Shape approximation of arc patterns using dynamic neural networks

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

[a]*Electronics and Communication Sciences Unit, Indian Statistical Institute, 203 BT Road, Calcutta 700 035, India*
[b]*Computer and Statistical Service Centre, Indian Statistical Institute, 203 BT Road, Calcutta 700 035, India*

Received 21 October 1993; revised 27 May 1994 and 30 September 1994

Abstract

A shape representation technique for two-dimensional patterns using a dynamic variation of Kohonen's self-organizing feature maps is discussed. In Kohonen's map, the number of processors is fixed and is to be specified a priori. This number, if not properly chosen, can cause either wastage or shortage of processors. The problem is overcome, in this paper, by incorporating dynamic growing and shrinking capabilities in the network. Shape of only arc patterns is considered here.

Zusammenfassung

Es wird eine Form-Repräsentationstechnik für zweidimensionale Muster diskutiert, die eine dynamische Variation von Kohonens selbstorganisierender Merkmalsliste darstellt. In Kohonens Liste ist die Anzahl von Prozessoren festgelegt und muß a-priori spezifiziert werden. Wenn diese Anzahl nicht geeignet gewählt ist, kann dieses entweder Verschwendung oder Mangel an Prozessoren bewirken. Um dieses Problem zu vermeiden, werden in dieser Arbeit dynamische Zuwachs- oder Schrumpfungs-Eigenschaften des Netzwerks eingebracht. Hierbei werden nur Formen von Bogen-Mustern betrachtet.

Résumé

Une technique de représentation de forme de gabarits bi-dimensionnels utilisant une variante dynamique des réseaux de Kohonen auto-organisants est discutée. Dans le réseau de Kohonen, le nombre de processeurs est fixe est doit être spécifié a priori. Ce nombre, s'il n'est pas correctement choisi, peut causer soit un gaspillage soit un manque de processeurs. Ce problème est résolu, dans cet article, en incorporant une capacité dynamique de croissance et de diminution dans ce réseau. La seule forme considérée ici est l'arc de cercle.

Keywords: Arc patterns; Self organization; Dynamic neural net; Skeletonization

1. Introduction

In the last few years, there has been a great interest in neural networks, especially in the field of pattern recognition and computer vision [1, 2, 3, 5, 6]. In this paper, a simple model of neural networks is proposed for a common image processing problem, namely, the skeletonization of two-dimensional patterns. Here we will deal with only arc patterns which may be in the form of a point set or a binary or gray level image. The neural net connections and the updation rules are similar to those in Kohonen's self-organizing one-dimensional feature map [4]. But while the number of processors in Kohonen's feature map is fixed, the present network is dynamic in the sense that during learning process new processors can be added to or old processors can be deleted from the network. A major advantage of such a dynamic model is that the number of processors need not be known a priori, but can be learnt. On the other hand, a model with a fixed number of processors can cause some processors to be overloaded and some processors to remain unused. Dynamic neural networks have been used to model the probability distribution in the plane [3] and to classify shapes [7]. In the present work, the structure of the neural net after convergence gives an approximation of shape in the form of a skeleton.

2. The network and updation rule

Kohonen's feature map uses an array of processors where each processor stores a weight vector. The map is learnt on the basis of a set of input signals. The dimension of the array of processors is normally either one or two and the dimension of the weight vectors is the same as that of the input signals which can be arbitrary. Suppose the array is one-dimensional and is represented as $\{\pi_1, \pi_2, \ldots, \pi_n\}$ where the processor π_i is connected to the processors π_{i-1} and π_{i-1} (the two end processors are connected to exactly one processor each). The processor π_i has a weight vector W_i whose dimension is m. Suppose the set of input signals is $S = \{P_1, P_2, \ldots, P_N\}$, where the dimension of each P_j is also m. The *neighbourhood* of a processor π_i is $\{\pi_p, p = i-q, i-q-1, \ldots, i+q-1, i+q\}$ for some positive integer q. Updation rule for the weight vectors is as follows.

Suppose at time instance t, P_j is presented to the feature map and let $W_k(t) = (W_{k1}(t), W_{k2}(t), \ldots, W_{km}(t))$ be the nearest weight vector to P_j. Then, only the $2q+1$ weight vectors (of the processors falling in the neighbourhood of π_k) are updated in the following way [4]:

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)] \quad (1)$$

$$\text{for } p = k-q, \ldots, k+q,$$

where the starting weight vectors $W_p(0)$ are chosen randomly, $\alpha(t)$ 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 convergence of $W_i(t)$.

Our network is a one-dimensional array $(\pi_1, \pi_2, \ldots, \pi_n)$ of processors as above with $q = 1$. Here the input signals are two-dimensional points and hence $m = 2$.

Here $S = \{P_1, P_2, \ldots, P_N\}$ is a set of N planar points where $P_j = (x_j, y_j)$. On the basis of the points in S, the weight vectors W_i of the processors π_i will be updated iteratively. Suppose, the initial weight vectors are $W_i(0) = (W_{i1}(0), W_{i2}(0))$. Let the point P_j be presented at the tth iteration. P_j updates the weight vectors in the following way. Let

$$\text{dist}(P_j, W_k) = \min_i[\text{dist}(P_j, W_i)].$$

That is, W_k is the nearest weight vector from the point P_j. The neighbourhood of π_k consists of $\{\pi_{k-1}, \pi_{k-1}\}$ (if $k-1=0$ or $k+1=n+1$, then we ignore the corresponding neighbour). P_j updates the weight vectors of π_{k-1}, π_k and π_{k+1} only in the following way:

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)]$$

$$\text{for } p = k-1, k, k+1. \quad (2)$$

One presentation each of all the points in S makes one *sweep* consisting of N iterations. After one sweep is completed, iteration for the next sweep starts again from P_1 to P_N. Several sweeps make one *phase*. One phase is completed when the weight

vectors of the current set of processors have converged, that is, when

$$\|W_i(t) - W_i(t+1)\| < \varepsilon \quad \text{for all } i, \tag{3}$$

where ε is a predetermined small positive value. The condition (3) is guaranteed from a property of Kohonen's feature map [4].

Only at the end of a phase are processors inserted or deleted. That is, during the iterative process in a phase, the set of processors remains unchanged. Suppose, after the sth phase is completed, 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, from the start of the first 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 as $[W_k(t_s) + W_{k+1}(t_s)]/2$. If, on the other hand,

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

then one processor is inserted between π_l and π_{l+1} and the new processor has the weight vector as $[W_l(t_s) + W_{l+1}(t_s)]/2$. Note that δ_1 and δ_2 are two predetermined positive quantities such that $\delta_1 < \delta_2$.

After the insertion and deletion of processors, the next phase starts with the new set of processors. The iterative process continues until

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

It can be seen that the process of insertion and deletion of π_i cannot fall in an infinite loop because $\alpha(t)$ goes to 0 as t tends to infinity in updation rule (2). 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.

The Algorithm

```
begin {begin of algorithm}
  while condition (6) is not true
    begin
      {phase}
      while condition (3) is not true
        begin
          {sweep}
          for each input vector P_j do
            begin
              present P_j to the net;
              update weights of the processors
              according to (2);
            end;
        end;
      merge (delete) or insert processor
      according to (4) or (5);
    end;
end {end of algorithm}
```

3. Results and conclusions

A planar point set S_1 ($N = 36$) and the starting weight vectors of the processors ($n = 3$) are shown in Fig. 1(a). The input points are indicated by dots and the weight vectors by circles. Fig. 1(b) shows the weight vectors of the processors at the end of the first phase (which occurred after 27 sweeps) when the first new processor (its weight vector is indicated by a solid circle in Fig. 1(b)) is inserted in the network. An intermediate output obtained after 211 sweeps has 9 processors and is shown in Fig. 1(c). The final output (that is, when condition (6) is satisfied) having 14 processors (obtained after 358 sweeps) is shown in Fig. 1(d).

Another point set S_2 ($N = 229$) generated at random from an 'S' shape is shown in Fig. 2(a). Three skeletons (in the form of the stabilized network) of S_2 using different numbers of processors are shown in Fig. 2(b)–(d). For both sets S_1 and S_2, our learning algorithm has been tested with several choices

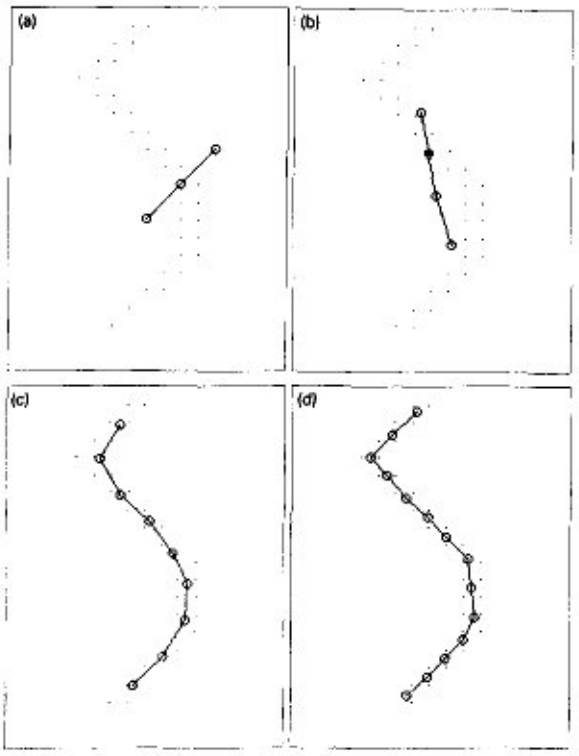

Fig. 1. A point set S_1 ($N = 36$) and the network at several stages of learning. Here $\alpha(t) = 0.02/[1 + t/1000]$, $\varepsilon = 0.001$, $\delta_1 = 0.5$, $\delta_2 = 1.0$. (a) Before learning starts; (b) after 27 sweeps; (c) after 211 sweeps; (d) final output (after 358 sweeps).

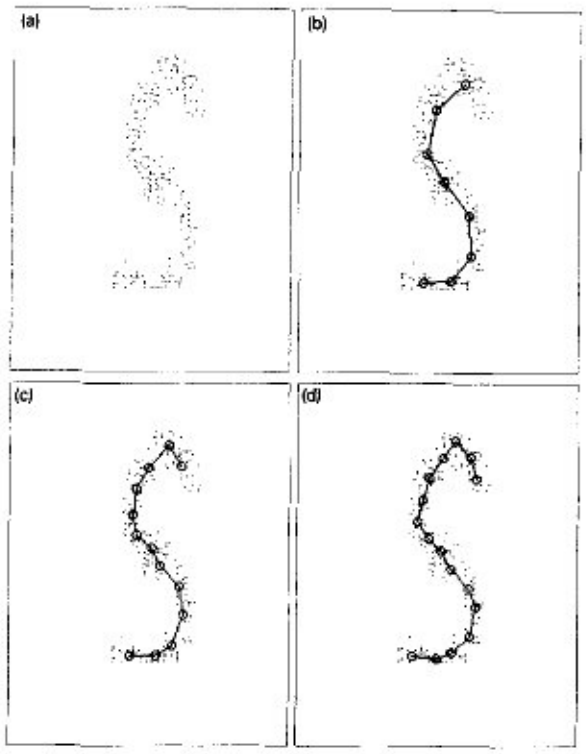

Fig. 2. (a) A point set S_2 ($N = 229$). Its skeletons (using different numbers of processors) are given in (b), (c) and (d) where $\alpha(t) = 0.2/[1 + t/10000]$, $\varepsilon = 0.01$. (b) The network after convergence has 8 processors where $\delta_1 = 18$, $\delta_2 = 36$ (after 217 sweeps). (c) The network after convergence has 13 processors where $\delta_1 = 12$, $\delta_2 = 24$ (after 310 sweeps). (d) The network after convergence has 16 processors where $\delta_1 = 11$, $\delta_2 = 22$ (after 481 sweeps).

of the starting weight vectors of the processors and the final output is found to be independent of the starting vectors.

The skeletonization process described in Section 2 can be applied to binary images in a straightforward manner. The input signals in that case will be the co-ordinates of the object pixels. The extension of the algorithm to gray level images can be done in the following way. Suppose for a gray level arc pattern, g_{rs} is the gray value of the pixel p at the rth row and sth column. Then the updation rules replacing Eq. (2) will be

$$W_p(t+1) = W_p(t) + \alpha(t)[P_j - W_p(t)]g_{rs}$$

$$\text{for } p = k-1,\ k,\ k+1.$$

Multiplication by g_{rs} means that the amount of updation will be more for pixels with high gray level values and less for pixels with low gray level values.

In the case of a binary image, a desirable property of its skeleton is that it should lie totally inside the object in the image. However, if, for some reason, the number of processors is small, our output skeleton may not lie totally inside the object (Figs. 1(c), 2(b)). But by having a sufficient number of processors (which can be controlled by δ_1 and δ_2), our algorithm can always produce a skeleton that will lie totally inside the object (Figs. 1(d), 2(d)).

In the present paper we have considered arc like patterns. For more complicated patterns that have forks, crossings, etc. (e.g. character patterns 'Y', 'X'), the algorithm will not work. We are currently

working on extending the present algorithm where we start with an initial one-dimensional array of processors and then grow it adaptively in two-dimensions so that the junctures (where the input pattern has forks or crossings) are identified.

References

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

[2] R. Eckmiller and C. v.d. Malsburg, eds., *Neural Computers*, Springer, Berlin, 1989.

[3] B. Fritzke, "Let it grow – Self-organizing feature maps with problem dependent cell structure", in: T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, eds., *Artificial Neural Networks*, North-Holland, Amsterdam, Vol. 1, 1991, pp. 403–408.

[4] T. Kohonen, *Self-Organization and Associative Memory*, Springer, Berlin, 1989.

[5] C.M. Lee and D. Patterson, "A hybrid neural network vision system for object identification", in: T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, eds., *Artificial Neural Networks*, North-Holland, Amsterdam, Vol. 1, 1991, pp. 69–74.

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

[7] M. Sabourin and A. Mitiche, "Modeling and classification of shape using a Kohonen associative memory with selective multiresolution", *Neural Networks*, Vol. 6, 1993, pp. 275–283.