Neural
Networks

Neural Networks 13 (2000) 377–384

Contributed article

A connectionist model for convex-hull of a planar set

A. Datta, S. Pal, N.R. Pal*

Computer and Statistical Service Centre, Electronics and Communication Sciences Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700 035, India

Received 18 March 1999; accepted 20 January 2000

Abstract

A neural network model is proposed for computation of the convex-hull of a finite planar set. The model is self-organizing in that it adapts itself to the hull-vertices of the convex-hull in an orderly fashion without any supervision. The proposed network consists of three layers of processors. The bottom layer computes the activation functions, the outputs of which are passed onto the middle layer. The middle layer is used for winner selection. These information are passed onto the topmost layer as well as fed back to the bottom layer. The network in the topmost layer self-organizes by labeling the hull-processors in an orderly fashion so that the final convex-hull is obtained from the topmost layer. Time complexities of the proposed model are analyzed and are compared with existing models of similar nature.

Keywords: Neural networks; Self-organization; Connectionist model; Planar set; Convex-hull

1. Introduction

In the present paper 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 two-dimension (2D) has been a topic of research for a long time. 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 of a planar set can be easily understood with the help of rubber band: stretch a rubber band to surround the set of points and then release it to shrink. On equilibrium, the rubber band defines the convex-hull.

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 (Agarwal, 1994; Akl & Lyons, 1993; Capoyleas, Rote & Woeginger, 1991; Devijver & Kittler, 1982; Duda & Hart, 1973; Earnshaw, 1988; Edelsbrunner, Kirkpatrick & Seidel, 1983; Hwang & Ahuja, 1993; Preparata & Shamos, 1985; Schwartz & Yap, 1987; Toussaint, 1985). Several conventional algorithms are available for this problem. What we aim at here is to formulate the convex-hull problem as an artificial neural network problem. We discuss how an interconnected set of neurons can iteratively self-organize itself on the basis of the input signals (points) to finally arrive at the convex-hull of the input points.

Since the 1970s, 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 (Akl & Toussaint, 1978; Bentley, Clarkson & Levine, 1993; Chank & Kapur, 1970; Eddy, 1977; Graham, 1972; Jarvis, 1973; Preparata & Hong, 1977; Wennmyr, 1989); and computing approximate convex-hull (Bently, Faust & Preparata, 1982; Bern, Karloff & Schieber, 1992; Guibas, Salesin & Stolfi, 1993; Leung, Zhang & Xu, 1997). 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 (2D or 3D which are the most common ones in real life) and some algorithms deal with higher dimensional input also. While processing, either all points can be presented together in a batch (the *off-line* mode) or they can be presented one by one (the *on-line* mode).

The algorithms in Leung et al. (1997) and Wennmyr (1989) are designed on artificial neural networks. Wennmyr

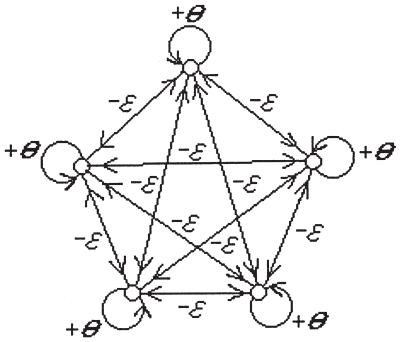

Fig. 1. Maxnet architecture.

(1989) proposed an exact convex-hull computation algorithm based on a multi-layer perceptron (Lippmann, 1987; Rumelhart & McClelland, 1986). The author designs a network that can decide whether a given point is inside a convex polygon (using the fact that a convex polygon is always the intersection of half-planes). In this network, every node at the lowest level has a different decision boundary. Leung et al. (1997) proposed an algorithm for the computation of an approximate convex-hull. The network consists of an input layer and an output layer of neurons. Similar to the adaptive resonance theory (ART) (Carpenter & Grossberg, 1987; Serrano-Gotarredona, Linares-Barranco & Andreou, 1998), the two layers of neurons can communicate via feedforward as well as feedback connections.

In this paper, we propose a self-organizing neural network model for computation of the convex-hull of a finite planar set. The term 'self-organization' here refers to the ability to learn from the input without having any prior supervising information and arrange (or order) the processors accordingly. It is argued that self-organization works as a basic principle of sensory paths of the human visual system. In this case, self-organization means a meaningful ordering of the neurons (processors) in the brain, where the 'ordering' does not mean moving of neurons physically. A well-known self-organizing neural network model is Kohonen's (1989) feature maps.

We also find dynamic versions of Kohonen's model proposed by different researchers where the network can grow in size. Sabourin and Mitiche (1993) proposed a "selective multi-resolution" approach in the context of shape classification. Such a dynamic model does not require a priori knowledge of the number of processors. Another dynamic version of Kohonen's model has been suggested by Fritzke (1991) to model the probability distribution in the plane. In the context of vector quantization, Choi and Park (1994) have proposed a dynamic version of Kohonen's model.

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 recognized. The learning takes place without any supervision.

2. The model

Before going into the proposed model, a network to compute the maximum (or minimum) of n given values will be discussed as it is used by our model. Such a network, called 'maxnet', is available in existing literature (Mehrotra, Mohan & Ranka, 1997) but we briefly describe it next for the sake of completeness.

2.1. Maxnet

The 'maxnet' is a recurrent network. It has only one layer of n nodes (Fig. 1) which compete to determine the node with the highest initial value. Every node is connected to every node including itself. 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 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

$$o_j = \max\left(0, \sum_{i=1}^{n} w_{ji} x_i\right). \tag{1}$$

where x_is ($i = 1, 2, ..., n$) are the inputs 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. In Fig. 1, the self-excitation weight $w_{jj} = \theta = 1$ and the inhibition weights w_{ji} ($i \neq j$) are $\epsilon \leq 1/n$.

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 number of data points n.

2.2. The proposed model

In the present model, we consider a set of n 2D points

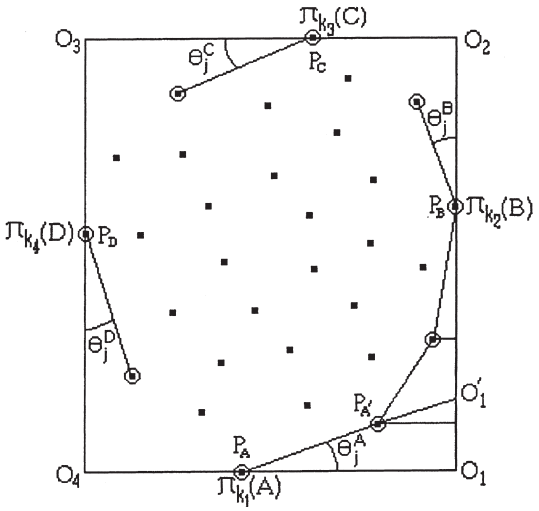

Fig. 2. Point-processors and hull-processors (circled). The labels in parentheses denote the processor types.

representing the input vectors (the signals)

$$S = \{(x_1,y_1),(x_2,y_2),...,(x_n,y_n)\} = \{P_1,P_2,...,P_n\}. \quad (2)$$

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

Definition 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.

Result 1. It is well known (Preparata & Shamos, 1985) 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 $\pi_1, \pi_2, ..., \pi_n$ be n processors associated with the points $P_1, P_2, ..., P_n$, respectively. The processor π_i stores its location (x_i, y_i) for $i = 1, 2, ..., n$ 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 2. A point-processor π_i is called a hull-processor if $P_i = (x_i, y_i)$ is a hull vertex.

Let π_k be a processor such that one of the following conditions holds:

$$y_k = \min_i \{y_i\} = y_{\min} \quad (3)$$

$$x_k = \max_i \{x_i\} = x_{\max} \quad (4)$$

$$y_k = \max_i \{y_i\} = y_{\max} \quad (5)$$

$$x_k = \min_i \{x_i\} = x_{\min}. \quad (6)$$

Then π_k is a hull-processor.

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 Eqs. (3)–(6), respectively (Fig. 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, $\pi_{k_1}, \pi_{k_2}, \pi_{k_3}$ 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 processor can be of the same type (according to Definition 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 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 mother hull-processor then compute the angles θ_i^A, $i = 1, 2, ..., n$, given by

$$\theta_i^A = \begin{cases} \cos^{-1}\dfrac{x_i - x_{k_1}}{\sqrt{(x_i - x_{k_1})^2 + (y_i - y_{k_1})^2}} & \text{if } y_i \geq y_{k_1} \\ 2\pi - \cos^{-1}\dfrac{x_i - x_{k_1}}{\sqrt{(x_i - x_{k_1})^2 + (y_i - y_{k_1})^2}} & \text{if } y_i < y_{k_1} \end{cases} \quad (i \neq k_1). \quad (7)$$

Note that θ_i^A measures the angle, as shown in Fig. 2, of 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. 2)

$$\theta_j^A = \min_i \theta_i^A. \quad (8)$$

The processor π_j is declared as a hull-processor (see the proof of Proposition 1 for justification) 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.

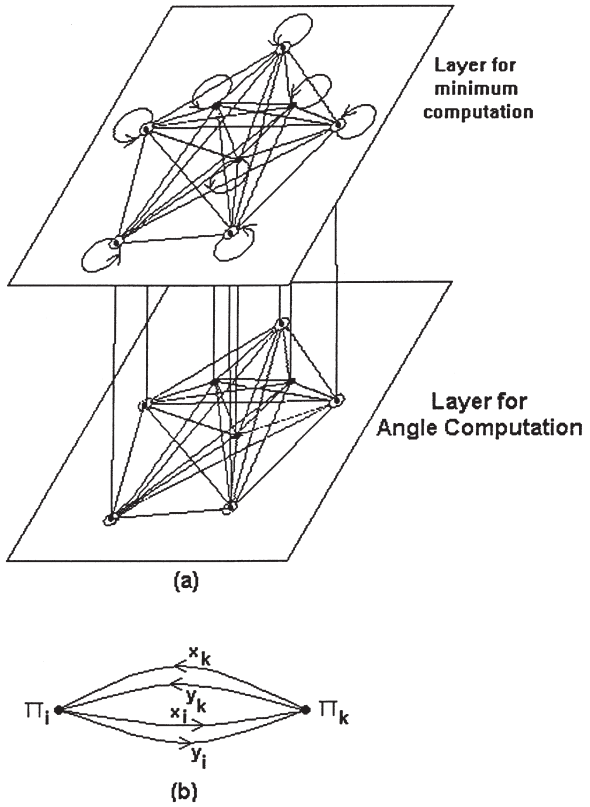

Fig. 3. The subnetwork architecture for a given type of hull-processor. Solid dots indicate point-processors and the circled dots represent hull-processors. All the links represented by lines are bidirectional.

Proposition 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. 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 $P_AO_1'O_1$ 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 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 $i = 1, 2, ..., n$, given by Eqs. (9)–(11), respectively.

$$\theta_i^B = \begin{cases} \cos^{-1}\dfrac{y_i - y_{k_2}}{\sqrt{(x_i - x_{k_2})^2 + (y_i - y_{k_2})^2}} & \text{if } x_i \le x_{k_2} \\ 2\pi - \cos^{-1}\dfrac{y_i - y_{k_2}}{\sqrt{(x_i - x_{k_2})^2 + (y_i - y_{k_2})^2}} & \text{if } x_i > x_{k_2} \end{cases} \quad (i \neq k_2) \tag{9}$$

$$\theta_i^C = \begin{cases} \cos^{-1}\dfrac{x_{k_3} - x_i}{\sqrt{(x_{k_3} - x_i)^2 + (y_{k_3} - y_i)^2}} & \text{if } y_{k_3} \ge y_i \\ 2\pi - \cos^{-1}\dfrac{x_{k_3} - x_i}{\sqrt{(x_{k_3} - x_i)^2 + (y_{k_3} - y_i)^2}} & \text{if } y_{k_3} < y_i \end{cases} \quad (i \neq k_3) \tag{10}$$

$$\theta_i^D = \begin{cases} \cos^{-1}\dfrac{y_{k_4} - y_i}{\sqrt{(x_{k_4} - x_i)^2 + (y_{k_4} - y_i)^2}} & \text{if } x_{k_4} \le x_i \\ 2\pi - \cos^{-1}\dfrac{y_{k_4} - y_i}{\sqrt{(x_{k_4} - x_i)^2 + (y_{k_4} - y_i)^2}} & \text{if } x_{k_4} > x_i \end{cases} \quad (i \neq k_4). \tag{11}$$

The computation of $\min\theta_i^B$, $\min\theta_i^C$, $\min\theta_i^D$ and the reproduction processes are similar to that of A-type processor. It can be seen that the process stops by Proposition 1. 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.

3. The architecture of the model

We shall now discuss the network architecture for a connectionist implementation of 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. 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. In the lower layer, there is no self-connection while in the upper layer every processor has a self-excitation connection as present

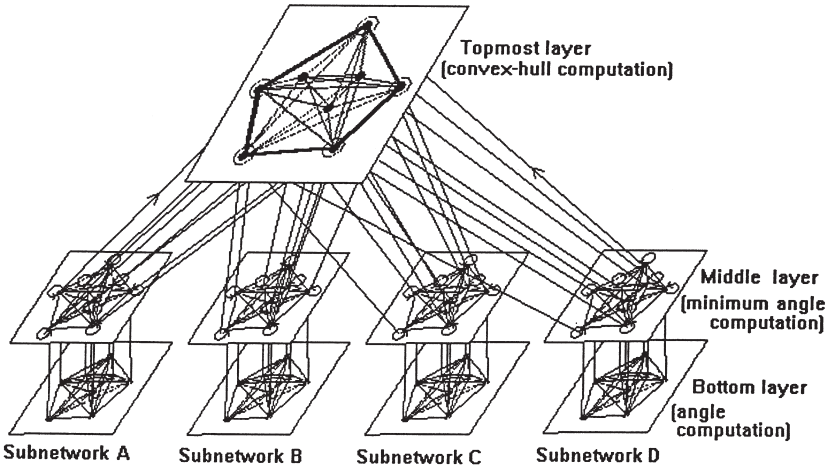

Fig. 4. The complete network architecture for the proposed model.

in the 'maxnet' described earlier. 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_1} is an A-type hull-processor. The within layer connections in Fig. 3(a) are shown in more detail in Fig. 3(b). A processor π_i is connected to π_{k_1} by four links (in Fig. 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

$$o_i^A = f^A(\alpha_i, \beta_i, x_k, y_k) = \begin{cases} \cos^{-1}\dfrac{\alpha_i - x_k}{\sqrt{(\alpha_i - x_k)^2 + (\beta_i - y_k)^2}} & \text{if } \beta_i \geq y_k \\ 2\pi - \cos^{-1}\dfrac{\alpha_i - x_k}{\sqrt{(\alpha_i - x_k)^2 + (\beta_i - y_k)^2}} & \text{if } \beta_i < y_k \end{cases}. \quad (12)$$

If we put $\alpha_i = x_i$ and $\beta_i = y_i$, then o_i^A is nothing but the angle θ_i^A (Eq. (7)). It is to be mentioned here that many connectionist models use sigmoidal function as the activation function. As we need to compute the angles, the arc cosine function has been used here.

The above activation function can be rewritten as

$$o_i^A = f^A(\alpha_i, \beta_i, x_k, y_k) = (1 + \delta)\pi - \delta \cos^{-1}\frac{\alpha_i - x_k}{\sqrt{(\alpha_i - x_k)^2 + (\beta_i - y_k)^2}} \quad (13)$$

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.

Every processor π_i, in the lower layer computes output o_i^A according to Eq. (12). These output values are passed to the respective processors in the upper layer by the forward links. The 'maxnet' layer then selects the winner (in respect of 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 processor 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 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. 3(a)) four times and adding another layer above these networks as shown in Fig. 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 Eqs. (14)–(16), respectively.

$$o_i^B = f^B(\alpha_i, \beta_i, x_{k_2}, y_{k_2}) = \begin{cases} \cos^{-1}\dfrac{\beta_i - y_{k_2}}{\sqrt{(\alpha_i - x_{k_2})^2 + (\beta_i - y_{k_2})^2}} & \text{if } \alpha_i \leq x_{k2} \\ 2\pi - \cos^{-1}\dfrac{\beta_i - y_{k_2}}{\sqrt{(\alpha_i - x_{k_2})^2 + (\beta_i - y_{k_2})^2}} & \text{if } \alpha_i > x_{k2} \end{cases} \quad (14)$$

$$o_i^C = f^C(\alpha_i, \beta_i, x_{k_3}, y_{k_3}) = \begin{cases} \cos^{-1}\dfrac{x_{k_3} - \alpha_i}{\sqrt{(x_{k_3} - \alpha_i)^2 + (y_{k_3} - \beta_i)^2}} & \text{if } y_{k_3} \geq \beta_i \\ 2\pi - \cos^{-1}\dfrac{x_{k_3} - \alpha_i}{\sqrt{(x_{k_3} - \alpha_i)^2 + (y_{k_3} - \beta_i)^2}} & \text{if } y_{k_3} < \beta_i \end{cases} \quad (15)$$

$$o_i^D = f^D(\alpha_i, \beta_i, x_{k_4}, y_{k_4}) = \begin{cases} \cos^{-1}\dfrac{y_{k_4} - \beta_i}{\sqrt{(x_{k_4} - \alpha_i)^2 + (y_{k_4} - \beta_i)^2}} & \text{if } x_{k_4} \leq \alpha_i \\ 2\pi - \cos^{-1}\dfrac{y_{k_4} - \beta_i}{\sqrt{(x_{k_4} - \alpha_i)^2 + (y_{k_4} - \beta_i)^2}} & \text{if } x_{k_4} > \alpha_i \end{cases} \quad (16)$$

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. 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. At the same time, the weight of the connection from the mother to the child processor (in the topmost layer) is set to 1 (denoted by thick lines in the topmost layer in Fig. 4). We now show, by Propositions 2 and 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 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 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 3. *The topmost layer finally gives the required convex-hull.*

Proof. Recall the proof of Proposition 1. It is clear (by Eq. (8)) that the extended edge $P_A P_{A'}$ (see Fig. 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 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 Eqs. (3)–(6). 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.

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 'maxnet' 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. As 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

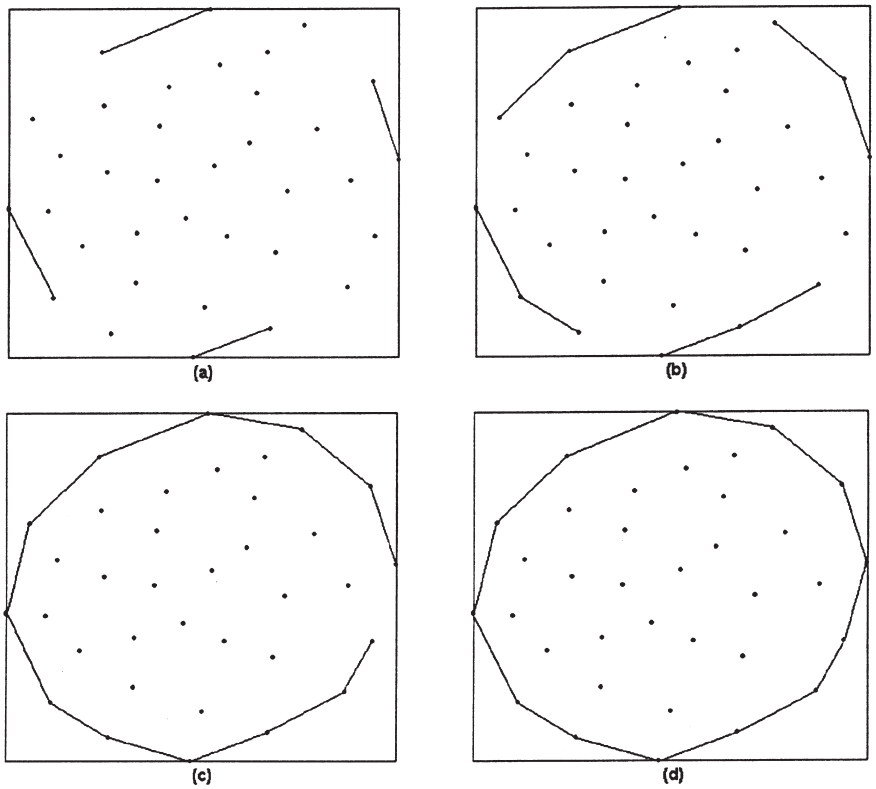

Fig. 5. The intermediate and final results on a planar set: (a)–(c) The results after iterations 1, 2 and 3, respectively; (d) the final result after 4 iterations.

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.

4. Results and conclusions

The proposed model is tested on several 2D point sets. Fig. 5 shows one such example with intermediate outputs. Throughout this paper, 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 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 (1989) and Leung et al. (1997) 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 paper provides a self-organizing connectionist model to compute convex-hull of a given set of 2D points in $O(h)$ time in off-line mode.

Acknowledgements

The authors gratefully acknowledge the reviewers for their valuable comments that led to considerable improvements of the paper.

References

Agarwal, P. K. (1994). Applications of parametric searching in geometric optimization. *Journal of Algorithms*, *17*, 292–318.

Akl, S. G., & Toussaint, G. T. (1978). A fast convex hull algorithm. *Information Processing Letters*, *7*, 219–222.

Akl, S. G., & Lyons, K. A. (1993). *Parallel computational geometry*. Englewood Cliffs, NJ: Prentice-Hall.

Bently, J. L., Faust, G. M., & Preparata, F. P. (1982). Approximation

algorithm for convex hulls. *Communication of the Association of Computational Machinery*, *25*, 64–68.

Bentley, J. L., Clarkson, K. L., & Levine, D. B. (1993). Fast linear expected time algorithm for computing maxima and convex hulls. *Algorithmica*, *9*, 168–183.

Bern, M. W., Karloff, H. L., & Schieber, B. (1992). Fast geometric approximation techniques and geometric embedding problems. *Theoretical Computer Science*, *106*, 265–281.

Capoyleas, V., Rote, G., & Woeginger, G. (1991). Geometric clustering. *Journal of Algorithms*, *12*, 341–356.

Carpenter, G.A., & Grossberg, S. (1987). ART2: self-organization of stable category recognition codes for analog input pattern. In: *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, CA, vol. II, pp. 727–736.

Chank, D. R., & Kapur, S. S. (1970). An algorithm for convex polytopes. *Journal of the Association of Computational Machinery*, *17*, 78–86.

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

Devijver, P. A., & Kittler, J. (1982). *Pattern recognition: a statistical approach*. Englewood Cliffs, NJ: Prentice-Hall.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley.

Earnshaw, R. A. (1988). Theoretical foundations of computer graphics and CAD *NATO ASI, F40*. Berlin: Springer.

Eddy, W. F. (1977). A new convex hull algorithm for planar sets. *Association of Computational Machinery, Transactions of Mathematical Software*, *3*, 398–403.

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

Fritzke, B. (1991). In T. Kohonen, K. Makisara, O. Simula & J. Kangas, *Let it grow—self-organizing feature maps with problem dependent cell structure*, *Artificial neural networks* (pp. 403–408). vol. 1. Amsterdam: North-Holland.

Graham, R. L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, *1*, 132–133.

Guibas, L., Salesin, D., & Stolfi, J. (1993). Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, *9*, 534–560.

Hwang, Y. K., & Ahuja, N. (1993). Cross motion planning—a survey. *Association of Computational Machinery Computational Survey*, *24*, 219–291.

Jarvis, R. A. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, *2*, 18–21.

Kohonen, T. (1989). *Self-organization and associative memory*. Berlin: Springer.

Leung, Y., Zhang, J.-S., & Xu, Z.-B. (1997). Neural networks for convex hull computation. *IEEE Transactions on Neural Networks*, *8*, 601–611.

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

Mehrotra, K., Mohan, C. K., & Ranka, S. (1997). *Elements of artificial neural networks*. Cambridge, MA: MIT Press.

Preparata, F. P., & Hong, S. J. (1977). Convex hulls of finite sets of points in two and three dimensions. *Communication of the Association of Computational Machinery*, *20*, 87–93.

Preparata, F. P., & Shamos, M. I. (1985). *Computational geometry: an introduction*. New York: Springer.

Rumelhart, D. E. & McClelland, J. L. (1986). *Parallel distributed processing*. vol. 1. Cambridge, MA: MIT Press.

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

Schwartz, J. T. & Yap, C. K. (1987). *Advances in robotics I: algorithmic and geometric aspects of robotics*. Hillsdale, NJ: Lawrence Erlbaum.

Serrano-Gotarredona, T., Linares-Barranco, B., & Andreou, A. G. (1998). *Adaptive resonance theory microchips—circuit design techniques*. Dordrecht: Kluwer Academic.

Toussaint, G. T. (1985). *Computational geometry*. New York: North-Holland.

Wennmyr, E. (1989). A convex hull algorithm for neural networks. *IEEE Transactions on Circuits and System*, *36*, 1478–1484.