Skeletons from dot patterns: A neural network approach

Amitava Datta [a], S.K. Parui [b,*]

[a] *Computer and Statistical Service Centre, Indian Statistical Institute, Calcutta 700 035, India*
[b] *Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India*

Received 5 December 1995; revised 11 September 1996

Abstract

Boundary detection is a well studied problem in the context of shape extraction from dot patterns and digital images. For images, particularly binary images, another frequently encountered issue is finding the skeleton of the object. Unfortunately, in the case of dot patterns, the skeletonization problem has not received much attention due to the lack of a proper definition of a dot pattern skeleton. We present a method, using artificial neural networks, to extract the skeletal shape of a dot pattern and demonstrate that the skeleton thus obtained is close to the perceptual skeleton. The neural network model proposed here is a modified version of Kohonen's self-organizing model. It is dynamic in the sense that processors can be inserted (or deleted) during the learning process. Unlike in Kohonen's map, the number of processors here need not be known a priori.

Keywords: Dot patterns; Shape extraction; Skeletonization; Neural network; Self organization

1. Introduction

Shape extraction from visual patterns in two dimensions has been widely studied in the field of pattern recognition and computer vision. Such patterns can be in the form of an image or they can be dot patterns. A suitable representation of shape of an object is often a prerequisite for recognition and classification of the object and hence the shape extraction problem has received considerable attention. Several techniques are developed for detecting the outer boundary of dot patterns (Preparata and Shamos, 1985; Edelsbrunner et al., 1983; Parui et al., 1993), for example, convex-hulls and alpha-hulls. But hardly any work has been done on skeletal shape of dot patterns excepting the "MST skeleton" mentioned in (Zahn, 1971). On the other hand, a number of techniques, known as *thinning*, have been developed for finding skeletons of image patterns (particularly binary images) (Lam et al., 1992; Datta and Parui, 1994).

For a dot pattern, unlike in the case of a binary image, a skeleton cannot be properly defined. But yet the human visual system can extract the perceptual skeleton from a dot pattern. For example, a dot pattern having a definite shape (say, "S"-like) is recognised by the human brain almost as easily as for a binary image having the same shape. But the conventional thinning algorithms that extract skeleton 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 the recent past, neural network technology has

been showing a great deal of promise in areas where conventional computing poses problems. Several neural network models have been proposed so far (Rumelhart and McClelland, 1986; Pao, 1989; Carpenter and Grossberg, 1992; Lippmann, 1987) for various applications. Neural network models or simply "neural nets" are massively parallel interconnections of computational elements (processors) that work as a collective system and thus provide a new form of parallel computing. Starting from an initial set of weights (usually random) these rules indicate how to adapt or adjust the initial weights to improve performance. Thus parallelism and adaptation are added advantages of neural network models. The present work uses a neural network based approach to get a piecewise linear approximation of a skeleton of a dot pattern. It is found that even without a proper definition of a skeleton the proposed technique is able to generate skeletons that are quite close to the perceptual skeleton.

In the present work, a dynamic self-organizing neural network model is proposed for extraction of shape from a 2D dot pattern (also called *a planar set*) in the form of a skeleton. In Kohonen's self-organizing model (Kohonen, 1989), the set of processors and their neighbourhoods are fixed (topologically) and do not change during learning. This may pose problems in many situations. For example, when the set of input vectors represents a prominent shape, it may happen that the reference vectors or weight vectors lying in zero-density areas are affected by input vectors from all the surrounding parts of the non-zero distribution (Kangas et al., 1990). This is particularly true for a problem like shape extraction which needs dynamic change in network topology. In the present paper, to overcome the limitations of Kohonen's model, we suggest some modifications of it in which the set of processors and their neighbourhoods change adaptively during learning, in order to extract the shape of a dot pattern in the form of a skeleton.

In the neural network model discussed here, the weight updating rules are similar to those in Kohonen's self-organizing feature map. But during the learning process new processors can be added to or old processors can be deleted from the network. The initial topology here is simple (linear with a small number of processors) which grows adaptively into a more complex topology to deal with higher order structures. Similar types of dynamic neural networks have been suggested by several authors for vector quantization (Choi and Park, 1994), estimation of probability distributions in the plane (Fritzke, 1991) and shape classification (Sabourin and Mitiche, 1993).

In Section 2.1 we propose a neural network model for simple patterns like arcs. In Section 2.2 we propose a modification of the model to deal with more complicated patterns having branchings, forks and crossings (we call them together tree-like patterns). Still another modification is made on the model in Section 2.3 to consider patterns that contain loop structures. In all these cases, the starting structure 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. The formal algorithm and its justification are given in Section 2.4. Results and conclusions are given in Section 3.

2. The network model and learning algorithm

Kohonen's feature mapping net 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 net of processors is normally represented in either one or two dimensions. The map 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 $\{\pi_1, \pi_2, \ldots, \pi_n\}$. The *neighbourhood* N_i of the processor π_i is $\{\pi_p \mid \pi_p$ is connected to $\pi_i\}$ which includes π_i. The processor π_i has a weight vector W_i whose dimension is m. 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 updating rule for the weight vectors is as follows.

At time instance t, let P_j be presented to the net and let $W_i(t) = (w_{i1}(t), w_{i2}(t), \ldots, w_{im}(t))$ be the nearest weight vector to P_j. Then, the weight vectors of the processors within the neighbourhood of π_i are updated in the following way (Kohonen, 1989):

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)],$$
$$\text{for } \pi_p \in N_i, \quad 0 < \alpha(t) < 1, \tag{1}$$

where the starting weight vectors $W_p(0)$ are chosen at random and $0 < \alpha(t) < 1$ is the gain term at time

t satisfying (i) $\alpha(t)$ decreases to 0 as t tends to ∞, (ii) $\sum \alpha(t) = \infty$ and (iii) $\sum \alpha^2(t) < \infty$. These conditions ensure that $W_i(t)$ converges.

2.1. Arc patterns

We now deal with input patterns having an arc shape, like character patterns "C", "L", "N", "S" (Parui et al., 1995). The structure of these patterns can be represented by a linear structure. We start with a net having a linear structure 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 coordinates of the points of the planar set and hence $m = 2$. $S = \{P_1, P_2, \ldots, P_N\}$ is a set of N points 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 tth iteration. Let $\text{dist}(P_j, W_k(t)) = \min_i[\text{dist}(P_j, W_i(t))]$ where $W_i(t)$ is the ith weight vector at the tth iteration. In other words, π_k is the processor whose weight vector is the closest to P_j. P_j updates the weight vectors in the following way:

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)], \quad \text{for } p = k-1, k, k-1. \tag{2}$$

If this updating continues then the weights tend to approximate the distribution of the input vectors in an orderly fashion. Note that the processors do not move physically during updating. It is the weight vectors 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, that is, when

$$|W_i(t) - W_i(t')| < \varepsilon \quad \text{for all } i, \tag{3}$$

where t and t' are the iteration numbers at the end of two consecutive sweeps and ε is a predetermined small positive quantity. Only after a phase is completed, are processors inserted or deleted. Suppose, at the end of the sth 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 sth phase and t_s is the total number of iterations needed to reach the end of the sth phase. If now the weight vectors of two neighbouring processors become very close, the processors are merged. If their weight vectors are far apart, a processor is inserted between them. More formally, if

$$|W_k(t_s) - W_{k-1}(t_s)| = \min_{i=1,\ldots,n(s)-1} |W_i(t_s) - W_{i-1}(t_s)| < \delta_1, \tag{4}$$

then the two processors π_k and π_{k+1} are merged and the new processor has the weight vector $[W_k(t_s) + W_{k+1}(t_s)]/2$. If, on the other hand,

$$|W_i(t_s) - W_{i+1}(t_s)| = \max_{i=1,\ldots,n(s)-1} |W_i(t_s) - W_{i+1}(t_s)| > \delta_2, \tag{5}$$

then one processor is inserted between π_i and π_{i+1} and the new processor has the weight vector $[W_i(t_s) + W_{i+1}(t_s)]/2$. Note that δ_1 and δ_2 are two predetermined positive quantities such that $\delta_1 < \delta_2$. After the insertion and merging of processors, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$\delta_2 > |W_i(t_s) - W_{i+1}(t_s)| > \delta_1 \quad \text{for all } i. \tag{6}$$

Condition (6) means that the weight vectors of no two neighbouring processors are either too close or 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. 1).

2.2. Tree-like patterns

Here we deal with patterns that have branchings, forks or crossings (we term these features as junctions). For example, consider character patterns "T", "X", "Y". Here it is required that a processor in the net 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 junction 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 degree of the processors.

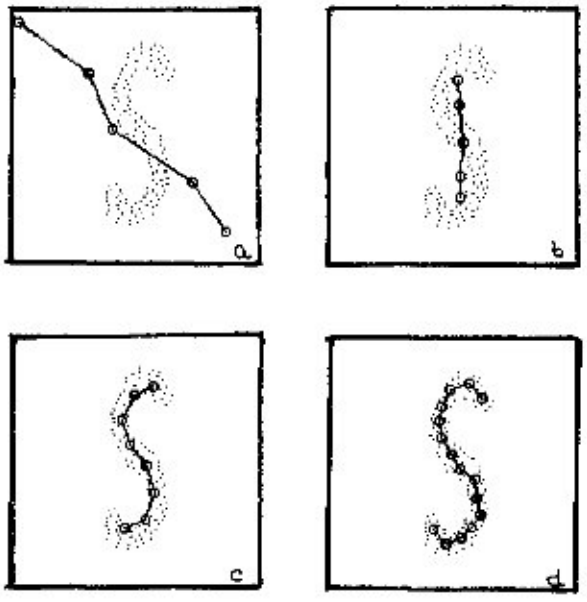

Fig. 1. Different steps of convergence of the net for an arc pattern "S". Circles represent processors and lines joining them represent links. (a) Initial net. (b) the net after 19 sweeps. (c) the net after 86 sweeps. (d) the final net after 261 sweeps.

Let us consider a pattern with a fork. As initially we do not have any topological information about the pattern, we start with a linear net with five processors (shown in Fig. 2(a)). After a number of iterations some processor forms a significantly small (decided on the basis of some threshold) acute angle with its two neighbours (Fig. 2(b)). This happens because by a property of Kohonen's feature map, the net tries to span the entire distribution of the input pattern and, also, the topological relationship of the pattern is preserved in the net. Suppose processor X forms an acute angle with its neighbouring processors Y and Z. This suggests that there is a junction lying between Y and Z. Hence we take the following actions (Fig. 2(c)) when processor X forms an acute angle with its neighbours.

Action 1.

(a) Create a new processor, say U (denoted by a solid circle), halfway between Y and Z.
(b) Delete the link between X, Y and the link between X, Z.
(c) Establish links between U, X; between U, Y and between U, Z.

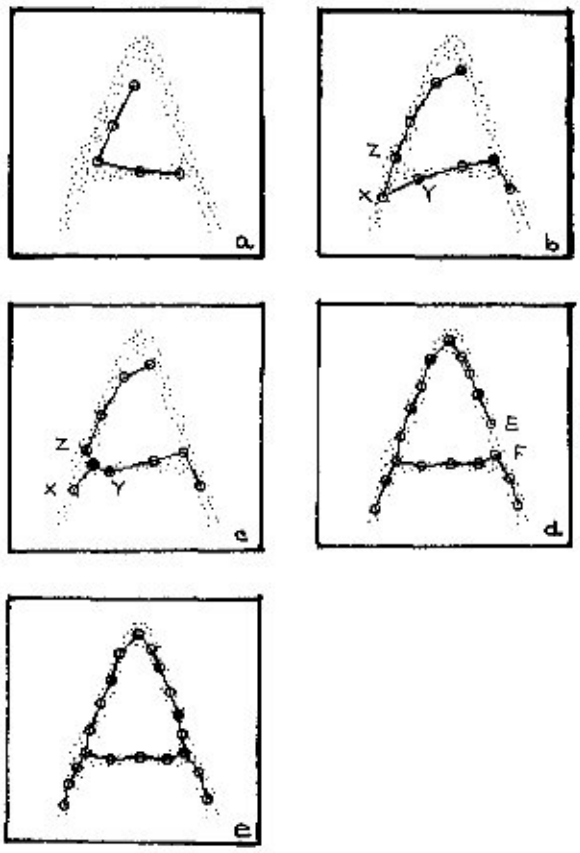

Fig. 2. Different steps of convergence of the net for pattern "A". (a) After 12 sweeps. (b) formation of an acute angle after 94 sweeps. (c) after 105 sweeps with a new processor (solid circle) created with degree = 3. (d) after 394 sweeps immediately before loop formation. (e) final net after 518 sweeps.

Thus a new processor U, with a higher degree which corresponds to a junction in the pattern, is created. The same actions are taken for all the processors forming a significantly small acute angle. These actions are taken after a phase is complete, to adapt the topology of the input and then the subsequent phases are continued to enable the net to approach towards a closer approximation of the shape of the input pattern. Similar principles used in the case of arc patterns are followed for insertion and deletion of processors and for convergence of the algorithm. For this the conditions (4), (5) and (6) are modified to accommodate all possible pairs of neighbouring processors as follows. If

$$|W_k(t_s) - W_{k'}(t_s)| = \min_{i=1,\ldots,n(S)} \min_{\pi_{i'} \in N_i - \{\pi_i\}} |W_i(t_s) - W_{i'}(t_s)| < \delta_1, \tag{7}$$

then the two processors π_k and $\pi_{k'}$ are merged and the new processor has the weight vector $[W_k(t_s) + W_{k'}(t_s)]/2$. If, on the other hand,

$$|W_i(t_s) - W_{i'}(t_s)| = \max_{i=1,\ldots,n(S)} \max_{\pi_{i'} \in N_i - \{\pi_i\}} |W_i(t_s) - W_{i'}(t_s)| > \delta_2, \tag{8}$$

then one processor is inserted between π_i and $\pi_{i'}$ and the new processor has the weight vector $[W_i(t_s) + W_{i'}(t_s)]/2$. After the insertion and merging of processors, the next phase starts with the new set of processors. The process continues until, at the end of a phase,

$$\text{for all } i, \quad \delta_2 > |W_i(t_s) - W_{i'}(t_s)| > \delta_1, \quad \forall \pi_{i'} \in N_i - \{\pi_i\}. \tag{9}$$

2.3. Loop patterns

The techniques discussed so far do not work for patterns that contain loops (for example, character patterns "A", "B", "P"). Let us look at pattern "A". Our algorithm can generate, on the basis of the principles discussed in Section 2.2, an incomplete skeleton as shown in Fig. 2(d). We are yet to complete the loop by means of bridging the gap (between processors E and F in Fig. 2(d)). This subsection proposes a method of doing this.

The asymptotic values of the weight vectors constitute some kind of vector quantization (Kohonen, 1989). In particular, the distance measure and the updating rules as considered in our algorithm, induce a partition of the input pattern space as

$$S_i = \{P_j \quad \text{dist}(P_j, W_i) \leqslant \text{dist}(P_j, W_k)\ \forall k\}.$$

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. In fact, by properties of Kohonen's model, two processors will be neighbours 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 processors having close weight vectors should be neighbours. 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_2. Formally, the loop joining step can be stated as follows.

Action 2. For every processor, its nearest processor among other processors (excepting its neighbours) is found. If the distance between these two processors is less than δ_2, then they are joined by a new link.

Note that after Action 2, the weight updating process continues until condition (9) is satisfied.

2.4. The algorithm

Summarizing the above discussion for all the three types of patterns, the proposed algorithm can be briefly stated as follows:

Step 1. Initialize t = 0;
  Initialize the weight vectors $W_i(t), i = 1, 2, \ldots, n$ with random values.
Step 2. For all input patterns $P_j, j = 1, 2, \ldots, N$, update weight vectors according to rule (2).
Step 3. If condition (3) is false goto Step 2.
Step 4. Merge or insert according to (4) or (5).
Step 5. If condition (6) is false goto Step 2.
Step 6. If no processor forms a spike goto Step 9.
Step 7. Create a new processor for each spike by Action 1.
Step 8. If condition (9) is false goto Step 2.
Step 9. If condition in Action 2 is false goto Step 13.
Step 10. Join each such pair of processors by Action 2.
Step 11. If condition (9) is false goto Step 2.
Step 12. Goto Step 6.
Step 13. Stop.

For arc patterns, it is easy to see that the resulting net, 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 (Kohonen, 1989, 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 either a processor is deleted or two consecutive processors are merged to a single one. In the process of insertion and merging, the existing global ordering of the processors is never disturbed. Note that each phase here can be regarded as a full execution of Kohonen's algorithm because each phase starts with a given number of processors and stops with the same number of processors. Thus the only difference between our model and the original model is that the former is a repetitive application of the latter, everytime increasing/reducing the size of the net without disturbing the global ordering. From the fact that once the processors are ordered they remain so for all t (Kohonen, 1989, p. 143), the output net in the proposed model will give the skeletal shape of the 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 net is replaced by a "T"-like structure locally. The starting net being linear, a spike here represents a junction in the local neighborhood of the pattern. The method is repeated after each phase. For an "X"-like junction two such replacements are required. For other parts of the pattern the argument holds as well since a tree-pattern is a union of arc patterns.

For loop patterns, the algorithm (upto Step 8) yields only a tree-structured net. Subsequently loops are formed on the basis of δ_2. If two processors are closer than δ_2, but not already joined, they are joined (Fig. 2(d,e)). The value of δ_2 is used here since for all other processors we allow maximum distance δ_2 between two neighbouring processors.

3. Results and conclusions

The above method has been tested on several character patterns. Some of the results are presented here. Figs. 1–4 show the intermediate steps and the final output skeletons for the "S"-, "A"-, "X"- and "a"-shaped patterns, respectively. The input dot patterns are generated from a uniform distribution over the underlying shapes. In the figures, a circle represents a processor and a line joins two circles if the corresponding processors are neighbours. For all the test patterns, the initial value of α is taken as 0.01 and the initial net

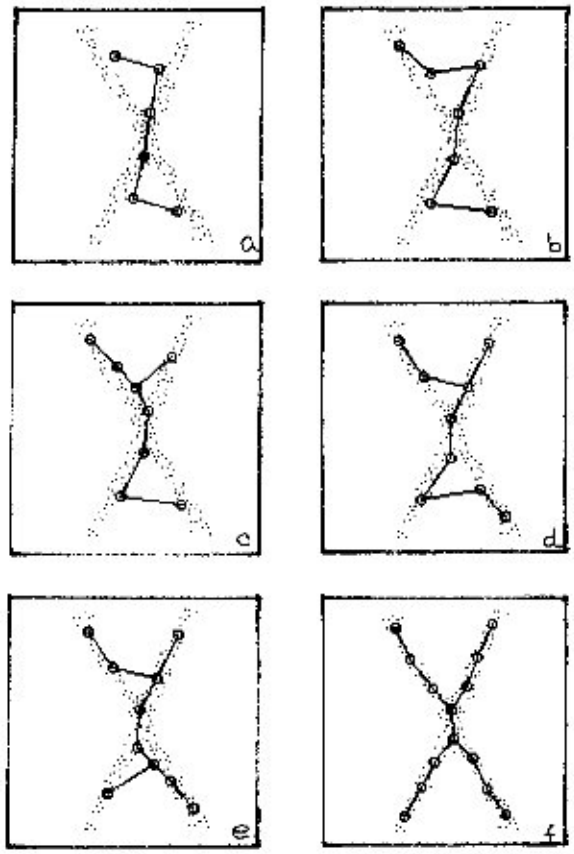

Fig. 3. Different steps of convergence of the net for pattern "X". (a) After 54 sweeps, (b)–(e) intermediate stages, (f) final result after 575 sweeps.

(Fig. 1(a)) is the same. The value of α is changed over time as $\alpha(t) = 0.01/(1+t/1000)$, where t is the number of iterations. The value of ε is taken as 0.001.

It is to be noted that the parameters δ_1 and δ_2 control the distances between two neighbouring processors. These distances are larger (and the processors are fewer in number) for higher values of δ_1 and δ_2 and smaller (and the processors are more in number) for lower values of δ_1 and δ_2. Fig. 5 demonstrates the sensitivity of the output skeleton to the values of δ_1 and δ_2. Note that small values of δ_1 and δ_2 cause a zigzag skeleton (Fig. 5(a)) and large values of δ_1 and δ_2 produce only a crude approximation of the shape of the pattern (Fig. 5(c)). The optimum values for the input pattern in Fig. 5 are δ_1=6 and δ_2=12 (Fig.5(b)). The proper values of δ_1 and δ_2 depend on the thickness of the pattern and not on its density or radius.

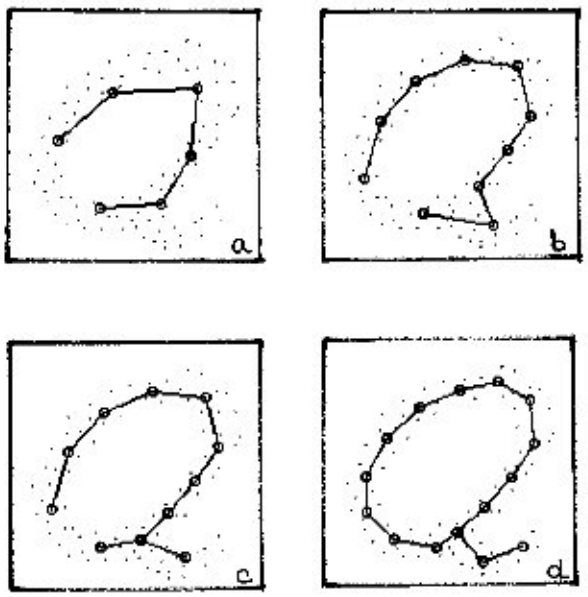

Fig. 4. Different steps of convergence of the net for pattern "a". (a) After 42 sweeps, (b)–(c) intermediate stages, (d) final net after 551 sweeps.

In the present work, δ_2 is always taken as $2\delta_1$ and δ_1 is approximately the thickness of the pattern. The parameters δ_1 and δ_2 are chosen manually and their optimum values are judged subjectively. Automatic determination of the optimum or near optimum values of these parameters is a topic of further research.

Higher values of ε may lead to a less accurate approximation of the shape of the pattern and its smaller values give more accurate approximation of the shape though the time taken for convergence will be longer. One possible value of ε that will work is half of the average nearest neighbour distance in the dot pattern.

A skeleton, representing a region-based global shape of an object in a binary image, is well-defined and can be computed in several ways. On the contrary, skeletons for dot patterns are not well-defined and hardly any technique is available to find a skeletal shape from a dot pattern. The lack of a proper definition for a dot pattern skeleton poses problems for its computation. The present work demonstrates that a neural network based technique to find such a skeleton can overcome these problems. This technique is able to produce a skeleton which is very close to the perceptual skeleton of the input pattern.

We have proposed certain modifications of Kohonen's self-organizing neural network model to get an applicable model for shape extraction from dot patterns. The initial net has the simplest possible structure and a small number of processors. Such a structure can deal with arc patterns only. We explain how to update the net structure to deal with tree-like and loop patterns also. In all these cases, in the initial net, each processor (except the two end ones) has only two neighbours. During the self-organizing process, some processor may acquire more than two neighbours, if

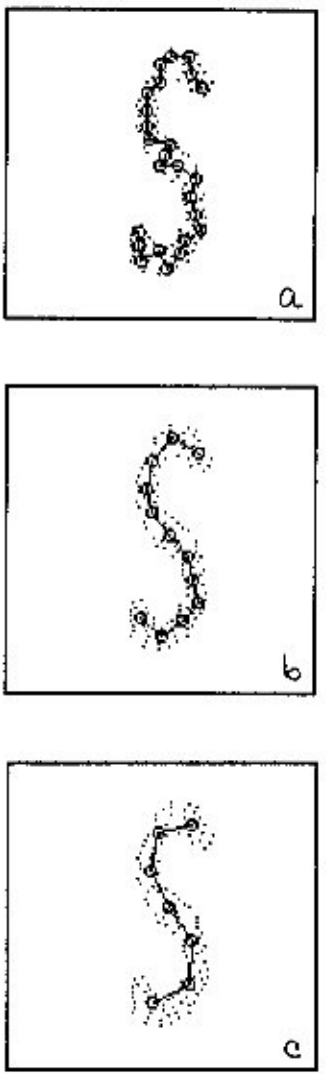

Fig. 5. Sensitivity of the output skeleton to the values of δ_1 and δ_2. (a) $\delta_1 = 4$ and $\delta_2 = 8$, (b) $\delta_1 = 6$ and $\delta_2 = 12$, (c) $\delta_1 = 9$ and $\delta_2 = 18$.

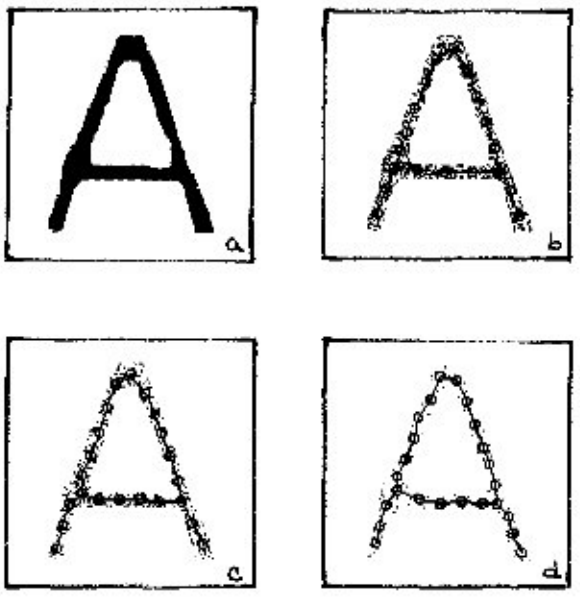

Fig. 6. (a) Original binary image, (b) NSR = 0.5, (c) NSR = 0.7, (d) NSR = 0.9.

necessary, and in this way, adapt their neighbourhoods dynamically on the basis of the input. Thus, not only the net grows in size but also the neighbourhoods of the processors change according to the local topology of the input pattern.

The proposed algorithm would be useful in skeletonization of binary images in the presence of noise. The conventional thinning algorithms are susceptible to noise. Although some of them can tackle boundary noise (Datta and Parui, 1994), they fail to preserve the essential topology of the pattern in the presence of noise in the interior of the object. For example, a noise pixel in the interior of the object normally leads to the creation of a hole in the output skeleton. On the contrary, the proposed algorithm is robust with respect to such noise pixels even when the noise-to-signal ratio (NSR) is very high where the NSR is defined as

$$\mathrm{NSR} = \frac{\text{Number of noise pixels inside object}}{\text{Number of object pixels}}. \quad (10)$$

When the NSR is very high, a binary object becomes a dot pattern. Hence the proposed algorithm for dot patterns can be used in such situations. It is found that even in the presence of 90% random noise, the output skeleton represents the skeletal shape of the original pattern (Fig. 6).

References

Carpenter, G.A. and S. Grossberg, eds. (1992). *Neural Networks for Vision and Image Processing*. MIT Press, Cambridge, MA.

Choi, D. and S. Park (1994). Self-creating and organizing neural networks. *IEEE Neural Networks* 5, 561–575.

Datta, A. and S.K. Parui (1994). A robust parallel thinning algorithm for binary images. *Pattern Recognition* 27, 1181–1192.

Edelsbrunner, H., D.G. Kirkpatrick and R. Seidel (1983). On the shape of a set of points in the plane. *IEEE Trans. Inform. Theory* 29, 551–559.

Fritzke, B. (1991). Let it grow – Self-organizing feature maps with problem dependent cell structure. In: T. Kohonen et al., eds., *Artificial Neural Networks*. North-Holland, Amsterdam, Vol. 1, pp. 403–408.

Kangas, J.A., T. Kohonen and J. Laaksonen (1990). Variants of self-organizing maps. *IEEE Neural Networks* 1, 93–99.

Kohonen, T. (1989). *Self-Organization and Associative Memory*. Springer, Berlin.

Lam, L., S.W. Lee and C.Y. Suen (1992). Thinning methodologies – A comprehensive survey. *IEEE Trans. Pattern Analysis and Machine Intelligence* 14, 869–885.

Lippmann, R.P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4–22.

Pao, Y.H. (1989). *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA.

Parui, S.K., S. Sarkar and B.B. Chaudhuri (1993). Computing the shape of a point set in digital images. *Pattern Recognition Lett.* 14 (2), 89–94.

Parui, S.K., A. Datta and T. Pal (1995). Shape approximation of arc patterns using dynamic neural networks. *Signal Processing* 42, 221–225.

Preparata, F.P. and M.I. Shamos (1985). *Computational Geometry: An Introduction*. Springer, New York.

Rumelhart, D.E. and J.L. McClelland, eds. (1986). *Parallel Distributed Processing*. MIT Press, Cambridge, MA, Vol. 1.

Sabourin, M. and A. Mitiche (1993). Modeling and classification of shape using a Kohonen associative memory with selective multiresolution. *Neural Networks* 6, 275–283.

Zahn, C.T. (1971). Graph-theoretical methods for detecting and describing Gestalt clusters. *IEEE Trans. Comput.* 20, 68–86.