leads to a higher order structure and finally gives the skeletal shape of the given

pattern. We explained how to update the network to accommodate tree and loop patterns. The network grows in size as well as the neighbourhoods of the processors change according to the local topology of the input pattern.

The DySONN model is, in fact, a combination of a modified version of Kohonen's SONN model and a few heuristics. The skeleton of arc patterns can be achieved by the weight update and the processor insertion/merging process only. But for a pattern with higher order structure, certain heuristic criteria have been used. To accommodate a fork in the pattern, whether a spike (measured in terms of a threshold acute angle) is present in the network is checked. For loops, input vectors are first labelled according to their nearest (winner) processors and then the adjacency of two labelled regions is checked to establish the loop. In this chapter we propose a new model which can get rid of such heuristic/postprocessing. We propose a model which is not only self-organizing but also topology-adaptive and can be used for skeletonization without any such heuristic/postprocessing. In other words, along with the weight update of the processors, the links between two processors are also evolved which leads the network to tend to the skeleton of the input pattern even when it has a nonlinear structure. Moreover, the DySONN model is not straightway applicable to all the three types of input data — binary images, gray-level images and dot patterns. The DySONN model for skeletonization of binary images, in its present form, does not work for the other two types of input pattern. (In the next chapter, we shall see how this model with minor modifications can be applied on other types of input namely, dot patterns and gray-level patterns.) The model, proposed in this chapter, provides a unified approach of skeletonization for all the three types of pattern. It should be noted that although many algorithms exist to find the skeleton of a binary object, hardly any algorithm is reported to find the skeleton of a dot pattern.

In Kohonen's self-organizing model the neighbourhood topology of the network is fixed at the beginning and does not change during the process. The network having either a linear topology or a planar topology (with a triangular or a rectangular or a hexagonal lattice structure) is used. The neighbourhood topology in the network is fixed. Recall, as discussed in Chapters 1 and 2, that such a network of fixed neighbourhood topology does not work well in skeletonization. Moreover, due to the

rigid topology of the network, the topology of the input pattern cannot be completely adapted. These pose problems in skeletonization tasks because the resulting network does not represent the required vector skeleton.

From the above issues it is felt that a dynamically growing network with a new type of local neighbourhood relationship (a dynamically defined neighbourhood) is required. The model proposed in this chapter meets this requirement. In the present model [26], the initial list of processors is empty and the resulting topology is completely data driven. That is, initially there is no processor and no neighbourhood is defined. The network grows in size by means of a certain processor creation/insertion mechanism. During a certain learning process the processors create/adapt their neighbourhoods dynamically, by means of connection building, on the basis of the input. The neighbourhood of a processor in the network is totally input driven which gets dynamically defined. The degrees of different processors (number of neighbours) may vary spatially and over time and hence the neighbourhood topology of the network is not fixed. The topology is adapted on the basis of the local input vectors. The model enables the network to learn the weight vectors as well as its topology from the input vectors in an unsupervised manner. Thus the present model is a *Topology-Adaptive and Self-Organizing Neural Network* (TASONN). The essential issues of this chapter are (i) to adapt the neighbourhood topology of the net while self-organizing and (ii) to allow the network to grow in size as required. The local topology varies not only over time but spatially too. The weight update process is similar to that in Kohonen's SONN model. The rules of processor creation/insertion and neighbourhood topology adaptation are introduced in this chapter. First, the TASONN model on continuous domain is discussed. In a subsequent section, the model is applied on digital domain where a binary image constitutes the set of input vectors.

3.2 The background

In Chapter 1, we mentioned "neural gas network" (NGN) model introduced by Martinetz and Schulten [73, 74, 77]. This model starts with a set of weight vectors without

any links. For each input vector, it ranks the weight vectors and updates first few (say, r) of them. It adapts the r nearest weights where r takes a large initial value and decreases over time according to a predefined schedule. For each input signal, the two nearest processors are joined by an edge. To remove an edge, an "edge aging" scheme is suggested. Fritzke [41] modified the NGN model and suggested a scheme to insert processors in the network. Thus in the "growing neural gas network" (GNGN) model of Fritzke, the initial number of weight vectors need not be known beforehand. One can start with just two processors and then go on growing the network by adding new processors and by linking/delinking edges until certain stopping criteria are achieved.

The models NGN and GNGN are found to be effective methods for topology learning. They are advantageous over the SONN model because the SONN model assumes a predefined topology of the network that remains fixed throughout. The major difference between the NGN and GNGN models is that the former starts with a given number of processors n but does not change it, while the latter starts with only two processors and dynamically inserts new processors until finally the network size becomes n. However, the NGN and GNGN models have some shortcomings. We shall discuss these shortcomings and describe how to overcome them. We first describe the main steps of the model in Algorithm GNGN (for more details see [73] and [41]).

Algorithm GNGN

Step 1: Start with two processors a and b with random weight vectors W_a and W_b. Initialize their local cumulative error counters $E(a)$ and $E(b)$ to 0.

Step 2: Present an input vector X.

Step 3: Find the nearest and second nearest processors i_0 and i_1 whose weight vectors are W_{i_0} and W_{i_1} respectively.

Step 4: Compute $E(i_0) = E(i_0) + ||W_{i_0} - X||^2$.

Step 5: Increase the age of all connections of processor i_0, i.e., set $t_{i_0j} = t_{i_0j} + 1$ for all j with $C_{i_0j} > 0$, where $C_{i_0j} > 0$ means processors i_0 and j are connected by

a link and $C_{i_0j} = 0$ means otherwise.

Step 6: Adjust the weights, to pull towards the input, of processor i_0 and its topological neighbours.

Step 7: If $C_{i_0i_1} = 0$, set $C_{i_0i_1} = 1$ and $t_{i_0i_1} = 0$, that is, construct a link between i_0 and i_1. If $C_{i_0i_1} > 0$, set $t_{i_0i_1} = 0$.

Step 8: Destroy all the links with ages higher than a_{max}. If this causes any singleton processor (processor having no emanating links), remove them.

Step 9: If number of input vectors $= M\lambda$, where M is an integer and λ is an input parameter, then insert a new processor halfway between the processor q with the maximum error $E(q)$ and its neighbour f with the highest error. Modify the links accordingly. Decrease the error values $E(q)$ and $E(f)$ by multiplying them with a constant c. Set error of the new processor equal to new value of $E(q)$.

Step 10: Decrease all error values by multiplying them with a constant d.

Step 11: Repeat from step 2 until the stopping criterion is reached.

3.3 Shortcomings of NGN/GNGN model

In NGN model, weights are updated for the r nearest processors. Initially the value of r is set high so that the chance of dead processors (processors not pulled by any input signal) is reduced. Ordering of the r distances, for each presentation of the input, makes the model computationally expensive.

In NGN and GNGN models, the edge destruction mechanism is based on an aging scheme. Connections at an early stage of the adaptation may not be valid anymore at an advanced stage. To take care of this, the authors develop an age counter for every link after the link is constructed. If the age of a link exceeds a predefined "lifetime" (a_{max} in GNGN algorithm) then the link is destroyed. The

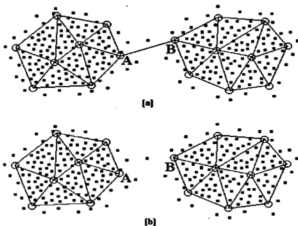


Figure 3.1: A clustering application

lifetime is chosen subjectively here. This technique may pose problems in a few situations as described below.

Consider a clustering problem. Suppose there are two clusters which are well separated except for a few points (Fig.3.1). Suppose we present the input points randomly. Note that since the number of points in the overlapping region is much less compared to the total number of points, they are expected to occur at long time gaps. Then, in the NGN/GNGN model, after some iterations, the output network can be any of the two as shown in Fig.3.1(a) and (b). In Fig.3.1(a), it happened that 1000 iterations were reached after the link AB was constructed and before its lifetime was exceeded. Thus the two clusters remain connected here. In Fig.3.1(b), 1000 iterations were reached between the time gap of the link AB being destroyed and not being constructed again. Therefore, the output networks here are disconnected and represent the two clusters properly. Thus the result may differ depending on the chosen value of the lifetime and the input sequence. Depending on the lifetime and the input sequence, the link AB may appear (due to the arrival of an input vector from the overlapping region) and disappear (due to a considerable time gap between consecutive arrivals of such an input vector) alternately. The asymptotic nature of

the network may be oscillating. Moreover, all the links are given equal importance. There is no way to know whether a link is supported by a considerable number of input vectors or supported by only a few.

Consider the so-called stability-plasticity dilemma. In adapting a new input region the NGN model performs poorly (as poorly as Kohonen's SONN model does) in that a new region cannot be adapted neatly when the input sequence is nonstationary (see [20]). This problem is demonstrated in Fig.3.2. In Figs.3.2(a),(b), inputs are drawn from the lower-left square for $t < 3000$ and from the upper-right square for $3000 < t < 6000$. In can be seen that the network cannot adapt the latter square as neatly as it can adapt the former square.

The proposed model here is free from these shortcomings. The proposed model has similarities with the GNGN model. But, in two major aspects it differs from the GNGN model to overcome the above problems. First, the aging of the links is done differently and the link adaptations are done more efficiently. In NGN and GNGN models, as we have seen, a link is constructed/destroyed in a sudden manner. A link is constructed between two processors as soon as a single input arises for which these two processors are the closest and the second closest. Similarly a link is destroyed as soon as its age exceeds the lifetime. In our model, the links are not constructed or destroyed suddenly. Instead, it assigns a strength variable to each edge and these strengths are adapted (lying between 0 and 1) gradually in the learning process. When the network converges, each link shows its importance in addition to the information provided by the NGN/GNGN model.

Second, the processors here are inserted in a different way so that new input regions can be adapted neatly (Fig.3.2(c)). In NGN model, all the processors are created initially at random positions. This requires prior knowledge of the network size. The GNGN model overcomes this problem by taking only two processors initially and then by growing the size of the network. The present model starts with an empty set of processors and processors are created where the input demands. Thus a new input region can be adapted easily and efficiently. Moreover, in GNGN model, new processors are inserted at constant (decided manually) time intervals while in our

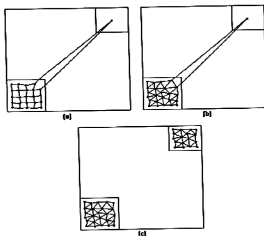


Figure 3.2: Stability-plasticity dilemma. (a) Kohonen's SONN model, (b) NGN model, (c) the proposed TASONN model.

model insertions are done when the current network stabilizes (with some predefined error). After the insertion the learning again continues.

The NGN and GNGN models are useful in creating topology learning networks. (In Chapter 2, we have seen that the skeletonization task needs a topology learning network.) But only topology learning property is not sufficient for skeletonization because a skeleton should closely approximate the medial axis too. The NGN/GNGN model cannot ensure this (we shall see later why) and hence it is not useful to skeleton extraction. To get a model suitable for skeletonization, we introduce a concept of *activation level* that decreases over time and enables the final output network to remain along the medial axis (approximately) of the pattern. A satisfactory skeleton of the pattern can thus be achieved.

Experiments show that in extraction of the skeleton of a pattern, the TASONN model produces much better results than the SONN, NGN and GNGN models. For O- and Y-like patterns, the resulting networks are shown in Figs.3.3-3.4. Figs.3.3(a),(b) and Figs.3.4(a),(b) are the output of Kohonen's SONN model, Fig.3.3(c) and Fig.3.4(c)

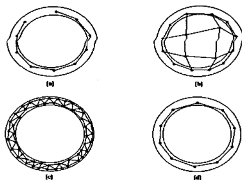


Figure 3.3: For uniform input over O-shaped pattern, Kohonen's output network with (a) linear topology (b) rectangular topology, (c) output network of NGN/GNGN models, (d) output network of TASONN (only links with nonzero β are shown).

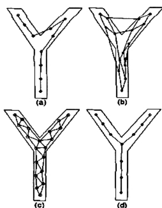


Figure 3.4: For uniform input over Y-shaped pattern, Kohonen's output network with (a) linear topology (b) rectangular topology, (c) output network of NGN/GNGN models, (d) output network of TASONN (only links with nonzero β are shown).

show the output of NGN/GNGN model while Fig.3.3(d) and Fig.3.4(d) show the output of the proposed TASONN model. It can be seen that the resulting networks do not represent the skeletal shape of the input patterns when the SONN and NGN/GNGN models are used. On the other hand, when the TASONN model is used, the resulting networks represent satisfactory skeletons of the patterns. In Figs.3.3(b),(c) and Figs.3.4(b),(c), it can be seen that the output networks form mesh-like structures rather than a vector skeleton. However, in GNGN model, this problem can be avoided by keeping the number of processors small (for which one needs to know the optimum number a priori). For example, the GNGN model may produce a skeleton as good as that of the TASONN model if the number of processors is optimally chosen (here 10). The proposed TASONN model gives an adaptive way to select the optimum number of processors. Moreover, the SONN model is found to not always preserve the topological properties of the input pattern. For example, in Fig.3.3, the output network does not form a loop (Fig.3.3(a)) while the TASONN model preserves the topology correctly (Fig.3.3(d)).

3.4 The TASONN model

We shall first describe our model in a general case and then demonstrate its applicability to skeletonization. Let the input vectors X come from a manifold defined in the m-dimensional space. Denote the array of processors by $\{\pi_1, \pi_2, \ldots, \pi_n\}$ and the neighbourhood N_i of a processor π_i by $\{\pi_p | \pi_p \text{ is connected to } \pi_i\}$ which includes π_i. The processor π_i stores an m-dimensional weight vector W_i which can be considered to be the location of the processor in R^m.

Definition 3.1: By *sensitive region* of a processor we mean an m-dimensional ball of a given radius centred at the weight vector of the processor. The radius is called the *sensitive level*. (It is the same for all the processors and does not change over time).

Definition 3.2: A processor is said to *respond* to an input signal if the input vector

falls inside the sensitive region of the processor.

Definition 3.3: A processor is said to *win* if it is the nearest to the input vector among all the processors. The processor is called the *winner* processor. The processor that is the second nearest to the input vector is called the *second winner.*

In the present model, the entire network is evolved. Initially there is no processor, that is, the set of processors is null. New processors can be added to the network in two ways (which will be discussed in a shortwhile) and we use two terms *create* and *insert* to distinguish between them. Both creation and insertion are performed according to the demand by the input. The former is to take care of new input regions while the latter is to take care of highly dense regions. If some input region is unattended, that is, if no other processor is found around, a new processor is created there. Again, if any input region is highly dense, new processors are inserted there.

An input vector is received through the input lines. The first processor is created at the location of the first input vector. A sensitive level is set and a sensitive region is determined for the processor. From the second input onwards the process continues as follows. If the presented input falls outside the sensitive regions of all the existing processors (that is, no processor responds) then a processor is created at the location of the input. If at least one processor responds, then (1) adjust the weights of the winner and the second winner (if it exists); and (2) strengthen the strength variable of the edge connecting the winner and the second winner and weaken that of all other edges.

Several iterations (presentations of input) make one *phase.* One phase is completed when the weight vectors of the current set of processors converge, that is, when the network becomes stable. (Here, a phase can be looked upon as one complete learning session in NGN model). After each phase, a new processor is inserted at the middle of the link with the maximum strength value. The links for the new processor are rebuilt after removing the old link and the strengths are readjusted (strength of the old link is distributed equally to the two new links). The next input vector is presented and the whole process is continued until certain convergence criterion is

met. Formally, the above process can be described as follows.

Processor creation

Suppose the sensitive level is τ. A new processor is created when the input vector $X(t)$, at time t, comes from a new region. A new input region is detected if no processor is found within a distance of τ from $X(t)$. In other words, on presentation of an input, if no processor responds with the given sensitive level τ (or, if the processor set is empty) then create a new processor at the location of $X(t)$.

Link construction

Establish a link between two nodes π_u and π_v $(u \neq v)$ if $X(t)$ is an input such that

$$max(||X(t) - W_u||, ||X(t) - W_v||) \leq ||X(t) - W_k|| \quad \forall k \ (k \neq u, k \neq v) \qquad (3.1)$$

The above criterion means, in the iterative process, if some input vector arises for which W_u and W_v are the weights of the winner and the second winner respectively then the nodes π_u and π_v are joined by a link, say, L_{uv}. A strength $\beta_{ij} (0 \leq \beta_{ij} \leq 1)$ is associated with every such link L_{ij}. These strengths are updated during learning. Initially, all β_{ij}'s are set to 0.

The strengths are updated as follows.

$$\beta_{uv}(t+1) = \beta_{uv}(t) + \frac{1}{t+1}(1 - \beta_{uv}(t)) \qquad (3.2)$$

For all other links,

$$\beta_{ij}(t+1) = \beta_{ij}(t) + \frac{1}{t+1}(0 - \beta_{ij}(t)) = \frac{t}{t+1}\beta_{ij}(t) \qquad (3.3)$$

This is as if $X(t)$ pulls only β_{uv} towards 1 and pulls all other β's towards 0.

Weight updating

After presentation of input $X(t)$ at the t-th iteration links are established by condition (3.1) and the weights are updated by

$$\begin{aligned} W_u(t+1) &= W_u(t) + \alpha_1(t)[X(t) - W_u(t)] \quad \text{if } \pi_u \text{ is the winner} \\ W_v(t+1) &= W_v(t) + \alpha_2(t)[X(t) - W_v(t)] \quad \text{if } \pi_v \text{ is the second winner} \end{aligned} \qquad (3.4)$$

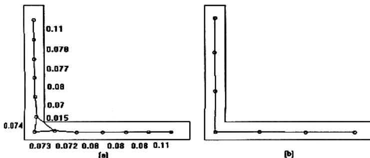


Figure 3.5: For L-shaped pattern, the output network achieved by TASONN after convergence (a) δ=25, (b) δ=50. The nonzero β values are shown against each link.

where $\alpha_1(t)$ and $\alpha_2(t)$ ($0 < \alpha_2(t) \leq \alpha_1(t) < 1$) are chosen as in condition (1.11).

The asymptotic relation between β_{ij}'s and W_i's will be as follows. Suppose, the asymptotic values of the weight vectors are W_i^*. Consider the Voronoi tiles [90] of the Voronoi diagram of order 2 of the set of weight vectors W_i^* which are

$$V_{ij} = \{X \in R^m : max(||X - W_i^*||, ||X - W_j^*||) \leq ||X - W_k^*|| \ \forall k \ (k \neq i, k \neq j)\} \quad (3.5)$$

As in NGN model [77], the present model considers pattern distribution $p(X)$ that has support only on a submanifold, not on the entire embedding space R^m. It can be noted that the asymptotic value of the strength β_{ij} is $\int_{V_{ij}} p(X)dX$. It is possible that some β_{ij} can have a positive value at a certain stage of learning although V_{ij} is in fact empty. In that case, a link introduced during learning will vanish in the limit. By similar arguments given by Martinetz and Schulten [77], the links with asymptotically non-zero β_{ij}'s will form a subgraph of the Delaunay triangulation of W_i^* ($i = 1, 2, \ldots, n$).

There are several advantages of TASONN model over the Kohonen's SONN model. First, folding of the network is a common problem in the SONN model while, as can easily be seen, the TASONN model does not have it. Second, the local neighbourhood or topology around a processor is fixed in the SONN model while it is data driven in the TASONN model making it more suitable in various applications.

Moreover, as mentioned earlier, in Kohonen's SONN model and in NGN/GNGN model, a link between two processors either exists or does not exist (deterministic). But in TASONN a link is allowed to have a kind of degree of its existence in the form of β and this can be of use in shape analysis. For example, for an L-shaped pattern (Fig.3.5(a)), there are 13 processors (after convergence) and a link between two processors is shown whenever the corresponding β value is greater than zero. Here $p(X)$ is the uniform distribution over the L-shaped area. The value of β for the diagonal link is 0.015 while the β values for the other links are between 0.070 and 0.080 (for the end processors they are 0.110). Removal of a link with a significantly low (in comparison to others) β value may thus result in a network that represents the input pattern more accurately. It is to be noted that such undesirable spurious triangles may occur at junctions of a pattern. They can be avoided by increasing the gaps between two processors (Fig.3.5(b)). Alternatively, such spurious triangles can be removed by removal of links as stated above.

Processor insertion

Only after a phase is completed, are processors inserted. Suppose, at the end of the s-th phase, the weight vectors of the processors are $W_1(t_s), \ldots, W_{n(s)}(t_s)$ where $n(s)$ is the number of processors during the s-th phase and t_s is the total number of iterations needed to reach the end of the s-th phase. The stopping criteria can be set depending upon the application (see next section). If, at the end of a phase, the stopping criteria are not met then a new processor π_l is inserted between π_k and $\pi_{k'}$ where

$$\beta_{kk'} = \max_{i=1,\ldots,n(s)} \max_{\pi_{i'} \in N_i - \{\pi_i\}} \beta_{ii'} \tag{3.6}$$

Set $\beta_{lk} = \beta_{lk'} = \frac{1}{2}\beta_{kk'}$, new $\beta_{kk'} = 0$ and the weight of $\pi_l = \frac{1}{2}[W_k(t_s) + W_{k'}(t_s)]$

By this process a processor is inserted at a location where the demand is maximum. The process continues until, at the end of a phase, the stopping criteria are met.

Algorithm TASONN

Step 1: Initialize $t = 0$.

Step 2: Present input pattern $X(t)$.

Step 3: If some processor responds to $X(t)$ then go to Step 5.

Step 4: Create a processor at the location of $X(t)$.

Step 5: If the second winner exists then update the strengths β_{ij}'s according to Eqns.(3.2) and (3.3).

Step 6: Update the weights of the winner and the second winner, according to Eqn.(3.4). (if the second winner does not exist then update the weight of the winner only).

Step 7: If the network is not stable, set $t = t + 1$ and go to Step 2.

Step 8: If the stopping criterion is met, then go to Step 10.

Step 9: Insert a processor according to condition (3.6), set $t = t + 1$. Go to Step 2.

Step 10: Stop.

We shall now see how the above model can be used in shape extraction. The main features of the TASONN model, useful in this problem, are the dynamic topology adaptivity and its dynamic growth in size.

3.5 Shape extraction by TASONN model

To make the proposed TASONN model applicable to shape extraction, a suitable stopping criterion is chosen. A parameter δ is introduced as an upper bound of the distances between two neighbouring processors. Skeletal shape extraction of binary images is considered here. As before, the set of input vectors $S = \{P_1, P_2, \ldots, P_N\}$

where P_j represents the positional co-ordinates of an object point and hence $m = 2$. One presentation of all the points in S makes one *sweep* consisting of N iterations. After one sweep is completed, the iterative process for the next sweep starts again from P_1 through P_N. With the new stopping criterion, the processor insertion step (discussed in the previous section) is modified as follows.

Processor insertion

Processors are inserted after each phase. One phase is completed when the network with the current set of processors converges, that is, when

$$||W_i(t) - W_i(t')|| < \varepsilon \qquad \forall i \tag{3.7}$$

where t and t' are the iteration numbers at the end of two consecutive sweeps and ε is a predetermined small positive quantity. If

$$||W_k(t_s) - W_{k'}(t_s)|| = \max_{i=1,\ldots,n(s)} \max_{\pi_{i'} \in N_i - \{\pi_i\}} ||W_i(t_s) - W_{i'}(t_s)|| > \delta \tag{3.8}$$

then one processor is inserted between π_k and $\pi_{k'}$. The weight of the new processor and the new β values are set as before. After the insertion of a processor, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$\text{for all } i, ||W_i(t_s) - W_{i'}(t_s)|| \leq \delta, \quad \forall\, \pi_{i'} \in N_i - \{\pi_i\} \tag{3.9}$$

It is to be noted that one phase here can essentially be looked upon as one complete execution of the NGN model. The output network of the algorithm gives an approximate global shape of the input pattern. The final network obtained by the above algorithm gives a vector skeleton for the given input pattern. The raster skeleton can easily be derived from the network by procedure *Raster-skel* as described in Chapter 2.

It should be mentioned that the output network depends on the parameter δ, and with a proper choice of δ, one can get a satisfactory vector skeleton. This issue is discussed in detail later.

Experimental results, on various input patterns, have shown that the above algorithm converges and the resulting network gives an approximation of the skeleton

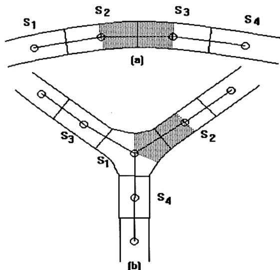


Figure 3.6: Link establishments in (a) arc pattern (b) fork pattern.

of the input pattern. The following discussions would help us to see how the resulting network gives an approximation of the skeletal shape of the pattern.

For arc patterns, it can easily be seen that the above node joining criteria would join the processors by single links (other link strengths are zero) so that the topology of the input is preserved. It is explained in Fig.3.6(a). Here two processors corresponding to the regions S_2 and S_3 are joined since they are the two nearest processors from the input vectors lying in the shaded region. Similarly, other links are established. The strengths of all other possible links are zero since no input vectors contribute to them. For a node representing a fork junction region S_1 (Fig.3.6(b)) the node pairs corresponding to the region pairs $(S_1, S_2), (S_1, S_3)$ and (S_1, S_4) are joined for the same reason.

3.6 Discussions and remarks

The parameter δ controls how densely the processors are to be located in the network. Like in the DySONN model, δ here controls the gap between two neighbouring processors. A desirable skeleton may not be achieved if δ is too high or too low. If δ is very high then the skeleton, although may preserve the essential topology of the pattern, does not properly represent the medial axis (Figs.3.7(a-b)). On the other hand, very low values of δ might produce spurious triangles in the network (Figs.3.7(e-f)) which does not represent the true skeletal shape of the pattern. It is to be noted that a similar problem occurs in the SONN model and in the DySONN model where the network becomes zigzag if too many processors are used. In the former case, a portion of the output skeleton may lie outside the object. Hence we require some adaptive mechanism to avoid this situation so that a satisfactory medial axis representation can be obtained. A similar mechanism, as introduced in the DySONN model in Chapter 2, is used here. We assign an activation level to each processor (see Definitions 2.2 and 2.3) so that an input vector falling within the activation region only can activate a processor. It should be mentioned at this stage that the sensitive level and activation level, although similar, are used in different contexts and hence are given different terminology. The sensitive level remains fixed over time and is used to create the initial set of processors. It is not used any more unless a new input region occurs. The activation level varies over time and may not be the same for all the processors at a point of time. It is used in competition and weight updating. It should also be noted that the sensitive level is also taken as a parameter in the TASONN model. But, since it is used to detect new regions and for selecting the initial few processors, its choice is not crucial. Within a wide range of values of the sensitive level, the model produces the same results.

Although the activation level overcomes the formation of spurious triangles in linear or arc segments of the pattern, it cannot overcome the problem completely at junctions of the pattern. If two such segments meet at a small angle then spurious triangles may occur even after incorporating activation levels. It is because of the fact that in the TASONN model the topology of the network is being continuously

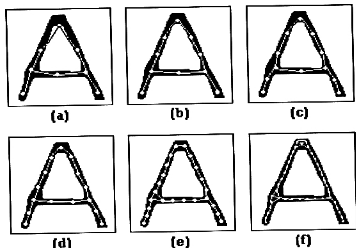


Figure 3.7: Output networks for an A-shaped pattern when (a) δ=30, (b) δ=25, (c) δ=20, (d) δ=15 (e) δ=10, (f) δ=5 without activation level.

updated (unlike in the DySONN model) by the input. However, this situation can be avoided by keeping δ high as seen in Fig.3.5.

With these observations, we implement the TASONN model for skeletonization as follows. The whole process is divided into two stages (a) finding an initial skeleton to get the overall topology of the pattern and (b) arriving at a more accurate final skeleton after incorporating the activation level. In the first stage, set δ high and get an initial skeleton that does not contain any spurious triangle but preserves the topology of the input pattern (Figs.3.7(a-d)). Note that higher the value of δ lower is the chance of forming a spurious triangle. If there is a spurious triangle then one of the following actions can be taken: (i) remove a link by choosing a threshold on the β values as described in Fig.3.5. (ii) take a higher value of δ and repeat the process to get an initial skeleton. However, this problem of spurious triangles will not arise in the final form of our algorithm as we shall see later.

By the above method, we get an initial skeleton, free of spurious triangles, that reflects the overall topology of the pattern but may not be close to the medial axis.

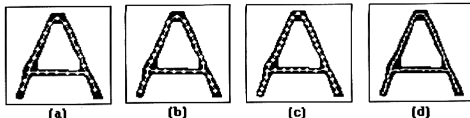


Figure 3.8: Output networks for an A-shaped pattern when (a) δ=10, (b) δ=7, (c) δ=5, (d) δ=3 with activation level.

Now, in the second stage, to get a close medial axis approximation, a lower value is assigned to δ and the weight updating process is continued. As the topology of the input pattern is already learned, the strengths (β) are ignored at this stage. That is, we do not make any topological changes in the network in the second stage. We only update the weight vectors in an effort to position them more accurately along the medial axis. As an illustration, output networks for the A-shaped pattern, using activation level, are shown for different δ values in Fig.3.8. It can be seen that these results give satisfactory skeletal shape of the pattern even for small δ's (compare with Fig.3.7 when activation level is not used). Processors are inserted simultaneously, as in the DySONN model, at this stage (see Eqn. 2.12).

As discussed in Chapter 2, in the DySONN model, the initial choice of δ can be made from a wide range within which the overall topology of the output network remains the same (Figs.3.7(a)-(d)). Hence one can easily choose a value within this range. Moreover, in applications like OCR, we can learn (by trial and error method) an estimate of δ from a number of training characters so that the initial skeleton gives the essential structure of the character pattern. This estimate can be used on the test patterns in the first stage. Often, the initial skeleton, obtained in the first stage, can be used in recognition tasks. Nevertheless, if one wants to get a more accurate skeleton, one can go for the second stage. A suitable value for the sensitive level can also be estimated in a similar way.

The TASONN model is tested on several English character patterns. Some of

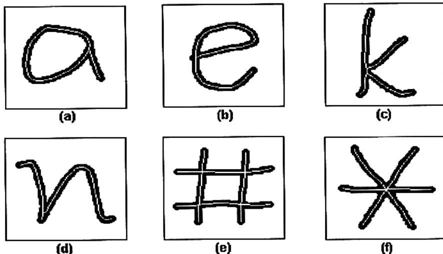


Figure 3.9: Output raster skeletons by TASONN model for different patterns.

the results are shown in Fig.3.9. The values of the gain terms $\alpha_1(s) = \frac{0.001}{[1+s/50]}$ and $\alpha_2(s) = \frac{0.001}{[1+s/20]}$ where s is the sweep number. Like in the DySONN model, with several choices of the starting weight vectors of the processors we have observed the final output to be independent of the starting vectors. The input vectors are presented to the network in a random order. The value of ε is taken as 0.001.

3.7 Conclusions

A topology-adaptive self-organizing neural network model is proposed which is applicable in skeletal shape extraction of an object. In Kohonen's SONN model, the network topology is initially set and it is maintained throughout. In the SONN model, the network is self-organizing in that the network tends to an approximation of the input pattern space in an orderly fashion without any supervision. In the proposed model the same is achieved and, in addition, the network topology is adapted automatically unlike in the SONN model. The topology adaptivity is a major issue of the

proposed model which makes it applicable to skeletonization tasks. Due to a fixed network topology set initially, Kohonen's model cannot always properly represent the shape of a pattern while the proposed model can do it more accurately. The proposed model evolves the topology of the network and it is self-organizing as well.

The proposed model is a modified version of an existing topology learning network. It overcomes some of the shortcomings (in view of shape extraction) of the latter and proposes a more efficient model. Moreover, the model is especially tuned to generate a vector skeleton (from which a raster skeleton can also be obtained) of the underlying pattern. Thus the proposed TASONN model provides a skeletonization technique which is a frequently encountered problem in pattern recognition.

Another important advantage of the TASONN model is fault tolerance. It is easy to see that damage of a few nodes or links can be automatically repaired in the TASONN model while it is not so in the SONN model. During learning, if any nodes and links are damaged, the input vectors in the surroundings will generate new nodes and new links. The link strengths (β values) are adapted from the subsequent input presentations. Moreover, the data driven topology adaptation of the network does not allow any folding to occur.

As skeletonization algorithms, the present TASONN model and the previously discussed DySONN model can claim to possess quite a few advantages over the existing conventional thinning algorithms. The most important one is high robustness of the present algorithms with respect to boundary and interior noise. Next, they are invariant to rotation by arbitrary angles while many of the conventional algorithms are not. The medial axis representation efficiency is also found to be quite satisfactory. The algorithms are also capable of higher data reduction. Finally, the proposed models can be seen as unified approaches to skeletonization because they are applicable to all the three types of input patterns, namely, binary images, dot patterns and gray-level images. These issues will be elaborated in the next chapter.

Chapter 4

COMPARISONS OF NEURAL NETWORK BASED AND CONVENTIONAL SHAPE EXTRACTION TECHNIQUES

Abstract: *A comprehensive comparison is made between the proposed neural network based algorithms and some well-cited conventional shape extraction algorithms.*

4.1 Introduction

In this chapter, we make a comprehensive comparative study of conventional shape extraction techniques with neural network based techniques. In Chapter 1, we have mentioned various conventional techniques for skeletal shape extraction available in the literature. Subsequently, in Chapters 2 and 3, we have proposed two shape extraction models which are based on neural networks. In the present chapter, several issues in skeletonization are studied [27] to compare the performance of the two said techniques. Since, a lot of algorithms are developed using conventional techniques, we have selected a few of them to compare with the two proposed neural algorithms.

For comparisons, algorithms developed over more than a decade are selected which are well-cited. Both qualitative and quantitative judgements have shown encouraging results to make us believe that the neural techniques are better in quite a few respects, especially, in terms of robustness to noise and rotations.

The fundamental and essential properties of skeletons are (i) connectivity preservation and (ii) unit thickness. But a good skeletonization algorithm should desirably possess a few more properties also. An algorithm should satisfy these desirable properties as much as possible. The comparative studies of the two above mentioned techniques are carried out in the light of the following properties of a good skeletonization algorithm: (a) medial axis representation; (b) noise immunity or robustness; (c) rotation invariance; (d) data reduction efficiency; (e) extendability to different types of input data.

Algorithm designers should keep all these properties in mind and should try to fulfill them as much as possible for various reasons. First, since one of the primary objectives of the skeleton is to approximate the medial axis, the better the approximation the more is the efficiency of the algorithm. The derived skeleton should not be biased, in other words, it should give a symmetric representation of the input pattern. Second, the performance of a skeletonization algorithm should be judged not only by measuring how well it works on perfect input data, but also by measuring how it works on noisy data. An algorithm may give good results with noise-free input but very poor results in the presence of even a little amount of noise. Such algorithms are not of much use. Since noise is almost unavoidable in practice, more robust algorithms are warranted. To judge the robustness of the algorithm, we should first test the algorithm for perfect data. We next add random noise to the input data. Then we should study the difference between the output of perfect input and that arising from noisy data.

Third, rotation of the object should not affect the output skeleton. In OCR applications, a tilt of the document may cause the character patterns to be rotated. Often italicized characters are found mixed in normal characters. If rotation invariance of the skeletonization algorithm cannot be achieved, it may not well suit in the application. In such cases, preprocessing steps called *skew correction* and *tilt correc-*

tion are required to detect the amount of rotation and to make necessary corrections. Fourth, as one more objective of skeletonization is storage reduction without loosing any structural information, the data reduction capacity of the algorithm is also to be kept in mind.

Finally, an algorithm should be considered more powerful if it is not restricted to only one type of input data. An algorithm will be considered more robust if it works on more than one of the three types of input data namely, binary patterns, dot patterns and gray-level patterns. It is desirable that the algorithm working on one type of data should at least be easily extendable to the other types. Keeping the above issues in mind, we shall now go into the comparisons of the two said approaches – conventional approach and neural approach. The algorithms are run on a test set of 62 character patterns (English upper and lower case characters and numerals) and the average figures mentioned hereafter are computed based on this test set.

4.2 Medial axis representation

As the basic purpose of skeletonization is to approximate the medial axis of the object pattern, it is important that the output skeleton should approximate the medial axis as close as possible. It is found in the previous chapters that the parameter δ play an important role so far as the medial axis representation is concerned. Very low values of this parameter might produce a zig-zag skeleton which does not represent the true skeletal shape of the pattern. On the other hand, if δ is very high the skeleton does not satisfactorily represent the medial axis. Moderate values of the parameter yield a good skeleton. Thus the medial axis representation of the output skeleton, in the proposed neural models, depends on the choice of δ. This dependency, as discussed in Chapters 2 and 3, can be avoided by introducing activation region. Subsequently a raster skeleton can also be derived. After getting the raster skeleton, the following measure is computed for comparison of the goodness of medial axis representation with a few other thinning algorithms.

$$\eta = \frac{Area[S'']}{Area[S]} \tag{4.1}$$

Table 4.1: Index of medial axis representation

ALGORITHMS	η
Chin et al. [19]	0.819
Holt et al. [53]	0.891
Hall [49]	0.902
Zhang and Suen [115]	0.886
Lu and Wang [70]	0.898
Arcelli and Baja [5]	0.867
Jang and Chin [56]	0.881
Datta and Parui [21]	0.931
Fan et al. [37]	0.811
Proposed neural algorithms	0.893

where S = set of all object pixels in the input pattern, S'' = union of the maximal digital disks (included in S) centred at all the skeletal pixels. $Area[.]$ is an operator that counts the number of pixels. Clearly, η ranges from 0 to 1 and the derived skeleton is identical to the ideal medial axis if η is 1. η measures the closeness of the extracted skeleton to the true medial axis and hence can be used as an index of medial axis representation.

For the neural algorithms, the average η value is found to be 0.893 (for computing η, first a vector skeleton with δ=5 is obtained from which a raster skeleton is computed). Average values of η are computed for a number of conventional algorithms for the same set of test patterns. The results are summarized in Table 4.1. It is found that for the neural algorithms, the medial axis representation index is as satisfactory as in the conventional algorithms.

4.3 Robustness

An important aspect of a skeletonization algorithm is noise immunity which makes the neural algorithms qualitatively different from the conventional ones. Two types

of noise, namely, boundary noise and object noise are considered here. Noise along the boundary of an object are boundary noise and noise within the object are object noise. If the original object contains noise, the skeleton should not deviate much from the skeleton of the same object without noise. A serious problem with many conventional algorithms is that they sometimes produce noisy skeletons if the input patterns contain noisy boundary (see [21] and [57]). Moreover, these algorithms cannot handle object noise from their design point of view. On the contrary, the proposed neural algorithms are designed to take care of both these types of noise and are highly robust to them.

The above claim of robustness of the neural algorithms can be argued as follows. The resulting vector skeleton here is given by the weight vectors after convergence, and their links. Its final position is highly insensitive to noise pixels because of two factors. First, the weight vector converges to the centre of gravity of the respective Voronoi region and this centre is not greatly affected by noise pixels. Second, the activation region of a processor decreases over time and as a result, the boundary noise pixels are kept outside it to a great extent. Thus, the noise insensitivity of the present algorithm is clear from its learning mechanism and convergence property.

Many existing conventional algorithms use a rigid definition of connectedness of the object – which in effect causes noise sensitivity. Our method implicitly relaxes the concept of connectivity and it is found that such a relaxation helps the robustness particularly in situations where SNR is very close to 1.

4.3.1 Boundary noise

The boundary noise is distributed on the boundary of the object (white noise) and on its immediate neighbourhood in the background (black noise). Many conventional thinning algorithms generate noise spurs in the presence of noise in the boundary of the object (see [21, 57] for more details). The proposed neural skeletonization algorithms (for the reasons stated above) are found to be very robust to such boundary noise. Here we add black and white boundary noise pixels and study their effect on the skeleton. We have experimented on several examples with different SNR values

where the SNR is defined as follows.

$$SNR_B = \frac{\text{Number of boundary black pixels}}{(B+W)} \tag{4.2}$$

where B = number of black noise pixels and W = number of white noise pixels.

As an illustration, the robustness of our algorithm to boundary noise is demonstrated in Fig.4.1. A conventional iterative thinning algorithm (for example, the algorithm by Jang and Chin [56]) is found to be noise sensitive (Fig.4.1(b)) while our proposed algorithm is insensitive to it (Fig.4.1(c), (d)). An error measure, as given below, has been suggested by Jang and Chin [57].

$$m_e = min\{1, \frac{Area[S_K - S'_K] + Area[S'_K - S_K]}{Area[S_K]}\} \tag{4.3}$$

where S_K and S'_K are the skeletons obtained from S and its noisy version respectively.

The average values of m_e against different SNR values are computed for comparing the algorithms. Using the above measure, Jang and Chin's algorithm [57] has been found to be superior to several other conventional algorithms, for example, [19, 49, 53, 70, 115], in terms of boundary noise immunity. The average m_e values of the two proposed neural algorithms are almost the same, are found to be more encouraging (see Fig.4.2) and reflect higher robustness to boundary noise.

4.3.2 Object noise

By object noise we mean the white noise distributed over the entire object including its boundary. The existing conventional algorithms are not able to handle such noise which is interior to the object. But such noise may occur in practice due to several reasons. The problem with the conventional algorithms (for example, the iterative ones) is that they use the property of local connectivity within a small window (mostly 3 x 3) and try to preserve such local connectivities throughout. Secondly, due to the use of a fixed set of templates of a small size, they treat a single white noise pixel as a hole consisting of a single pixel. As a result, in the output skeleton it appears as a big hole (Fig.4.3(c)). Fig.4.3 demonstrates how only two noise pixels misclassify a

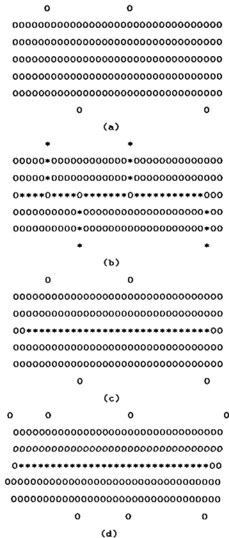


Figure 4.1: Robustness to boundary noise, an illustration. '0' represents object and '*' represents skeletal pixel. (a) A line segment with four boundary noise pixels, (b) result of a conventional iterative thinning algorithm, (c)-(d) results of our neural algorithms with the same noise and higher noise.

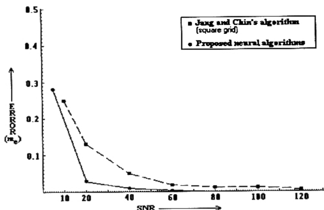


Figure 4.2: Boundary noise immunity.

'1'-like pattern to an '8'-like pattern. On the contrary, the proposed neural algorithms use the connectivity concept in a more global sense (for example, while joining two processors we check whether the two respective regions are adjacent). The algorithm treats small holes as white noise at the cost of the possibility of missing a true small hole. Thus, very small holes have hardly any effect on the resulting skeleton (Fig.4.3(d)-(e)). But if the hole is large enough and is in fact a part of the pattern, it is output as a hole in the resulting skeleton (as we have seen in an A-shaped pattern in the previous chapters). We have experimented and found that the algorithm is robust and performs satisfactorily with moderately low SNR (moderate amount of noise) where the SNR is defined by (assuming the noise are uniformly distributed)

$$SNR = \frac{\text{Number of object pixels}}{\text{Number of noise pixels on the object}} \tag{4.4}$$

In fact, if the amount of noise is not so high that the connectivity of the object is no more preserved, the neural algorithms can be straightway implemented. For a very low SNR, that is, for a very high amount of noise, when the connectivity is lost, the algorithm in the DySONN model needs a minor modification while the TASONN model needs none. With very high amount of noise a binary object becomes merely a set of scattered pixels or a dot pattern. The proposed neural algorithms can still yield the global skeletal shape of the pattern. This issue will be discussed in details

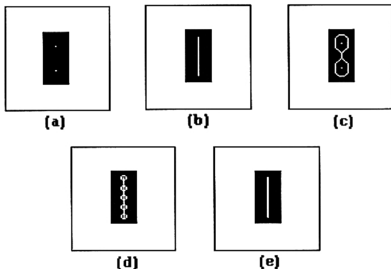


Figure 4.3: (a) Input with two white noise pixels inside the object; output of conventional thinning algorithms (b) without noise; (c) with noise; (d) vector skeleton by the two proposed neural algorithms with the same noise as in (b); (e) output raster skeleton for the same.

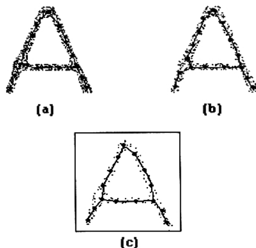


Figure 4.4: The final skeletons obtained for the pattern 'A' with object noise. (a) SNR= 2 (b) SNR= 1.5 (c) SNR= 1.1.

in a later section where the extendability feature of the neural algorithms are dealt with.

An illustration is presented in Figs.4.4(a)-(c) for the pattern 'A' with different SNR values. We have taken a very high amount of object noise and tested the algorithm for several character patterns. It has been found that even in the presence of very high noise (SNR = 2.0, 1.5 and 1.1), the proposed algorithm is able to extract the skeletal shape of the original object as can be seen in the example figure (Fig.4.4). The existing conventional algorithms fail to work in such situations.

4.4 Rotation invariance

Another advantage of the neural algorithms, which is easy to see, is that the output skeleton does not depend on rotation of the input pattern by arbitrary angles. In other words, the skeleton of the original object and that of the object rotated by any

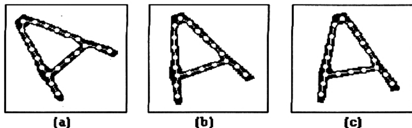


Figure 4.5: Effect of rotation by angles (a) 45° (b) 22° and (c) 11°.

angle are the same in the sense that the latter is a rotated version of the former by the same angle. That is, the skeletonization process which is a transformation from an object to its skeleton, is invariant under rotation. This is due to the fact that the proposed method does not assume any underlying grid. The iterative methods for skeletonization that use square grid are invariant under rotation by multiples of 90° only. Rotations by any other angles generate redundant spurs and hence the quality of the skeleton becomes poor (for details see [57]). The work [57], based on derived grid, reports invariance under 45° rotation also. In applications like OCR, as mentioned earlier, the rotation invariance property is very much required since it saves a lot of preprocessing time that is taken by the correction measures due to rotation.

We now demonstrate the rotation invariance of the proposed neural techniques. As an illustrative example, we have rotated the pattern 'A' by different angles and shown the output skeletons in Fig.4.5. All the vector skeletons in Figs.4.5(a-c) give essentially the same structure.

4.5 Data reduction efficiency

One of the basic purposes of skeletonization is to reduce the storage space required to store the image data without losing the essential structural information. As a raster image occupies lot of space, it becomes highly expensive to handle them in

applications where multiple frames of large sizes are to be processed together. Thus it is often required to store only its essential structural information in a suitablly chosen data structure (e.g., a linked list) and thus storage requirement can be drastically reduced. The neural algorithms can achieve more data reduction compared to many existing skeletonization algorithms. In the neural algorithms, the set of weight vectors along with their interconnections, or the graph (planar straight line graph) with the weight vectors as its nodes and the interconnections as its edges represents the skeletal shape of the input binary pattern. The graph requires much less space than the original input image and hence a considerable data reduction is achieved. It can be seen that the less the number of processors in the network the more is the data reduction. By choosing larger values of δ, that is, by making the average distance between neighbouring processors larger, we can make higher data reduction. But this might worsen the accuracy of medial axis representation. A proper choice of δ (as mentioned earlier) can balance this trade-off.

4.6 Extendability to dot pattern and gray-level pattern

Extendability of the neural algorithms to dot patterns and gray-level patterns is a major achievement. In the class of conventional techniques, the algorithms are designed for one particular type of input data (mostly binary patterns). In contrast, the neural approach gives a unified skeletonization algorithm for all the three types of data. The DySONN model proposed for skeletonization of binary patterns needs a minor modification for handling dot patterns while in the TASONN model all the three data types can be handled by the same algorithm.

4.6.1 Dot pattern

For binary patterns, as discussed in Chapter 1, a skeleton is well-defined. Thus a skeleton from a binary pattern can be computed. But, for dot patterns the situation is different. For a dot pattern, the skeleton cannot be properly defined. Hence

skeletonization algorithms, similar to that for binary patterns, are hardly available. But however, the human visual system can extract the perceptual skeleton from even a dot pattern. For example, a dot pattern having a definite shape (say, 'S'-like) is recognized by the human brain almost as easily as for a binary image having the same shape. Unfortunately, the conventional skeletonization algorithms that extract skeletons from binary images do not work for dot patterns due to the lack of a proper definition which poses a problem in formulating a computational method for a dot pattern skeleton. In fact, perhaps due to this reason, hardly any work on finding dot pattern skeletons has been reported in the literature. In the recent past, neural network technology has been showing a great deal of promise in areas where conventional computing poses problems. The proposed neural algorithms establish that even without a proper definition, the proposed techniques are able to generate skeletons that are quite close to the perceptual skeleton.

Modification in DySONN model

It is clear that the loop joining process by checking the adjacency of two respective regions S_i's is meaningful as long as the connectivity of the object is preserved (after noise addition). Otherwise, the loop joining has to be done in a different way. In fact, by properties of Kohonen's model, two processors will be neighbours to each other if the two respective Voronoi regions are close in the pattern space since Kohonen's self-organizing map preserves the topological relationship. Hence in our case, it is expected that two close processors should be neighbours to each other. Therefore, we join two processors by a link if they are close enough, but are not already joined. The closeness is determined on the basis of δ. The procedure *Loop-join* in the DySONN model should be replaced by the following procedure.

Procedure Loop-join

For every processor, its nearest among other processors (excepting its neighbours) is found. If the distance between these two processors is less than δ, then they are joined by a new link.

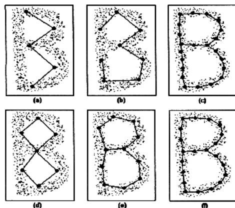


Figure 4.6: Results on a B-shaped dot pattern (a)-(c) by DySONN and (d)-(f) by TASONN model. (c) and (f) are the final vector skeletons and others are intermediate results.

Thus after the initial convergence, we join the processors satisfying the above criteria. Then we continue the algorithm until final convergence is reached, that is, until condition (2.9) is satisfied. Results on a dot pattern are demonstrated in Fig.4.6.

4.6.2 Gray-level pattern

The skeletonization technique discussed here can be extended to gray-level thinning [6, 31]. A gray-level image may concern either the whole image or a subset of it. Here the latter case is assumed. We consider gray-level images where the area of interest (i.e., the object portion) can be interpreted as constituting a multi-valued foreground emerging from a single-valued background. This simplifies the general gray-level case, but such simplified situation has been reported in the literature (for example, see [6]). The extension can be done in the following way.

Suppose, for a gray-level pattern, g_{rs} is the gray value of the pixel at the r-th row and s-th column. Then, in DySONN model, the weight update rule replacing

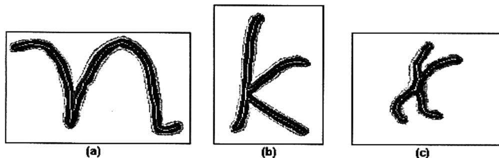


Figure 4.7: Output skeletons of gray-level characters and chromosome pattern by DySONN model ($\delta = 5$).

Eqn. (2.2) will be

$$\begin{aligned} W_k(t+1) &= W_k(t) + \alpha_1(t)[P_j - W_k(t)]g_{rs} \\ W_p(t+1) &= W_p(t) + \alpha_2(t)[P_j - W_p(t)]g_{rs} \quad \text{for } \pi_p \in N_k - \{\pi_k\} \end{aligned} \tag{4.5}$$

Multiplication by g_{rs} means that the amount of update will be more for pixels with high gray values and less for pixels with low gray values. Note that for binary images g_{rs} is either 0 or 1. Similarly the update rule (3.4) in TASONN model will be replaced by

$$\begin{aligned} W_u(t+1) &= W_u(t) + \alpha_1(t)[P_j - W_u(t)]g_{rs} \quad \text{if } \pi_u \text{ is the winner} \\ W_v(t+1) &= W_v(t) + \alpha_2(t)[P_j - W_v(t)]g_{rs} \quad \text{if } \pi_v \text{ is the second winner} \end{aligned} \tag{4.6}$$

In the DySONN model, the gray-level extensions for arc and tree-like patterns are straightforward. But for loop patterns, the extension is not trivial because the definition of adjacency is not well defined in that case. If criteria for the adjacency are available then the extension is possible for loop patterns also. In some situations the loop joining criteria used for dot patterns may work. Alternatively, one can use the adjacency as defined for binary images after local thresholding (locality being defined by the regions S_i). The DySONN algorithm has been tested on several gray-level images and Fig.4.7 shows the results on character and chromosome patterns. However, in the TASONN model, the gray-level extension is more straightforward. In fact, the TASONN model can be implemented on gray-level patterns only by

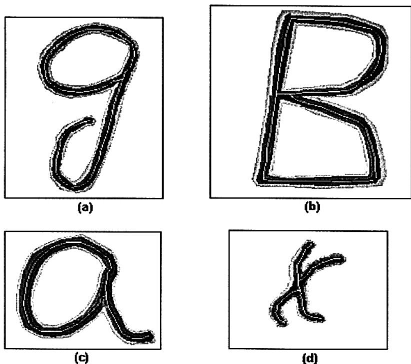


Figure 4.8: Output skeletons of gray-level characters and chromosome pattern by TASONN model ($\delta = 5$).

replacing the weight update rule by Eqn. (4.6). The raster skeletons of gray-level character and chromosome patterns are presented in Fig.4.8.

4.7 Computational aspects

In general, finding skeletons is a computationally expensive task. That is why parallel algorithms have been encouraged in this field. The proposed neural network based methods are parallel in nature (in general, neurocomputing has been considered to be a new form of parallel computing). In the proposed skeletonization algorithms, the input vectors are presented to the network sequentially. All the processors present in

the network compute the distances from their respective weight vectors in parallel in a constant time and the winner is selected by a 'maxnet' like network. The winner processor and its neighbours update their weights simultaneously. Thus for a single input, the time taken by the above process is not dependent on the network size. This process is repeated for all the input vectors. Hence one complete pass of the input, that is, one sweep requires $O(N)$ time where N is the total number of object pixels. It can be seen that the total number of sweeps does not depend on the input size N.

It should be mentioned here that in existing parallel thinning algorithms also multiple processors are used where all the processors work in parallel. But in these algorithms, all the input values are fed together in an iteration and all the processors (or a large subset of them) compute their output in parallel which would be the input in the next iteration. The total number of passes that are required by this class of algorithms is $O(width_{max})$ where $width_{max}$ is the maximum width of the input pattern. These algorithms use *cellular networks* where each pixel in the image is assigned a processor. Thus the number of processors required is $O(N_I)$ where N_I is the total number of pixels in the whole (including the background) image. In our algorithms, the size of the network is $O(N_S)$ where N_S is the number of skeletal pixels. In most of the applications, $N_I >> N_S$.

4.8 Conclusions

A comparative study of a few classical skeletonization techniques and two neural skeletonization techniques (proposed in this thesis) are carried out in this chapter. The proposed neural algorithms have quite a few advantages over many existing conventional thinning techniques. Many existing conventional algorithms are not robust to noise. Some of them are to some extent robust, but that too for boundary noise only. In contrast, the neural algorithms are found to be highly robust to both boundary and interior noise in the sense that it can produce satisfactory skeleton even under very low SNR (close to 1) or very high amount of noise. Noise is unavoidable in real-life applications. During transmission of signals, the data may accumulate

noise of any of the above mentioned types. The amount of noise can vary within a wide range which depends on several external factors. The neural techniques would be useful particularly when the input data contain high amount of noise.

Another advantage of the proposed neural algorithms is that it is invariant under rotation by arbitrary angles. Many conventional algorithms are invariant only under rotation of 90°. The data reduction efficiency of the neural algorithms is higher than that of the conventional algorithms. Moreover, the neural approach is found to be insensitive to the type of input data. That is, in addition to binary patterns, it works even for gray-level patterns and dot patterns.

Finally, parallel skeletonization algorithms (based on template matching techniques) use a cellular network of processors which work in parallel. Each pixel in the image, even background ones, requires a processor. Thus large images needs huge number of processors which becomes very expensive. Moreover, most of the processors are idle (wasted) because the number of object pixels is much less (usually) than the number of background pixels. But in the neural algorithms, the number of processors is of the order of the number of skeleton pixels.

There exist numerous algorithms for extraction of skeletal shape of an object. Every algorithm has some defects over the others. Although the neural techniques are found to be superior to the conventional techniques in several respects, they have some disadvantages also. Since they are very much noise insensitive, they may treat very small (real) holes as noise. Thus such holes may be missed in the output skeleton and hence the proposed models do not strictly guarantee homotopy property (see Fig.4.3) unlike many conventional algorithms.

Chapter 5

A SELF-ORGANIZING MODEL TO COMPUTE CONVEX HULL

Abstract: *A self-organizing model is proposed for convex hull computation of a finite planar set. A complete network, consisting of all the input points as its nodes, adapts in an unsupervised way such that the relevant nodes and links are mapped as the corresponding hull-vertices and hull-edges.*

5.1 Introduction

After discussing self-organizing neural network models for skeletal shape extraction of an object, we now switch over to a self-organizing model for a different problem. Like the earlier one, this problem is also well studied. In the present chapter we deal with a well-known problem, namely, computation of *convex-hull* which is a central problem to the theory and applications of computational geometry in various fields. In computational geometry, the problem to compute the convex-hull of a finite number of points in $2D$ has been a topic of research for a long time. The concept of convex-hull is quite natural and easy to understand. It can be imagined as a stretched rubber band surrounding the set of points and then being released to shrink. The convex-hull of a given set of points is defined as the smallest convex polygon containing all the

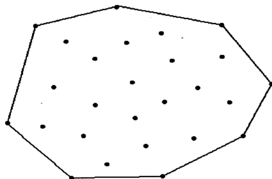


Figure 5.1: A point set in $2D$ and its convex-hull.

points in the set. In many situations, it provides a rough estimate of the boundary-based shape of the underlying set of points. An illustration of the convex-hull is given in Fig.5.1.

The computation of the convex-hull of a finite set of points, particularly in the plane, has been studied extensively and has wide applications in pattern recognition, image processing, cluster analysis, statistics, robust estimation, operations research, computer graphics, robotics and several other fields [1, 3, 13, 29, 30, 32, 54, 90, 102, 111]. Several conventional algorithms are available for this problem. Here we formulate the convex-hull problem as an artificial neural network problem. We discuss how an initial network can iteratively adapt itself on the basis of the input signals (points) and self-organize accordingly to finally arrive at the convex-hull.

Since 1970's, the problem of convex-hull computation has been an interesting area of research. As a result, a number of algorithms are available in the literature to solve this problem. These algorithms can broadly be classified into two categories: computing exact convex-hull [2, 16, 33, 47, 58, 89, 113]; and computing approximate convex-hull [7, 8, 48, 68]. There can be another classification of these algorithms: sequential (using single processor) and parallel (using multiple processors). Again, some of the convex-hull algorithms consider input vectors of a specific dimension (2-dimension or 3-dimension which are the most common ones in real life) and some

algorithms deal with higher dimensional input also. While processing the input, either they can be presented together in batch (the *off-line* problem) or they can be presented one by one (the *on-line* problem).

Out of the above algorithms, the algorithms [68, 113] are designed on artificial neural networks. Wennmyr [113] proposed an exact convex-hull computation algorithm based on multi-layer perceptron [69, 96]. The author designs a network that can decide whether a given point is inside a convex region (using the fact that a convex region is always the intersection of half-planes). In this network every node at the lowest level has a different decision boundary. Leung et al. [68] proposed an algorithm for the computation of an approximate convex-hull. The network consists of one input layer and one output layer of neurons. Similar to the adaptive resonance theory (ART) [14], the two layers of neurons can communicate via feedforward as well as feedback connections.

In this chapter, we propose a self-organizing neural network model [22] for the computation of the convex-hull of a given planar set. Recall that the term 'self-organization' here refers to the ability to learn from the input without having any prior supervising information and arrange the processors accordingly. A well-known self-organizing neural network model is Kohonen's feature maps [64]. In Kohonen's feature maps, a mapping is established between the input space and the network in an orderly fashion.

We also found, as discussed earlier, dynamic versions of Kohonen's model proposed by different researchers. Sabourin and Mitiche [98] proposed a "selective multi-resolution" approach in the context of shape classification. Such a dynamic model can overcome some shortcomings of Kohonen's model. Another dynamic version of Kohonen's model has been suggested by Fritzke [39] to model the probability distribution in the plane. In vector quantization context, Choi and Park [20] have proposed a dynamic version of Kohonen's model. The preceding chapters of this thesis dealt with dynamic self-organizing neural network models that are designed to extract skeletons from the input pattern [24, 25, 26, 87].

The present neural network model behaves, as we shall see in the next section,

like a self-organizing network. We start with a network where every point in the given set is assigned a processor. The network consists of three layers. The bottom layer is used for the computation of angles which are passed onto the middle layer. The middle layer computes the minimum angles. These information are passed onto the topmost layer as well as fed back to the bottom layer. Using these information, the topmost layer and the bottom layer self-organize, to label the hull-processors in an orderly fashion. Initially the model identifies a small number of points as hull-points. Gradually the network self-organizes to identify other processors that correspond to the rest of the hull-points. These hull-points are generated in an orderly fashion so that the convex-hull of the data points can be easily generated. The learning takes place without any supervision.

5.2 The proposed model

In the present model we consider a finite set of n 2-dimensional points representing the input vectors (the signals)

$$\begin{aligned} S &= \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \\ &= \{P_1, P_2, \ldots, P_n\}. \end{aligned} \tag{5.1}$$

The convex-hull of the planar set S is defined as follows:

Definition 5.1. The convex-hull of a planar set S is the smallest convex set containing S. The convex-hull here is in fact a convex polygon. Each edge of the polygon is a hull edge and each of its vertices is a hull vertex. (see Fig.5.1)

Result 5.1. It is well-known [90] that every hull edge (of S) partitions the plane into two half-planes such that one of them contains all the points of S and the other contains no point of S.

Let $\pi_1, \pi_2, \cdots, \pi_n$ be n processors associated with the points $P_1, P_2, \cdots, P_n$ respectively. The processor π_i stores its location (x_i, y_i) for $i = 1, 2, \cdots, n$ as its weight vector. Also assume that each processor π_i is connected with itself and all other

processors π_j. These processors are termed as *point-processors.*

Definition 5.2. A point-processor π_k is called a *hull-processor* if one of the following conditions holds:

$$y_k = \min_i\{y_i\} = y_{min} \tag{5.2}$$

$$x_k = \max_i\{x_i\} = x_{max} \tag{5.3}$$

$$y_k = \max_i\{y_i\} = y_{max} \tag{5.4}$$

$$x_k = \min_i\{x_i\} = x_{min}. \tag{5.5}$$

It can be seen that for $n > 1$, initially there can be two or more types of hull-processors out of the four types – A, B, C and D depending upon the Eqns. (5.2), (5.3), (5.4) and (5.5) respectively (Fig.5.2). Let us assume that all the four types of processors are present initially (later on we shall see that this assumption does not restrict us). Suppose, π_{k_1}, π_{k_2}, π_{k_3} and π_{k_4} are the initial four hull-processors of the types A, B, C and D respectively. We emphasize here that initially when we assign the processor types, more than one processors can be of the same type (according to Definition 5.2). For example, there could be several points with the same smallest value of the y—coordinate resulting in more candidates for the A-type hull-processors. In such a case only one of them (no matter which one) is assigned as a hull-processor of the respective type and others are assigned as point-processors. The processor assigned as hull-processor, at this stage, is the *first mother* of A-type.

Definition 5.3. Two processors π_i and π_j are said to be equivalent if their weight vectors are the same.

Let us first consider the A-type hull-processor. If processor π_{k_1} is an A-type

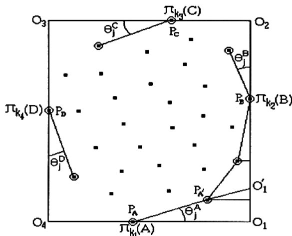


Figure 5.2: Point-processors and hull-processors (circled). The labels in parentheses denote the processor types.

mother hull-processor then compute the angles θ_i^A, $i = 1, 2, \cdots, n$, given by

$$\theta_i^A = \begin{cases} \cos^{-1} \frac{x_i - x_{k_1}}{\sqrt{(x_i - x_{k_1})^2 + (y_i - y_{k_1})^2}} & \text{if } y_i \geq y_{k_1} \\ 2\pi - \cos^{-1} \frac{x_i - x_{k_1}}{\sqrt{(x_i - x_{k_1})^2 + (y_i - y_{k_1})^2}} & \text{if } y_i < y_{k_1} \end{cases} \quad (i \neq k_1). \tag{5.6}$$

Note that θ_i^A measures the angle, as shown in Fig.5.2, between the line segment joining the mother hull-processor π_{k_1} and the processor π_i with respect to the x-axis. Let π_j be the processor such that (see Fig.5.2)

$$\theta_j^A = \min_i \theta_i^A. \tag{5.7}$$

The processor π_j is declared as a hull-processor and as a *child* of the mother hull-processor π_{k_1}. Since π_j is a child of π_{k_1}, it inherits the type of its mother π_{k_1}, that is, π_j is also an A-type hull-processor. The creation of the child makes π_{k_1} *inactive*, and makes π_j an *active* mother hull-processor. By inactivating a processor we mean that it is no longer a mother but it remains as a hull-processor. The process of angle calculation and finding the processor with minimum angle is repeated with the active mother hull-processor π_j. Consequently π_j also gives birth to a child processor of its

own type. This process of *reproduction* is continued until an A-type child becomes equivalent to an existing hull-processor of another type. Our next proposition shows the process indeed terminates.

Proposition 5.1. If the reproduction process is continued, at one point of time the child of an A-type mother will eventually be equivalent to another type (B, C or D).

Proof. Here S is a given set of finite number of points.

Let P_A, P_B, P_C and P_D be the points corresponding to the initial four hull-processors π_{k_1}, π_{k_2}, π_{k_3} and π_{k_4}.

By definition, P_A, P_B, P_C and P_D are hull-points and the rectangle $O_1O_2O_3O_4$ (the sides of which pass through the points P_A, P_B, P_C and P_D) in Fig.5.2 contains all the points of S.

Let $P_{A'}$ be the point corresponding to the child processor of π_{k_1}. Suppose, the line $P_AP_{A'}$ cuts O_1O_2 at O'_1. Thus the triangle $P_AO'_1O_1$ does not contain any point of S other than P_A and $P_{A'}$. So the line $P_AP_{A'}$ partitions the plane into two half-planes such that one of them contains all the points of S and the other contains no point of S. Therefore, by Result 5.1, $P_AP_{A'}$ is a hull-edge. Hence $P_{A'}$ is a hull-point. Now, if $P_{A'} = P_B$ then the reproduction of A-type processors stops. If not, the reproduction of A-type processors continues.

Renaming $P_{A'}$ as P_A the same logic can be repeated. Note that with the creation of a new child processor (hull-point) the number of points inside the triangle $P_AP_BO_1$ is reduced at least by one. Since S is finite, in a finite number of steps, this number would be zero in which case the most recent A-type child processor will be created at P_B. And the reproduction process stops.

Similar arguments hold for the other three types of processors. Thus the reproduction process of a particular type of hull-processor stops when a child of its own type is equivalent to another type of processor. Hence the proof. ■

Like A-type processors, other types of hull-processors use similar reproduction rules when the respective formulae for angle calculation are as follows. If π_{k_2}, π_{k_3}, and π_{k_4} are B-, C- and D-type hull-processors respectively, then compute the angles θ_i^B, θ_i^C and θ_i^D $i = 1, 2, \cdots, n$, given by Eqns. (5.8), (5.9) and (5.10) respectively.

$$\theta_i^B = \begin{cases} \cos^{-1} \frac{y_i - y_{k_2}}{\sqrt{(x_i - x_{k_2})^2 + (y_i - y_{k_2})^2}} & \text{if } x_i \leq x_{k_2} \\ 2\pi - \cos^{-1} \frac{y_i - y_{k_2}}{\sqrt{(x_i - x_{k_2})^2 + (y_i - y_{k_2})^2}} & \text{if } x_i > x_{k_2} \end{cases} \quad (i \neq k_2) \qquad (5.8)$$

$$\theta_i^C = \begin{cases} \cos^{-1} \frac{x_{k_3} - x_i}{\sqrt{(x_{k_3} - x_i)^2 + (y_{k_3} - y_i)^2}} & \text{if } y_{k_3} \geq y_i \\ 2\pi - \cos^{-1} \frac{x_{k_3} - x_i}{\sqrt{(x_{k_3} - x_i)^2 + (y_{k_3} - y_i)^2}} & \text{if } y_{k_3} < y_i \end{cases} \quad (i \neq k_3) \qquad (5.9)$$

$$\theta_i^D = \begin{cases} \cos^{-1} \frac{y_{k_4} - y_i}{\sqrt{(x_{k_4} - x_i)^2 + (y_{k_4} - y_i)^2}} & \text{if } x_{k_4} \leq x_i \\ 2\pi - \cos^{-1} \frac{y_{k_4} - y_i}{\sqrt{(x_{k_4} - x_i)^2 + (y_{k_4} - y_i)^2}} & \text{if } x_{k_4} > x_i \end{cases} \quad (i \neq k_4). \qquad (5.10)$$

The computation of $\min \theta_i^B$, $\min \theta_i^C$, $\min \theta_i^D$ and the reproduction processes are similar to that of the A-type processor. It can be seen, by Proposition 5.1, that the process of reproduction for all the types of processors stops. And when it happens, as we shall see later, all the hull-processors are ordered. The initial four processors are ordered by their type definitions. Again, the child hull-processors are ordered according to their parent-child relations.

5.3 The architecture of the model

We shall now discuss the network architecture for the proposed model. The network consists of two layers: lower layer and upper layer. The lower layer is used for the angle computation and the upper layer is used for finding the minimum angle (see Fig.5.3). The number of processors in each layer is n, the number of data points. Both layers have similar structures where every processor is connected to every other.

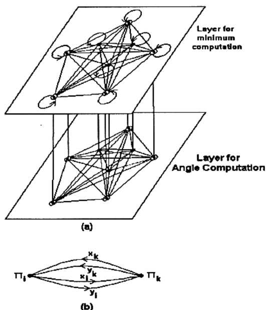


Figure 5.3: The subnetwork architecture for a given type of hull-processor. Solid dots indicate point-processors and the circled dots represent hull-processors. All the links represented by lines are bidirectional.

In the lower layer, there is no self-connection while in the upper layer every processor has a self-excitation connection as present in the 'maxnet' described earlier in Chapter 1. Moreover, the ith processor in the lower layer is connected to the ith processor in the upper layer by feedforward as well as by feedback links.

First, consider the A-type processors. Suppose, in the lower layer, π_k is an A-type hull-processor. The within layer (lower) connections in Fig.5.3(a) are shown in more details in Fig.5.3(b). A processor π_i is connected to π_k by four links (in Fig.5.3(a), they are represented by a single straight line). At the beginning, the connection weights are assigned as shown in the figure against each link. The motivation behind such connection weights is that π_k must know the co-ordinates of π_i, and π_i must know the co-ordinates of π_k. The output o_i^A of the ith processor is computed by the activation function

$$o_i^A = f^A(\alpha_i, \beta_i, x_k, y_k) = \begin{cases} \cos^{-1} \frac{\alpha_i - x_k}{\sqrt{(\alpha_i - x_k)^2 + (\beta_i - y_k)^2}} & \text{if } \beta_i \geq y_k \\ 2\pi - \cos^{-1} \frac{\alpha_i - x_k}{\sqrt{(\alpha_i - x_k)^2 + (\beta_i - y_k)^2}} & \text{if } \beta_i < y_k \end{cases} . \tag{5.11}$$

If we put $\alpha_i = x_i$ and $\beta_i = y_i$, then o_i^A is nothing but the angle θ_i^A (Eqn.(5.6)). It is to be mentioned here that many neural network models use sigmoidal function as the activation function. As we need to compute the angles, the arccosine function has been used here.

The above activation function can be rewritten as

$$o_i^A = f^A(\alpha_i, \beta_i, x_k, y_k) = (1 + \delta)\pi - \delta \cos^{-1} \frac{\alpha_i - x_k}{\sqrt{(\alpha_i - x_k)^2 + (\beta_i - y_k)^2}} \tag{5.12}$$

where

$$\delta = \begin{cases} -1 & \text{if } \beta_i \geq y_k \\ +1 & \text{otherwise.} \end{cases}$$

This δ can easily be realized by an analog *comparator* circuit which compares the voltage β_i with the reference voltage y_k.

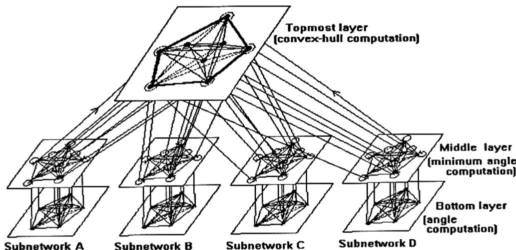


Figure 5.4: The complete network architecture for the proposed model.

Every processor π_i, in the lower layer, computes its output o_i^A according to Eqn.(5.11). These output values are passed to the respective processors in the upper layer by the forward links (Fig.5.3(a)). The upper layer then selects the winner (with respect to the minimum value) processor π_j and passes this information back to the lower layer. The processor π_j is then declared as an A-type hull-processor and as a child of π_k, and the processor π_k is then inactivated. Thus, at any point of time, only one A-type hull-processor is active in the entire lower layer although there could be more than one such processors present in the layer. In the next iteration, the only active A-type processor (most recently created child) plays the role of an A-type mother hull-processor and the process continues until the stopping criteria are met (see Proposition 5.1).

The above process is replicated for all other types (i.e., B, C and D) of processors. This is done by replication of the above network (Fig.5.3(a)) four times and adding another layer above these networks as shown in Fig.5.4. Each such replication (subnetwork) takes care of a distinct type of hull-processors. These subnetworks are structurally identical but their activation functions are different. The activation

function of a lower layer node for Subnetwork-B, Subnetwork-C and Subnetwork-D are given in Eqns. (5.13), (5.14) and (5.15) respectively.

$$o_i^B = f^B(\alpha_i, \beta_i, x_{k_2}, y_{k_2}) = \begin{cases} \cos^{-1} \frac{\beta_i - y_{k_2}}{\sqrt{(\alpha_i - x_{k_2})^2 + (\beta_i - y_{k_2})^2}} & \text{if } \alpha_i \leq x_{k_2} \\ 2\pi - \cos^{-1} \frac{\beta_i - y_{k_2}}{\sqrt{(\alpha_i - x_{k_2})^2 + (\beta_i - y_{k_2})^2}} & \text{if } \alpha_i > x_{k_2} \end{cases} \tag{5.13}$$

$$o_i^C = f^C(\alpha_i, \beta_i, x_{k_3}, y_{k_3}) = \begin{cases} \cos^{-1} \frac{x_{k_3} - \alpha_i}{\sqrt{(x_{k_3} - \alpha_i)^2 + (y_{k_3} - \beta_i)^2}} & \text{if } y_{k_3} \geq \beta_i \\ 2\pi - \cos^{-1} \frac{x_{k_3} - \alpha_i}{\sqrt{(x_{k_3} - \alpha_i)^2 + (y_{k_3} - \beta_i)^2}} & \text{if } y_{k_3} < \beta_i \end{cases} \tag{5.14}$$

$$o_i^D = f^D(\alpha_i, \beta_i, x_{k_4}, y_{k_4}) = \begin{cases} \cos^{-1} \frac{y_{k_4} - \beta_i}{\sqrt{(x_{k_4} - \alpha_i)^2 + (y_{k_4} - \beta_i)^2}} & \text{if } x_{k_4} \leq \alpha_i \\ 2\pi - \cos^{-1} \frac{y_{k_4} - \beta_i}{\sqrt{(x_{k_4} - \alpha_i)^2 + (y_{k_4} - \beta_i)^2}} & \text{if } x_{k_4} > \alpha_i \end{cases} \tag{5.15}$$

The ith processor in the upper layer of every subnetwork (now it is the middle layer in the complete network) is connected to the ith processor of the topmost layer (see Fig.5.4). So the ith node of the topmost layer will have four input connections. Within the topmost layer every processor is connected to every other and the connection weights initially are set to zero. In every iteration, the angles (with respect to a given mother processor) are computed in the bottom layer. These angles are passed to the middle layer where the winner (the child processor) is selected. After the winner is selected, the child is marked as a hull-processor on the topmost layer. At the same time, the weight of the connection from the mother to the child processor (on the topmost layer) is set to 1 (denoted by thick lines in the topmost layer in Fig.5.4). We now show, by Propositions 5.2 & 5.3, that the network formed by the hull-processors and the links with connection weights as 1 on the topmost layer provides the required convex-hull.

Proposition 5.2. On termination all hull-processors on the topmost layer are ordered and every processor will have exactly two links. In other words, the processors

and the links form a closed loop.

Proof. We explained earlier that if a mother hull-processor π_k creates a child processor π_j, then a link is established (the link weight is set to 1) from π_k to π_j on the topmost layer to get a hull-edge. Hence the link defines the ordering. Thus every subnetwork produces an (linearly) ordered set of hull-processors. Since there is an implicit *cyclic* ordering among the A, B, C and D type hull-processors ($A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$) and the child of one type eventually becomes equivalent to another type (coming as the next in the ordering) by Proposition 5.1, all the processors are ordered and they form a closed loop. [Note that if a particular type of processor is absent (say, C-type processor is absent) then a B-type child will eventually be equivalent to a D-type processor, when D-type is present.] ■

Proposition 5.3. The topmost layer finally gives the required convex-hull.

Proof. Recall the proof of Proposition 5.1. It is clear (by Eqn. 5.7) that the extended edge $P_A P_{A'}$ (see Fig.5.2) partitions the plane into two half-planes such that only one half-plane contains all the points of S and the other half-plane contains no point of S. Similar argument holds for all the links with weight 1 (on the topmost layer). Moreover, by Proposition 5.2, all the links with weight 1 on the topmost layer form a closed loop. Hence the proof. ■

The algorithm for computation of the convex-hull can be now briefly stated as:

Algorithm CHULL

Step 1. [Initialization]

Create the initial active hull-processors according to the Eqns. (5.2),(5.3), (5.4) and (5.5). Mark these first mothers as A^*, B^*, C^* and D^* respectively.

Step 2. [Reproduction]

Each active mother hull-processor creates a child of its own type. The mother

then becomes inactive and the child becomes active. This process is continued unless the child is equivalent to the first mother of a different type.

Step 3. [Convex-hull construction]
Construct the convex-hull from the topmost layer.

Step 4. Stop.

5.3.1 Computational aspects

After the initial four hull-processors are created, every processor, in each subnetwork, computes its output independently (in parallel). Thus this computation takes constant amount of time. Since, for every type of hull-processors (i.e., A, B, C and D), there is a separate subnetwork, the winner processor can be selected in parallel by the respective minimum-calculating layers. Hence, computational complexity of this step will not depend on size of the data set. It is easy to see that each subnetwork, in the worst case, needs to compute the output (in the lower layer) and then select the winner (in the upper layer) h times where h is the number of hull-points. Thus, the whole process takes, in the worst case, $O(h)$ number of iterations.

The above complexity analysis assumes that no three points of S fall on a single side of the rectangle $O_1O_2O_3O_4$. Without this assumption we may have three or more initial hull-processors of the same type (say, B type). We select arbitrarily one of them as the respective first mother (B^*). Then, in Step 2 of our algorithm, there will be a tie while computing the minimum angle. If we break the tie arbitrarily, we may not hit the first mother B^* at the first chance. Hence the Subnetwork A may take longer time than $O(h)$. However, from theoretical point of view, the probability of three points (in the Real plane) falling on a single side is zero.

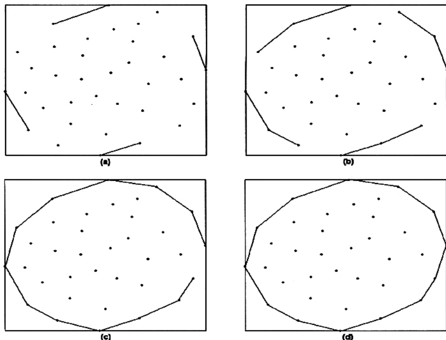


Figure 5.5: The intermediate and final results on a planar set. (a)-(c) The results after iterations 1, 2 and 3 respectively; (d) the final result after 4 iterations.

5.4 Results and conclusions

The proposed model is tested on several $2D$ point sets. Fig.5.5 shows one such example with intermediate outputs. Throughout this chapter, we have assumed that all the four types of hull-processors are present. Note that for $n > 1$, there may be only two or three types of hull-processors also. It is easy to see that the algorithm remains the same as we can simply forget the relevant subnetworks for the types not being present. At the time of network initialization (Step 1 in Algorithm CHULL) when the A, B, C and D type hull-processors are identified, the respective subnetworks are put 'on' or 'off'.

We proposed a neural network model for computing the exact convex-hull of planar set. The proposed model is self-organizing in the sense that the learning is unsupervised. In the output layer (top layer), the hull-processors are organized in such a way that they are mapped (recall Kohonen's self-organizing feature maps) as the hull vertices of the required convex hull. Moreover, only the links between two successive hull-processors are set to 1 and these links are mapped as the corresponding hull edges.

Computation of convex-hull of a planar set has been a problem of considerable interests and several researchers have developed various algorithms. While most of them are based on conventional techniques, Wennmyr [113] and Leung et al. [68] suggested neural network based techniques. For computing the exact convex-hull, Wennmyr proposed a multi-layer perceptron based model. Leung et al. proposed an ART based model that can compute an approximate convex-hull. Both algorithms have complexities $O(n)$ in off-line mode. The algorithm proposed in this chapter provides a self-organizing model to compute convex-hull of a given set of $2D$ points in $O(h)$ time in off-line mode.

Like the computation of convex-hull of a planar set, another interesting problem studied in computational geometry that has drawn considerable attentions over a long time is the computation of circular hull of a planar set. While the convex-hull provides the minimum convex polygon enclosing the set of input points, the circular hull provides the minimum circle that encloses all the points. The circular hull is also called the minimum spanning circle and has a lot of applications in optimizing problems. In the the next chapter we shall discuss some such applications and propose a self-organizing neural network model for computing the minimum spanning circle.

Chapter 6

MINIMUM SPANNING CIRCLE BY SELF-ORGANIZATION

Abstract: *A self-organizing model is proposed that computes the centre and radius of the smallest circle enclosing a finite set of given points. The model is straightway implementable to higher dimensions.*

6.1 Introduction

The present chapter deals with the problem of finding minimum spanning circle. The problem is to find the radius and the centre of the smallest circle such that the circle encloses n given points in the Euclidean plane. Such a circle is called the *minimum spanning circle* (MSC). In location theory this is the unweighted Euclidean 1-centre problem in the plane.

The MSC has applications in optimization, pattern recognition, image analysis, statistical estimation etc. It has applications in transmission and transportation problems. Suppose a community of users needs the service of some facility, for example, radio/T.V. receivers receiving signals from a single transmitter. The problem is to find the optimum location of the transmitter (facility). In other words, we need to find where the facility should be located so as to minimize the maximum distance

from the facility to any user. It is clear that, if we imagine the users as points in the plane, then the centre of the MSC for this set of points is the solution of the problem. In a different kind of application, for example in transportation, the centre of the MSC can be the optimal location for a service station if we want to minimize the maximum distance that a customer would have to travel from his place to the service station. Here the locations of the residences specify the set of points.

The MSC problem has a long history. After it was posed by Sylvester [109] in 1857, it attracted several researchers and as a result many solutions have been suggested [35, 38, 50, 78, 80, 83, 103, 104, 105, 106, 110]. The (worst-case) time complexities for the above solutions range from $O(n^3)$ to $O(n \log n)$, where n is the number of input points. The exception is the work due to Megiddo [78]. Megiddo formulated this problem as a linear programming problem which could be solved in $O(n)$ time. Apart from the time complexities, many of these algorithms suffer from implementation complexities. That is, they are not simple from the point of view of actual programming and implementation.

In this chapter, an efficient and parallel algorithm is proposed for finding the MSC for a given set of points using a self-organizing neural network [23]. The algorithm finds the optimum position of the centre of the MSC iteratively. Against each presentation of input signals (here points) the network adapts without any supervision and finally converges to the optimum solution. Modeling of the MSC problem into neural networks has various justifications. Instead of performing operations sequentially neural network models are capable of doing the same in parallel using networks composed of many processors and thus provide high computational rate. Moreover, in neural network technology, a complex problem can be solved by a number of processors, working collectively, while each processor needs to do much simpler computations only. Thus simple processors can work collectively and can solve a much complex problem. Finally, the neural network modeling helps extendability to higher dimensions without any extra computational complexity.

6.2 The MSC problem

Let us consider the following problem :
Let $S = \{P_1, P_2, ..., P_n\}$ be a set of n given points in the Euclidean plane. The problem is to find the centre and radius of the smallest circle such that no point of S falls outside the circle. Such a circle is called the minimum spanning circle. The problem can be stated as :

Find the point $W = (w_1, w_2)$ so that

$$\max_{j} \|P_j - W\| \tag{6.1}$$

is minimized over all choices of W. In other words, we compute

$$r = \min_{(w_1,w_2)} \max_{j} \{((w_1 - a_j)^2 + (w_2 - b_j)^2)^{\frac{1}{2}}\} \tag{6.2}$$

where (a_j, b_j) is the co-ordinates of $P_j, j = 1, 2, \ldots, n$. The radius of the MSC is r and the optimum (w_1, w_2) is the centre.

After describing the MSC problem, we now formulate it into a self-organizing neural network model. As discussed in the preceding chapters, the term self-organization refers to the ability to adapt or learn from the input signals without having any prior supervising information. The input signals are presented to a network consisting of a number of processors. Against the presentation of each input, some feedbacks are generated and the network is adjusted (i.e., the weight vectors are updated in a certain fashion) according to this feedback. The proposed model use the principle of simple competitive learning, discussed in Chapter 1, and learns the position of the centre of the MSC from the input without any supervision.

In a simple competitive network model what essentially happens is : There is a set of input (signal) vectors; and an array of processors (normally a 1- or 2-dimensional network) interconnected among themselves. Each processor is associated with a m-dimensional weight vector $W = (w_1, w_2, \ldots, w_m)$ with initial random values.

Let S be the set of input points where each $P \in S$ is an m-dimensional vector.

Then, against each presentation of the input vector, all the processors compete and the weight of the winner processor (closest from the input) is updated by the following equation:

$$W(t+1) = W(t) + \alpha(t)(P - W(t)) \tag{6.3}$$

where $0 < \alpha(t) < 1$. $\alpha(t)$ and $W(t)$ stand for the value of α and W respectively at iteration t. This update process is iteratively repeated until the network converges (that is, when some stopping criterion is met). The variable α is a training rate coefficient that starts with some initial value and may be gradually reduced during the training process. This allows large steps for initial rapid training and smaller steps as final value is approached.

It is easy to see that the effect of the above update is to move the weight vector toward the input vector. This fact will be used in the present self-organizing model. The movement of the weight vector has also been stated in the text as movement of the respective processor. It should be kept in mind that the processors never move physically. The movement is in terms of the changes in the weight vector.

6.3 The proposed model for the MSC problem

The present problem considers a set S of $2D$ input points $\{P_j = (a_j, b_j),\ j = 1, 2, \ldots, n\}$ that contains the co-ordinates of the points in S mentioned earlier. The $2D$ co-ordinates are treated as the input signals and thus $m = 2$ in the present situation. Assign one processor to each point of S. Another processor π stores a weight vector $W = (w_1, w_2)$. This weight vector will be updated iteratively in the self-organization process. The weight vector here represents a position in the plane where the processor π can be thought to be located.

We begin with the centre of gravity of all the points in the set S as the initial value of the weight vector W. At iteration t we find the farthest point (w.r.t. Euclidean distance) from $W(t)$. Let $P_k = (a_k, b_k)$ be the farthest point, $k \in \{1, 2, ..., n\}$.

That is,

$$\|P_k - W\| = \max_j \|P_j - W\| \tag{6.4}$$

Then the weight vector W is updated as follows :

$$W(t+1) = W(t) + \alpha(t)(P_k - W(t)) \tag{6.5}$$

The idea is as follows. We start with the centre of gravity of the points as the initial approximation of the centre ($W(0)$) of the MSC. Find the farthest point P_k from $W(0)$. Since we desire to minimize the distance $dist(P_k, W(0))$, we try to move W toward P_k. With a suitable value of α (as discussed later) we update the weight vector W according to Eqn. (6.5). This process is repeated enabling W to gradually move towards P_k. After each weight update the farthest point from W is recalculated. It is clear that during this process, some new point $P_{k'}$ $(k' \neq k)$ becomes the farthest point from the weight vector W and, as soon as it happens, W starts moving toward $P_{k'}$ instead of moving toward P_k. The weight update process is continued repeatedly with $\alpha(t) \to 0$ as $t \to \infty$ enabling the process to stabilize when the difference between the weight vectors at two successive iterations is negligible.

We shall see later that $W(t)$ converges. Suppose $W(t) \to W^*$ as $t \to \infty$. There will be two or more points in S which are farthest from W^*. Let T be the set of such points in S. Thus, if for $P \in T$, $r = ||P - W^*||$, then the circle with radius r and centre at W^*, will be the required MSC.

Definition 6.1 The points in T are called the *contact points* of the MSC.

Denote the number of points in T by $C(T)$. We shall also see that if $C(T) = 2$ then the MSC is determined by the two points in T which form the diameter of the MSC. If $C(T) > 2$ then the MSC is determined by some three points in T where these three points do not lie on an open semicircumference of the MSC.

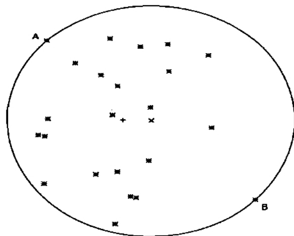


Figure 6.1: The minimum spanning circle determined by two points. '+' represents the initial position of the weight vector and 'x' represents the final centre of the circle ($t = 50$).

Algorithm MSC

Step 1. Initialize iteration number $t = 0$;
$w_1(t) = \frac{1}{n}\sum a_j$ and $w_2(t) = \frac{1}{n}\sum b_j$

Step 2. Calculate $d_j = dist(W(t), P_j)$ for all $j = 1, 2, ..., n$.

Step 3. Find the maximum d_j, and the point P_k such that
$\max_j d_j = dist(W(t), P_k)$.

Step 4. Adjust weight vector according to Eqn. (6.5).

Step 5. Set $t = t + 1$ and repeat from step 2 until the weight vectors of two successive iterations are sufficiently close.

One should note that $\alpha(t)$ should be made a decreasing function of iteration number t. It should decrease neither too fast nor too slow. The former case may cause an early (incorrect) convergence before another point $P_{k'}$ becomes the farthest from W. On the other hand, the latter case may unnecessarily delay the convergence.

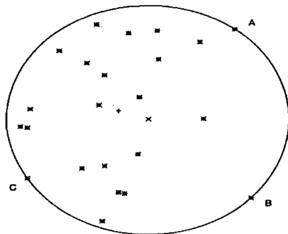


Figure 6.2: The minimum spanning circle determined by three points ($t = 70$).

In the present simulations initial $\alpha(0)$ is taken to be 0.40. Here the function is chosen as $\alpha(t) = \frac{1}{t+1}\alpha(0)$. Mathematical treatments of present model are done in the next section. The algorithm has been tested on a number of point sets and it has converged to the desired centres of the respective smallest circles. Some of the results are presented in Fig.6.1 and Fig.6.2. The input points are chosen randomly.

6.4 Convergence

It can be seen that **Algorithm-MSC** converges so that the centre of the minimum spanning circle can be obtained with any given level of accuracy. Before going into the details of mathematical proofs we need the following definition to be stated.

Definition 6.2. For the set S, the *farthest point Voronoi diagram* (FPVD) is a partition of the plane defined by the convex regions V_r, $1 \leq r \leq n$ where

$$V_r = \{P : \|P - P_r\| > \|P - P_t\| \;\; \forall\, t \neq r\} \tag{6.6}$$

V_r is the farthest point Voronoi polygon corresponding to P_r, $1 \leq r \leq n$. Farthest point Voronoi diagrams are shown in Fig.6.3.

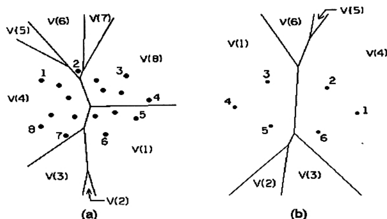


Figure 6.3: Farthest point Voronoi diagrams for two planar sets. MSC is determined by (a) three points, (b) two points.

We shall now prove that in **Algorithm-MSC**, the weight vector $W(t)$ converges to the centre of the minimum spanning circle. Before going into the proof, we state below a few known results ([80, 90, 104]).

Result 6.1. If there is a circle C passing through three points, P_i, P_j and P_k such that they do not lie on any open semicircumference of C and if C contains all the points, then C is the MSC.

Result 6.2. If the MSC passes through exactly two points P_i and P_j, then the line segment P_iP_j forms a diameter of the MSC.

Result 6.3. If no two points of S form a diameter of the MSC, then the MSC passes through at least three points which do not lie on any open semicircumference of the MSC.

Result 6.4. The MSC is unique.

Let $X(t),\ t = 0, 1, 2, \ldots$ be such that $\|X(t) - W(t)\| = \max_i \|P_i - W(t)\|$, $0 < \alpha(t) < 1$ for all t and $\alpha(t)$ decreases to 0 as t tends to ∞ such that $\sum_{t=0}^{\infty} \alpha(t) = \infty$.

Note that the points of S come from a bounded region. Hence, $\|W(0) - P_i\| < K$ for all i, for some $K > 0$. It is easy to see that $\|W(t) - P_i\| < K$ for all i, for all t.

Lemma 6.1. $\prod_{t=0}^{\infty} [1 - \alpha(t)] = 0$.

Proof. Note that since $0 < \alpha(t) < 1$ for all t, $\sum_{t=0}^{\infty} \alpha(t) = \infty$ if and only if $\prod_{t=0}^{\infty} [1 - \alpha(t)] = 0$ (see [4]).

We shall now prove a result on $\alpha(t)$. For any given t_0, let $S(0) = \alpha(t_0)$ and $S(t+1) = [1 - \alpha(t_0 + t + 1)]S(t) + \alpha(t_0 + t + 1)$ for $t = 0, 1, 2, \ldots$.

Note that

$$\begin{aligned}
S(1) &= \alpha(t_0)[1 - \alpha(t_0 + 1)] + \alpha(t_0 + 1) \\
S(2) &= \alpha(t_0)[1 - \alpha(t_0 + 1)][1 - \alpha(t_0 + 2)] + \alpha(t_0 + 1)[1 - \alpha(t_0 + 2)] + \alpha(t_0 + 2) \\
S(3) &= \alpha(t_0)[1 - \alpha(t_0 + 1)][1 - \alpha(t_0 + 2)][1 - \alpha(t_0 + 3)] \\
&\quad + \alpha(t_0 + 1)[1 - \alpha(t_0 + 2)][1 - \alpha(t_0 + 3)] \\
&\quad + \alpha(t_0 + 2)[1 - \alpha(t_0 + 3)] + \alpha(t_0 + 3)
\end{aligned}$$

In general,

$$\begin{aligned}
S(t) &= \alpha(t_0)[1 - \alpha(t_0 + 1)][1 - \alpha(t_0 + 2)] \ldots [1 - \alpha(t_0 + t)] \\
&\quad + \alpha(t_0 + 1)[1 - \alpha(t_0 + 2)] \ldots [1 - \alpha(t_0 + t)] \\
&\quad + \alpha(t_0 + 2)[1 - \alpha(t_0 + 3)] \ldots [1 - \alpha(t_0 + t)] + \ldots \\
&\quad + \alpha(t_0 + t - 2)[1 - \alpha(t_0 + t - 1)][1 - \alpha(t_0 + t)] \\
&\quad + \alpha(t_0 + t - 1)[1 - \alpha(t_0 + t)] + \alpha(t_0 + t)
\end{aligned}$$

Lemma 6.2. $S(t)$ goes to 1 as t goes to ∞.

Proof. It is easy to see that

$$\begin{aligned} 1-S(1) &= [1-\alpha(t_0)][1-\alpha(t_0+1)] \\ 1-S(2) &= [1-S(1)][1-\alpha(t_0+2)] \\ &= [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)] \\ 1-S(3) &= [1-S(2)][1-\alpha(t_0+3)] \\ &= [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)][1-\alpha(t_0+3)] \end{aligned}$$

In general,

$$1-S(t) = [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]$$

From Lemma 6.1, $\prod_{t=t_0}^{\infty}[1-\alpha(t)] = 0$. Hence Lemma 6.2. ■

Applying Eqn.(6.5) repeatedly we get

$$\begin{aligned} W(t_0+t+1) = \; & [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]W(t_0) \\ & +\alpha(t_0)[1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]X(t_0) \\ & +\alpha(t_0+1)[1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]X(t_0+1) \\ & +\alpha(t_0+2)[1-\alpha(t_0+3)]\dots[1-\alpha(t_0+t)]X(t_0+2)+\dots \\ & +\alpha(t_0+t-2)[1-\alpha(t_0+t-1)][1-\alpha(t_0+t)]X(t_0+t-2) \\ & +\alpha(t_0+t-1)[1-\alpha(t_0+t)]X(t_0+t-1) \\ & +\alpha(t_0+t)X(t_0+t) \end{aligned}$$

Case 1 : $n = 1$.

Here, $X(t) = P_1$ for all t. Hence, from above, we get

$$W(t_0+t+1) = [1-S(t)]\, W(t_0) + S(t)\, P_1$$

Lemma 6.3. For Case 1, $W(t_0+t)$ goes to P_1 as t goes to ∞.

Case 2 : $n = 2$. The centre of the MSC is $\frac{1}{2}(P_1+P_2)$.

Lemma 6.4. For Case 2, $W(t_0+t)$ goes to $(P_1+P_2)/2$ as t goes to ∞.

Proof. It is based on Proposition 6.1 and Proposition 6.2 below.

Proposition 6.1. $W(t_0+t+1)$ can be made arbitrarily close to the straight line L passing through P_1 and P_2.

Note that

$$
\begin{aligned}
&P_1 - W(t_0+t+1) \;=\\
&[1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)][P_1 - W(t_0)]\\
&+\alpha(t_0)[1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)][P_1 - X(t_0)]\\
&+\alpha(t_0+1)[1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)][P_1 - X(t_0+1)]\\
&+\alpha(t_0+2)[1-\alpha(t_0+3)]\dots[1-\alpha(t_0+t)][P_1 - X(t_0+2)]+\dots\\
&+\alpha(t_0+t-2)[1-\alpha(t_0+t-1)][1-\alpha(t_0+t)][P_1 - X(t_0+t-2)]\\
&+\alpha(t_0+t-1)[1-\alpha(t_0+t)][P_1 - X(t_0+t-1)]\\
&+\alpha(t_0+t)[P_1 - X(t_0+t)]
\end{aligned}
$$

where X is either P_1 or P_2.

Now,

$$
\begin{aligned}
&\|P_1 - W(t_0+t+1)\|\\
\le\;&[1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]\|P_1 - W(t_0)\|\\
&+\alpha(t_0)[1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]\|P_1 - X(t_0)\|\\
&+\alpha(t_0+1)[1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]\|P_1 - X(t_0+1)\|\\
&+\alpha(t_0+2)[1-\alpha(t_0+3)]\dots[1-\alpha(t_0+t)]\|P_1 - X(t_0+2)\| \;+\;\dots\\
&+\alpha(t_0+t-2)[1-\alpha(t_0+t-1)][1-\alpha(t_0+t)]\|P_1 - X(t_0+t-2)\|\\
&+\alpha(t_0+t-1)[1-\alpha(t_0+t)]\|P_1 - X(t_0+t-1)\|\\
&+\alpha(t_0+t)\|P_1 - X(t_0+t)\|
\end{aligned}
$$

Similarly,

$$\begin{aligned}
& \|P_2 - W(t_0+t+1)\| \\
\leq\ & [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]\|P_2 - W(t_0)\| \\
& + \alpha(t_0)[1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]\|P_2 - X(t_0)\| \\
& + \alpha(t_0+1)[1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)]\|P_2 - X(t_0+1)\| \\
& + \alpha(t_0+2)[1-\alpha(t_0+3)]\dots[1-\alpha(t_0+t)]\|P_2 - X(t_0+2)\| + \dots \\
& + \alpha(t_0+t-2)[1-\alpha(t_0+t-1)][1-\alpha(t_0+t)]\|P_2 - X(t_0+t-2)\| \\
& + \alpha(t_0+t-1)[1-\alpha(t_0+t)]\|P_2 - X(t_0+t-1)\| \\
& + \alpha(t_0+t)\|P_2 - X(t_0+t)\|
\end{aligned}$$

Hence,

$$\begin{aligned}
& \|P_1 - W(t_0+t+1)\| + \|P_2 - W(t_0+t+1)\| \\
\leq\ & [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)][\|P_1 - W(t_0)\| \\
& +\|P_2 - W(t_0)\|] \\
& +\alpha(t_0)[1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)][\|P_1 - X(t_0)\| \\
& +\|P_2 - X(t_0)\|] \\
& +\alpha(t_0+1)[1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)][\|P_1 - X(t_0+1)\| \\
& +\|P_2 - X(t_0+1)\|] \\
& +\alpha(t_0+2)[1-\alpha(t_0+3)]\dots[1-\alpha(t_0+t)][\|P_1 - X(t_0+2)\| \\
& +\|P_2 - X(t_0+2)\|] + \dots \\
& +\alpha(t_0+t-2)[1-\alpha(t_0+t-1)][1-\alpha(t_0+t)][\|P_1 - X(t_0+t-2)\| \\
& +\|P_2 - X(t_0+t-2)\|] \\
& +\alpha(t_0+t-1)[1-\alpha(t_0+t)][\|P_1 - X(t_0+t-1)\| + \|P_2 - X(t_0+t-1)\|] \\
& +\alpha(t_0+t)[\|P_1 - X(t_0+t)\| + \|P_2 - X(t_0+t)\|]
\end{aligned}$$

$$
\begin{aligned}
&= [1-\alpha(t_0)][1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)] \; [\|P_1 - W(t_0)\| \\
&\quad +\|P_2 - W(t_0)\|] \\
&\quad + \; \alpha(t_0)[1-\alpha(t_0+1)][1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)] \; \|P_1 - P_2\|] \\
&\quad + \; \alpha(t_0+1)[1-\alpha(t_0+2)]\dots[1-\alpha(t_0+t)] \; \|P_1 - P_2\| \\
&\quad + \; \alpha(t_0+2)[1-\alpha(t_0+3)]\dots[1-\alpha(t_0+t)] \; [\|P_1 - P_2\| + \dots \\
&\quad + \; \alpha(t_0+t-2)[1-\alpha(t_0+t-1)][1-\alpha(t_0+t)] \; [\|P_1 - P_2\| \\
&\quad + \; \alpha(t_0+t-1)[1-\alpha(t_0+t)] \; \|P_1 - P_2\| \\
&\quad + \; \alpha(t_0+t) \; \|P_1 - P_2\| \\
&= [1-S(t)] \; [\|P_1 - W(t_0)\| + \|P_2 - W(t_0)\|] \quad + \quad S(t) \; \|P_1 - P_2\|
\end{aligned}
$$

Now, $S(t)$ goes to 1 as t tends to ∞. So, $W(t_0+t+1)$ gets arbitrarily close to the line segment joining P_1 and P_2. Hence Proposition 6.1. ■

Proposition 6.2. $W(t_0+t+1)$ can be made arbitrarily close to the perpendicular bisector L_1 of the line segment P_1P_2.

Let H_1 and H_2 be the two half planes defined by L_1 such that P_i belongs to H_i $(i = 1, 2)$. Without loss of generality, let $W(t_0)$ lie in H_1. From Lemma 6.3, we know $W(t_0+t)$ will belong to H_2 for some t. Suppose $W(t_0+1), W(t_0+2), \dots, W(t_0+t_1-1)$ belong to H_1 and $W(t_0+t_1)$ belongs to H_2. Now, the perpendicular distance between $W(t_0+t_1)$ and L_1 is less than or equal to $\|W(t_0+t_1-1) - W(t_0+t_1)\|$. (Fig.6.4)

Note that $\|W(t_0+1) - W(t_0)\| = \alpha(t_0)\|X(t_0) - W(t_0)\| \leq \alpha(t_0)K$.

Since $\alpha(t) \to 0$ as $t \to \infty$, for any $\delta > 0$ we can take t_0 to be sufficiently large so that $\|W(t_0+1) - W(t_0)\| < \delta$ and in general, $\|W(t_0+t) - W(t_0+t-1)\| < \delta$ for all $t > 0$.

Now, $W(t_0+t_1+t)$ will belong to H_1 for some t. Suppose, $W(t_0+t_1), \dots, W(t_0+t_1+t_2-1)$ belong to H_2 and $W(t_0+t_1+t_2)$ belongs to H_1. From Fig.6.4(a), it is clear that the distance between $W(t_0+t_1+t)$ and L_1 is less than δ for all $t > 0$. Hence Proposition 6.2. ■

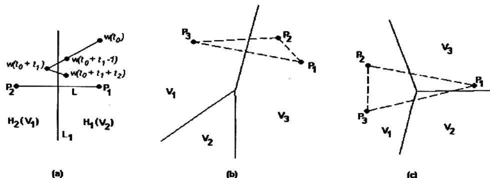


Figure 6.4: The trajectories of $W(t)$. (a) Case 2, (b) Case 3(a), (c) Case 3(b). The FPVD is denoted by solid lines and the triangle formed by the input points is denoted by dashed lines.

Case 3. Suppose $n = 3$. Consider the farthest point Voronoi diagram (FPVD) of the points. The only vertex of the FPVD of P_1, P_2, P_3 may lie either (a) outside (Fig.6.4(b)) or (b) inside (Fig.6.4(c)) the triangle $\Delta P_1P_2P_3$ formed by P_1, P_2, P_3. The FPV polygon corresponding to the point P_i is V_i $(i = 1, 2, 3)$.

Case 3(a). In the former case, for some large t_0, $W(t)$ will lie outside V_2 for all $t > t_0$. From Case-2, it is then clear that $W(t)$ converges to $(P_1 + P_3)/2$ which is the centre of the MSC.

Case 3(b). In the latter case, consider the three perpendicular bisectors L_{ij} of the line segments P_iP_j $(1 \leq i \neq j \leq 3)$. From the arguments given in the proof of Proposition 6.2 in Case-2, $W(t)$ will be arbitrarily close to each of the three L_{ij}'s. Thus $W(t)$ converges to the intersection point of the three bisectors which is the FPV vertex where the FPV polygons V_1, V_2, V_3 meet. Here, the FPV vertex is the centre of the MSC. Note that for any large t_0 and for any i, there will be a $t_i > t_0$ such that $W(t_i)$ lies in V_i. In other words, each V_i will be visited by $W(t)$ infinitely many times.

Denote the limiting value of $W(t)$ by W^*.

Case 4. Suppose $n > 3$.

We claim that W^* will lie either (a) on an open edge (an edge excluding the vertex points) or (b) on a vertex of the FPV diagram of the point set S. Suppose not. Suppose, W^* lies in the interior of some V_p. Then, after some time, $W(t)$'s will always lie in the interior of V_p in which case, the limiting value of $W(t)$ will be P_p (from Case 1). It is a contradiction since V_p cannot contain P_p. Hence we have two subcases:

Case 4(a). The limiting value W^* lies on an open FPV edge. Suppose this edge is the common edge of the two FPV polygons V_i and V_j. That means, for sufficiently large t_0, $W(t)$ will lie either in V_i or in V_j for all $t > t_0$. Moreover, for any t' there will exist t_i, $t_j > t'$ such that $W(t_i)$ and $W(t_j)$ are in V_i and V_j respectively. In other words, both V_i and V_j (and only they) will be visited by $W(t)$ infinitely many times. This case is similar to Case 2 and hence $W^* = \frac{1}{2}(P_i + P_j)$ (see Fig.6.3). Now construct a circle C, centred at W^* passing through P_i, P_j. Obviously, P_iP_j forms the diameter of the circle C and since P_i, P_j are the farthest points from W^*, C contains all the points of S. Hence, by Results 6.1 - 6.4, C is the MSC.

Case 4(b). The limiting value of W^* lies on an FPV vertex. Suppose W^* lies on the common FPV vertex v_{ijk} of three FPV polygons say, V_i, V_j and V_k. That means, for sufficiently large t_0, $W(t)$ will lie either in V_i or V_j or V_k for all $t > t_0$ and each of V_i, V_j, V_k will be visited by $W(t)$ infinitely many times. Then it will be similar to Case 3(b). Note that W^* will be equidistant from P_i, P_j and P_k. Construct a circle C, centred at W^* and passing through P_i, P_j and P_k. We claim that P_i, P_j and P_k do not lie on an open semicircumference of C. Suppose not. Then v_{ijk} falls outside the triangle $\Delta P_iP_jP_k$. Hence, from Case 3(a), W^* cannot lie on v_{ijk} (see Fig.6.4(b)). This is a contradiction since W^* lies on v_{ijk}. Moreover, since P_i, P_j and P_k are the farthest points from v_{ijk}, C contains all the points of S. Hence C is the MSC from Results 6.1 - 6.4.

Summarizing the above we have:

Case-1: If $S = \{P_1\}$ then $W^* = P_1$.

Case-2: If $S = \{P_1, P_2\}$ then $W^* = \frac{1}{2}(P_1 + P_2)$.

Case-3: If $S = \{P_1, P_2, P_3\}$ then W^* is either (a) the midpoint of one of the sides of the triangle $P_1P_2P_3$ or (b) the common FPV vertex of the three FPV polygons V_1, V_2, V_3.

Case-4: If $S = \{P_1, P_2, \ldots, P_n\}$ then

(a) If the MSC is determined by two points P_i, P_j then $W^* = \frac{1}{2}(P_i + P_j)$.

(b) If the MSC is determined by three points P_i, P_j, P_k then W^* = the common FPV vertex of the three FPV polygons V_i, V_j, V_k.

6.5 Discussions

It can be seen that the above model works as a self-organizing process. We present a set of input vectors to a network of processors against which a competition takes place and the network generates some feedback (the maximum d_j and the point P_k in Step 3 of the algorithm). Depending on the feedback the processor π adapts and moves accordingly. If the process is repeated iteratively, the process converges and the processor π positions itself to the desired centre of the MSC. Moreover, the above learning is unsupervised. In brief, from the inputs the network learns without any supervision and finally outputs the MSC. Similar things essentially happen in self-organization.

The present model uses the principle of simple competitive learning rule but it is quite different from the same. It is self-organizing in that it learns the centre and the radius of the MSC in an unsupervised manner. A few comparisons of the present model with Kohonen's model can be stated as follows. In Kohonen's model, all the processors in the network can take part in the weight update process. The present model uses a network of processors where all the processors have fixed weights

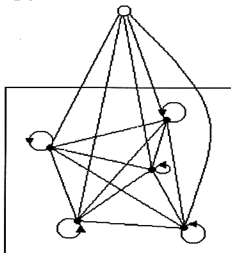


Figure 6.5: The architecture of the MSC model. Solid tiny circles represent point processors.

excepting one i.e., all other processors are fixed while a single (floating) processor takes part in the process of positioning itself in the proper location. Other processors are used for parallel computations. In Kohonen's model, the 'winner' processor moves itself to the input vector presented while in the present model, the winner processor (the processor associated with the point P_k in Step 3 of the algorithm) draws the processor π toward itself (toward its associated input point). Secondly, all the input vectors are presented here simultaneously instead of sequentially presenting them as in Kohonen's model.

Computational cost

The present algorithm works in parallel with the help of multiple processors (see Fig.6.5). Suppose the input points are assigned to one processor each and every processor is connected to every processor forming a network. Let us call these *point processors*. The point processors are *fixed* in the sense that their weight vectors (i.e., their location coordinates) do not change. On the contrary, a processor storing the weight $W(t)$ is *floating* in that its weight is updated at every t. The floating processor is connected to all the point processors. Consider one iteration. All the point processors in the network can simultaneously compute the distances between themselves and the weight vector $W(t)$. Therefore, in Step 2, the distance calculation can be done in constant time. The point processors in the network are also connected by lateral connections. Thus by using a 'maxnet' type model, the time needed to compute the maximum distance does not depend on the input size. Hence Step 3 of the above algorithm has the same time complexity. Thus, in the whole process, complexity does not depend on the size of the input.

6.6 Conclusions

In the last few years, there has been a great interest in applying neural networks technology in various fields of conventional computing [15] and this trend, in turn, has been enriching the neurocomputing technology. This is due to the facts that neurocomputing provides parallelism, it is adaptive and it sometimes simplifies a

problem. A number of processors, each performing simple computation, when work collectively can solve a complex problem. The present work is an application of self-organization principle to compute the minimum spanning circle for a given set of points. An algorithm is proposed to find the centre of the circle with any given degree of accuracy.

The MSC problem has attracted several researchers and as a result varieties of solution exist [35, 38, 50, 80, 83, 103, 104, 105, 106, 110]. The proposed solution is iterative in nature and is a parallel implementation of the problem. The (worst-case) time complexities for the existing solutions range from $O(n^3)$ to $O(n\,logn)$, excepting the work due to Megiddo [78]. The worst-case time complexity of the present algorithm does not depend on the input size when $O(n)$ processors are used. As can be seen, the individual processors perform very simple calculations like computing the Euclidean distance (d_j) here, which is an important phenomenon of neural network models. Moreover, the present work can be straightway implemented for any higher dimension. For a similar problem in d-dimension ($d > 2$) one has to simply calculate the Euclidean distances in d-dimension. The complexity of the algorithm remains the same. In the existing algorithms the time complexities and implementation difficulties increase significantly for higher dimensions.

Chapter 7

CONCLUSIONS

Abstract: *The contributions of the thesis are briefly stated. Some future scopes of work are also outlined.*

7.1 Contributions of the thesis

Artificial neural networks, man-made models for biological neural networks, have been established to be a promising area of research. Artificial neural networks are composed of a number of simple processing elements (called processors, neurons or nodes) interconnected by connection weights which are adaptive in nature. These models are characterized by the characteristics of the neurons, the architecture of the connections and the learning rules. Depending on the learning rules the neural network models can be of two types — unsupervised and supervised. The present thesis has covered a few models of the former type. These models are self-organizing in the sense that they can learn without any supervision.

During last few decades neural network models have been used to solve problems in different fields like pattern recognition, computer vision, image processing, optimization etc. for various reasons. Neural network models provide a new form of parallel computing which is performed by a number of processors simultaneously. These models are capable of adaptation in a similar way as biological systems do.

Another important property of neural network models is that the individual processors needs to do very simple computations and these models are cost effective. Simple processors while working collectively can solve a much more complex problem. Neural network models provide robustness to some extent. Overall performance is not greatly affected due to the damage of a few processors or links and thus such models have sometimes higher fault tolerance. Moreover, neurocomputing is found to work efficiently where conventional computing poses problems or performs poorly.

In the present thesis, we have proposed a few self-organizing neural network models and discussed their applications. The whole work is divided into two divisions according the application domains namely, image processing and computational geometry.

Chapter 1: Introductory discussion on artificial neural network models are done with an emphasis on self-organizing neural network. Various properties and extensions of Kohonen's self-organizing neural network model, published in the existing literature, are described. Some of them have been used to solve various problems in the subsequent chapters.

Chapter 2: A dynamic self-organizing neural network (DySONN) model is proposed here that generates a skeleton of a binary pattern. The proposed model has a few advantages over Kohonen's self-organizing model [64] so far as shape extraction is concerned. The model uses Kohonen's weight update rules but the network here can grow in size by means of processor insertion mechanism. It uses a few heuristics to capture the topology of the input pattern.

Chapter 3: A topology adaptive and dynamically growing self-organizing neural network (TASONN) model is proposed that is especially designed to produce a vector skeleton of a binary pattern. Partly it is similar to Martinetz and Schulten's neural gas network model [73] and Fritzke's growing neural gas network model [41]. The link adaptation and processor insertion, in the proposed model, is different from the said models. The proposed model has a few advantages over the models of Kohonen, Martinetz and Fritzke in view of shape extraction.

Chapter 4: A comprehensive comparative study between the proposed neural net-

work models, discussed in Chapters 2 and 3, and a few existing conventional thinning algorithms are carried out. The study establishes that the proposed neural network based models are highly robust to noise (boundary and interior noise) as compared to existing conventional thinning algorithms and are invariant under arbitrary rotations of the input pattern. They are also efficient in medial axis representation and in data reductions. Moreover, they work on binary patterns, dot patterns and also gray-level patterns. Thus the proposed neural network based models provide unified approaches to skeletonization.

Chapter 5: Computing the convex-hull of a planar point set is a well known problem and several conventional algorithms are available to solve it. We have formulated this problem into a self-organizing neural network model. The network can adapt itself without supervision. It has been found that the network can self-organize to compute the convex-hull of a finite planar set. Complexity of the proposed algorithm is compared with some existing algorithms.

Chapter 6: An efficient and simple technique for computation of the minimum spanning circle for a given finite set of points is proposed. The minimum spanning circle has applications in optimization problems. The concept of self-organizing neural networks is used here. Starting with an initial approximation the process learns from the input iteratively and finally finds the centre of the minimum circle. The complexity of the algorithm is computed and it does not depend on the input size. The technique can be implemented straight in higher dimensions also without any extra complexity.

7.2 Future scopes

7.2.1 Shapes in higher dimensions

The shape extraction problem discussed in Chapters 2-4 can be generalized. In Chapters 2-4, the underlying objects are 2-dimensional. The neural network models, discussed there, for extraction of skeletons from 2-dimensional objects can be

extended to get suitable models for extraction of the skeletal shape of 3-dimensional objects. Thinning 3-dimensional objects has drawn attention in the last few years since 3-dimensional data capture has become easier. 3-dimensional data are now a days available in Magnetic Resonance Imaging (MRI), Confocal Laser Scanning Microscope (CLSM), X-ray Computed Tomography (X-ray CT) etc.

In 3-dimensional extension, the 3-dimensional object points will constitute the set of input vectors. The weight vectors of the neural network will also be 3-dimensional. The weight update rule will remain similar to those discussed in DySONN and TASONN models. In the 3-dimensional DySONN model, the criteria for detecting forks need to be redefined. For loop joining also, the definition of adjacency should be extended to 3 dimensions. In the 3-dimensional TASONN model, however, the extension will be more straightforward.

7.2.2 Extraction of outer shape

In Chapter 5, we have discussed a neural network model for extraction of the convex-hull of a given finite planar set. The convex-hull is often used to describe the shape of a finite set of points. But in many situations, the underlying shape from which the input points emerge is not convex. Edelsbruner et al. [34] have proposed a general definition of the outer shape, convex or otherwise, called the α-hull of a finite planar set. For a finite set of planar points S, the α-hull of S is the complement of the union of all open discs of radius α containing no points of S. Edelsbruner et al. [34] also proposed a technique to compute the α-hull of finite planar set. The computed α-hull (for a given α) provides an approximation of outer shape of a finite planar set even when the underlying shape is not convex. The neural network model in Chapter 5 may be suitably modified to get a model that can extract a similar outer shape from a planar set provided there is no hole in the input pattern.

7.2.3 Extension of minimum spanning circle model

Given a finite planar set S, finding a given number (say, k) of circles of equal radius whose union contains S so that the radius is minimized, is a challenging problem in computational geometry. The proposed MSC model, in Chapter 6, can be useful in developing a self-organizing model to solve this problem. This has application in optimization, shape analysis etc. For example, suppose k TV/radio transmission centres are to be installed to provide service to a community of users so that the maximum distance of any user from its nearest facility is minimized. The minimization here is over k centres and one radius.

The extension of the proposed MSC model to higher dimensions is straightforward as mentioned earlier. This extension can provide solutions to some non-trivial problems. For example, it can be applied to an estimation problem in statistics. Suppose, there are n independent observations from uniform distribution over a p-dimensional hypersphere whose centre and radius are unknown. Here the maximum likelihood estimation on the basis of the n observations is given by the smallest hypersphere containing these observations.

Bibliography

[1] P. K. Agarwal. "Aplications of parametric searching in geometric optimization," *J. Algorithms*, vol. 17, pp. 292-318. 1994.

[2] S. G. Akl and G. T. Toussaint, "A fast convex hull algorithm," *Inform. Processing Lett.*, vol. 7, pp. 219-222, 1978.

[3] S. G. Akl and K. A. Lyons, *Parallel Computational Geometry*, Englewood Cliffs, NJ: Prentice-Hall, 1993.

[4] T. M. Apostol, *Mathematical Analysis*, second edition, Addison-Wesley, 1979.

[5] C. Arcelli and G. Sanniti di Baja, "A width-independent fast thinning algorithm," *IEEE Trans. Patt. Anal. Mechine Intell.*, vol. PAMI-7. no. 4, pp. 463-474, 1985.

[6] C. Arcelli and G. Ramella, "Finding grey-skeletons by iterated pixel removal," *Image Vision Comput.*, vol. 13, no. 3, pp. 159-167, 1995.

[7] J. L. Bently, G. M. Faust and F. P. Preparata, "Approximation algorithm for convex hulls," *Comm. ACM.*, pp. 64-68, 1982.

[8] M. W. Bern, H. L. Karloff and B. Schieber, "Fast geometric approximation techniques and geometric embedding problems," *Theoretical Comput. Sci.*, vol. 106, pp. 265-281, 1992.

[9] P. Bhattacharya and X. Lu, "A width-independent sequential thinning algorithm," *Int. J. Patt. Recogn. Artificial Intell.*, vol. 11, pp. 393-403, 1997.

[10] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford: Clarendon Press. 1995.

[11] H. Blum, "A transformation for extracting new descriptions of shape," in *IEEE Proc. Symp. on Models for the Speech and Vision Form* (Boston), 1964, pp. 362-380.

[12] J. F. Canny, "A computational approach to edge detection," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-8, pp. 679-698, 1986.

[13] V. Capoyleas, G. Rote, and G. Woeginger, "Geometric clustering," *J. Algorithms*, vol. 12, pp. 341-356, 1991.

[14] G. A. Carpenter and S. Grossberg, "ART2: Self-organization of stable category recognition codes for analog input pattern," in *Proc. IEEE Int. Conf. Neural Networks*, San Diego, CA, 1987, pp. II 727-736.

[15] G. A. Carpenter and S. Grossberg, Eds., *Neural Networks for Vision and Image Processing.* MIT Press, 1992.

[16] D. R. Chank and S. S. Kapur, "An algorithm for convex polytopes," *J. ACM*, vol. 17 , pp. 78-86, 1970.

[17] B. B. Chaudhuri and D. Dutta Majumder, *Two-tone Image Processing and Recognition*, Calcutta:Wiley Eastern, 1993.

[18] Y.-S. Chen and W.-H. Hsu, "A comparison of some one-pass parallel thinnings," *Patt. Recogn. Lett.*, vol. 11, no. 1, pp. 35-41, 1990.

[19] R. T. Chin, H.-K. Wan, D. L. Stover and R. D. Iverson, "A one-pass thinning algorithm and its parallel implementation," *Comput. Vision Graphics Image Processing*, vol. 40, pp. 30-40, 1987.

[20] D. Choi and S. Park, "Self-creating and organizing neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 561-575, 1994.

[21] A. Datta and S. K. Parui, "A robust parallel thinning algorithm for binary images," *Pattern Recognition*, vol. 27, pp. 1181-1192, 1994.

[22] A. Datta, S. Pal and N. R. Pal, (a) "A neural network model for 2D convex-hull," accepted in *6th Int. Conf. Neural Infor. Processing* (Perth, Australia), Nov 16-20, 1999; and (b) "A connectionist model for convex-hull of a planar set," revised copy resubmitted to *Neural Networks*.

[23] A. Datta, "Computing minimum spanning circle by self-organization," *Neurocomputing*, vol. 13 , pp. 75-83 , 1996.

[24] A. Datta, T. Pal and S. K. Parui, "A modified self-organizing neural net for shape extraction," *Neurocomputing*, vol. 14, pp. 3-14, 1997.

[25] A. Datta and S. K. Parui, "Skeletons from dot patterns: a neural network approach," *Pattern Recognition Letters*, vol. 18, pp. 335-342, 1997.

[26] A. Datta and S. K. Parui, "A skeleton generating neural network: a prerequisite for character recognition," in *Proc. Int. Conf. Comp. Ling., Speech and Doc. Processing*, Indian Statistical Institute (Calcutta), Feb 18-20, 1998.

[27] A. Datta and S. K. Parui, "Shape extraction: a comparative study between neural network-based and conventional techniques," *Neural Computing and Applications*, vol. 7, pp. 343-355, 1998.

[28] L. S. Davis, "A survey of edge detection techniques," *Comput. Graphics Image Processing*, vol. 4, pp. 248-270, 1975.

[29] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, NJ: Prentice-Hall, 1982.

[30] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, New york: Wiley. 1973.

[31] C. R. Dyer and A. Rosenfeld, "Thinning algorithms for gray-scale pictures," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-1, pp.88-90, 1979.

[32] R. A. Earnshaw. Ed. "Theoretical foundations of computer graphics and CAD," *NATO ASI.*, vol. F40, Berlin: Springer-Verlag, 1988.

[33] W. F. Eddy, "A new convex hull algorithm for planar sets," *ACM Trans. Math. Software*, vol. 3, pp. 398-403, 1977.

[34] H. Edelsbrunner, D. G. Kirkpatrick and R. Seidel, "On the shape of a set of points in the plane," *IEEE Trans. Infor. Theory*, vol. 29, pp. 551-559, 1983.

[35] J. Elzinga and D. E. Hearn, "Geometrical solutions for some minimax location problems," *Transportation Sci.*, vol. 6, pp. 379-394, 1972.

[36] E. Erwin, K. Obermayer and K. Schulten, "Self-organizing maps: ordering, convergence properties and energy functions," *Biological Cybernetics*, vol. 67, pp. 47-55, 1992.

[37] K. C. Fan, D. F. Chen and M. G. Wen, "Skeletonization of binary images with nonuniform width via block decomposition and contour vector matching," *Pattern Recognition*, vol. 31, pp. 823-838, 1998.

[38] R. L. Francis and J. A. White, *Facility Layout and Location*, Prentice-Hall, 1974.

[39] B. Fritzke, "Let it grow - self-organizing feature maps with problem dependent cell structure," in *Artificial Neural Networks* (T. Kohonen et al., Eds.), North-Holland, 1991, vol. 1, pp. 403-408.

[40] B. Fritzke, "Growing cell structures - a self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441, 1994.

[41] B. Fritzke, "A growing neural gas network learns topologies," in *Advances in Neural Information Processing Systems* (G. Tesauro et al., Eds.), MIT Press, Cambridge MA, 1995, vol. 7, pp. 625-632.

[42] J. Ghosh and A. C. Bovik, "Neural networks for textured image processing," in *Artificial Neural Networks and Statistical Pattern Recognition* (I. Sethi et al., Eds.), North-Holland, 1991, pp. 133-154.

[43] J. Ghosh and Y. Shin, "Efficient higher order neural networks for classification and function approximation," *Int. J. Neural Systems*, vol. 3, pp. 323-350, 1992.

[44] J. Ghosh and S. V. Chakravarthy, "The rapid kernel classifier: A link between the SOFM and the Radial Basis Function network," *J. Intelligent Material Systems and Structures*, vol. 5, pp. 211-219, 1994.

[45] J. Ghosh, "Vision-based inspection," in *Artificial Neural Networks for Intelligent Manufacturing* (C. H. Dagli, Ed.), Chapman & Hall, 1994, pp. 265-298.

[46] W. J. Gordon and R. F. Riesenfeld, "B-spline curves and surfaces," in *Computer Aided Geometric Design* (R. E. Barnhill and R. F. Riesenfeld, Eds), Academic Press, New York, 1974, pp. 95-126.

[47] R. L. Graham. "An efficient algorithm for determining the convex hull of a finite planar set," *Inform. Processing Lett.*, vol. 1, pp. 132-133, 1972.

[48] L. Guibas, D. Salesin and J. Stolfi, "Constructing Strongly convex approximate hulls with inaccurate primitives," *Algorithmica*, vol. 9, pp. 534-560, 1993.

[49] R. W. Hall, "Fast parallel thinning algorithms: Parallel speed and connectivity preservation," *Comm. ACM*, vol. 32, no. 1, pp. 124-131, 1989.

[50] D. W. Hearn and J. Vijay, Efficient algorithms for the (weighted) minimum circle problem, *Operation Research*, vol. 30, pp. 777-795, 1982.

[51] J. A. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Reading, MA: Addison-Wesley, 1991.

[52] C. J. Hilditch, "An application of graph theory in pattern recognition," in *Machine Intell.* (B.Meltzer and D.Michie, Eds.), New York:Amer. Elsevier 1968, pp. 325-347, vol. 3.

[53] C. M. Holt, A. Stewart, M. Clint and R. H. Perrott, "An improved parallel thinning algorithm," *Comm. ACM*, vol. 30, no. 2, pp. 156-160, 1987.

[54] Y. K. Hwang and N. Ahuja, "Cross motion planning-A survey," *ACM Comput. Survey*, vol. 24, no. 3, pp. 219-291, 1993.

[55] N. Izzo and W. Coles, "Blood-cell scanner identifies rare cells," *Electron.*, vol.35, pp. 52-55, 1962.

[56] B. K. Jang and R. T. Chin, "Analysis of thinning algorithms using mathematical morphology," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-12, pp. 541-551, 1990.

[57] B. K. Jang and R. T. Chin, "One-pass parallel thinning : analysis, properties, and quantitative evaluation," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-14, no. 11, pp. 1129-1140, 1992.

[58] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Inform. Processing Lett.*, vol. 2, pp. 18-21, 1973.

[59] J. A. Kangas, T. Kohonen and J. Laaksonen, "Variants of self-organizing maps," *IEEE Neural Networks*, vol. 1, pp. 93-99, 1990.

[60] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.

[61] T. Kohonen, "Analysis of a simple self-organizing process," *Biological Cybernetics*, vol. 44, pp. 135-140, 1982.

[62] T. Kohonen, "Clustering, taxonomy and topological maps of patterns," in *Proc. 6th Int. Conf. Patt. Recog.*, Munich, 1982.

[63] T. Kohonen, K. Makisara and T. Saramaki, "Phonotopic maps - insightful representation of phonological features for speech recognition," in *Proc. 7th Int. Conf. Patt. Recog.*, Montreal, 1984.

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

[65] T. Kohonen, "Self-organizing maps: optimization approaches," in *Artificial Neural Networks* (T. Kohonen et al., Eds.), Amsterdam: North Holland 1991, pp. 981-990, vol. II.

[66] B. Kosko, "Stochastic competitive learning," in *Proc. IJCNN-90*, vol. 2, 1990, pp. 215-226.

[67] L. Lam, S. W. Lee and C. Y. Suen, "Thinning methodologies — a comprehensive survey,," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-14, pp. 869-885, 1992.

[68] Y. Leung, J. -S. Zhang and Z. -B. Xu, "Neural networks for convex hull computation," *IEEE Trans. Neural Networks*, vol. 8, pp. 601-611, 1997.

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

[70] H. E. Lu and P. S. Wang, "A comment on a fast parallel algorithm for thinning digital patterns," *Comm. ACM*, vol. 29, pp. 239-242, 1986.

[71] D. Marr and E. C. Hildreth, "Theory of edge detection," in *Proc. R. Soc. Lond.*, vol. 270, Series B, pp. 187-217, 1980.

[72] T. M. Martinetz, H. Ritter and K. J. Schulten, "Three-dimensional neural net for learning visuomotor-coordination of a robot arm," *IEEE Trans. Neural Networks*, vol. 1, pp. 131-136, 1990.

[73] T. M. Martinetz and K. J. Schulten, "A neural gas network learns topologies", in *Artificial Neural Networks* (T. Kohonen et al., Eds.), North-Holland, 1991, vol. 1, pp. 397-402.

[74] T. M. Martinetz, "Competitive Hebbian learning rule forms perfectly topology preserving maps," in *Proc. ICANN'93: Int. Conf. Artificial Neural Networks* (Amsterdam, Netherland), 1993, pp. 427-434.

[75] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, "Neural-Gas network for vector quantization and its application to time-series prediction," *IEEE Trans. Neural Networks*, vol. 4, pp. 558-569, 1993.

[76] T. M. Martinetz, P. Protzel, O. Gramckow and G. Srgel, "Neural network control for rolling mills," in *Proc. Second European Congress on Intelligent Techniques and Soft Computing (EUFIT-94)*, Aachen, vol. I, 1994, pp. 147-152.

[77] T. M. Martinetz and K. J. Schulten, "Topology representing networks," *Neural Networks*, vol. 7, no. 3, pp. 507-522, 1994.

[78] N. Megiddo, "Linear time algorithms for linear programming in R3 and related problems," *SIAM J. of Computing*, vol. 12, pp. 759-776, 1983.

[79] K. Mehrotra, C. K. Mohan and S. Ranka, *Elements of Artificial Neural Networks*, MIT Press, 1997.

[80] R. C. Melville, "An implementation study of two algorithms for the minimum spanning circle problem," in *Computational Geomerty* (G. T. Toussaint, Ed.), North-Holland, 1985, pp. 267-294.

[81] B. Moayer and K. S. Fu, "A syntactic approach to fingerprint pattern recognition," *Pattern Recogn.*, vol. 7, pp. 1-23,1975.

[82] J. L. Mundy and R. E. Joynson, "Automatic visual inspection using syntactic analysis," in *Proc. Int. Conf. Patt. Recogn. Image Processing*, 1977, pp. 144-147.

[83] K. P. K. Nair and R. Chandrasekaran, "Optimal location of a single service center of certain types," *Naval Res. Logist. Quart.*, vol. 18, pp. 503-510, 1971.

[84] J. F. O'Callaghan and J. Loveday, "Quantitative measurement of soil cracking patterns," *Patt. Recogn.*, vol. 5, pp. 83-98, 1973.

[85] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern Recogn.*, vol. 26, pp. 1277-1294, 1993.

[86] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, 1989.

[87] S. K. Parui, A. Datta and T. Pal, "Shape approximation of arc patterns using dynamic neural networks," *Signal Processing*, vol. 42, pp. 221-225, 1995.

[88] T. Pavlidis, "A vectorizer and feature extractor for document recognition," *Comput. Vision Graphics Image Processing*, vol. 35, pp. 111-127, 1986.

[89] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Comm. ACM.*, vol. 20, pp. 87-93, 1977.

[90] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, New York: Springer-Verlag, 1985.

[91] K. Preston, "The CELLSCAN system — A leucocyte pattern analyzer," in *Proc. West. Joint Comput. Conf.* (Los Angeles, CA), 1961, pp. 173-183.

[92] K. Preston, M. J. B. Duff, S. Levialdi, P. E. Norgren, and J. Toriwaki, "Basics of cellular logic with some applications in medical image processing," *Proc. IEEE*, vol. 67, no. 5, pp. 826-857, 1979.

[93] H. Ritter and K. Schulten, "Topology conserving mappings for learning motor tasks," in *Proc. of the AIP Conf., 151: Neural Networks for Computing*, pp. 376-380, 1986.

[94] A. Rosenfeld and J. L. Pfaltz, " Sequential operations in digital picture processing," *Journal of the ACM*, vol.13, no. 4, pp. 471-494, 1966.

[95] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, U.K.: Academic Press, 1982.

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

[97] D. Rutovitz, "Pattern recognition," *J. Roy. Stat. Soc.*, vol. 129, Series A, pp. 504-530, 1966.

[98] M. Sabourin and A. Mitiche, "Modeling and classification of shape using a Kohonen's associative memory with selective multiresolution," *Neural Networks*, vol. 6, pp. 275-283, 1993.

[99] V.D. Sanchez A. and G. Hirzinger, "The state of the art of robot learning control using artificial neural networks - An overview," in *The Robotics Review 2* (O. Khatib, J.J. Craig, and T. Lozano-Perez, Eds.), The MIT Press, Cambridge, 1992, pp. 261-283.

[100] V. D. Sanchez A., "Robustization of a learning method for RBF networks," *Neurocomputing*, vol. 9, pp. 85-94, 1995.

[101] V. D. Sanchez A., "On the design of a class of neural networks," *J. Network Computer Appl.*, vol. 19, pp. 111-118, 1996.

[102] J. T. Schwartz and C. K. Yap, Eds., *Advances in Robotics I: Algorithimic and Geometric Aspects of Robotics*, Hillsdale, NJ: Lawrence Erlbaum, 1987.

[103] M. I. Shamos and D. Hoey, "Closest-point problems," in *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science* (Los Angeles), 1975, pp. 151-162.

[104] M. Shamos, *Problems in Computational Geometry*, Carnegie-Mellon Univ., 1977.

[105] R. Skyum, "A simple algorithm for smallest enclosing circle," *Infor. Proc. Letters*, vol. 37, pp. 121-125, 1991.

[106] R. D. Smallwood, "Minimax detection station placement," *Operation Research*, vol. 13, pp. 636-646, 1965.

[107] R. W. Smith, "Computer processing of line images: a survey," *Pattern Recognition*, vol. 20, pp. 7-15, 1987.

[108] T. Suzuki and S. Mori, "Structural description of line images by the cross section sequence graph," *Int. J. Patt. Recogn. Artificial Intell.*, vol. 7, pp. 1055-1076, 1993.

[109] J. J. Sylvester, "A question in the geometry of situation," *Quart. J. Math.*, vol. 1, p. 79, 1857.

[110] G. Toussaint and B. Bhattacharya, "On geometric algorithms that use the farthest-point Voronoi diagram," *Tech. Rept. SOCS-81.3*, McGill Univ. School of Computer Sci., 1981.

[111] G. T. Toussaint, Ed., *Computational Geometry*, New York: North-Holland 1985.

[112] S. Ubeda, "A parallel thinning algorithm using the bounding boxes techniques," *Int. J. Patt. Recogn. Artificial Intell.*, vol. 7, pp. 1103-1114, 1993.

[113] E. Wennmyr, "A convex hull algorithm for neural networks," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1478-1484, 1989.

[114] Q. Z. Ye and P. E. Danielsson. "Inspection of printed circuit boards by connectivity preserving shrinking," *IEEE Trans Pattern Anal. Mach. Intell.*, vol. 10, no. 5, pp. 737-742, 1988.

[115] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Comm. ACM*, vol. 27, no. 3, pp. 236-239, 1984.

AUTHOR'S PUBLICATIONS RELATED TO THE THESIS

1. A. Datta and S. K. Parui, "A robust parallel thinning algorithm for binary images," *Pattern Recognition*, vol. 27, pp. 1181-1192, 1994.

2. A. Datta, T. Pal and S. K. Parui, "Skeletonization of binary objects by dynamic neural net", in *Proc. 4th Symposium on Intelligent Systems (IEEE Bangalore Section)*, (Bangalore, India), 1994.

3. S. K. Parui, A. Datta and T. Pal, "Shape approximation of arc patterns using dynamic neural networks," *Signal Processing*, vol. 42, pp. 221-225, 1995.

4. A. Datta, S. K. Parui and B. B. Chaudhuri, "Skeletal shape extraction from dot patterns by self-organization," in *Proc. 13th Int. Conf. Pattern Recogn.*, (Vienna, Austria), 1996, vol. IV, pp. 80-84.

5. A. Datta and S. K. Parui, "A dynamic neural net to compute convex-hull," *Neurocomputing*, vol. 10, pp. 375-384, 1996.

6. A. Datta, "Computing minimum spanning circle by self-organization," *Neurocomputing*, vol. 13 , pp. 75-83 , 1996.

7. A. Datta, T. Pal and S. K. Parui, "A modified self-organizing neural net for shape extraction," *Neurocomputing*, vol. 14, pp. 3-14, 1997.

8. A. Datta and S. K. Parui, "Skeletons from dot patterns: a neural

network approach," *Pattern Recognition Letters*, vol. 18, pp. 335-342, 1997.

9. A. Datta and S. K. Parui, "A skeleton generating neural network: a prerequisite for character recognition," in *Proc. Int. Conf. Comp. Ling., Speech and Doc. Processing (Indian Statistical Institute)*, (Calcutta, India), Feb 18-20, 1998, pp. D28-D33.

10. A. Datta and S. K. Parui, "Shape extraction: a comparative study between neural network based and conventional techniques," *Neural Computing and Applications*, vol. 7, pp. 343-355, 1998.

11. A. Datta, S. Pal and N. R. Pal, "A neural network model for 2D convex-hull," accepted in *6th Int. Conf. Neural Infor. Processing*, Perth, Nov 16-20, 1999.

12. A. Datta, S. K. Parui and B. B. Chaudhuri, "Skeletonization by a topology adaptive self-organizing neural network," communicated to *Pattern Recognition.*

13. A. Datta, S. Pal and N. R. Pal, "A connectionist model for convex-hull of a planar set," revised copy resubmitted to *Neural Networks.*