A Multilayer Self-Organizing Model for Convex-Hull Computation

Srimanta Pal, Amitava Datta, and Nikhil R. Pal, *Senior Member, IEEE*

***Abstract*—A self-organizing neural-network model is proposed for computation of the convex-hull of a given set of planar points. The network evolves in such a manner that it adapts itself to the hull-vertices of the convex-hull. The proposed network consists of three layers of processors. The bottom layer computes some angles which are passed to the middle layer. The middle layer is used for computation of the minimum angle (winner selection). These information are passed to 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 complexity of the proposed model is analyzed and is compared with existing models of similar nature.**

***Index Terms*—Convex-hull, neural networks, planar set, RANKNET, self-organization.**

I. Introduction

IN THIS paper we deal with a well-known problem, namely, the computation of *convex-hull* of a finite number of points in two dimensions (2-D). 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 2-D convex-hull of a set of points on the plane 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 on 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, shape analysis, and several other fields [1], [3], [7], [13]–[15], [17], [21], [31], [33], [37]. For example, convex-hull can be used for representation of shapes in soil micro-structure study [27] where the objects to be recognized or classified have almost random shapes, and there are rarely two geometrically similar objects. A shape representation scheme should be normalized and invariant with respect to coordinate rotation, translation, and scaling. A convex-hull based shape representation is suitable for classification and recognition of irregular objects because it is invariant with respect to coordinate rotation, translation, and scaling.

Since 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 [2], [5], [16], [9], [18], [22], [30], [40] and computing approximate convex-hull [4], [6], [19], [24]. 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 in 2-D or three dimensions (3-D), 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 [11], [12], [24], and [40] are designed on artificial neural networks. Wennmyr [40] proposed an exact convex-hull computation algorithm based on a multilayer network [25], [32]. The author designed 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. Datta and Parui [11] proposed a dynamic neural-network model for the computation of an exact convex-hull with complexity $O(n \log n)$. In this model, the initial network size is very small and the network is able to grow through certain neuron insertion–deletion mechanism. Leung *et al.* [24] 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) [8], [34] the two layers of neurons can communicate via feedforward as well as feedback connections and use a new training strategy named "excited" learning. The network produces two approximations, one from inside and one from outside, of the convex-hull. Datta *et al.* [12] proposed a self-organizing model for the computation of an exact convex-hull in 2-D with complexity $O(h)$, where h is the number of hull-points. The present paper proposes a different self-organizing model for computation of the 2-D convex-hull.

Conceptually the proposed algorithm is a connectionist implementation of Preparata and Hong's gift wrapping algorithm [30]. We implement the algorithm by a self-organizing neural network [10], [23]. The term "self-organization" here refers to the ability to learn from the input without having any prior supervising information and to arrange (or order) the processors accordingly. 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 angle. These information are passed to 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 processors (as hull-processors) in an orderly fashion. Initially the model identifies one hull-vertex. Gradually the network self-organizes to identify other processors that correspond to the rest of the hull-vertices. These hull-vertices are generated in an ordered fashion so that the convex-hull of the data points is obtained. The learning takes place without any supervision.

Before going into the proposed model, a network to compute the extremum (maximum or minimum) of n given values will be discussed as it is used by our model. We first briefly discuss different existing *MAXNET* models and propose an efficient *MAXNET* model which requires $O(1)$ time.

II. The MAXNET Model

Several neural networks are available that compute the maximum of n given values [25], [26], [28], [41]. The MAXNET model proposed by Lippmann *et al.* [26] is a fully connected net made up of n neurons with internal thresholds set to zero. The input values x_i, $i = 1, 2, \ldots, n$, are fed at time zero to the input neurons. The output $o_i(t)$, at time $t = 0$, is initialized to the input value x_i, for $i = 1, 2, \ldots, n$. That is

$$o_i(0) = x_i, \qquad i = 1, 2, \ldots, n. \tag{1}$$

The network then iterates to find the neuron with the maximum input value by

$$o_i(t+1) = f\left[o_i(t) - \sum_{j \neq i} w_{ji} x_j(t)\right] \tag{2}$$

where $w_{ji} < 1/n$ is the inhibitory connection weight between neurons i and j, $i \neq j$, and f is the threshold logic function defined by

$$f(\alpha) = \begin{cases} \alpha & \text{if } \alpha \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

By (2), every neuron inhibits all other neurons by an amount equals to the neuron's output multiplied by a small negative weight. Each neuron also feeds back itself with a unity gain. After convergence, the single neuron whose value is initially the maximum prevails as the "winner" neuron, and the outputs of all other neurons subside to zero.

To find the maximum of n values, the above MAXNET uses n neurons and requires $O(n)$ iterations in the worst case. To reduce the number of iterations, Pecht and Gur [29] modified this model by dynamically changing the weights after every iteration as follows:

$$w_{ji}(t) = \frac{1}{n(t)} \tag{4}$$

where $n(t)$ stands for the number of neurons at time t with nonzero output and $n(0) = n$. With these dynamic weights, the time-complexity of the modified MAXNET reduces to $O(\log n)$. But, to implement the dynamic change of weights, additional hardware is necessary. Another MAXNET model has been proposed by Suter and Kabrisky [36] which has the worst-case time-complexity $O(n)$. They observed that the model converges in $O(\log n)$ iterations for some distributions. This model demands more hardware than required by the model in [29]. Winters and Rose [41] have proposed an improved MAXNET which takes $O(\log n)$ time but requires special switching neuron elements. Tseng and Wu [38] have used $O(n^2)$ neurons to achieve a model that can compute the maximum in $O(1)$ time.

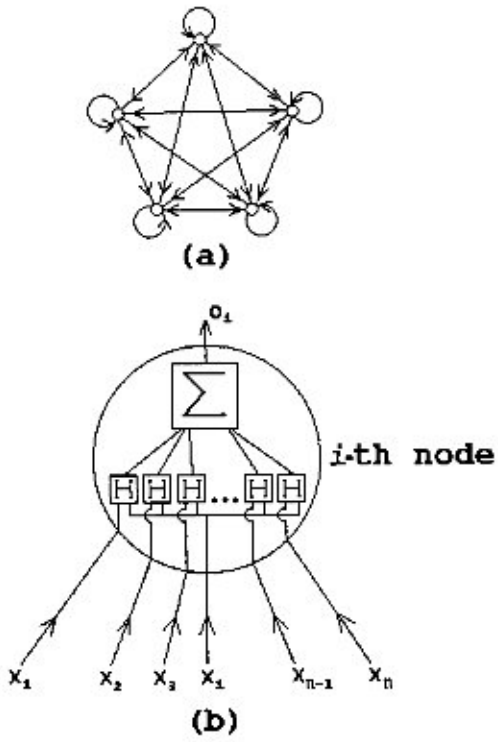

Fig. 1. (a) The schematic diagram of RANKNET. Every neuron is connected to every neuron. (b) The architecture of each neuron in (a). The internal threshold (used by function H) of the ith neuron is its input value x_i.

The MAXNET model proposed here is simple in terms of hardware and uses only hard-limiters and summers. The model ranks all input values (and henceforth is called RANKNET) in a single iteration and thus it can be used in k winner-take-all ($k \geq 1$) networks.

The RANKNET is a fully connected network composed of n neurons, one for each input. Every neuron is connected to every neuron. A schematic diagram of the network is given in Fig. 1(a). Let x_i denote the ith input value which is fed to the ith neuron. The ith neuron receives each of the n input values x_j, $j = 1, 2, \ldots, n$ through a hard limiter, $H(x_j, x_i)$. Denote the output of the ith neuron by o_i. The ith neuron, $i = 1, 2, \ldots, n$, computes

$$o_i(0) = \sum_{j=1}^{n} H(x_j, x_i) \tag{5}$$

where

$$H(x_j, x_i) = \begin{cases} 1 & \text{if } x_j \geq x_i \\ 0 & \text{otherwise.} \end{cases}$$

Thus, H is a hard-limiter that uses the respective neuron's input as the internal threshold. The ith neuron computes $H(x_j, x_i)$, $j = 1, 2, \ldots, n$ and then adds them to get the output o_i. Hence, o_i gives the rank $r_i \in \{1, 2, \ldots, n\}$ of the input value x_i. All the neurons compute their outputs o_is simultaneously. Thus, the ranks of all input values are obtained in a single iteration. The neuron which is associated with the maximum input value gives its output (rank) as one, if there is no tie (for tie, see Case II of examples below).

The hardware implementation of the proposed network is much simpler than the existing MAXNET (or MINNET) models. Every neuron in Fig. 1(a) (denoted by a circle) is composed of a building block given in Fig. 1(b). The difference in architecture between our neuron and that in the MAXNET model proposed by Lippmann *et al.* [26] is important. Each neuron in the MAXNET model first computes the weighted sum of the input values and then uses a threshold logic function to compute the output. The neurons use linear synapses. On the contrary, we here use nonlinear synapses [35], [39]. Every neuron in the RANKNET model first uses hard-limiters and then computes the sum. Note that similar architecture for "nonlinear-synapse neurons" have been used by several researchers (for example, see [35] and [39]). The nonlinear synapses used in the RANKNET model help us to reduce the time-complexity of our model to $O(1)$. Let us illustrate the RANKNET using some examples.

A. Examples

Case I: Without Tie: Let the input values be {5, 7, 2, 17, 9, 15 }.

These input values are assigned to six neurons.

Denote $H_{ji} = H(x_j, x_i)$. Writing H_{ji} in jth row and ith column we get the following matrix, say $\mathcal{H}$:

$$\mathcal{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The column totals of $\mathcal{H}$ produce (5 4 6 1 3 2). The ith column total gives the rank of x_i.

Case II: With Tie: Let the input values be {5, 7, 2, 17, 7, 15}.

These input values are assigned to six neurons. In this case the matrix $\mathcal{H}$ obtained is

$$\mathcal{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The column totals of $\mathcal{H}$ produce (5 4 6 1 4 2) as the ranks of different x_is. Note that, the input value seven is repeated twice and both get the rank of four.

III. The Proposed Model

Consider a set of n 2-D points representing the input vectors (the signals)

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} = \{P_1, P_2, \ldots, P_n\}. \quad (6)$$

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

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 [31].

Result 1: 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 [31].

Suppose $\pi_1, \pi_2, \ldots, \pi_n$ are n processors (neurons) associated with the points $P_1, P_2, \ldots, P_n$, respectively. The processor π_i stores a vector $(x_i, y_i, w_i^{(x)}, w_i^{(y)}) = (X_i, W_i)$ where $X_i = (x_i, y_i)$ is the coordinate of P_i and $W_i = (w_i^{(x)}, w_i^{(y)})$ is its weight vector (later on we shall see that the weight vector is the *direction ratio*) for $i = 1, 2, \ldots, n$. Each processor π_i is connected with itself and all other processors π_j forming a complete network. These processors are termed as *point-processors*.

Definition 2: A point-processor π_k is called a *hull-processor* if $P_k = (x_k, y_k)$ is a hull-vertex.

Initially, choose a processor at random, say π_f. Call it a *seed-processor*. Now consider the processor π_k such that

$$\|X_k - X_f\| = \max_i \|X_i - X_f\|. \quad (7)$$

That is, π_k is the farthest processor from π_f. Note that a "MAXNET"-like network can be used here.

A standard result of computational geometry shows that, π_k is a hull-processor [20]. Call it a *mother* (first mother) hull-processor. Also assume that the initial weights for jth neuron; $j = 1, 2, \ldots, n$ are

$$W_j(0) = \begin{cases} X_j - X_k & \text{if } j \neq k \\ X_j - X_f & \text{if } j = k. \end{cases} \quad (8)$$

The convex-hull computation algorithm consists of three major steps: angle calculation, finding the minimum angle and weight update as described next.

A. Angle Computation

Compute the angles θ_j, with respect to the mother hull-processor π_k, $j = 1, 2, \ldots, n (j \neq k)$, as

$$\theta_j = \cos^{-1} \frac{\langle W_k, W_j \rangle}{\|W_k\| \, \|W_j\|}. \quad (9)$$

Here, θ_j initially measures the angle (by angle here we mean the smaller angle) between the vectors $\overline{P_k P_j}$ and $\overline{P_k O_1}$ (see Fig. 2). O_1 is a point on the extension of the line segment $\overline{P_f P_k}$.

B. Finding the Minimum Angle

Let π_m be the processor such that (see Fig. 2)

$$\theta_m = \min_j \{\theta_j\}. \quad (10)$$

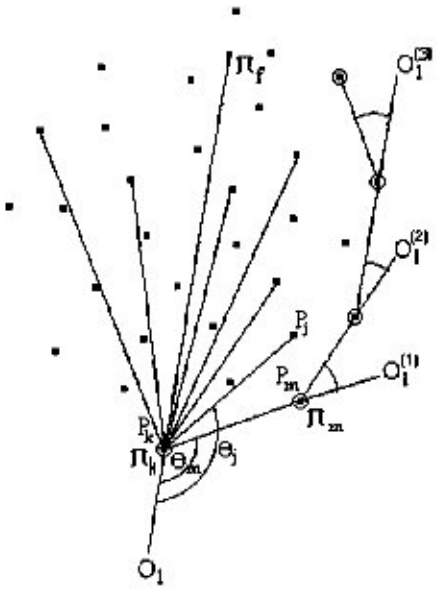

Fig. 2. Point-processors and hull-processors (circled).

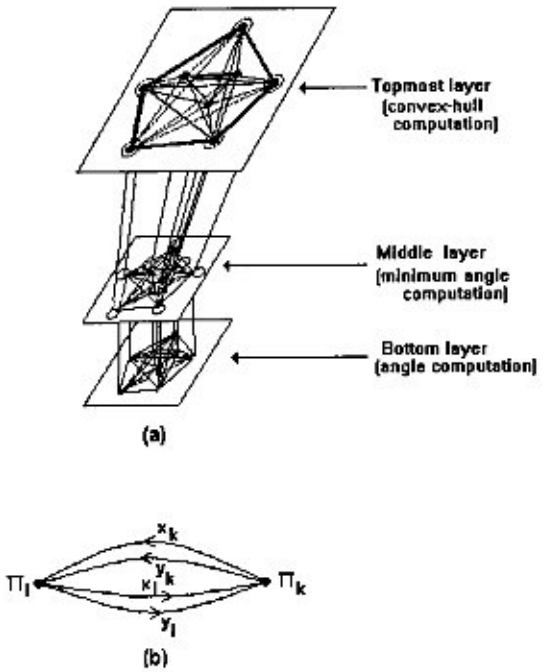

Fig. 3. (a) The neural-network architecture for convex hull computation. Solid dots indicate point-processors and the circled dots represent hull-processors. All the links represented by lines are bidirectional. (b) The detailed within layer connection used in the bottom layer of (a).

The processor π_m is declared as a hull-processor and as a *daughter* of the mother hull-processor π_k. Again a "MAXNET"-like network can be used to compute $\min_j\{\theta_j\}$. The mother hull-processor π_k now becomes inactive and the newly created daughter π_m becomes an active mother hull-processor. By inactivating a processor we mean that it is no longer a mother hull-processor in the sense of *reproduction* (i.e., it cannot produce a daughter) but its status as a hull-processor remains so that it can participate in subsequent angle computation.

C. Updating the Weights

After finding the minimum angle weights are updated as follows:

$$W_j(t+1) = \begin{cases} W_j(t) - W_m(t) & \text{for } j = 1, 2, \ldots, n \\ & \text{and } j \neq m, k \\ W_j(t) & \text{for } j = m \\ -W_m(t) & \text{for } j = k. \end{cases} \tag{11}$$

As mentioned earlier, in the first iteration θ_m measures the minimum of the angles between vectors, $\overline{P_kP_j}$ and $\overline{P_kO_-}$; $j = 1, 2, \ldots, n$; $j \neq k$ (see Fig. 2). In the subsequent iterations, as a result of the weight updating, θ_m gives the minimum of the angles between the vectors (after renaming P_m as P_k), $\overline{P_kP_j}(\forall j \neq k)$, and the vector $\overline{m(P_k)P_k}$, where $m(P_k)$ stands for the input point corresponding to the mother of π_k. The above process of angle calculation and finding the processor with the minimum angle is repeated with the active mother hull-processor until the most recent daughter processor is identical with the first mother hull-processor.

IV. The Network Architecture of the Model

We now discuss the neural-network architecture for implementation of the proposed model. The network consists of three layers: bottom, middle, and topmost layer [see Fig. 3(a)]. The bottom layer is used for the angle computation and the middle layer is used for finding the minimum angle. The number of processors in each layer is n, the number of data points. All the layers have similar structures where every processor is connected to every other. In the bottom and topmost layers, there is no self-connection while in the middle layer every processor has a self-excitation connection as present in the "RANKNET" described earlier. Moreover, the ith processor in the bottom layer is connected to the ith processor in the middle layer by feedforward as well as by feedback links. Similarly, the ith processor in the middle layer is connected to the ith processor of the topmost layer [see Fig. 3(a)].

Suppose, in the bottom layer, π_k is a hull-processor. The within layer connections in the bottom layer in Fig. 3(a) are shown in more details in Fig. 3(b). A processor π_i is connected to π_k by four links [in Fig. 3(a), they are represented by a single straight line]. In the beginning, the connection weights are set as shown in the figure. The motivation behind such connection weights is that π_k must know the coordinates of π_i, and π_i must know the coordinates of π_k. The output o_j of the jth neuron is computed by the activation function in (12).

When π_k is the most recently generated hull-processor (mother), the neuron $j \neq k$ computes its output o_j as

$$\begin{aligned} o_j &= f(W_j, W_k) \\ &= \cos^{-1}\frac{\langle W_k, W_j\rangle}{\|W_k\|\,\|W_j\|} \\ &= \cos^{-1}\frac{w_k^{(x)}w_j^{(x)} + w_k^{(y)}w_j^{(y)}}{\sqrt{(w_k^{(x)})^2 + (w_k^{(y)})^2}\sqrt{(w_j^{(x)})^2 + (w_j^{(y)})^2}}. \end{aligned} \tag{12}$$

Clearly, o_j is nothing but the angle θ_j in (9).

Every processor $\pi_j (j \neq k)$, in the bottom layer, computes output o_j according to (12). These output values are passed to the respective processors in the middle layer by the forward links. The middle layer then selects the winner (in respect of the minimum value) processor π_m and passes this information back to the bottom layer and to the topmost layer. The processor π_m is then declared as a hull-processor and as a daughter of π_k; and π_k is then inactivated. At the same time, the weight of the connection from the mother to the daughter processor (in the topmost layer) is set to one [denoted by thick lines in the topmost layer in Fig. 3(a)]. In the next iteration, the connection weights in the bottom layer are updated according to (11) and the most recently created daughter processor plays the role of the mother processor. The process continues until the most recent daughter becomes equal to the first mother processor. Initially all connection weights on the topmost layer are set to zero and at every iteration only one weight is set to one.

Now we shall show, in Propositions 1 and 2, that the network, formed by the hull-processors and the links with connection weights as one in the topmost layer, provides the required convex-hull.

Proposition 1: If the reproduction process is continued, the algorithm stops in a finite number of iterations.

Proof: Let

S = set of given points, that is, $S = \{P_1, P_2, \ldots, P_n\}$;

Π = $\{\pi_1, \pi_2, \ldots, \pi_n\}$ be the corresponding set of processors;

C_t = set of hull processors after the tth iteration, where $C_0 = \{\}$;

T_t = set of winner processors formed at the tth iteration;

π'_1 = first mother processor, such that $\pi'_1 \in \Pi$ after iteration one;

π'_t = daughter (winner) processor, selected at the tth iteration; hence, $\pi'_t \in T_t$ and $T_t \subset \Pi \quad C_{t-1} \cup \{\pi'_1\}$ where $C_{t-1} = \{\pi'_1, \ldots, \pi'_{t-1}\}$.

At iteration $t+1$, the daughter π'_t becomes a mother and produces a daughter π'_{t+1} where $\pi'_{t+1} \in T_{t+1}$ and $T_{t+1} \subset \Pi - C_t \cup \{\pi'_1\}$ which is produced in the middle layer. If $|T_{t+1}| > 1$ then π'_{t+1} is chosen arbitrarily. C_t is modified to $C_{t+1} = C_t \cup \{\pi'_{t+1}\}$. If $\pi'_{t+1} = \pi'_1$, the algorithm stops. Otherwise, it continues. Since, S is finite and $|\Pi - C_t \cup \{\pi'_t\}|$ decreases with iterations, if the first mother does not tie with any other processor, the algorithm terminates in a finite number of iterations $m (\leq n)$. Now consider the case where first mother ties with a set of processors V ($|V| \geq 2$). This corresponds to a situation when the first mother falls on the line segment formed by the points corresponding to the processors in V. Our method of selection of the first mother ensures that the first mother is one of the end points of this line segment. Hence, the most recent daughter will be the first mother in at most $|V| - 1$ additional steps. Thus the algorithm again terminates in $m (\leq n)$ iterations.

Proposition 2: The output network in the topmost layer provides the convex-hull of S.

Proof: Let π'_1 be the first mother (Fig. 2) which corresponds to the input point P_k. Therefore, π'_1 is a hull-processor. Now suppose that π'_2 is the daughter of π'_1 which corresponds to the input point P_m. Note that, the line $P_k P_m$ partitions the plane into two half-planes in such a way that one of them contains all points of S and other contains none [3]. Therefore, $P_k P_m$ is a hull-edge and P_m is a hull-vertex. In the next iteration, π'_1 is inactivated and π'_2 is activated as a mother processor. By similar logic, every link, joining a mother processor and its daughter, partitions the plane satisfying Result 1. Moreover, every link is directed from a mother to its daughter. Hence, all the hull-vertices of S are mapped as the hull-processors in an ordered fashion in the topmost layer of the resulting network.

The algorithm for computation of the convex-hull can now be stated as follows.

Algorithm:

Step 1. **[Initialization]** Select the seed-processor π_f at random. Compute the first mother hull-processor π_k satisfying (7) and declare it as the first mother hull-processor. Set the initial weights using (8).

Step 2. **[Reproduction]** The mother hull- processor creates a daughter processor as follows:

Step 2(a) **[Angle computation]** All processors, except the mother, in the bottom layer compute the angle (with respect to the mother hull-processor) using the activation function in (12).

Step 2(b) **[Minimum angle computation]** The minimum angle is computed in the middle layer, where the mother hull-processor does not participate. The processor corresponding to the minimum angle is declared the daughter processor.

Step 2(c) **[Report to topmost and bottom layer]** The minimum angle information is sent to the topmost layer to set the connection weight as one from the mother to the daughter. This information is also passed to the bottom layer.

Step 3. **[Weight updating]** Update the weights in the bottom layer using (11).

Step 4. Inactivate the mother and activate the daughter as the mother.

Step 5. Repeat Steps 2) through 4) until the most recent daughter processor be the same as the first mother.

Step 6. Obtain the convex-hull from the topmost layer.

Step 7. Stop.

A. Computational Aspects

After the initial hull-processor is created every processor in the network computes its own output independently (in parallel). Thus this computation takes a constant amount of time. The winner processor is selected in parallel by the "RANKNET" in the middle layer (described in Section II) which requires $O(1)$ time. It is easy to see that the network, in the worst case, needs to compute the output (in the bottom layer) and then selects the winner (in the middle layer) h times where h is the number of hull-points. Thus, the whole process, in the worst case, takes $O(h)$ time.

Computation of convex-hull of a planar set has been a problem of considerable interest and several researchers have developed various algorithms. While most of them are based on conventional techniques, Wennmyr [40], Leung *et al.* [24], Datta and Parui [11] and Datta *et al.* [12] suggested neural-network-based techniques. For computing the exact

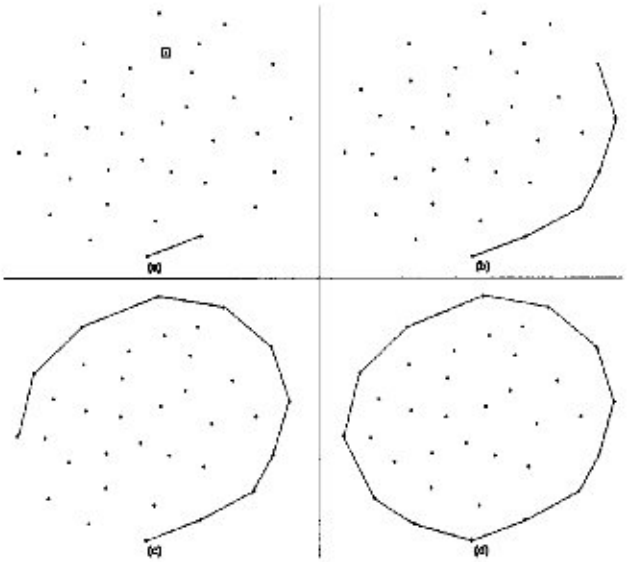

Fig. 4. The intermediate and final results on a planar set. (a)–(c) The results after iterations one, five, and ten, respectively (where the dot enclosed by a box in panel (a) indicates the seed-processor). (d) The final result after 13 iterations.

convex-hull, Wennmyr proposed a multilayer 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. Datta and Parui [11] proposed a dynamic neural-network model for the computation of the exact convex-hull with complexity $O(n \log n)$; while Datta *et al.*'s self-organizing model computes the exact convex-hull with complexity $O(h)$. The algorithm proposed in this paper provides a different self-organizing connectionist model also to compute the exact convex-hull of a given set of 2-D points in $O(h)$ time in off-line mode.

V. Results and Conclusion

The proposed model is tested on several 2-D point sets. Fig. 4 shows one such example with intermediate outputs. The point corresponding to the randomly selected seed processor is enclosed by a square in Fig. 4(a). After one iteration, the first hull edge, as shown in Fig. 4(a), is determined. The hull edge obtained after five and ten iterations are shown, respectively, in Fig. 4(b) and (c). The complete hull [Fig. 4(d)] is obtained in just 13 iterations.

We proposed a neural-network model for computing the exact convex-hull of a planar set. It is shown that the neural network is capable of learning from the input and can arrange itself to generate the convex-hull. The model is self-organizing in the sense that the learning is unsupervised. Moreover, the network organizes itself in an orderly fashion in such a way that a mapping is established from the hull-vertices to the hull-processors and form the hull-edges to the respective links in the topmost layer of the network.

In this context it is worth noting the differences between the model by Datta *et al.* [12] (model **A**) and the present model (model **B**). In model **A**, there are four subnetworks each consisting of two layers. These four subnetworks are connected to another layer placed on their top. Every layer has n neurons, where n is the number of points. So each subnetwork requires $2n$ neurons and the topmost layer needs n neurons. Thus, the total number of neurons in model **A** is $4 \times 2n + n = 9n$. Model **B** uses only one network consisting of three layers each having n neurons. So model **B** requires only $3n$ neurons. In model **B**, the initial mother hull-processor is selected in a manner different from that in model **A**. The activation functions of different types of hull-processors in model **A** are different while a single activation function for all neurons is used in model **B**. The activation function in model **B** uses direction ratios and these direction ratios are treated as weight vectors. The weight vectors are updated in such a way that a single activation function serves the purpose.

The Gift-wrapping concept used here can be extended to compute the convex-hull of a set of 3-D points also [31]. In that case, instead of rotating a line, a plane has to be rotated about a line which is passing through a hull-point. Accordingly the activation function is to be redefined. We are currently investigating such possibilities.

Acknowledgment

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

References

[1] P. K. Agarwal, "Applications 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] L. J. Bently, G. M. Faust, and F. P. Preparata, "Approximation algorithm for convex hulls," *Comm. ACM.*, pp. 64–68, 1982.
[5] J. L. Bentley, K. L. Clarkson, and D. B. Levine, "Fast linear expected time algorithm for computing maxima and convex hulls," *Algorithmica*, vol. 9, pp. 168–183, 1993.
[6] 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.
[7] V. Capoyleas, G. Rote, and G. Woeginger, "Geometric clustering," *J. Algorithms*, vol. 12, pp. 341–356, 1991.
[8] 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*, vol. II, San Diego, CA, 1987, pp. 727–736.
[9] D. R. Chank and S. S. Kapur, "An algorithm for convex polytopes," *J. ACM*, vol. 17, pp. 78–86, 1970.
[10] D. Choi and S. Park, "Self-creating and organizing neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 561–575, 1994.
[11] A. Datta and S. K. Parui, "A dynamic neural net to compute convex-hull," *Neurocomput.*, vol. 10, pp. 375–384, 1996.
[12] A. Datta, S. Pal, and N. R. Pal, "A connectionist model for convex-hull of a planar set," *Neural Networks*, vol. 13, pp. 377–384, 2000.
[13] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, NJ: Prentice-Hall, 1982.
[14] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[15] R. A. Earnshaw, "Theoretical foundations of computer graphics and CAD," in *NATO ASI*. Berlin, Germany: Springer-Verlag, 1988, vol. F40.
[16] W. F. Eddy, "A new convex hull algorithm for planar sets," *ACM Trans. Math. Software*, vol. 3, pp. 398–403, 1977.
[17] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 551–559, 1983.
[18] 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.

[19] L. Guibas, D. Salesin, and J. Stolfi, "Constructing strongly convex approximate hulls with inaccurate primitives," *Algorithmica*, vol. 9, pp. 534–560, 1993.
[20] G. Hadley, *Linear Programming*. Reading, MA: Addison-Wesley, 1962.
[21] Y. K. Hwang and N. Ahuja, "Cross motion planning—A survey," *ACM Comput. Survey*, vol. 24, pp. 219–291, 1993.
[22] 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.
[23] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Germany: Springer-Verlag, 1989.
[24] 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.
[25] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, 1987.
[26] R. P. Lippmann, B. Bold, and M. L. Malpass, "A comparison of Hamming and Hopfield neural nets for pattern classification," Lincoln Lab., Massachusetts Inst. Technol., Cambridge, MA, Tech. Rep. 769, May 1987.
[27] D. Luo, *Pattern Recognition and Image Processing*. Chichester, U.K.: Horwood, 1998.
[28] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of Artificial Neural Networks*. Cambridge, MA: MIT Press, 1997.
[29] O. Y. Pecht and M. Gur, "A biologically-inspired improved MAXNET," *IEEE Trans. Neural Networks*, vol. 6, pp. 757–759, 1995.
[30] 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.
[31] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. New York: Springer-Verlag, 1985.
[32] D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing*. Cambridge, MA: MIT Press, 1986, vol. 1.
[33] J. T. Schwartz and C. K. Yap, Eds., *Advances in Robotics I: Algorithmic and Geometric Aspects of Robotics*. Hillsdale, NJ: Lawrence Erlbaum, 1987.
[34] T. Serrano-Gotarredona, B. Linares-Barranco, and A. G. Andreou, *Adaptive Resonance Theory Microchips-Circuit Design Techniques*. Boston, MA: Kluwer, 1998.
[35] N. Suetake, N. Yamauchi, and T. Yamakawa, "Wavelet neuron filter with the local statistics oriented to the preprocessor for the image signals," in *Proc. 2nd Int. Conf. Inform. Fusion*, vol. I, 1999, pp. 626–633.
[36] B. W. Suter and M. Kabrisky, "On a magnitude preserving iterative MAXnet algorithm," *Neural Comput.*, vol. 4, pp. 224–233, 1992.
[37] G. T. Toussaint, Ed., *Computational Geometry*. New York: North-Holland, 1985.
[38] Y.-H. Tseng and J.-L. Wu, "On a constant-time, low-complexity winner-take-all neural network," *IEEE Trans. Comput.*, vol. 44, pp. 601–604, 1995.
[39] T. Yamakawa and T. Samatsu, "Wavelet neural networks realizing high-speed learning," in *Proc. Int. Conf. Neural Inform. Processing*, Seoul, Korea, 1994, pp. 1571–1576.
[40] E. Wennmyr, "A convex hull algorithm for neural networks," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1478–1484, 1989.
[41] J. H. Winters and C. Rose, "Minimum distance automata in parallel networks for optimum classification," *Neural Networks*, vol. 2, pp. 127–132, 1989.

Srimanta Pal received the B.Sc. (Hons.) degree in mathematics from the University of Calcutta, Calcutta, India, in 1978, the B.Tech. degree in instrumentation and electronics engineering from Jadavpur University, Jadavpur, India, in 1982, the M.Tech. degree in computer science from the Indian Statistical Institute, Calcutta, in 1984, the M.B.A. degree in operations research from Jadavpur University in 1989, and the Ph.D. degree in computer science from the Indian Institute of Technology, Kharagpur, in 1992.

He was with Semi-Conductor Complex Ltd., India, and with Indian Institute of Management, Calcutta, from 1984 to 1991. Currently he is an Associate Professor in the Electronics and Communication Sciences Unit, Indian Statistical Institute. His research interests include image processing, pattern recognition, artificial intelligence, neural networks, computational geometry, and brain modeling.

Amitava Datta received the Master's degree in statistics from the Indian Statistical Institute, Calcutta, India, in 1977 and the postgraduate diploma in computer science from the same institute in 1978. He received the Ph.D. degree from the Indian Statistical Institute in 2000.

After working for a few years in reputed computer industries he joined the Indian Statistical Institute as a System Analyst in 1988. Since then, he has been working in image processing and pattern recognition. From 1991 to 1992, he visited GSF, Munich, Germany, as a Guest Scientist and worked on a query-based decision support system. His current interests include neural networks, image processing, and pattern recognition.

Nikhil R. Pal (M'91–SM'00) received the B.Sc. degree with honors in physics and the Master of Business Management degree from the University of Calcutta, Calcutta, India, in 1978 and 1982, respectively. He received the M.Tech. and Ph.D. degrees, both in computer science, from the Indian Statistical Institute in 1984 and 1991, respectively.

Currently he is a Professor in the Electronics and Communication Sciences Unit of the Indian Statistical Institute, Calcutta. From September 1991 to February 1993, July 1994 to December 1994, October 1996 to December 1996, and January 2000 to July 2000, he was with the Computer Science Department of the University of West Florida, Pensacola. He was a Guest Faculty Member of the University of Calcutta also. His research interest includes image processing, pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, genetic algorithms, and fuzzy logic controllers. He has coauthored a book titled *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, (Boston, MA: Kluwer, 1999), coedited a volume *Advances in Pattern Recognition and Digital Techniques, ICAPRDT99,* and edited a book titled *Pattern Recognition in Soft Computing Paradigm,* (Singapore, World Scientific, 2001). He is an Associate Editor of the *International Journal of Fuzzy Systems, International Journal of Approximate Reasoning,* the IEEE TRANSACTIONS ON FUZZY SYSTEMS and the IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS—B (ELECTRONIC VERSION), and a Steering Committee member of *Applied Soft Computing*, Elsevier Science.