Rival Penalized Competitive Learning for Clustering Analysis, RBF Net, and Curve Detection

Lei Xu, Adam Krzyżak, *Member, IEEE,* and Erkki Oja, *Senior Member, IEEE*

***Abstract*—It is well known that the classical k-means clustering algorithm has a problem of selecting an appropriate k, the number of clusters. It is a hard problem and affects the performance of k-means strongly. When used for clustering analysis, the conventional competitive learning (CL) algorithms also have a similar crucial problem: the selection of an appropriate number of neural units, although the problem remains largely unrevealed (or ignored) in the CL literature. As shown in this paper, frequency sensitive competitive learning (FSCL), one version of the recently improved CL algorithms, also significantly deteriorates its performance when the number of units is inappropriately selected. This paper proposes a new algorithm called rival penalized competitive learning (RPCL). The basic idea is that for each input not only the winner unit is modified to adapt to the input, but also its rival (the 2nd winner) is delearned by a smaller learning rate. RPCL can be regarded as an unsupervised extension of Kohonen's supervised LVQ2. RPCL has the ability of automatically allocating an appropriate number of units for an input data set. The experimental results show that RPCL outperforms FSCL when used for unsupervised classification, for training a radial basis function (RBF) network, and for curve detection in digital images.**

I. Introduction

As an adaptive version of the classical *k-means clustering* algorithm, *competitive learning* (CL) has a number of applications. First, it can function as an adaptive method for clustering analysis problems encountered in statistical data analysis or unsupervised pattern recognition [1]. Second, it can be used for vector quantization which is widely used in image processing and speech signal processing for compressing data and message coding [2], [3]. Third, it has also been recently incorporated into some supervised learning methods for training multilayer feedforward nets more effectively, e.g., into the *radial basis function (RBF)* nets for locating the centers of Gaussian receptive fields [4]–[6].

However, it was found by Rumelhart *et al.* [7], Grossberg [8], and Hecht-Nielsen [8] that the simple classical CL algorithm has the so called *under-utilized* or *dead unit* problem. Many efforts have been made to solve the problem. Grossberg's ART series [7], [10]–[13] and Kohonen Map [14], [15] are two main developments of the classical competitive learning. ART provides a stable model for unsupervised pattern recognition and associative memory. However, as pointed out by Lippmann [16], the ART method does not correspond to k-means algorithm for clustering analysis and vector quantization in the global optimization sense. Although Kohonen Map is related to adaptive k-means, its main purpose is to form a topographic feature map which is a more complex task than just clustering analysis. It is therefore also computationally more complicated.

Focusing on improving the dead units problem of the classical CL, there are also several other techniques proposed, e.g., *leaky learning* by Rumelhart [7] and Grossberg [8], and *convex bridge* by Hecht-Nielsen [9]. A notable improvement is the strategy of reducing the winning rate of the frequent winners [8], [17]–[19], sometimes called *conscience* [18]. *Frequency Sensitive Competitive Learning* (FSCL) [19] is a good example which uses this strategy. The idea is fairly straightforward but the method does improve the classical CL significantly.

In this paper, we will show that there is another critical problem with the present CL algorithms: the selection of an appropriate number of the units needed. It is well known that one key problem with the k-means algorithm is that k (the number of clusters) should be appropriately pre-selected, otherwise the algorithm will perform badly. In a competitive learning net, k directly corresponds to the number of the neural units used. This number should also be externally preselected appropriately, since the number of units corresponds to the number of resulted clusters when CL is used for solving a clustering problem, or to the the number of hidden units when CL is used in a RBF network. As will be shown later, inappropriately selected k will result in a poor clustering result that will in turn affect the performances of FSCL for unsupervised classification, RBF net, and curve detection significantly.

For tackling this problem, we propose a new version of competitive learning algorithm called *rival penalized competitive learning* (RPCL), which is developed by adding a new mechanism into FSCL. The basic idea is that for each input, not only the weights of the winner unit are modified to adapt to the input, but also the weights of its rival (the 2nd winner) are delearned by a smaller learning rate. The idea can be regarded as an unsupervised extension of Kohonen's LVQ2 [15] which is a supervised vector quantization algorithm, and closer in effect to Bayes decision theory. We have applied RPCL to the problems of unsupervised classification, RBF network training, and curve detection, and compared its performance with FSCL. The experimental results show that significant improvements have been obtained by RPCL.

Manuscript received December 12, 1991; revised August 17, 1992.

L. Xu is with the Harvard Robotics Laboratory, Harvard University, Cambridge, MA 02138 and is also with the Department of Mathematics, Peking University, Beijing, P. R. China.

A. Krzyżak is with the Department of Computer Science, Concordia University, Montreal, Quebec H3G 1M8, Canada.

E. Oja is with the Department of Information Technology, Lappeenranta University of Technology, SF-53851 Lappeenranta, Finland.

IEEE Log Number 9204279.

In the sequel, the problem of selecting the number of units in a competitive learning net will be addressed in Section II. Then RPCL is proposed and analyzed in Section III. In Section IV, FSCL, and RPCL are experimentally compared in the problems of unsupervised classification and RBF net's training. In Section V, the curve detection problem is modeled into a problem of competitive learning, and both FSCL and RPCL are tested with a problem of detecting four lines in a diamond-shaped frame.

II. A Crucial Problem For Competitive Learning

Given a layer of units with the output of each unit denoted by u_i and its weight vector by $\vec{w}_i$ for $i = 1, \cdots, k$, the classical CL algorithm consists of the following two steps.

Step 1: Randomly take a sample $\vec{x}$ from a data set D, and for $i = 1, \cdots, k$, let

$$u_i = \begin{cases} 1, & \text{if } i = c \text{ such that } \|\vec{x} - \vec{w}_c\|^2 = \min_j \|\vec{x} - \vec{w}_j\|^2, \\ 0, & \text{otherwise.} \end{cases} \tag{1a}$$

Step 2: Update the weight vectors $\vec{w}_i$ by

$$\Delta\vec{w}_i = \alpha_i u_i (\vec{x} - \vec{w}_i)$$

$$\text{i.e., } \Delta\vec{\mathrm{w}}_\mathrm{i} = \begin{cases} \alpha_\mathrm{i}(\vec{\mathrm{x}} - \vec{\mathrm{w}}_\mathrm{i}), & \text{if } \mathrm{u_i} = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{1b}$$

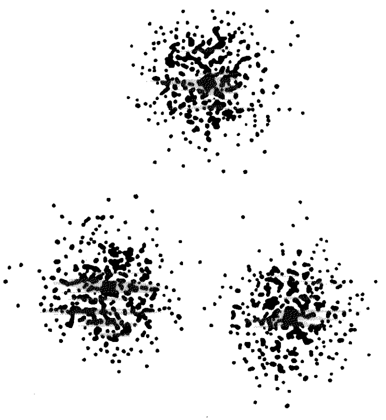
Thus in practice only the *winning unit* or *best matching unit* $\vec{w}_c$ is updated. In (1b), the parameter α_c with $0 \le \alpha_c \le 1$ is the learning rate which is usually a small real number, or it starts from a reasonable initial value and then reduces to zero in some way [14], [15], e.g., in the way used in the Robbins–Monro stochastic approximation procedure [21]. The explicit dependence of α_c on time is not shown above.

Note: Equations (1a) and (1b) are often called the Winner-Take-All rule. In addition, there are also some other versions of the classical CL [26]. However, the basic idea is the same; the weight vector of the neural unit which tunes to an input most strongly is adjusted most strongly to tune to the input even stronger.

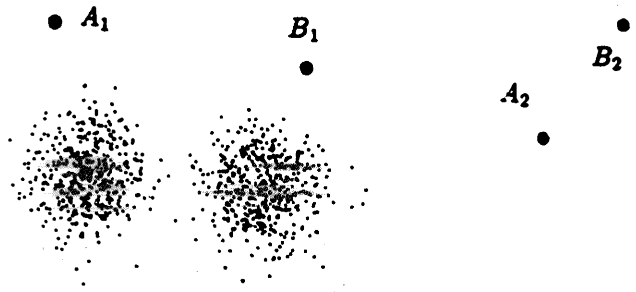
The implementation of the above algorithm is quite easy; each weight vector $\vec{w}_i$ is randomly initialized, and then the above two steps are iterated until the iteration converges or freezes as the learning rate α_c becomes zero or very small, or until the number of iterations reaches a prespecified value.

For a data set D consisting of several clusters of samples (e.g., three clusters as shown in Fig. 1(a)), when the number of units k is equal to the number of these clusters, the desired results of implementing the above algorithm should be that the weight vector of each unit is moved to the center of one of the clusters (Fig. 1(a)) regardless of the initial values of these weight vectors. However, the actual results highly depend on the initial weight vectors. For example, even for a two clusters problem shown in Fig. 1(b), when the two weight vectors $\vec{w}_1, \vec{w}_2$ are initialized at positions A_1, B_1, respectively, they finally move to the center points of the two clusters after the learning has converged. However, when $\vec{w}_1$ is initialized at position A_2 and $\vec{w}_2$ at B_2, the result of (1) will always be $u_1 = 1$ and $u_2 = 0$ during the implementation of the above algorithm. Thus $\vec{w}_1$ is always moving and eventually oscillates around the mean vector of the two clusters of samples, while $\vec{w}_2$ is always the loser and remains fixed during the whole learning process. Such a loser unit is usually called a dead unit.

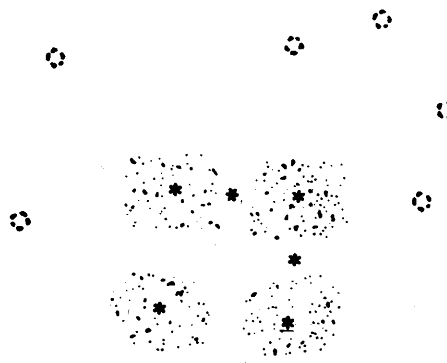
Unfortunately, the situation of dead units happens quite often for the above simple CL algorithm unless the weight



(a)



(b)



(c)

Fig. 1. A crucial problem for competitive learning. (a) The desired result of using CL: each of weight vectors moved to a cluster center regardless of the initial values of these weight vectors. (b) The actual result highly depends on the initial weight vectors. $\vec{w}_1, \vec{w}_2$ will move to cluster centers when they are initialized at points A_1, B_1, respectively; but when $\vec{w}_1$ is initialized at A_2 and $\vec{w}_2$ at B_2, $\vec{w}_1$ will be located around the middle point of the two clusters, while $\vec{w}_2$ remains unmoved as a dead unit. (c) A typical result of using the conscience strategy and FSCL when the number k of used units is larger than the number of clusters in a data set, where the two extra units have also been moved to some boundary points between different clusters, and may confuse quite a large portion of samples from different clusters.

vectors are initialized near the cluster centers which are the eventual points of attraction. As mentioned previously in Section I, many efforts have been made to solve the problem. An often used strategy, sometimes called *conscience*, is to reduce the winning rate of the frequent winners. For example, in Fig. 1(b), when $\vec{w}_1$ initialized at A_2 wins a certain number of times, we force $u_1 = 0$ and $u_2 = 1$ to bring $\vec{w}_2$ according to (1b) towards its point of attraction. By doing so, $\vec{w}_2$ will be gradually brought towards the right-side cluster while $\vec{w}_1$ keeps oscillating between the two clusters. Finally $\vec{w}_1$ will stop oscillating and the two weight vectors will converge to the two cluster centers. The so called FSCL [19] is a recent algorithm using this strategy. It is a straightforward extension of CL, obtained by modifying (1a) into the following.

$$u_i = \begin{cases} 1, & \text{if } i = c \text{ such that} \\ & \quad \gamma_c\|\vec{x} - \vec{w}_c\|^2 = \min_j \ \gamma_j\|\vec{x} - \vec{w}_j\|^2, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

where $\gamma_j = n_j / \sum_{i=1}^{k} n_i$ and n_i is the cumulative number of the occurrences of $u_i = 1$.

The conscience strategy and FSCL do solve the problem of dead units well. Unfortunately, they also bring a new problem. As shown in Fig. 1(c), when the number k of units used in a CL net is larger than the number of clusters in the input data set, all the k weight vectors will be finally moved to some places in the data set and some weight vectors (at least one) will no longer be located at the centers of the clusters but either at some boundary points between different clusters or at points biased from some cluster centers (the phenomenon will be further verified by experiments shown in the next section).

The new problem is quite crucial, since it raises some serious problems for the CL applications. For example, some problems are as follows.

1) For an unsupervised classification application, we expect results like the ones shown in Fig. 1(a), where we obtained three weight vectors located at the mean vectors of the three classes or clusters. In this case, the recognition rate will be the highest when all the samples are classified by distance classifiers based on the three weight vectors. However, in the case shown in Fig. 1(c), the recognition rate will significantly decrease since there are two disturbing units located at the boundary points of different clusters and the two units will draw quite a large portion of samples from three of the four clusters to form two mixture groups which give an incorrect clustering result.

2) For the application of training the hidden units of a RBF net, we would like to expect that in Fig. 1(c) only four weight vectors of the hidden units will be located at the centers of the four clusters, while the weights of the other units should be driven away from the input data set so that the linear units in the output layer of the RBF net become capable to perform further classifications. But the results presented in Fig. 1(c) show two hidden units becoming disturbing units which may confuse quite a large portion of input samples from different clusters. In the case of supervised learning, a class may consist of more than one cluster, so there may be no problem if the confused clusters belong to the same class. However, if the confused clusters do not belong to the same class, the two disturbing units will not only increase the difficulty of learning for the units in the output layer, but will also reduce the recognition rate considerably since the linear output units may not be able to separate the samples confused by the two disturbing hidden units.

The above addressed problem may not notably deteriorate the performance of CL on an application of vector quantization, where the goal is not to find any clusters or classes. However, even in this case the input data density is usually not uniform, and results like the one shown in Fig. 1(a) are desired. For the codebook we would now have three weight vectors, each located at the center of one of the clusters. Results like that shown in Fig. 1(c) will increase the number of vectors in the codebook, without much improvement on the coding performance due to the fact that the code vectors which are located at the boundary points between clusters can contribute only a little in reducing the distortion.

Therefore, we see that the conscience strategy and FSCL work well only when the number of clusters in the input data set is known in advance so that we can let our CL net have the same number of units. This is not an easy task since we usually do not know the number of clusters in the input data *a priori*. The same problem exists in the conventional k-means clustering method: if the number of clusters k is selected inappropriately, we may obtain very poor clustering results. Unfortunately, the selection of k is a hard problem. It could only be solved heuristically by some prior knowledge, or by enumerating a number of different values and doing clustering for each of these values so that a better value could be obtained according to some rule, e.g., finding the value with a sharp change on the curve of the average least square error versus the values of k [1].

III. Rival Penalized Competitive Learning

A. The Algorithm

For tackling the crucial problem described above, we propose here an new version of CL—RPCL, by adding a new mechanism into FSCL. The basic idea is that for each input not only the weight vector $\vec{w}_c$ of the unit which wins the competition is modified to adapt to the input, but also the weight vector $\vec{w}_r$ of its rival (i.e., the second winner) is delearned by a learning rate smaller than that used by $\vec{w}_c$. Specifically, we modify the algorithm given in the beginning of Section II into the following one.

Step 1: Randomly take a sample $\vec{x}$ from a data set D, and for $i = 1, \cdots, k$, let

$$u_i = \begin{cases} 1, & \text{if } i = c \text{ such that} \\ & \quad \gamma_c\|\vec{x} - \vec{w}_c\|^2 = \min_j \ \gamma_j\|\vec{x} - \vec{w}_j\|^2, \\ -1, & \text{if } i = r \text{ such that} \\ & \quad \gamma_r\|\vec{x} - \vec{w}_r\|^2 = \min_{j \neq c} \ \gamma_j\|\vec{x} - \vec{w}_j\|^2, \\ 0, & \text{otherwise.} \end{cases} \tag{3a}$$

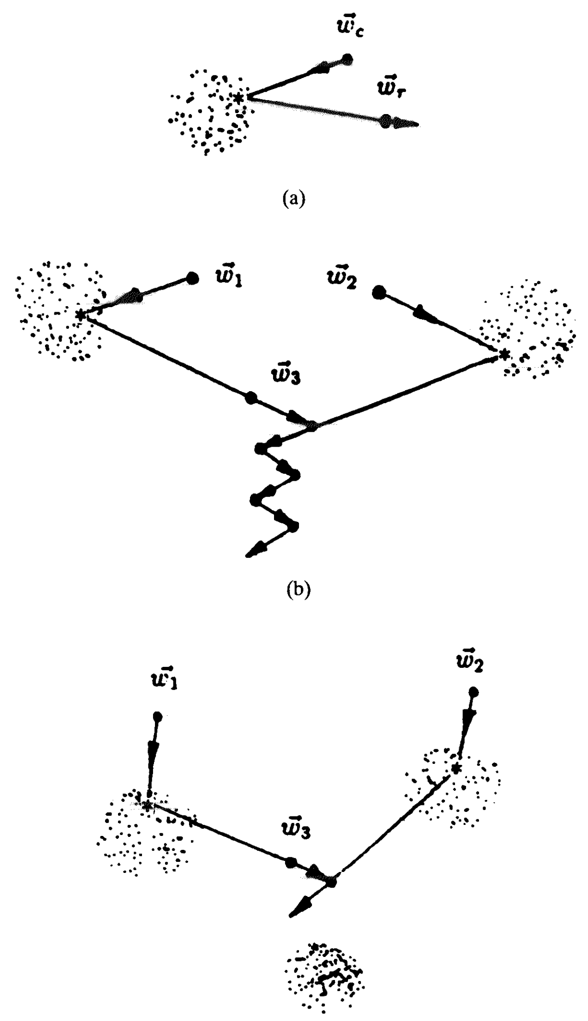


Fig. 2. Rival penalized competitive learning. (a) The rival is pushed away from the cluster that the winner is learning. (b) The rival $\vec{w}_3$ of both $\vec{w}_1$ and $\vec{w}_2$ is driven out along a zig-zag path. (c) Learning of $\vec{w}_3$ is faster because it is pushed towards its correct cluster by the two rivals $\vec{w}_1$ and $\vec{w}_2$.

Step 2: Update the weight vector $\vec{w}_i$ by

$$\Delta\vec{w}_i = \begin{cases} \alpha_c(\vec{x}-\vec{w}_i), & \text{if } u_i = 1 \\ -\alpha_r(\vec{x}-\vec{w}_i), & \text{if } u_i = -1, \\ 0, & \text{otherwise.} \end{cases} \quad (3b)$$

where $0 \le \alpha_c, \alpha_r \le 1$ are the learning rates for the winner and rival unit, respectively. In practice they may depend on time and usually at each iteration step t it holds $\alpha_c(t) \gg \alpha_r(t)$. Moreover, $\gamma_j = n_j / \sum_{i=1}^{k} n_i$ is the same parameter as that in the FSCL algorithm introduced earlier.

As shown in Fig. 2(a), the rival penalized mechanism tries to push its rival far away from the cluster towards which the winner is moving, thus implicitly producing a force which attempts to make sure that each cluster is learned by only one weight vector. This force is just a balance to the force generated by the conscience strategy of FSCL, which encourages both weight vectors to share one cluster. This balancing role can be more clearly seen from Fig. 2(b). Assuming that three weight vectors have already been brought somewhere between two classes, the rival penalized force will gradually drive away the weight vector $\vec{w}_3$ along a zig-zag path as the input samples come randomly and alternatively from both classes. Similarly, we can also imagine that the two disturbing units given in Fig. 1(c) can be driven away by this force.

So, we see that the key point of using the rival penalized mechanism is that the appropriate number of units will be selected automatically for representing an input data set by gradually driving extra units far away from the distribution of data set in the case that the number of units in a competitive learning net is larger than the number of clusters in the input data set. Thus the crucial problem described in Section I can be tackled. In addition, the extra units become now spare units which are ready to learn some new clusters if some additional data are input in the future.

Another important point is that the rival penalized mechanism may sometimes speed up the learning process: as shown in Fig. 2(c), the de-learning of $\vec{w}_3$ caused by the learning of $\vec{w}_1, \vec{w}_2$ will push $\vec{w}_3$ toward its correct cluster.

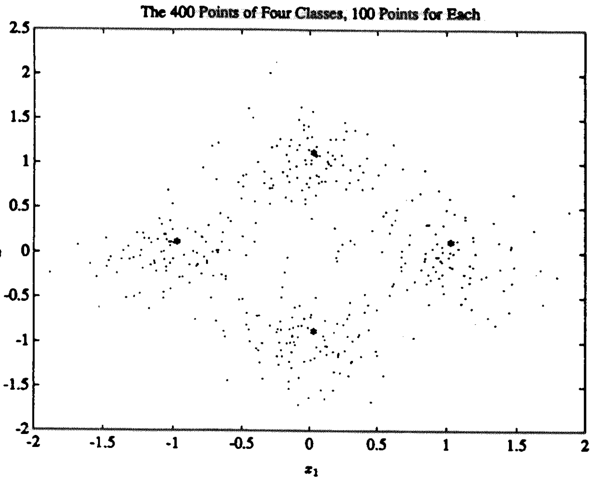


Fig. 3. The data set used in the experiments. Each of four clusters has 100 samples from four Gaussian distributions with variance 0.1 and centered at (-1,0), (1,0), (0,1), (0,-1), marked by an asterisk.

It may be also interesting to note that the basic idea of RPCL can be regarded as a kind of unsupervised extension of Kohonen's supervised learning vector quantization algorithm LVQ2 [15] which can give a result closer in effect to Bayes decision theory by simultaneously modifying the weight vectors of both the winner and its rival when the winner is in a wrong class but the rival is in a correct class for an input vector.

In the sequel, the characteristics of RPCL will be further illustrated through some experiments.

B. Simulation Results

The data set used here is given in Fig. 3. There are four clusters of samples, and each cluster has 100 samples from four Gaussian distributions with variance 0.1 and centered at (-1,0), (1,0), (0,1), (0,-1), marked by an asterisk. At each learning step, one sample $\vec{x}$ is randomly selected from the four clusters with anyone of 400 samples being chosen with equal probability. For simplicity, in all the experiments below, we fixed the learning rates at $\alpha_c = 0.05$, $\alpha_r = 0.002$, although it is usually assumed that better results may be obtained by some specific schedule for changing the rates like that used by the Robbins–Monro stochastic approximation procedure [21]. In addition, we always initialize the weight vectors by random numbers in the interval between 3.0 and 4.0.

First, we choose the number of units in our CL net as 4, the same as the number of clusters in the data set. Fig. 4(a) shows the learning traces (i.e., trajectories of weight vectors during the learning process) obtained by the classical CL algorithm. Obviously there are three dead units, and only the weight vector $\vec{w}_4$ of one unit quickly moves towards a cluster center point (0,0) in less than 50 learning steps and then oscillates around it, which can be observed from the learning curves (i.e., the changes of the component variables in every weight vector versus the learning steps) given in Fig. 4(b).

The fluctuations are due to the fact that the learning rate α_c is fixed at 0.05. If the rate is gradually reduced to zero, then the fluctuations will vanish.

The results obtained by FSCL are given in Figs. 4(c) and (d). It can be seen that after about 160 learning steps (see the corresponding learning curves in Fig. 4(d)) each of the four weight vectors has smoothly moved into one of the four cluster centers. This result again confirms that FSCL solves the "dead unit" problem [19].

Fig. 4(e) shows the learning traces obtained by using our RPCL. Comparing it with Fig. 4(c), we see that the performances of RPCL and FSCL are almost the same in this case. Moreover, the learning curves of RPCL are also almost the same as those in Fig. 4(d), thus we omit them here.

Second, we choose the number of units in our CL net larger than the number of clusters in the data set. Fig. 5(a) and (b) gives the learning traces (curves) obtained by FSCL with five units in the CL net. As argued in Section II, there now occurs the problem that all the five units are brought among the four clusters (it will be more clear by comparing the coordinates given in Fig. 3 and Fig. 5(a)). In addition to the four units located around the four cluster centers (-1,0) (1,0), (0,-1), (0,1), there is also one disturbing unit located around (-0.4, 0.4)—a boundary point between the two clusters centered at (-1,0), (0,1). Moreover, due to the effect of the disturbing unit, the other two units, although located around the cluster centers (-1,0) and (0,1), are somewhat biased from these points. As expected, such problems are absent in Fig. 5(c), showing the results obtained by RPCL. There are only four units moved to the four cluster centers, while the extra unit has been dragged towards the data points only for a while, and then the rival penalized mechanism has driven it back and far away from the four clusters as a spare unit. This phenomenon can also be clearly observed from the dashed learning curve of $\vec{w}_2$ in Fig. 5(d).

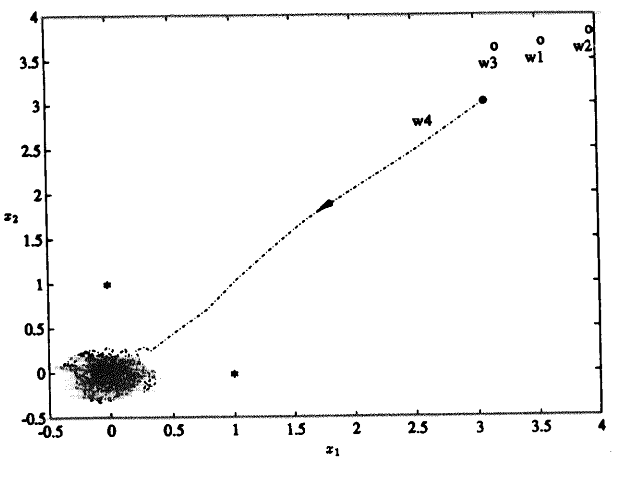
The advantage of RPCL over FSCL can also be observed in Fig. 6(a) and (b), showing results obtained with six units in the CL networks. Fig. 6(a) shows the learning traces obtained by FSCL. Again, all the six units were brought among the four clusters, with four units located around the centers of the clusters, one disturbing unit at a boundary point between the two clusters centered at (-1,0), (0,1), and the other disturbing unit near (0.2,-1). Fig. 6(b) is the result obtained by RPCL. Similar to the case in Fig. 5(c), again only four units moved to each of the four cluster centers, while the two extra units moved towards the clusters only for a while, and then were driven back far away as spare units.

Furthermore, by comparing Fig. 5(b) and (d), one can observe that the learning curves of FSCL became nearly stabilized after about 150 steps, while those of RPCL became near stabilized only after 80 steps. So as we mentioned earlier, the rival penalized mechanism introduced some speedup to the learning process.

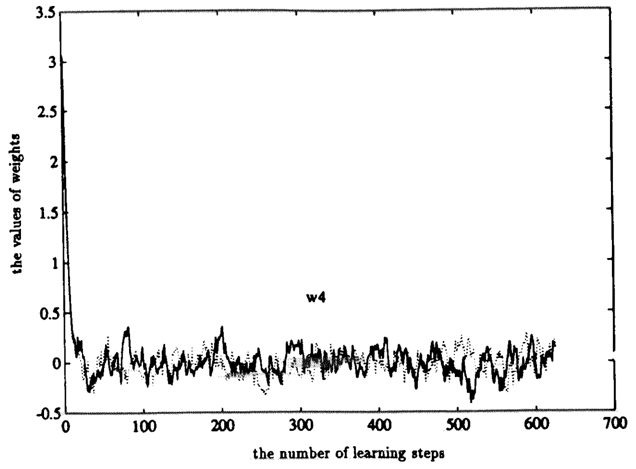
IV. Comparisons on the Performances of RPCL and FSCL for Unsupervised and Supervised Classification

A. Unsupervised Classification

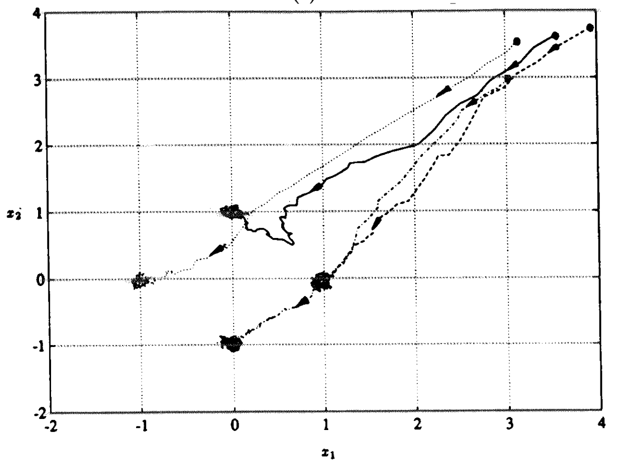
Given a set of data consisting of unlabeled samples from several classes, the task of unsupervised classification (or clustering analysis) is to label every sample in the same



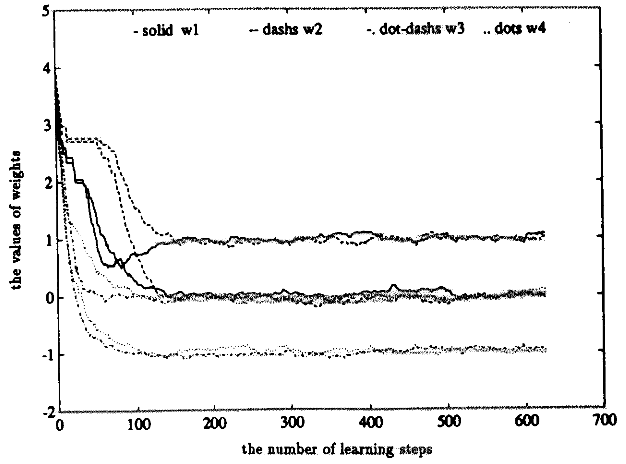
(a)



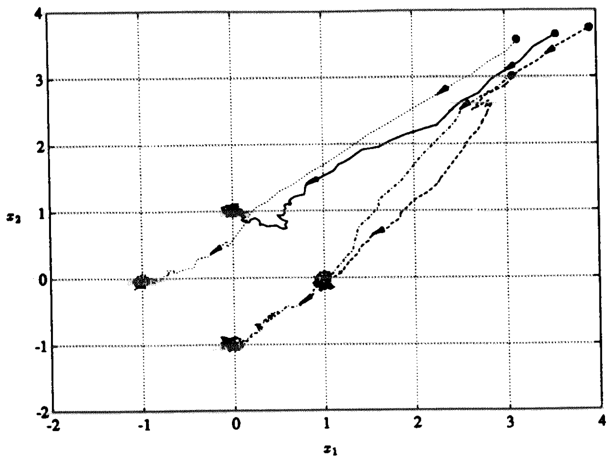
(b)



(c)



(d)



(e)

Fig. 4. Experimental results obtained in the case $k = 4$, i.e., the number of units is equal to the number of clusters in the data. (a) The learning traces, i.e., trajectories of weight vectors during the learning process, obtained by the classical CL algorithm. In this figure, as well as in all the following figures showing learning traces, each of the target locations is marked by an asterisk (in the present figure, only two of the four target locations appear). (b) The learning curves, i.e., the changes of the component variables in every weight vector versus the learning step in the classical CL algorithms. (c) The learning traces obtained by FSCL. (d) The learning curves obtained by FSCL. (e) The learning traces obtained by RPCL.

class by the same symbol such that the data set is divided into several clusters (classes) each associated with a different symbol. The task is harder than supervised classification for two reasons: we have no training samples with known labels, and usually we have no knowledge of the number of classes contained in the data set. In the sequel, we compare the performances of RPCL and FSCL in carrying out the task of unsupervised classification.

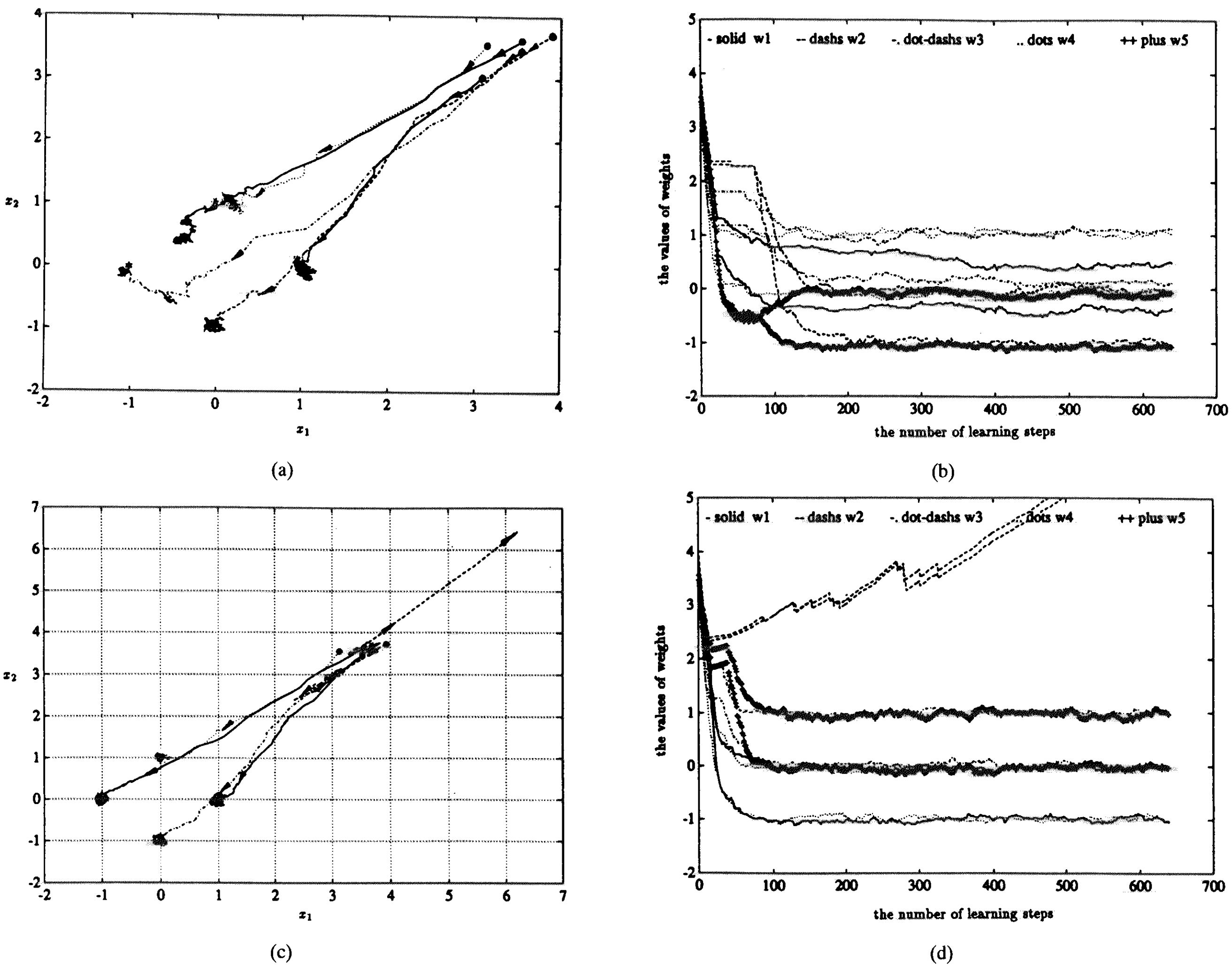


Fig. 5. Experimental results obtained in the case $k = 5 > 4$, i.e., the number of units is larger than the number of clusters in the data. (a) The learning traces obtained by FSCL. (b) The learning curves obtained by FSCL. (c) The learning traces obtained by RPCL. (d) The learning curves obtained by RPCL.

The task is carried out in two steps. First, the weight vectors $\vec{w}_1, \cdots \vec{w}_k$ of units in a CL net are trained by samples from the data set using FSCL or RPCL. Second, each of the trained weight vectors is used as the center of each possible class, and every sample $\vec{x}$ of the data set is labeled as follows.

$$\text{if } i = \arg\min_j ||\vec{x} - \vec{w}_j||^2, \quad \text{then label } \vec{x} \text{ by index } i. \tag{4}$$

As a result, all the samples with the same index i constitute a class, and the number of indices is just the number of classes contained in the data set. Each index has been assigned to at least one sample (or at least m samples, m being a predefined number; samples less than this number are regarded as forming a cluster of noise points).

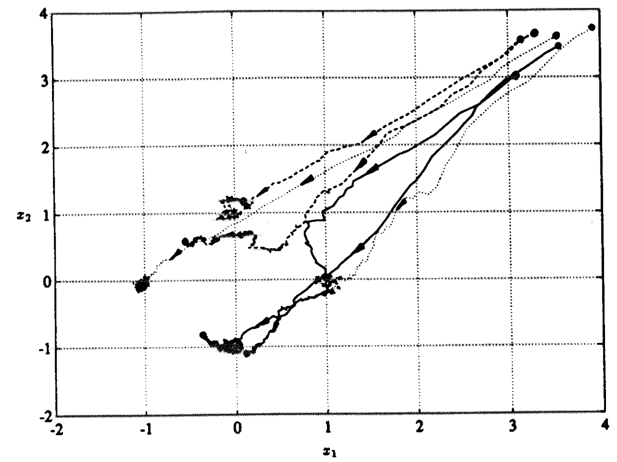
In the following experiments, the data set is again the same as in Fig. 3. The learning rates α_c, α_r are also the same as those used in the experiments of Fig. 4(c)–(e). In the sequel, for convenience we denote the classes with centers located at (0,-1), (0,1), (1,0), and (-1,0) by class 1, 2, 3, and 4, respectively.

We first consider a simpler case of $k = 4$. The classification results obtained by FSCL and RPCL are given in Tables I and II, respectively.

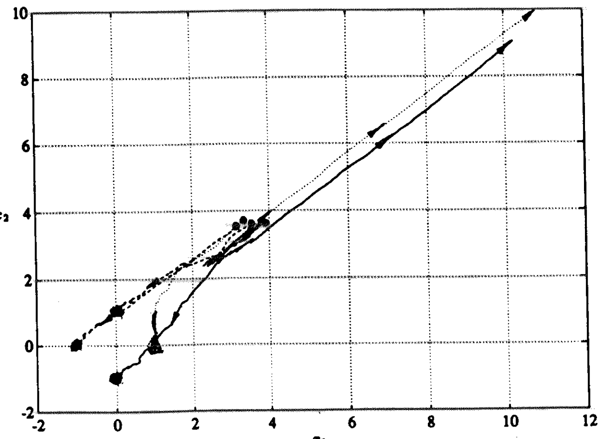
It is seen from the tables that both FSCL and RPCL perform similarly in this case. A recognition rate of about 97% is obtained. This rate is already very good since the four classes in Fig. 3 are not separated and one cannot expect a zero error rate. This suggests that FSCL can work equally well as RPCL if we know the number of classes in the data set in advance.

Second we consider the cases of k larger than 4. Tables III and IV present the classification results obtained when $k = 5$, and Tables V and VI show the results obtained when $k = 6$. Again, the learning processes of FSCL and RPCL are similar to those given in Fig. 5(a)–(d) and Fig. 6(a) and (b), respectively.

It can be seen from the above tables that now the performances of FSCL and RPCL are significantly different. RPCL can still obtain recognition rates of $(96+98+97+97)/400 = 97\%$ when $k = 5$ and $(98 + 98 + 98 + 95)/400 = 97.25\%$ when $k = 6$, which are as good as in the case when $k = 4$. However, the recognition rates obtained by FSCL have been considerably reduced to $(96 + 69 + 85 + 93)/400 = 85.75\%$ when $k = 5$ and $(54 + 75 + 97 + 84)/400 = 77.5\%$ when $k = 6$, due to the influences of the disturbing units. By using FSCL, it follows from Table III that unit number 4 becomes a disturbing unit which forms a mixture group with 48 samples



(a)



(b)

Fig. 6. Experimental results obtained in the case $k = 6 > 4$. (a) The learning traces obtained by FSCL. (b) The learning traces obtained by RPCL.

TABLE I
THE CONFUSION MATRIX OBTAINED BY FSCL WHEN $k = 4$.

	unit$_1$	unit$_2$	unit$_3$	unit$_4$
class 1	0	1	3	96
class 2	98	1	1	0
class 3	1	97	0	2
class 4	2	0	97	1

TABLE II
THE CONFUSION MATRIX OBTAINED BY RPCL WHEN $k = 4$

	unit$_1$	unit$_2$	unit$_3$	unit$_4$
class 1	0	1	2	97
class 2	98	1	1	0
class 3	1	97	0	2
class 4	2	0	97	1

from each of the four classes when $k = 5$; and it follows from Table V that when $k = 6$, units number 4 and 6 become two disturbing units with one drawing 49 samples from classes 1,4 and the other drawing 36 samples from classes 2, 4. In contrast, by using RPCL, unit number 4 in Table IV has been

TABLE III
THE CONFUSION MATRIX OBTAINED BY FSCL WHEN $k = 5$

	unit$_1$	unit$_2$	unit$_3$	unit$_4$	unit$_5$
class 1	0	0	3	1	96
class 2	69	0	1	30	0
class 3	1	85	0	12	2
class 4	1	0	93	5	1

TABLE IV
THE CONFUSION MATRIX OBTAINED BY RPCL WHEN $k = 5$

	unit$_1$	unit$_2$	unit$_3$	unit$_4$	unit$_5$
class 1	0	96	3	0	1
class 2	98	0	1	0	1
class 3	1	2	0	0	97
class 4	2	1	97	0	0

TABLE V
THE CONFUSION MATRIX OBTAINED BY FSCL WHEN $k = 6$

	unit$_1$	unit$_2$	unit$_3$	unit$_4$	unit$_5$	unit$_6$
class 1	1	1	0	45	54	0
class 2	0	1	75	0	0	24
class 3	0	97	1	0	2	0
class 4	84	0	0	4	0	12

TABLE VI
THE CONFUSION MATRIX OBTAINED BY RPCL WHEN $k = 6$

	unit$_1$	unit$_2$	unit$_3$	unit$_4$	unit$_5$	unit$_6$
class 1	1	0	0	98	1	0
class 2	1	0	98	1	1	0
class 3	0	0	1	98	98	0
class 4	95	0	2	0	0	0

driven far away from the other four units. It is just a spare unit which draws none of the samples. Similarly, units 2 and 6 in Table VI also become spare units.

Thus we see that for the case when k is larger than the number of classes in the data set, the performance of the FSCL deteriorates, while the RPCL is able to maintain a good performance. Furthermore, RPCL can automatically find the number of classes in the data set (e.g., in both Tables IV and VI, there are only four clusters obtained by four units) and leave the extra units unused as spare units.

B. Supervised Classification through RBF Net

The RBF network has recently become popular in supervised classification due to its good performance and fast training [4]–[6]. An RBF net consists of one hidden layer of units with the Gaussian radial basis activation functions given by

$$g_j(\vec{x}) = \frac{exp[-\|\vec{x} - \vec{w}_j\|^2/2\sigma_j^2]}{\sum_{i=1}^{k} exp[-\|\vec{x} - \vec{w}_i\|^2/2\sigma_i^2]} \tag{5a}$$

and an output layer of linear units given by $o_i = \sum_{j=1}^{k} w_{ij}^o g_j$, where $\vec{w}_i^o = [w_{i1}^o, \cdots, w_{iq}^o]^t$ is the weight vector of the ith

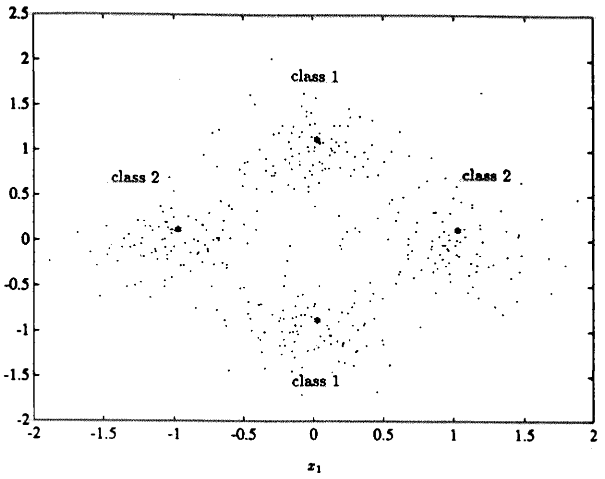


Fig. 7. The data set used for testing a RBF net. There are clusters which are produced in the same way as the data shown in Fig. 3. The two clusters centered at (0,1), (0,1) form class 1, and the two centered at (1,0), (-1,0) form class 2.

output unit, and k, q are the numbers of units in the hidden and output layer, respectively.

As suggested by Moody and Darken [4], the training of a RBF net consists of two steps. First, the center weight vectors $\vec{w}_j, j = 1, \cdots k$ are moved in an unsupervised manner by a version of the k-means algorithm so that they become located at the centers of clusters in the input data set. Then the weight vectors $\vec{w}_i^o$ are simply trained by the delta rule using a set of desired output values d_i and a learning rate α^o:

$$\Delta \vec{w}_{ij}^o = \alpha^o (d_i - o_i) g_j. \tag{5b}$$

A competitive learning algorithm, especially FSCL, has the same function as the k-means algorithm, and thus can take the same role in training the RBF net. However, here we should point out that FSCL as well as the *k-means* algorithm work well only when the hidden unit number k is appropriately selected to be the same as the number of clusters in the input data set, otherwise the performance of the RBF net will deteriorate considerably. The problem is again that one usually does not know the number of clusters in the data set in advance.

In the sequel, we will show through the results of experiments how this problem affects the recognition performance of FSCL, and how RPCL can automatically solve the problem so that good performances are insensitive to the selection of k.

The data set is given in Fig. 7. It consists of two classes which have in total four clusters which are produced in the same way as the data shown in Fig. 3. However, here two clusters form a class. Specifically, the two clusters centered at (0,1), (0,1) form class 1, and the the two centered at (1,0), (-1,0) form class 2. In fact, we confront a "noisy" XOR problem. Two sets of such data are generated. One is used as a training set which consists of 400 samples. They are the same as those in Fig. 3, with each cluster having 100 samples and thus each class having 200 samples. The other is used as a testing set, the 400 samples of which come from the same four Gaussian distributions as the training set, but they are obtained by different random samplings. Again, each cluster has 100 samples, and thus each class has 200 samples.

TABLE VII
THE CONFUSION MATRIX OBTAINED BY FSCL WHEN $k = 4$

	Training		Set	Testing		Set
	class 1	class 2	reject	class 1	class 2	reject
class 1	189	6	5	189	5	6
class 2	6	187	7	5	189	6

TABLE VIII
THE CONFUSION MATRIX OBTAINED BY RPCL WHEN $k = 4$

	Training		Set	Testing		Set
	class 1	class 2	reject	class 1	class 2	reject
class 1	189	6	5	189	6	5
class 2	6	186	8	5	189	6

TABLE IX
THE CONFUSION MATRIX OBTAINED BY FSCL WHEN $k = 5$

	Training		Set	Testing		Set
	class 1	class 2	reject	class 1	class 2	reject
class 1	174	4	22	167	2	31
class 2	16	170	14	13	170	17

TABLE X
THE CONFUSION MATRIX OBTAINED BY RPCL WHEN $k = 5$

	Training		Set	Testing		Set
	class 1	class 2	reject	class 1	class 2	reject
class 1	189	6	5	189	5	6
class 2	6	187	7	5	190	5

The first step of training, i.e., locating the weight vectors of hidden RBF units, is performed in the same way as we did earlier in the case of unsupervised classification. Then these trained weight vectors are fixed, and we train the weight vectors of the top linear units by (5b). The Gaussian activation functions g_j of (5a) are calculated under the parameter $\sigma^2 = 0.1$. The same parameter is used in the four Gaussian distributions when we generate the data set. Again, we first record the recognition results obtained in the case $k = 4$, in the following Tables VII and VIII.

We see that FSCL and RPCL work similarly well in this case, with a recognition rate around 94% and an error rate below 3% for both the training and testing sets.

However, we find that the performance changes considerably in the cases when the number of hidden units k is larger than 4 or the number of clusters in the data set. Tables IX and X present the classification results obtained when $k = 5$, and Tables XI and XII show the results for $k = 6$.

RPCL can still maintain a recognition rate around 93% and an error rate below 3% for both the training and testing sets for both cases $k = 5$ and $k = 6$. However, the recognition rates obtained by FSCL have been reduced down to around

TABLE XI
THE CONFUSION MATRIX OBTAINED BY FSCL WHEN $k = 6$

	Training		Set	Testing		Set
	class 1	class 2	reject	class 1	class 2	reject
class 1	171	3	26	171	1	28
class 2	10	170	20	12	172	17

TABLE XII
THE CONFUSION MATRIX OBTAINED BY RPCL WHEN $k = 6$

	Training		Set	Testing		Set
	class 1	class 2	reject	class 1	class 2	reject
class 1	190	4	6	185	6	9
class 2	7	189	4	3	185	12

85% in the cases of $k = 5$ and $k = 6$, while the error rate is still around 3%. That is, the performance of FSCL deteriorates considerably.

Before closing this section, we would like to further note that in fact the extra units could be discarded from the RBF net after the weight vectors of the hidden units have been trained by using RPCL. These extra units can be detected simply by doing unsupervised classification as indicated in the first part of this section. The exclusion of the extra units can save some computation cost of the RBF net. Moreover, it may also help to keep a good generalization ability. By comparing the results given in Tables VIII, X, and XII, we can see that as the number of extra units increases, the generalization ability is slightly reduced although the good recognition rates prevail.

V. CURVE DETECTION BY RPCL

Detecting curves (straight line, circle, ellipse, etc.) from an image is one of the basic tasks in image processing and machine vision. In the traditional pattern recognition literature, a lot of methods have been proposed for solving this problem. However, the study of using neural network techniques for such problems may cast some new light on this classical but important task. In [22], motivated by an investigation into the learning characteristics of the Kohonen self-organizing map [14], we developed a new neural network technique, a Hough-like technique called *Randomized Hough Transform* (RHT), for detecting curves. It has been shown [22] that RHT has several advantages over the conventional Hough transform (HT) and can improve the performance of HT significantly. In the sequel, we further study the possibility of using a competitive learning model for the curve detection tasks.

Let us assume a binary image (which is usually obtained from a gray-level image by some conventional or neural techniques, , e.g., the Canny operator [23] or the Marr–Hildreth operators [24]). Let $F(\vec{a}, \vec{x}) = 0$ be the parametric equation of a curve with $\vec{a} = [a_1, \cdots, a_m]$ a vector of parameters and $\vec{x} = [x_1, x_2]$ the coordinates of a pixel in the image. If the image contains one curve expressible by the equation, then there must be a number of "white" or "on" pixels $\vec{x}_i, i = 1, \cdots, n$ which all satisfy the equation $F(\vec{a}, \vec{x}_i) = 0$. When $n \geq m$, the parameter vector $\vec{a}$ can often be solved. This problem is usually called a curve fitting problem and it is not difficult to solve. However, if the image contains a number of groups of pixels (e.g., a number of straight line segments or a number of circles) with each group lying on its own curve expressible by the parametric equation $F(a, \vec{x}) = 0$, with the parameter $\vec{a}$ differing from curve to curve, we need to classify every pixel into one of the groups. Thus it is a type of clustering analysis problem. However the clusters here are more complex than what we have studied in the earlier section.

Let's try to express the problem as a competitive learning problem. Assume we have a number k of neural units with their outputs denoted by $u_i, i = 1, \cdots, k$ and weight vectors by $\vec{a}_i, i = 1, \cdots, k$. For an input pixel $\vec{x}$, we let the output of each unit be given by

$$u_i = \epsilon_i^2, \quad \epsilon_i = F(\vec{a}_i, \vec{x}). \tag{6a}$$

Now each unit stands for one possible parametric curve in the image. Our basic idea is to use competitive learning to modify the weight vectors of these units so that after training the output of each unit can selectively become the minimum for the pixels drawn from a specific curve. Then the weight vectors of the units would directly give estimates for the parameter vectors of the curves. The implementation of this idea is quite simple—just the repeated use of the following two steps.

Step 1: Randomly take a pixel $\vec{x}$ from D, compute $u_i, i = 1, \cdots, k$ by (6a) and then find the index c with $u_c = \min_i u_i$.

Step 2: Update the weight vector $\vec{a}_c$ by $\vec{a}_c := \vec{a}_c - \alpha_c(\partial\epsilon_c^2/\partial\vec{a}_c)$.

There D is a set containing all the "on" pixels of the image, α_c is the learning rate, and each $\vec{a}_i$ is initialized randomly.

This simple procedure is just what was proposed earlier by two of the authors in [23]. Several further improvements are possible as suggested in the following.

• First, ϵ_i given in (6a) can be replaced by

$$r_i = \frac{F(\vec{a}_i, \vec{x})}{\|\partial F(\vec{a}_i, \vec{x})/\partial\vec{x}\|} \tag{6b}$$

where $|r_i|$ is the orthogonal distance from $\vec{x}$ to the curve, yielding a better error criterion than ϵ_i. Minimization of ϵ_i^2 actually corresponds to the minimization of the conventional least square (LS) fitting errors, while the minimization of r_i^2 corresponds to the minimization of the so called total least square (TLS) fitting errors [20]. We have shown that the results of curve fitting obtained from TLS are considerably better than those from LS [20].

• Second, FSCL can be used to replace the simple competitive learning, i.e., we can replace the above two learning steps by the following.

Step 1: Randomly take a pixel $\vec{x}$ from D, compute $u_i = r_i^2$ by (6b) and then find the index c with $\gamma_c u_c = \min_j \gamma_j u_j$. Here $\gamma_j = n_j / \sum_{i=1}^{k} n_i$ is the same frequency parameter as the one used for FSCL in the earlier sections of this paper.

Step 2: Update the weight vector $\vec{a}_c$ by $\vec{a}_c := \vec{a}_c - \alpha_c(\partial r_c^2/\partial\vec{a}_c)$.

• Third, it is better to use RPCL to replace FSCL, i.e.,

Step 1: Randomly take a pixel $\vec{x}$ from D, compute $u_i = r_i^2$ by (6b) and then find the index c with $\gamma_c u_c = \min_j \gamma_j u_j$ and index r with $\gamma_r u_r = \min_{j \neq c} \gamma_j u_j$.

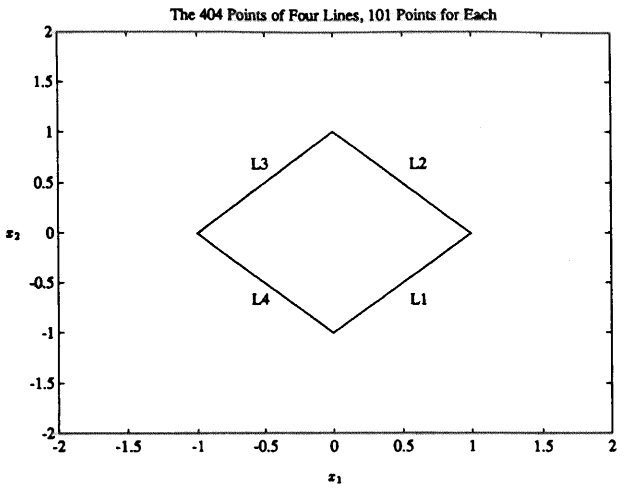


Fig. 8. The data set used for testing curve detection by FSCL and RPCL. It consists of pixels lying on four line segments which constitute the diamond shape. The four lines are $L1 : x_1 - x_2 = 1$, $L2 : x_1 + x_2 = 1$, $L3 : -x_1 + x_2 = 1$, $L4 : -x_1 - x_2 = 1$, and on each line there are 101 pixels.

Step 2: Update the weight vectors $\vec{a}_c$, $\vec{a}_r$ by $\vec{a}_c := \vec{a}_c - \alpha_c \partial r_c^2 / \partial \vec{a}_c$ and $\vec{a}_r := \vec{a}_r + \alpha_r \partial r_r^2 / \partial \vec{a}_r$.

That is, the winner u_c is awarded using a learning rate α_c, and the rival u_r of u_c is penalized using a smaller learning rate $\alpha_r \ll \alpha_c$.

In the above, the curve is considered in its general form $F(\vec{a}, \vec{x}) = 0$. For a specific curve, the learning rules can be written more precisely. For example, for detecting a line $\vec{a}^t \vec{x} + c = 0$, we have

$$\frac{\partial r_i^2}{\partial \vec{a}_i} = \frac{2r_i(\vec{x} - r_i \vec{a}_i / \|\vec{a}_i\|)}{\|\vec{a}_i\|}, \qquad r_i = \frac{\vec{a}^t \vec{x} + c}{\|\vec{a}_i\|} \tag{7a}$$

from which $\vec{a}_c := \vec{a}_c - \alpha_c \partial r_c^2 / \partial \vec{a}_c$ in both FSCL and RPCL could be computed by direct substitution. However, let us replace it by

$$\vec{a}_c := \vec{a}_c - \alpha_c \frac{r_c(\vec{x} - r_c \vec{a}_c / \|\vec{a}_c\|)}{\|\vec{x}\|} \tag{7b}$$

and let us replace $\vec{a}_r := \vec{a}_r + \alpha_r \partial r_r^2 / \partial \vec{a}_r$ in RPCL by

$$\vec{a}_r := \vec{a}_r + \alpha_r \frac{r_r(\vec{x} - r_r \vec{a}_r / \|\vec{a}_r\|)}{\|\vec{x}\|}. \tag{7c}$$

Now the scalar factor $\frac{1}{\|\vec{a}_i\|}$ in (7a) has been replaced by $1/\|\vec{x}\|$. Mathematically, this is equivalent to redefining the learning rates α_c and α_r. The reason is to let the learning rate be adjusted according to input $\vec{x}$, resulting in more stable learning under noise. The details are similar to those given in [20].

In the sequel, we compare the performances of FSCL and RPCL on a four-line detection problem through several experiments.

As shown in Fig. 8, our data consists of the "on" pixels (in this case, black pixels) in an image containing four line segments which form a diamond. The theoretical equations of the four lines are $L1 : x_1 - x_2 = 1$, $L2 : x_1 + x_2 = 1$, $L3 : -x_1 + x_2 = 1$, $L4 : -x_1 - x_2 = 1$, and each line segment in the actual digital binary image consists of 101 pixels. Now the problem is to detect the two parameters of each line under the line equation $F(\vec{a}, \vec{x}) = a_1 x_1 + a_2 x_2 - 1 = 0$ in an unsupervised manner, using only the available pixels. In all the experiments, we let for simplicity the number of neural units be $k = 4$, which is the same as the number of line segments. The weight vectors are all initialized by random numbers between [0, 1]. The learning rates are $\alpha_c = 0.01$, $\alpha_r = 0.001$.

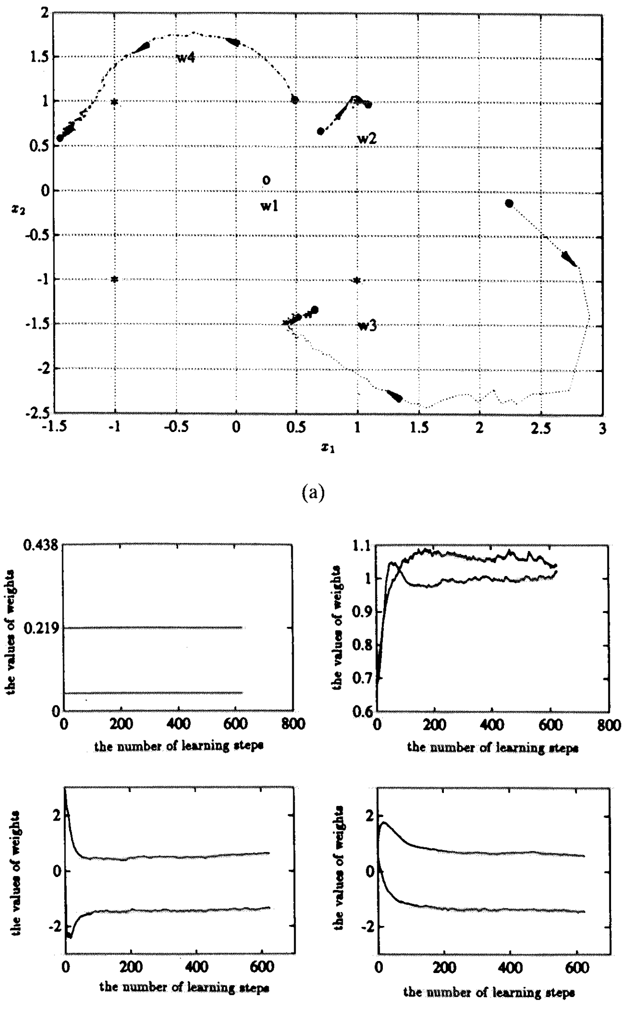
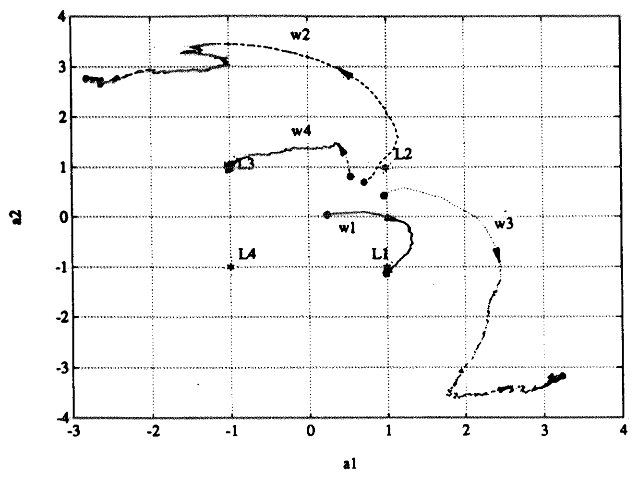


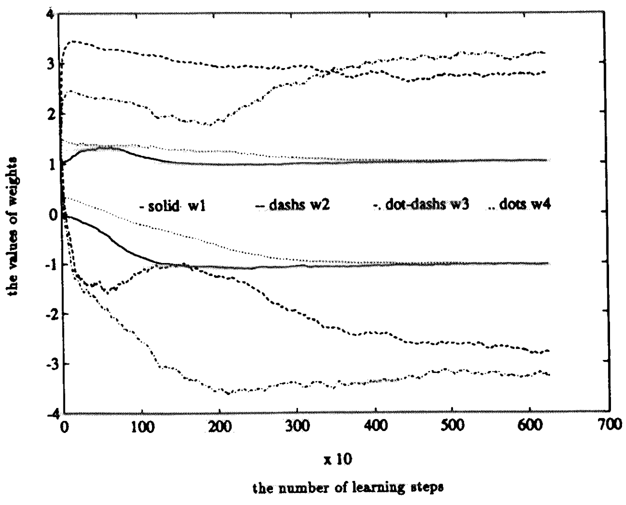
Fig. 9. The results obtained by the classical CL with $k = 4$. (a) The learning traces. (b) The learning curves. Only one curve was detected.

Fig. 9(a) and (b) shows the learning traces and learning curves for the four weight vectors obtained by using the classical CL. The result is very poor although the learning is quickly stabilized (see Fig. 9(b)). Now only one vector $\vec{w}_2$ reached a correct point (1,1), which is the parameter pair of line $L2$. Two other vectors converged to wrong points, and another vector was simply dead from the beginning.

Fig. 10(a) and (b) give results obtained by FSCL. Now, some improvement is obtained. Two vectors $\vec{w}_1$ and $\vec{w}_4$ reached two parameter pairs (1,-1) and (-1,1) of lines $L1, L3$. However, there are still two vector $\vec{w}_2, \vec{w}_3$ which moved towards the wrong points $(-3,3), (3,-3)$. So, we see that the four-line problem seems much harder than the four-cluster problem given in Fig. 5, where FSCL can very easily produce

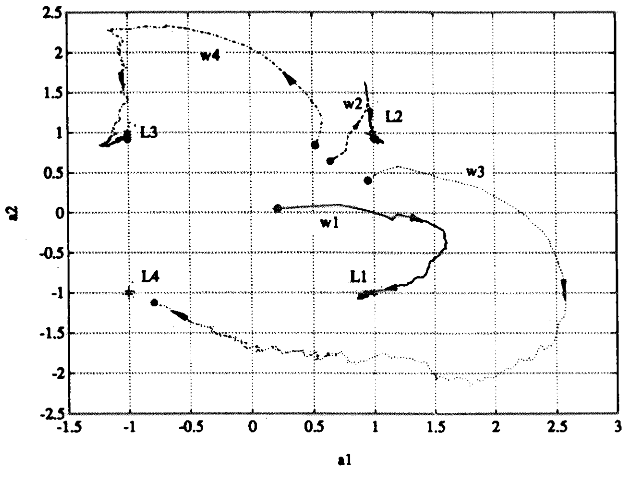


(a)

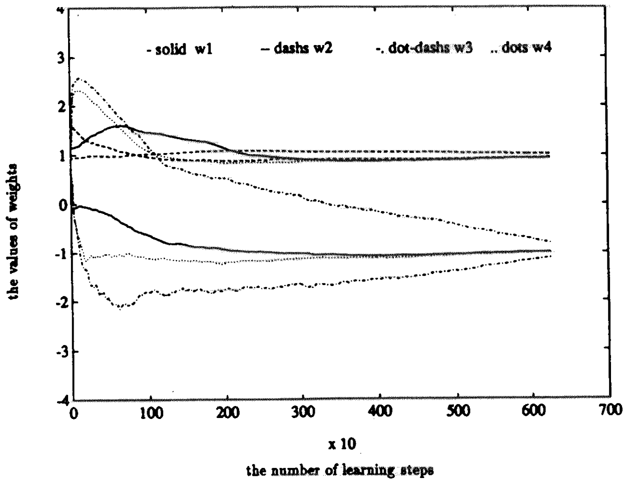


(b)

Fig. 10. The results obtained by FSCL with $k = 4$. (a) The learning traces, (b) The learning curves. Two curves were detected.



(a)



(b)

Fig. 11. The results obtained by RPCL with $k = 4$. (a) The learning traces, (b) The learning curves. All four curves were detected.

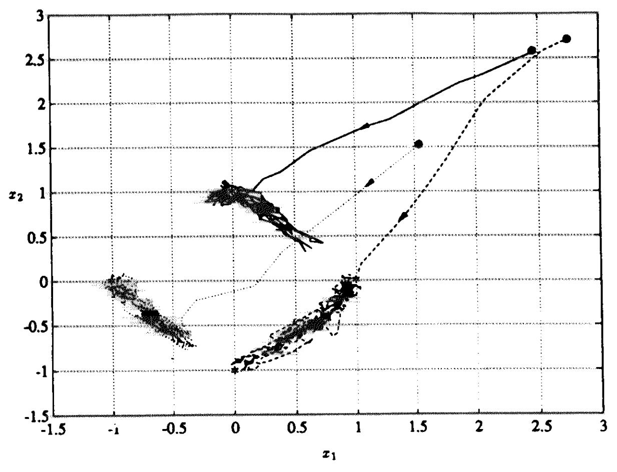
good results. But, here in Fig. 10(a) and (b) the performance of FSCL is obviously quite suboptimal.

However, this problem can be solved by RPCL as shown in Fig. 11(a) and (b). Now the two vectors $\vec{w}_2$ and $\vec{w}_4$ rapidly converge to two parameter pairs (1,1) and (-1,1) of lines $L2, L3$ (see Fig. 11(b)). The other two vectors $\vec{w}_1, \vec{w}_3$ also moved towards two parameter pairs (1,-1), (-1,-1) of lines $L1$ and $L4$ although they converge more slowly. In particular, $\vec{w}_3$ has taken a long indirect way to avoid the barrier produced by the trace of $\vec{w}_1$.

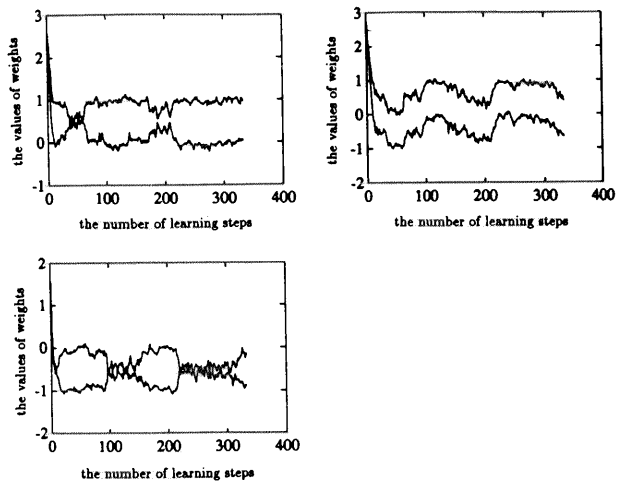
VI. Some Further Remarks On The Number Of Units

In the previous sections we have only considered the cases that the number k of units in the CL net is equal to or larger than the number of clusters in the input data. There will naturally arise a question: What will happen for FSCL and RPCL when k is smaller than the number of clusters, and is it possible to detect this occurrence and use it to find a good value for k? This section will answer this question in a positive way for the RPCL network.

Fig. 12(a) and (b) show the results obtained by both FSCL and RPCL when $k = 3$. The data set used here is still the four clusters given in Fig. 3, and thus k is too small. It can be seen from Fig. 12(a) and more clearly from Fig. 12(b) that all the three weight vectors oscillate strongly between different clusters and the learning can not stabilize with the fixed learning rate $\alpha_c = 0.05$. Moreover, if we reduce α_c to zero as the learning proceeds, then each of the three weight vectors will be finally stabilized at some point which is either away from a cluster center at a certain distance or is a boundary point of two different clusters, as marked by the small black rectangles in Fig. 12(a). Clearly, when this kind of learning behavior occurs for unsupervised classification or RBF net, the performance will not be good, since each of three weight vectors will draw samples from two different classes. Thus, like the k-means algorithm, both FSCL and RPCL will work poorly if k is too small.



(a)



(b)

Fig. 12. Experimental results by both FSCL and RPCL on the data set given in Fig. 3, with $k = 3$. In this case, all three weight vectors oscillate strongly between different clusters and the learning cannot stabilize with the fixed learning rate $\alpha_c = 0.05$. (a) The learning traces, (b) The learning curves.

Fortunately, it is quite easy for RPCL to avoid such situations, e.g., just by the simple method of choosing a large number as k such that it is surely larger than the number N_c of clusters in the data set. This may not be desirable, however, since a large $k \gg N_c$ will increase the training time. A better way is the following: we can start with a small number or a guess for k and perform unsupervised classification as was done in Section IV. By checking whether there are spare units left, we would know if $k > N_c$. If yes, then take the present results as the solutions; if not, increase k by adding a positive increment, or double it and then perform unsupervised classification again. The same procedure can be repeated until $k > N_c$. Actually, few iterations are needed for a moderately large number as the starting value of k, especially when there are not too many clusters in the data set.

VII. Conclusions

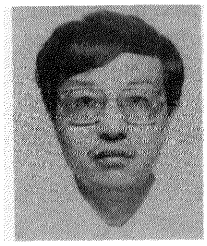
Similar to the problem of selecting an appropriate k in the classical k-means algorithm, the present Competitive Learning algorithms have also a similar crucial problem: the selection of an appropriate number of neural units. It has been shown that FSCL significantly deteriorates its performance when the number of units is inappropriately chosen. A new algorithm called RPCL has been developed from the basic idea that for each input not only the winner unit is modified to adapt to the input, but also its rival is de-learned by a smaller learning rate. RPCL can automatically allocate an appropriate number of units for an input data set. The experimental results have shown that RPCL outperforms FSCL when they are used for unsupervised classification, for training a RBF network, and for curve detection in digital images.

References

[1] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. London, U.K.: Prentice-Hall, 1982.
[2] J. Makhoul, S. Rpucos, and H. Gish, "Vector quantization in speech coding," *Proc. IEEE*, vol.73, no. 11, pp. 1551-1558, 1985.
[3] N. Nasrabadi and R. A. King, "Image coding using vector quantization: A review," *IEEE Trans. Commun.* vol. 36, no. 8, pp. 957-971, 1988.
[4] J. Moody and C. Darken,"Fast learning in networks of locally tuned processing units," *Neural Computation*, vol. 1, pp. 281-294, 1989.
[5] T. Poggio and F. Girosi, "Regularization algorithms for learning that are equivalent to multilayer networks," *Science*, vol. 247, pp. 978-982, 1990.
[6] S. J. Nowlan and G. E. Hinton, "Evaluation of adaptive mixtures of competing experts," in *Advances in Neural Information Processing System 3*,R. P. Lippmann, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufman, 1991, pp. 774–780.
[7] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Science*, vol. 9, pp. 75-112, 1985.
[8] S. Grossberg, "Competitive learning: from iterative activation to adaptive resonance," *Cognitive Science*, vol. 11, pp. 23-63, 1987.
[9] R. Hecht-Nielsen, "Counterpropagation networks," *Appl. Opt.*, vol. 26, pp. 4979–4984, 1987.
[10] S. Grossberg, "Adaptive patten classification and universal recording: I. parallel development and coding of neural feature detectors," *Biol. Cybern.*, vol. 23, pp. 121-134, 1976.
[11] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54–115, 1987.
[12] G. A. Carpenter and S. Grossberg, "ART 2: Self-organization of stable category recognition codes for analog output patterns," *Appl. Opt.*, vol. 26, pp. 4919-4930, 1987.
[13] G. A. Carpenter and S. Grossberg, "ART 3: Hierarchical searching using chemical transmitters in self-organizing pattern recognition architectures," *Neural Networks*, vol. 3, pp. 129-152, 1990.
[14] T. Kohonen, *Self-Organization and Associative Memory*. Berlin: Springer-Verlag, 1988.
[15] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, pp. 1464-1480, 1990.
[16] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.
[17] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex," *J. Neuroscience*, vol. 2, pp. 32–48, 1982.
[18] D. Desieno, "Adding a conscience to competitive learning," in *Proc. IEEE Int. Conf. Neural Networks*, 1988, vol. I, pp. 117-124.
[19] S. C. Ahalt, A. K. Krishnamurty, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, pp. 277-291, 1990.
[20] L. Xu, E. Oja, and C. Y. Suen, "Modified Hebbian learning for curve and surface fitting," *Neural Networks*, vol. 5, pp. 441-457, 1992.
[21] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.*, vol. 22, pp. 400-407, 1951.
[22] L. Xu, E. Oja, and P. Kultanen, "A new curve detection methods: Randomized Hough Transform (RHT)," *Pattern Recognition Letters*, vol. 11, no. 5, pp. 331-338, 1990.
[23] L. Xu and E. Oja, "Extended self-organizing map for curve detection," *Research Report 16*, Dept. Information Technology, Lappeenranta University of Technology, Sept. 1989.
[24] J. Canny, "A computational approach to edge detection," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. 8, pp. 679-698, 1986.

[25] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. Roy. Soc. London, B*, vol. 207, 1980, pp. 187-217.

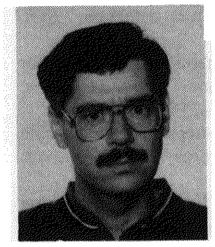
[26] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to Theory of Neural Computation*. New York: Addison-Wesley, 1991.



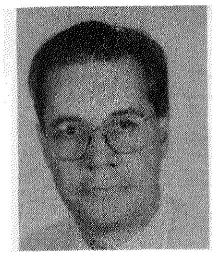
Lei Xu received the master's and doctoral degrees, both in pattern recognition and signal processing, from Tsinghua University in 1984 and 1987, respectively,

From June 1987 to June 1988, he was a postdoctoral fellow at he Department of Mathematics, Peking University, where he is currently an Associate Professor. He visited the Department of Information Technology, Lappeenranta University of Technology, Finland, and the Department of Computer Science, Concordia University, Canada, as a Senior Researcher and Research Associate, each for one year, respectively. From September 1991 to September 1992, he was a Visiting Scientist in the Division of Applied Sciences, Harvard University. From September 1992 to May 1993, he was a postdoctoral Research Associate in the Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology. He has published several dozen papers in the areas of neural networks, computer vision/pattern recognition, artificial intelligence, and signal processing. He has served as a reviewer for a number of international journals and conferences on neural networks, and as a Guest Editor of a special issue of the *Journal of Artificial Neural Networks* and as the organizer of one of the NIPS92 Postconference Workshops.

Dr. Xu was one of the winners of the First Fok Ying Tung Education Foundation Prize for young university teachers of the People's Republic of China, 1988. He was also the second of the winners of the 2nd Beijing Young Scientists Prize awarded by the Beijing Association for Science and Technology in 1988, and the first of the winners of the excellent paper award for young researchers in the 1988 National Conference of the Chinese Automation Society, May 1988.



Adam Krzyżak (M'86), biography not available at the time of publication.



Erkki Oja (S'76–M'77–SM'90) was born in Helsinki, Finland, in 1948. He received the M.Sc. degree and the Dr.Tech. degree (with distinction) from the Helsinki University of Technology in 1972 and 1977, respectively.

He was a visiting scientist at Brown Univeristy in 1977–1978, and at the Tokyo Institute of Technology, Japan, in 1983–1984. For the academic year 1990–1991, he held the Toshiba Visiting Professor's Chair at the Department of Computer Science, Tokyo Institute of Technology. Since 1987, he has been Professor of Computer Science at the Lappeenranta University of Technology, Finland. He is the author of a number of journal papers and book chapters on pattern recognition, computer vision, and neural computing, and the book *Subspace Methods of Pattern Recognition*, which has been translated into Chinese and Japanese. His present research interests are in applying neural networks to computer vision and the study of subspace, PCA, and self-organizing networks. He has lectured on neural computation and pattern recognition in universities, and for industry, in Europe and Japan.

Dr. Oja has served in the scientific and organization committees of a number of conferences, recently including INNC Paris '90, Cognitiva '90, ICANN '91, IJCNN '91, ECAI '91, ICPR '92, IEEE ICNN '93, WCNN '93, and IJCNN '93. He was a General Chairman of the 6th Scandinavian Conference on Image Analysis in 1989. He is a member of ACM, INNS, ENNS, Finnish Academy of Sciences, and the Finnish Academy of Technical Sciences. He is a past Chairman of the Finnish Pattern Recognition Society, past Vice President of the IEEE Finland Section, a member of the Governing Board of the International Association of Pattern Recognition , and a member of the editorial boards of the *International Journal of Neural Systems, Neural Networks, Neural Computation*, and the IEEE TRANSACTIONS ON NEURAL NETWORKS.