Self-Splitting Competitive Learning: A New On-Line Clustering Paradigm

Ya-Jun Zhang and Zhi-Qiang Liu, *Senior Member, IEEE*

***Abstract*—Clustering in the neural-network literature is generally based on the competitive learning paradigm. This paper addresses two major issues associated with conventional competitive learning, namely, sensitivity to initialization and difficulty in determining the number of prototypes. In general, selecting the appropriate number of prototypes is a difficult task, as we do not usually know the number of clusters in the input data *a priori.* It is therefore desirable to develop an algorithm that has no dependency on the initial prototype locations and is able to adaptively generate prototypes to fit the input data patterns. In this paper, we present a new, more powerful competitive learning algorithm, self-splitting competitive learning (SSCL), that is able to find the natural number of clusters based on the one-prototype-take-one-cluster (OPTOC) paradigm and a self-splitting validity measure. It starts with a single prototype randomly initialized in the feature space and splits adaptively during the learning process until all clusters are found; each cluster is associated with a prototype at its center. We have conducted extensive experiments to demonstrate the effectiveness of the SSCL algorithm. The results show that SSCL has the desired ability for a variety of applications, including unsupervised classification, curve detection, and image segmentation.**

***Index Terms*—Clustering, competitive learning, one-prototype-take-one-cluster (OPTOC), self-splitting, unsupervised learning, validity measure, winner-take-all (WTA).**

I. Introduction

DATA CLUSTERING aims at discovering and emphasizing structure which is hidden in a data set. Thus the structural relationships between individual data points can be detected. In general, clustering is an unsupervised learning process [1], [2]. Traditional clustering algorithms can be classified into two main categories: One is based on model identification by parametric statistics and probability, e.g., [3]–[7]; the other that has become more attractive recently is a group of vector quantization-based techniques, e.g., self-organizing feature maps (SOFMs) [8]–[12], the adaptive resonance theory (ART) series [13]–[17], and fuzzy logic [18]–[26]. In the neural-networks literature, clustering is commonly implemented by distortion-based competitive learning (CL) techniques [2], [27]–[31] where the prototypes correspond to the weights of *neurons*, e.g., the center of their receptive field in the input feature space. A common trait of these algorithms is a competitive stage which precedes each learning steps and decides to what extent a neuron may adapt its weights to a new input pattern [32]. The goal of competitive learning is the minimization of the *distortion* in clustering analysis or the *quantization error* in vector quantization.

A variety of competitive learning schemes have been developed, distinguishing in their approaches to competition and learning rules. The simplest and most prototypical CL algorithms are mainly based on the *winner-take-all* (WTA) [33] (or hard competitive learning) paradigm, where adaption is restricted to the *winner* that is the single neuron prototype best matching the input pattern. Different algorithms in this paradigm such as LBG (or generalized Lloyd) [34]–[36] and k-Means [37] have been well recognized. A major problem with the simple WTA learning is the possible existence of *dead nodes* or the so-called *under-utilization* problem [38]–[40]. In such cases, some prototypes, due to inappropriate initialization can never become a winner, therefore, have no contribution to learning. Significant efforts have been made in the literature to deal with this problem. By relaxing the WTA criterion, soft competition scheme (SCS) [31], neural-gas network [41] and fuzzy competitive learning (FCL) [20] treat more than a single neuron as winners to a certain degree and update their prototypes accordingly, resulting in the *winner-take-most* (WTM) paradigm (*soft competitive learning*). WTM decreases the dependency on the initialization of prototype locations; however, it has an undesirable side effect in clustering analysis [28]: since all prototypes are attracted to each input pattern, some of them are detracted from their corresponding clusters. As a consequence, these prototypes may become biased toward the global mean of the clusters. Kohonen's SOFM [8] is a learning process which takes WTM strategy at the early stages and becomes a WTA approach while its neighborhood size reduces to unity as a function of time in a predetermined manner. However, its main purpose is to form a topographic feature map which is a more complex task than just clustering analysis [29]. Several other algorithms, such as additive conscience competitive learning [38] and *convex bridge* [40], modulate the *sensitivity* of prototypes, so that less frequent winners increase their chances to win next time. Reference [42] introduced a *conscience* parameter to reduce the rate of frequent winners by making them "guilty." Frequency sensitive competitive learning (FSCL) [43] uses such a strategy which in some cases significantly improves the classical CL algorithms. Moreover, fuzzy frequency sensitive competitive learning (FFSCL) [20] combines the frequency sensitivity with fuzzy competitive learning. Since both FSCL and FFSCL use non-Euclidean distance to determine the winner, they may lead to the problem of *shared clusters* in the sense that a number of prototypes may be

Manuscript received March 21, 2000; revised November 15, 2000 and November 14, 2001.

Y.-J. Zhang is with the Department of Computer Science and Software Engineering, The University of Melbourne, Victoria 3010, Australia.

Z.-Q. Liu is with the School of Creative Media, City University of Hong Kong, Hong Kong, China (e-mail: z.liu@computer.org).

Publisher Item Identifier S 1045-9227(02)02403-7.

updated into the same cluster during the learning process. This problem was considered by Xu *et al.* in their rival penalized competitive learning (RPCL) algorithm [29]. The basic idea in RPCL is that for each input pattern, not only the weight of the frequency-sensitive winner is learned to shift toward the input pattern, but also the weight of its rival (the *2nd* winner) is delearned by a smaller learning rate. The rival is always pushed away reducing its interference in the competition. However, recently Liu *et al.* have pointed out that RPCL also has some new problems: it is sensitive to the penalizing rate. If the penalizing rate is not appropriately selected, a prototype can be unfairly penalized resulting in overpenalization, or, on the contrary, underpenalization [28].

Another well-known critical problem with competitive learning is the difficulty in determining the number of clusters [21]. It must be appropriately presumed, otherwise the algorithm will perform badly. Determining the optimum number of clusters is a largely unsolved problem. The algorithms discussed above do not adequately tackle the problems caused by the inappropriate number of initial prototypes. Although RPCL is applicable to some cases that the number of prototypes are larger than that of clusters, it is unable to deal with the situation that the number of prototypes is less than the actual number of clusters. To avoid this problem, Xu *et al.* suggested to use a large number of prototypes initially [29]. However, it is difficult in most cases to choose a reasonably large number because of the lack of prior knowledge in the data set. In addition, this solution will result in unnecessary training and computation. There are similar problems in the *fuzzy C spherical shells* algorithm proposed by [24] and the *robust competitive clustering algorithm* (RCA) very recently proposed by [21] which start with an over-specified number of clusters and merges the compatible clusters during the learning process. The growing cell structure (GCS) [45] and growing neural gas (GNG) [46] algorithms are different from the previously described models since the number of prototypes is increased during the self-organization process. Both of them, however, have neither insertion validity measure nor stop validity measure. The insertion is judged at each prespecified number of iterations and the stop criterion is simply the network size or some *ad hoc* subjective criteria on the learning performance. In addition, the new prototype is inserted near the neuron that has accumulated most distortion. This, however, is not applicable to classifying the clusters with different sizes because a well-partitioned large cluster may still have the most distortion, which may trick GCS or GNG to generate a new, redundant prototype to share this cluster. Moreover, the distortion for each neuron will not be reset after a new one has been inserted. It is also required that the initial number of prototypes be at least two, which is not always the right choice since sometimes a single cluster may exist in the data set.

In this paper, we present a new competitive learning algorithm, self-splitting competitive learning (SSCL) that is capable of tackling the two critical, difficult problems in competitive learning. In SSCL, we introduce a new learning paradigm: One-prototype-take-one-cluster (OPTOC). To our best knowledge, OPTOC is different from all the existing algorithms in the CL literature. With the OPTOC learning paradigm, SSCL starts from only a single prototype which is randomly initialized in the feature space. During the learning period, one of the prototypes (initially, the only single prototype) will be chosen to split into two prototypes based on a split validity measure. This self-splitting behavior terminates if no more prototype is suitable for further splitting. After learning by the SSCL algorithm, each cluster is labeled by a prototype located at its center. We have performed extensive experiments to demonstrate the performance of SSCL algorithm including unsupervised clustering analysis, curve detection, and image segmentation. Since SSCL does not need *a priori* knowledge about how many clusters or how many curves, or in general, how many types of *things* in the input data set, which is normally the case when humans are doing the sorting, consider for instance, the cashier at the supermarket. SSCL is a valuable alternative to unsupervised learning and offers a great potential in many real-world applications.

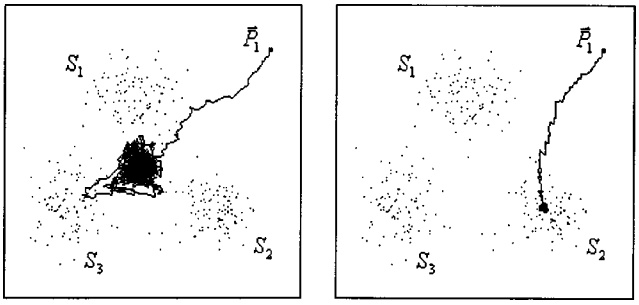
The remainder of this paper is organized as follows. In Section II, we describe in detail the SSCL Algorithm and analyze its properties. Sections III and IV presents the experimental results on clustering analysis and curve detection, and Section V demonstrates the capabilities of SSCL in range image segmentation. Finally, Section VI gives the summary and conclusion.

II. Self-Splitting Competitive Learning Algorithm

A. One Prototype Takes One Cluster

Clustering in the neural-network literature can be viewed as distortion-based competitive learning. The *nearest neighbor* and the *centroid* conditions are the two necessary conditions to achieve optimal learning. To start the learning process, a set of prototypes should be initialized for the purpose of characterizing the clusters. In conventional CL algorithms, either the prototype locations or their numbers may have a significant effect on the result. In particular, how to predetermine the appropriate number of prototypes remains largely unsolved or just ignored due to the difficulty. Let us assume that the number of prototypes is less than that of the natural clusters in a data set, e.g., three clusters $S = \{S_1, S_2, S_3\}$ and only a single prototype $\vec{P}_1$ for characterizing the clusters. For a pattern randomly taken from S, according to the nearest neighbor condition, $\vec{P}_1$ is the only winner since there is no other prototype competing with it. Consequently, $\vec{P}_1$ tries to move toward each pattern from S_1, S_2, or S_3, which results in the oscillation phenomenon shown in Fig. 1(a). In general, if the number of prototypes is less than that of the natural clusters, there must be at least one prototype that wins patterns from more than two clusters. We call this behavior one-prototype-take-multiclusters (OPTMC). The behavior of OPTMC is not desirable in data clustering since we expect each prototype characterizes only one natural cluster. One way to tackle this problem is that, as a first step, $\vec{P}_1$ is biased to one of the three clusters, either S_1, S_2, or S_3, and ignores the other two clusters. Then, further judgment and action may be carried out to explore the other clusters. We call this new learning paradigm OPTOC.

The ideas in OPTOC are in great contrast to that in OPTMC. The key technique used in OPTOC is that, for each prototype,



(a) (b)

Fig. 1. Two learning behaviors: OPTMC versus OPTOC. (a) One prototype is trying to take three clusters, resulting in oscillation phenomenon (OPTMC); (b) one prototype takes one cluster and ignores the other two clusters (OPTOC).

an online learning vector, asymptotic property vector (APV) is assigned to guide the learning of this prototype. With the "help" of the APV, each prototype will locate only one natural cluster and ignore other clusters in the case that the number of prototypes is less than that of clusters. Fig. 1(b) shows an example of learning based on the OPTOC paradigm. In this figure, $\vec{P}_1$ actually wins all the three clusters, but it finally settles at the centroid of S_2 and ignores the other two clusters S_1 and S_3.

The learning of a prototype is affected by its APV. For simplicity, $\vec{A}_i$ represents the APV for prototype $\vec{P}_i$ and $n_{\vec{A}_i}$ denotes the learning counter (or say, *winning counter* in accordance with the CL literature) of $\vec{A}_i$. As a necessary condition of OPTOC mechanism, $\vec{A}_i$ is required to initialize at a random location that is far from its associated prototype $\vec{P}_i$. The winning counter $n_{\vec{A}_i}$ is initially zero. Let $|\vec{\mu}\vec{\nu}|$ denote the Euclidean distance from $\vec{\mu}$ to $\vec{\nu}$, i.e., $||\vec{\mu} - \vec{\nu}||$, where $||\cdot||$ is the Euclidean norm. Let us construct a dynamic circle area that is defined as the neighborhood of prototype $\vec{P}_i$ with the radius $|\vec{P}_i\vec{A}_i|$. If the current input pattern $\vec{X}$ satisfies the condition $|\vec{P}_i\vec{X}| \leq |\vec{P}_i\vec{A}_i|$, we may say that this pattern is inside the neighborhood of $\vec{P}_i$. Patterns inside the neighborhood of $\vec{P}_i$ will contribute to the learning of $\vec{P}_i$ much more than those outside. $\vec{A}_i$ affects the learning of $\vec{P}_i$ by dynamically updating its neighborhood size which plays a key role in discriminating the input patterns.

Since $\vec{A}_i$ is initialized far from $\vec{P}_i$, at the very beginning of the learning process, the neighborhood of $\vec{P}_i$ is large enough to contain patterns from all the clusters. In other words, there is no discrimination among the patterns. All of them have the same significance to the learning of $\vec{P}_i$. Thus, in the worst case, if $\vec{A}_i$ were kept static at a distant location from $\vec{P}_i$ during the entire learning process, OPTOC would reduce to the OPTMC learning paradigm resulting in an oscillation phenomenon. In SSCL, to implement the OPTOC paradigm, $\vec{A}_i$ is updated online to construct a *dynamic* neighborhood of $\vec{P}_i$. Initially, the neighborhood is large; as time progresses, however, its size reduces to zero. The reduction of the neighborhood size guarantees the convergence of the prototype, since no more pattern will be eligible for its learning. To achieve this, we devise a process such that the patterns "outside" of the dynamic neighborhood will contribute less to the learning of $\vec{P}_i$ as compared to those "inside" patterns.

The update of $\vec{A}_i$ depends on the relative locations of the input pattern $\vec{X}$, prototype $\vec{P}_i$, and $\vec{A}_i$ itself. Each time when $\vec{X}$ is presented, the winning prototype is judged by the nearest neighbor criterion. Assume $\vec{P}_i$ is the winner, the learning of $\vec{A}_i$ can be given by

$$\vec{A}_i^* = \vec{A}_i + \frac{1}{n_{\vec{A}_i}} \cdot \delta_i \cdot \left(\vec{X} - \vec{A}_i\right) \cdot \Theta\left(\vec{P}_i, \vec{A}_i, \vec{X}\right) \quad (1)$$

where Θ is a general function given by

$$\Theta\left(\vec{\mu}, \vec{\nu}, \vec{\omega}\right) = \begin{cases} 1 & \text{if } |\vec{\mu}\vec{\nu}| \geq |\vec{\mu}\vec{\omega}| \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

and $0 < \delta_i \leq 1$; it can be either a constant or a varied fraction. In this paper, δ_i is defined as follows:

$$\delta_i = \left(\frac{\left|\vec{P}_i\vec{A}_i\right|}{\left|\vec{P}_i\vec{X}\right| + \left|\vec{P}_i\vec{A}_i\right|}\right)^2. \quad (3)$$

For each update of $\vec{A}_i$, its winning counter $n_{\vec{A}_i}$ is computed as follows:

$$n_{\vec{A}_i} = n_{\vec{A}_i} + \delta_i \cdot \Theta\left(\vec{P}_i, \vec{A}_i, \vec{X}\right). \quad (4)$$

With this learning scheme, we can see that the key point of updating $\vec{A}_i$ is to make sure that $\vec{A}_i$ always shifts toward the patterns located in the neighborhood of its associated prototype $\vec{P}_i$ and gives up those data points out of this area. In other words, $\vec{A}_i$ tries to move closer to $\vec{P}_i$ by recognizing those "inside" patterns that may help $\vec{A}_i$ to achieve its goal while ignoring those "outside" patterns that are of little benefit. Suppose that $\vec{P}_i$ is at a fixed location and $\vec{A}_i$, $\vec{A}_i^*(0)$, ..., $\vec{A}_i^*(n)$ are the sequential locations of $\vec{A}_i$ during the learning period. According to the learning rule described above, we have

$$\left|\vec{P}_i\vec{A}_i^*(n)\right| \leq \cdots \leq \left|\vec{P}_i\vec{A}_i^*(0)\right| \leq \left|\vec{P}_i\vec{A}_i\right|.$$

It thus may be observed that in the finite input space, the asymptotic property vector $\vec{A}_i$ always tries to move toward $\vec{P}_i$, i.e., $\vec{A}_i$ has the *asymptotic* property with respect to its associated prototype $\vec{P}_i$.

Since $\vec{P}_i$ is the winner, the input pattern $\vec{X}$ makes contribution to its update, meanwhile, $\vec{A}_i$ guides its learning as well. Thus, both $\vec{X}$ and $\vec{A}_i$ play a significant role on the adaptation of $\vec{P}_i$. An update schedule for $\vec{P}_i$ is given as below. Later we will show the learning process with a dynamic analysis

$$\vec{P}_i^* = \vec{P}_i + \alpha_i\left(\vec{X} - \vec{P}_i\right) \quad (5)$$

where

$$\alpha_i = \left(\frac{\left|\vec{P}_i\vec{A}_i\right|}{\left|\vec{P}_i\vec{X}\right| + \left|\vec{P}_i\vec{A}_i\right|}\right)^2 = \left(1 + \left|\vec{P}_i\vec{X}\right| \Big/ \left|\vec{P}_i\vec{A}_i\right|\right)^{-2} \quad (0 < \alpha_i \leq 1). \quad (6)$$

For the current input pattern $\vec{X}$, if $|\vec{P}_i\vec{X}| \gg |\vec{P}_i\vec{A}_i|$ (that is, $\vec{X}$ is far out of the neighborhood of $\vec{P}_i$), $\alpha_i \to 0$ according to (6). As a result, $\vec{X}$ will have little influence on the learning of $\vec{P}_i$; whereas if $|\vec{P}_i\vec{X}| \ll |\vec{P}_i\vec{A}_i|$, according to (1), $\vec{A}_i$ will be shifted toward $\vec{X}$ to some degree, as seen in (6), $\vec{P}_i$ will have a large learning rate, $\alpha_i \to 1$.

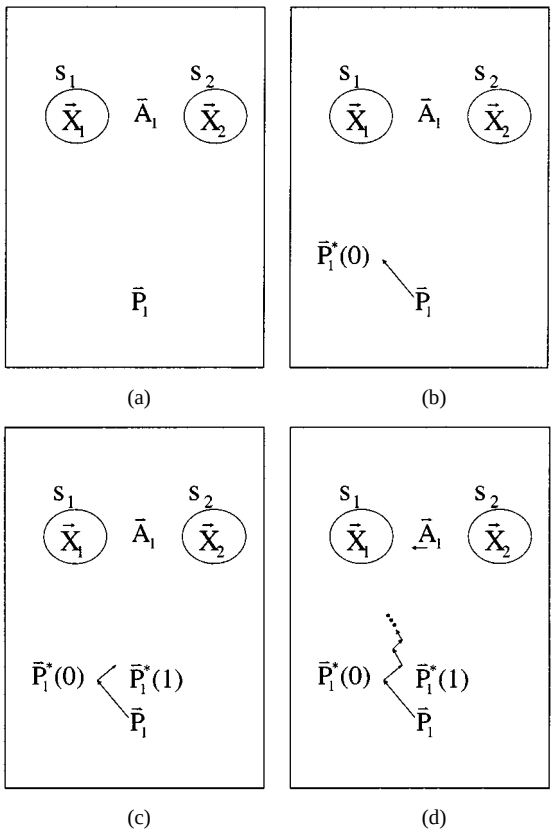


Fig. 2. (a) Initial locations of the single prototype $\vec{P}_1$ and its asymptotic property vector $\vec{A}_1$; (b) the locations of $\vec{P}_1$ and $\vec{A}_1$ after one pattern from S_1 was presented; (c) the locations of $\vec{P}_1$ and $\vec{A}_1$ after another pattern from S_2 was presented; (d) the trajectories of $\vec{P}_1$ and $\vec{A}_1$ during learning.

Given the update schedules above, we analyze how $\vec{A}_i$ and $\vec{P}_i$ can implement the OPTOC paradigm. Suppose there are two clusters $S = \{S_1, S_2\}$, $\vec{X}_1$ and $\vec{X}_2$ denote the randomly grabbed patterns from S_1 and S_2, respectively. Assume the worst-case scenario that $\vec{A}_1$ has moved into the center point between S_1 and S_2, and $\vec{P}_1$ is currently located on the vertical line of S and far from $\vec{A}_1$ (because in this case $\vec{X}_1$ and $\vec{X}_2$ will have the same effect on $\vec{P}_1$), as shown in Fig. 2(a). Our analysis below will show that both $\vec{P}_1$ and $\vec{A}_1$ will move into one of the two clusters step by step, and ignore the other cluster eventually.

We select $\vec{X}_1$ randomly from S_1 and S_2. Since $|\vec{P}_1\vec{X}_1| > |\vec{P}_1\vec{A}_1|$, according to (1), $\vec{A}_1$ remains static, and $\vec{P}_1$ moves toward $\vec{X}_1$ with the learning rate α_1 computed from (6), the bigger the distance $|\vec{P}_1\vec{A}_1|$ between $\vec{P}_1$ and $\vec{A}_1$, the bigger the learning rate α_1. Let $\vec{P}_1^*(0)$ denote the new location of $\vec{P}_1$ after being trained by $\vec{X}_1$, as shown in Fig. 2(b). It is reasonable to assume that the training sample is taken from S_1 and S_2 with equal probability. After $\vec{P}_1$ has been updated, now assume $\vec{X}_2$ is taken from S_2. Since $|\vec{P}_1^*(0)\vec{X}_2| > |\vec{P}_1^*(0)\vec{A}_1|$, $\vec{A}_1$ will remain static, the learning rate for $\vec{P}_1^*$ becomes

$$\alpha_1^* = \left(1 + \left|\vec{P}_1^*(0)\vec{X}_2\right| \Big/ \left|\vec{P}_1^*(0)\vec{A}_1\right|\right)^{-2}. \tag{7}$$

In this case we may reasonably assume $|\vec{P}_1\vec{X}_1| \approx |\vec{P}_1^*(0)\vec{X}_2|$, because we have initialized $\vec{P}_1$ far from $\vec{A}_1$ so that $|\vec{P}_1\vec{X}_1|$ is big enough. As at the first step $\vec{P}_1$ won a big learning step toward S_1 [see Fig. 2(b)], $|\vec{P}_1^*(0)\vec{A}_1| < |\vec{P}_1\vec{A}_1|$; as a result, $\alpha_1^* < \alpha_1$, thus $\vec{P}_1^*(0)$ moves toward $\vec{X}_2$ with a smaller learning rate, as shown in Fig. 2(c). Since $\vec{P}_1$ has been biased toward cluster S_1, in the following learning steps, the same analysis applies. $\vec{P}_1$ aggravates this tendency: first it moves closer to cluster S_1; when it is closer to S_1 than to $\vec{A}_1$, patterns in S_1 take effect on the learning of $\vec{A}_1$, as a result, it moves toward cluster S_1 as well. This is illustrated in Fig. 2(d).

After both $\vec{P}_1$ and $\vec{A}_1$ have moved into cluster S_1, data from S_2 will have little effect on the learning of $\vec{P}_1$ since $|\vec{P}_1\vec{X}_2|/|\vec{P}_1\vec{A}_1| \to \infty$; that is, the learning rate caused by patterns from S_2 is very close to zero. Therefore, we may assume that only cluster S_1 affects $\vec{P}_1$ after a number of iterations. As a result, $\vec{P}_1$ moves to the center of S_1. Alternatively, we can say that $\vec{P}_1$ takes only one cluster and gives up the other cluster.

Fig. 3(a) shows a real example using the OPTOC paradigm. In Fig. 3(a), prototype $\vec{P}_1$ and its APV $\vec{A}_1$ are randomly ini-

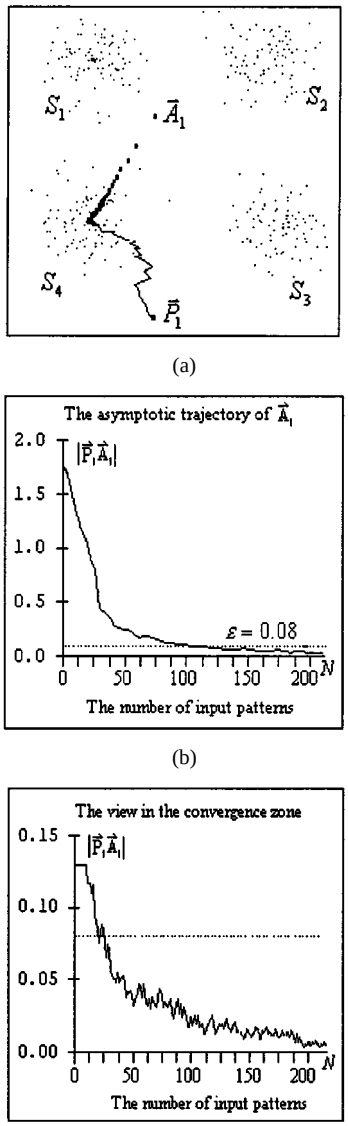


Fig. 3. The example of One-prototype-take-one-cluster: (a) the learning trajectories of $\vec{P}_1$ (solid curve) and $\vec{A}_1$ (dotted curve); (b) the asymptotic property of $\vec{A}_1$ with respect to its associated prototype $\vec{P}_1$ at a larger scale; (c) close-up view around the convergence zone in (b).

tialized. As required, $\vec{A}_1$ is initially far away from $\vec{P}_1$. During the learning process, the trajectory of $\vec{A}_1$ is plotted as a dotted curve, whereas the trajectory of prototype $\vec{P}_1$ is plotted as a solid curve. Fig. 3(b) shows that $\vec{A}_1$ has the asymptotic property with respect to $\vec{P}_1$ at a larger scale. When $|\vec{P}_1\vec{A}_1|$ is less than 0.08 in this example, we say $\vec{A}_1$ has been converged to $\vec{P}_1$. Fig. 3(c) shows a close-up view around the convergence zone in Fig. 3(b).

Thus, upon each input pattern, the update schedule (5) for prototypes and the update schedule (1) for their corresponding APVs guarantee that a prototype takes only one cluster and ignores other clusters.

B. Split Validity Criterion

So far we have presented the learning schemes for implementing the OPTOC paradigm. OPTOC has proved to be successful in that it enables each prototype to find only one natural cluster when the number of clusters is greater than that of the prototypes. This in itself is a major improvement over most other clustering algorithms in the literature [47]. However, at this stage we are still unable to classify all the data into proper natural groups not just to find some groups of the data. What we need to do next is how to find those clusters that have failed in competing for a prototype and have been ignored by the OPTOC learning process.

In this section, we introduce a split validity criterion to judge if all the clusters have been classified. If not yet, one prototype will be chosen to split into two prototypes to expand the prototype set. Then in the next OPTOC learning iteration, one more cluster will be partitioned by the *spawned* prototype. This split process will be repeated until no prototype satisfies the split validity criterion. Starting from one prototype, the prototype set expands to 2, 3, 4 … prototypes and finally terminates when each cluster has been labeled by a prototype. Considering the learning process in Fig. 1(b), because after a certain number of learning iterations, clusters S_2 and S_3 will have little influence on the learning of $\vec{P}_1$, we can reasonably assume that only cluster S_1 influences $\vec{P}_1$. Consequently, $\vec{P}_1$ will satisfy the centroid condition

$$\vec{P}_1 = \frac{1}{N_1}\sum_{\vec{X}\in S_1}\vec{X}; \qquad N_1 = \sum_{\vec{X}\in S_1} 1. \tag{8}$$

In most traditional clustering algorithms, for example in Fig. 1(a), as $\vec{P}_1$ wins all the clusters S_1, S_2, and S_3, $\vec{P}_1$ should move to the centroid of S_1, S_2, and S_3. In contrast, the OPTOC learning scheme forces $\vec{P}_1$ to move into cluster S_2, and ignores the other two clusters, S_1 and S_3 [see Fig. 1(b)]. To make the analysis simpler, let us consider that each prototype has a *body* and a *heart.* We may say that in Fig. 1(b) S_2 holds the *body* of $\vec{P}_1$, but not its *heart.* The *heart* of $\vec{P}_1$ stays at the centroid of S_1, S_2, and S_3 since it wins all the patterns from the three clusters. Only when the *body* and the *heart* of a prototype coincide in position, can we say that this prototype represents a cluster "sincerely," otherwise, "reluctantly." Reluctance hints that there must be other clusters attracting this prototype; however, due to the OPTOC pressure, they fail in contending for this prototype. Let us define the center property vector (CPV) $\vec{C}_i$ for prototype $\vec{P}_i$ which indicates the centroid of all the patterns for which $\vec{P}_i$ has been the winner. Then in Fig. 1(b) we have

$$\vec{C}_1 = \frac{1}{\hat{N}_1}\sum_{\vec{X}\in S_1\cup S_2\cup S_3}\vec{X}; \qquad \hat{N}_1 = \sum_{\vec{X}\in S_1\cup S_2\cup S_3} 1. \tag{9}$$

It is obvious that there is an apparent bias $|\vec{P}_1\vec{C}_1|$ between $\vec{C}_1$ ("heart" in our sense) and the prototype $\vec{P}_1$ in (8) ("body" so to speak). This bias indicates that there exist extra clusters in the input space that try to pull the cluster center from $\vec{P}_1$ to $\vec{C}_1$. If there were only one cluster, $|\vec{P}_1\vec{C}_1|$ should theoretically be 0 according to (8) and (9).

In SSCL, the CPVs are updated by the k-Means learning scheme [37], which is used to calculate the exact arithmetic mean of the input data points for which a prototype has been the winner so far. Let $n_{\vec{C}_i}$ denote the winning counter of $\vec{C}_i$. $\vec{C}_i$ is updated as follows:

$$\vec{C}_i^* = \vec{C}_i + \frac{1}{n_{\vec{C}_i}}\left(\vec{X} - \vec{C}_i\right). \tag{10}$$

Every time when $\vec{P}_i$ converges into a cluster (it is judged by $|\vec{P}_i\vec{A}_i|$), we check the distortion between $\vec{P}_i$ and $\vec{C}_i$. If it is larger than a predefined threshold, there are other clusters that fail in competing for this prototype and need to be labeled in subsequent learning iterations; this prototype is then suitable for splitting to expand the prototype set. For each split validity measure, the suitable prototype with the most distortion $|\vec{P}_i\vec{C}_i|$ will be chosen for splitting.

$$\left|\vec{P}_i\vec{A}_i\right| < \varepsilon_1 \bigcap \left|\vec{P}_i\vec{C}_i\right| > \varepsilon_2 \Longrightarrow \vec{P}_i \text{ is suitable for splitting.} \tag{11}$$

where ε_i is a small positive value. In practice, when $|\vec{P}_i\vec{A}_i|$ is small enough, we may say that $\vec{P}_i$ has converged into a cluster. To make it simpler, ε_1 and ε_2 can be the same value ε. This gives an alternative splitting criterion

$$\left|\vec{P}_i\vec{A}_i\right| < \varepsilon \bigcap \left|\vec{P}_i\vec{C}_i\right| > \varepsilon \Longrightarrow \vec{P}_i \text{ is suitable for splitting.} \tag{12}$$

The bigger ε is, the worse the accuracy of clustering, because some adjacent clusters will be merged as one cluster. Since, usually, no information about the bounding distance is available, it must be determined adaptively from the analysis of the feature space, e.g., ε may be defined as the average variance for Gaussian distributed clusters. Assume there are n such clusters, v_i denotes the variance of ith cluster and N_i denotes the number of data points in ith cluster, we may define ε as

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{n} v_i N_i; \qquad N = \sum_{i=1}^{n} N_i. \tag{13}$$

For those non-Gaussian distributed clusters, e.g., in curve detection, ε can be set as two percent of the maximum scale of the coordinates in the input feature space. For instance, if the feature space is M-dimensional, ε can be defined as,

$$\varepsilon = \tfrac{1}{50}\left(Max\{ScaleC_1, ScaleC_2, \ldots, ScaleC_M\}\right) \tag{14}$$

where $ScaleC_i$ denotes the scale of the ith coordinate in the M-dimensional space.

Since $\vec{P}_i$ and $\vec{A}_i$ are always able to converge to a cluster, for simplicity, we may judge $|\vec{P}_i\vec{C}_i|$ at every N learning iterations. N is appropriately large enough so that the APVs have enough time to converge to their associated prototypes.

The definition of ε is not absolute. We will address its significant influence on the split quality in the experimental sections. We suggest that ε be determined adaptively from an analysis of the feature space.

C. Distant Property Vector for Splitting

When one prototype $\vec{P}_i$ is judged to satisfy the split validity criterion after a number of iterations, it splits into two prototypes: one stays at its current location, the other is initialized at a distant location. It is not mandatory to arrange the new one far from its "mother;" however, putting it at a distant location can help it avoid the unnecessary competition against its mother prototype in the new learning period and make the learning process converge more quickly and efficiently. For each prototype, we assign a distant property vector (DPV) at which $\vec{P}_i$ splits if it satisfies the split validity criterion. Each distant property vector is designed to update with respect to its associated prototype during the learning process.

We use $\vec{R}_i$ to represent the DPV for $\vec{P}_i$. It is initialized at the same location as $\vec{P}_i$. Let $n_{\vec{R}_i}$ denote the learning counter for $\vec{R}_i$, which is initialized to zero. During the learning process, $\vec{R}_i$ will be updated to a distant location from $\vec{P}_i$. It is better to determine the update schedule of $\vec{R}_i$ adaptively from the analysis of the feature space to improve the efficiency of splitting. Contrary to the APV $\vec{A}_i$, the DPV $\vec{R}_i$ always tries to move away from $\vec{P}_i$. In classifying Gaussian distributed clusters, when a pattern $\vec{X}$ is presented, one update schedule for $\vec{R}_i$ can be given by

$$\vec{R}_i^* = \vec{R}_i + \frac{1}{n_{\vec{R}_i}} \cdot \rho_i \cdot \left(\vec{X} - \vec{R}_i\right) \cdot \Theta\left(\vec{P}_i, \vec{X}, \vec{R}_i\right) \quad (15)$$

where ρ_i is the learning rate for $\vec{R}_i$, it can be given by

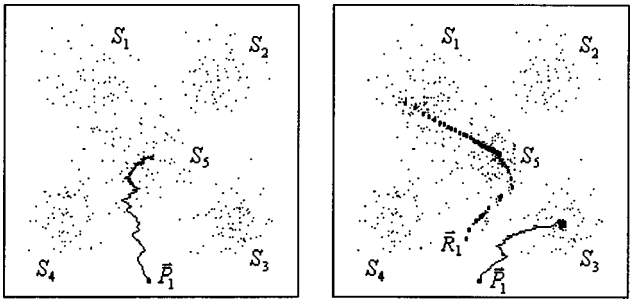
$$\rho_i = \left(\frac{\left|\vec{P}_i\vec{X}\right|}{\left|\vec{P}_i\vec{X}\right| + \left|\vec{P}_i\vec{R}_i\right|}\right)^2. \quad (16)$$

Alternatively, ρ_i can be set as a constant value $0 < \rho_i \leq 1$. $n_{\vec{R}_i}$ will increase ρ_i if $\vec{R}_i$ is updated.

In some cases, e.g., line detection, a simple update schedule for $\vec{R}_i$ would be efficient enough: $\vec{R}_i$ always follows the farthest pattern for which its associated prototype has been the winner so far

$$\vec{R}_i^* = \vec{R}_i \cdot \left(1 - \Theta\left(\vec{P}_i, \vec{X}, \vec{R}_i\right)\right) + \vec{X} \cdot \Theta\left(\vec{P}_i, \vec{X}, \vec{R}_i\right). \quad (17)$$

In some (albeit rare) cases, e.g., Fig. 4(a), the prototype $\vec{P}_i$ could move into a cluster whose centroid is coincided with the global centroid; consequently the prototype will not satisfy the split validity criterion. Since no prototype is suitable for splitting, the algorithm then announces that only one cluster is found finally. This is undesirable as it fails to find the other four clusters. There are two ways to tackle this problem. One is that, when no prototype is suitable for splitting, a further test is used to check whether there are other separate clusters. This idea is not practical since we have no prior knowledge of the cluster distribution and additional test will make the split criterion more complex. Another method is to introduce a bias factor into the learning rate of the prototype that causes the prototypes to give priority to those clusters that are far from the global center so that it may prevent the case shown in Fig. 4(a) from happening. When a pattern is presented, a bias influence for the winning prototype is defined as follows:



(a) (b)

Fig. 4. (a) $\vec{P}_1$ can be trapped at the global centroid; (b) the bias factor enables $\vec{P}_1$ to move away from the global centroid. $\vec{R}_1$ tries to move far away from $\vec{P}_1$ during the learning process; when a split happens, the new prototype will be spawned at the location indicated by $\vec{R}_i$.

$$\beta_i = \left(\frac{\left|\vec{X}\vec{R}_i\right|}{\left|\vec{P}_i\vec{X}\right| + \left|\vec{P}_i\vec{R}_i\right|}\right)^2 \cdot \left(\frac{\left|\vec{P}_i\vec{R}_i\right|}{\left|\vec{X}\vec{R}_i\right|}\right)^{2\cdot\exp(-n_{\vec{P}_i}\cdot(|\vec{P}_i\vec{X}|^2/V^2))} \quad (18)$$

where $0 < \beta_i \leq 1$, $n_{\vec{P}_i}$ is the winning counter of $\vec{P}_i$. V denotes the radius of the finite input space and remains constant during the whole learning process.

As time progresses, $n_{\vec{P}_i}$ gradually increases and the exponential item decreases to zero, thus, the first term in (18) will play a major role. Let us suppose that at a moment an input $\vec{X}$ close to $\vec{R}_i$ is presented, since $\vec{R}_i$ always moves away from $\vec{P}_i$ during the learning process, we have $|\vec{P}_i\vec{X}| \gg |\vec{X}\vec{R}_i|$. As a result, β_i is very small so that $\vec{X}$ has little influence on the learning rate of $\vec{P}_i$. In other words, clusters closer to $\vec{P}_i$ will attract $\vec{P}_i$ more than those closer to $\vec{R}_i$, which results in a bias effect. As shown in Fig. 4(b), $\vec{P}_1$ moves to S_3 while its DPV R_1 moves far from it. The new prototype will be spawned at the location indicated by $\vec{R}_1$.

When $\vec{P}_i$ moves into a cluster, $|\vec{P}_i\vec{X}|^2/V^2$ in (18) decreases sharply for those patterns from the occupied cluster due to the small value of $|\vec{P}_i\vec{X}|^2$, which means that the second term in (18) recovers its influence on β_i. The bias effect is reduced by the denominator of the second term. This indicates that the bias factor has important influence on the global learning but has little influence on the local learning. In most cases, we can simplify (18) by

$$\beta_i = \left(\frac{\left|\vec{P}_i\vec{R}_i\right|}{\left|\vec{P}_i\vec{X}\right| + \left|\vec{P}_i\vec{R}_i\right|}\right)^2. \quad (19)$$

Initialization :

Set the number of clusters $K = 1$;
Initialize $\vec{P}_1(\vec{A}_1, \vec{C}_1, \vec{R}_1)$ as below,

Set $\vec{P}_1 = \vec{R}_1$ at a random location in the input feature space;
Set $\vec{A}_1 = \vec{C}_1$ at a random location far from $\vec{P}_1$;
Set all their winning counters as 0;

Set ε as a small positive value using (13) or (14);
Initialize *Iteration* to 0;
Make all input patterns active(unpresented);

Learning Loop :

Step 1: Repeat

substep 1: Randomly take an active pattern $\vec{X}$ from the data set, set it inactive;
If no active pattern, increment *Iteration* and set all patterns active;

substep 2: Competition, find the winner prototype $\vec{P}_i$,
$|\vec{P}_i\vec{X}| = \min_l |\vec{P}_l\vec{X}|$ for $l = 1, \ldots, K$;
Label $\vec{X}$ with i;

substep 3: Update the *Asymptotic Property Vector* $\vec{A}_i$ using (1);

substep 4: Update the *Center Property Vector* $\vec{C}_i$ using (10);

substep 5: Update the *Distant Property Vector* $\vec{R}_i$ using (15) or (17);

substep 6: Update the prototype $\vec{P}_i$ using (20);

Until $|\vec{P}_l\vec{A}_l| < \varepsilon$ for all $l = 1, \ldots, K$ OR
iteration reaches a predefined number;

Step 2: Split
$|\vec{P}_j\vec{C}_j| = \max_l |\vec{P}_l\vec{C}_l|$ for $l = 1, \ldots, K$;
if $|\vec{P}_j\vec{C}_j| > \varepsilon$ then $\vec{P}_j$ splits:
Increment K;
Set $\vec{P}_K = \vec{R}_j$;
Assign $(\vec{A}_K, \vec{C}_K, \vec{R}_K)$ for $\vec{P}_K$;
goto Step 3;
else goto Step 4;

Step 3: Reset
For all $l = 1, \ldots, K$ do follows,
set $\vec{R}_l = \vec{P}_l$;
set $\vec{A}_l = \vec{C}_l$ at a location far from $\vec{P}_l$;
set all the winning counters as 0;
Make all patterns active;
Initialize *Iteration* to 0;
Goto Step 1;

Step 4: Find K clusters, quit Learning Loop.

Fig. 5. The SSCL algorithm.

D. Self-Splitting Competitive Learning

Finally, we use $\vec{P}_i(\vec{A}_i, \vec{C}_i, \vec{R}_i)$ to denote the ith prototype in terms of its three property vectors. Upon the presentation of $\vec{X}$, if $\vec{P}_i$ is the winner, it is updated in the general form as following:

$$\vec{P}_i^* = \vec{P}_i + \alpha_i \cdot \beta_i \cdot \left(\vec{X} - \vec{P}_i\right) \tag{20}$$

where α_i is computed with (6) and β_i with (18) or (19). Fig. 5 shows the pseudocode for the SSCL algorithm.

The SSCL represents a new competitive learning paradigm in that one prototype takes one cluster. After the first split happens, the prototypes will start to compete for updating when a pattern is presented. Each time when a prototype splits into two prototypes, one stays at the same location as the mother prototype, the other is spawned at a distant location which is indicated by the distant property vector of its mother prototype. Then all the prototypes available will reset their winning counters and start to compete according to the nearest neighbor condition. It has no dependency on the initial locations of prototypes. Moreover, it may find the right number of natural clusters via the adaptive splitting processes. Thus, it provides one effective alternative algorithm in the CL literature.

III. Clustering Analysis

The goal of clustering analysis (unsupervised classification) is to label every data point in the same class by the same symbol

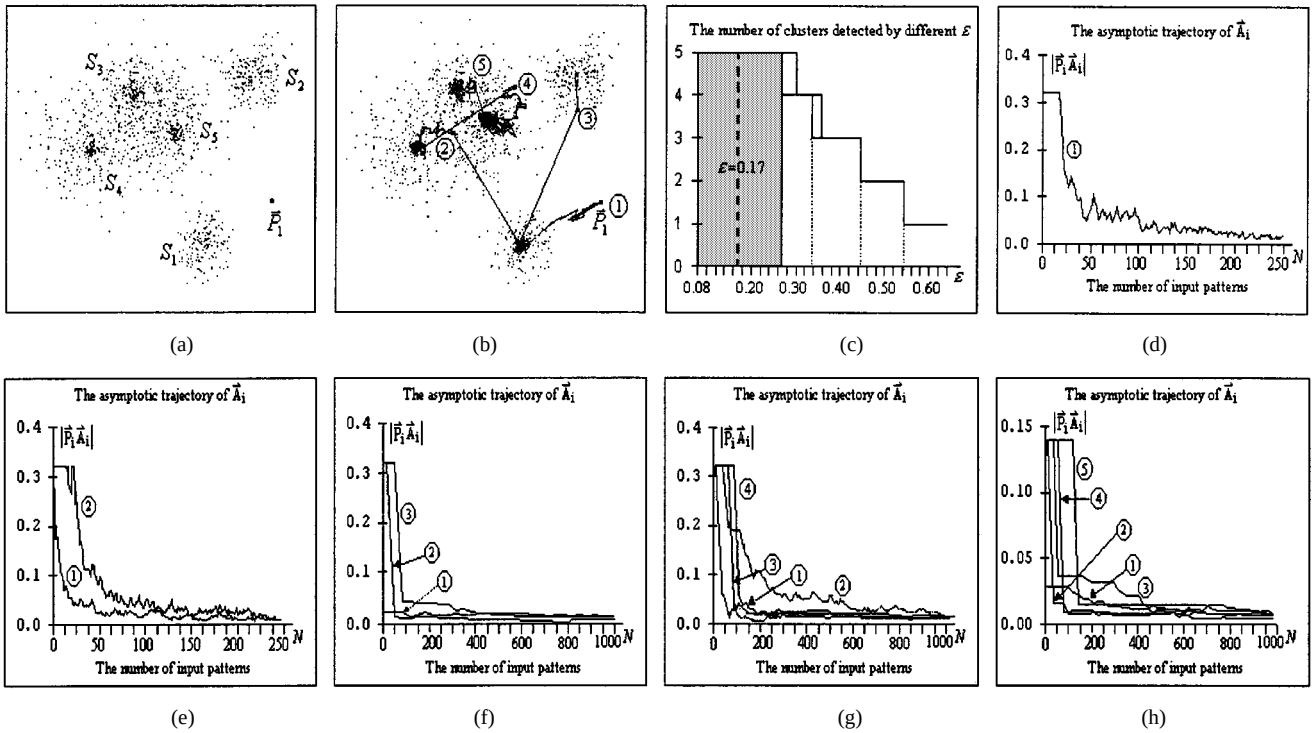
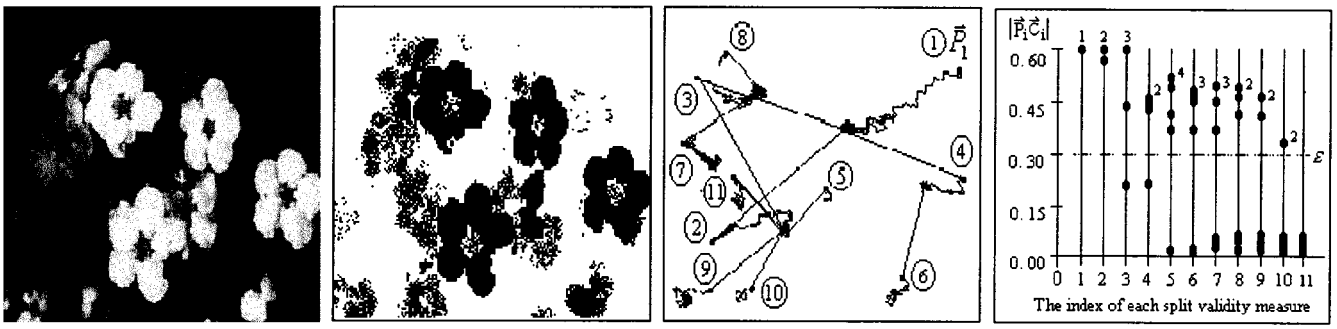


Fig. 6. (a) The original five clusters and the single prototype; (b) the learning trajectory and splitting processes by SSCL; (c) the influence of the split constant ϵ on the final number of clusters detected by SSCL; (d)–(h) the asymptotic property of each APV with respect to its associated prototype.

or the same color such that the data set is divided into several clusters each associated with a different symbol or color. The task is more difficult than supervised classification because we have little prior knowledge of the number of classes contained in the data set. Since SSCL does not need to know how many clusters in the input data set *a priori*, it was used for unsupervised classification in this section. To show the splitting details, we draw a line to connect the *mother* prototype and its newly *spawned* prototype. The initial location of each prototype is indicated by its order. SSCL terminates if no prototype satisfies the split validity measure.

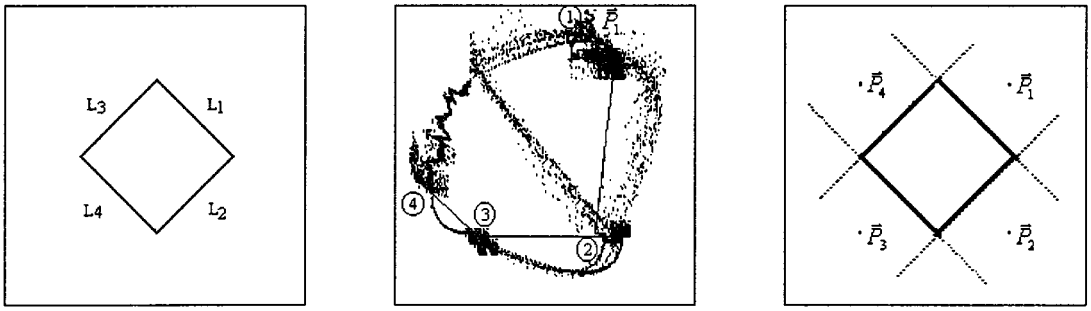
In Fig. 6(a), there are five clusters, S_1, S_2, ..., S_5, in the data set. Among them S_3, S_4, and S_5 are overlapped to some degree. The numbers of sample points for these five clusters are 150, 200, 300, 250, and 250, respectively; and their corresponding Gaussian variances are 0.10, 0.12, 0.15, 0.18, and 0.20. The split criterion constant ϵ was 0.17 according to (13). Only one prototype, $\vec{P}_1$, was randomly initialized, as shown in Fig. 6(a). Fig. 6(b) demonstrates the splitting processes and learning trajectories obtained by SSCL. As we can see, the splitting occurred four times. Therefore, five clusters were discovered finally; each cluster was associated with a prototype located at its center. According to the nearest neighbor condition, each sample point was labeled by its nearest prototype. Fig. 6(c) shows that the split constant ϵ has a significant influence on the learning performance. For $0.08 \leq \epsilon \leq 0.27$, SSCL performed consistently on the number of clusters finally detected, i.e., five clusters, as shown in the gray area in Fig. 6(c). SSCL is robust since it performed well in a large range of $\epsilon \in [0.08, 0.27]$. When $0.27 < \epsilon < 0.30$ it got into a nonsteady decision zone in which SSCL detected either five or four clusters in different processes. In general, the larger ϵ is, the worse the accuracy of the clustering. For $0.36 < \epsilon < 0.45$, the three overlapped clusters, S_3, S_4, and S_5 were merged into one cluster, therefore, together with S_1 and S_2, three clusters were obtained by SSCL. When ϵ was large enough, e.g., $\epsilon > 0.54$, no prototype was suitable for splitting: SSCL treated all the clusters as one cluster. Fig. 6(d)–(h) show the asymptotic property of $\vec{A}_i$ for each stage with $\epsilon = 0.17$. At the first stage, just $\vec{P}_1$ was initialized in the feature space, thus, only one curve shows the asymptotic property of $\vec{A}_1$. While at the last stage, there were five prototypes after splitting four times. Thus, we show five curves to demonstrate the asymptotic properties.

Fig. 7 shows a gray-scale image of flowers. It contains grass, leaves, and flowers. The task is to identify the locations of the flowers in the image, not concerned whether they are part of a certain plant or not. We model this task in terms of clustering analysis. Since the flowers are brighter than their surroundings, they may easily be "filtered" out by a thresholding process first. Then, the clustering analysis labels each flower as a cluster. As we do not know how many flowers in each image input for analysis, SSCL is a good choice for this task. Fig. 7(a) shows the original digital image containing several (about 11) flowers. In Fig. 7(b) the flowers were obtained after thresholding. Fig. 7(c) demonstrates the learning trajectories and splitting processes with $\epsilon = 0.30$ by SSCL. As can be seen, in this figure, the splitting process occurred ten times, and finally eleven flowers were labeled; each one was associated with a prototype. SSCL was able to accurately identify the locations of the flowers in the image. Fig. 7(d) gives the statistical data of the distortion between each prototype and its center property vector each time the split validity was measured. On the first split validity measure, there was only one prototype with a distortion value of 0.6. Since this distortion was greater than the split constant ϵ,



(a) (b) (c) (d)

Fig. 7. (a) The original digital image containing several flowers; (b) the corresponding clusters after the thresholding process; (c) the Learning trajectories and splitting processes by SSCL; finally 11 clusters have been discovered; (d) the distortion between each prototype and its CPV at each split judgment; the prototype with the highest distortion was chosen for splitting. The values along the horizontal axis indicate the 11 rounds of validity measure. The number next to the black dots at the top indicate the prototype that have the most distortion during their corresponding validity measure rounds.



(a) (b) (c)

Fig. 8. Line detection: (a) the diamond image consists of four line segments; (b) the learning trajectories, splitting processes and the feature points generated during SSCL learning process; (c) four lines have been identified by four prototypes; each prototype can be viewed as a parametric vector for a specific line.

this prototype was chosen for the first splitting. On the second round, two prototypes were available for evaluating the split validity measure. Although both prototypes were above the decision line, only the one with more distortion, i.e., the second prototype [the number "2" next to the top black dot in Fig. 7(d)] in this experiment, was selected for splitting. At the 11th round for split measure, there was no prototype with a distortion value greater than ϵ. Therefore, no prototype was suitable for splitting and SSCL terminated.

IV. Curve Detection

Detecting curves (line, circle, ellipse, etc.) in images is an important task in image processing and machine vision. Assume there is a binary image, we use $F(\vec{a}, \vec{X}) = 0$ as the parametric equation of a curve with a vector of parameters $\vec{a} = [a_1, \ldots, a_m]$ and $\vec{X} = [x_1, x_2]$ the coordinates of a pixel in the image. If the image contains a number of groups of pixels (e.g., a number of line segments or a number of circles) with each group lying in its own curve expressible by the parametric equation $F(\vec{a}, \vec{X}) = 0$, with the parameter vector $\vec{a}$ differing from curve to curve. We need to classify every pixel into one of the groups. If we have k groups of pixels, there must be k distinct parametric vectors $\vec{a}_1, \ldots, \vec{a}_k$ to characterize them. Each pixel can be mapped into one of these k parametric vectors according to the rule

$$\vec{X} \mapsto \vec{a}_j \qquad \text{if } F\left(\vec{a}_j, \vec{X}\right) = 0, \qquad j \in \{1, \ldots, k\}. \quad (21)$$

All the pixels mapped into the same parametric vector constitute a curve; each parametric vector stands for one curve in the image. The feature space in this section refers to the parametric vector space.

Our basic idea is to think each parametric vector as a prototype in the feature space and transform each sample pixel into the feature space. In this feature space, at the beginning, we set only one prototype as the parametric vector, then we apply SSCL to perform the clustering task. After it is finished, the number of prototypes should be the same as that of curves and each prototype can selectively become the parametric vector for the pixels drawn from a specific curve with the least distortion. Let $\vec{Z}$ denote the data point in the feature space associated with input pixel $\vec{X}$ and parametric vector $\vec{a}$. First, we define the error function between the parametric vector $\vec{a}$ and real input pixel $\vec{X}$ as

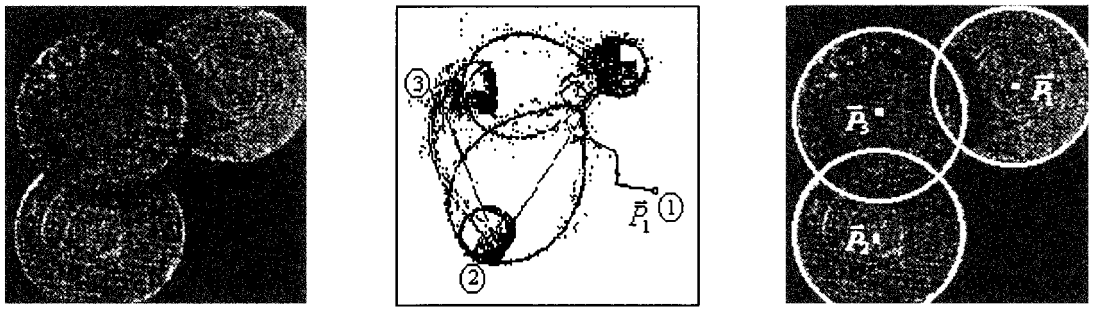
$$\mu = F^2\left(\vec{a}, \vec{X}\right). \quad (22)$$

We define the error function in the feature space as

$$\mu = |\vec{a}\vec{Z}|^2. \quad (23)$$

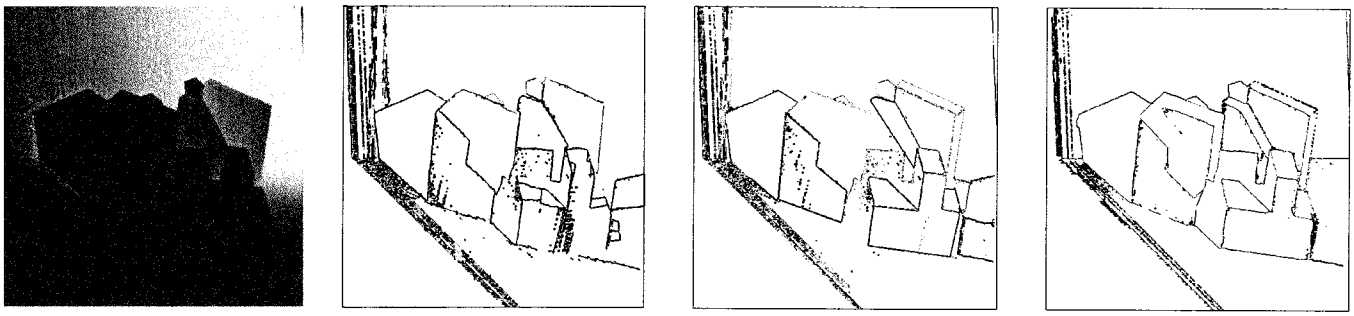
Then the transformation from $\vec{X}$ to $\vec{Z}$ can be carried out from the following:

$$\frac{\partial \mu}{\partial \vec{a}} = \frac{\partial \left|\vec{a}\vec{Z}\right|^2}{\partial \vec{a}} = \frac{\partial F^2\left(\vec{a}, \vec{X}\right)}{\partial \vec{a}}. \quad (24)$$



(a) (b) (c)

Fig. 9. Circle detection: (a) an image of three overlapped discuses; (b)the Learning trajectories, splitting processes and the feature points generated during SSCL learning process; (c) the estimated prototypes representing the centers of the circles and their corresponding circles for the discs in the image.



(a) (b) (c) (d)

Fig. 10. Segmentation of an ABW training image: (a) the range image; (b) segmented image by $\epsilon = 12$; 15 clusters in the feature space have been detected; (c) segmented image by $\epsilon = 8$; 17 clusters in the feature space have been detected; (d) segmented image by $\epsilon = 4$; 20 clusters in the feature space have been detected.

TABLE I
RESULTS OF SEGMENTATIONS WITH DIFFERENT ϵ

	$\epsilon = 12$		$\epsilon = 8$		$\epsilon = 4$	
	$(\beta_0, \beta_1, \beta_2)$	Variance	$(\beta_0, \beta_1, \beta_2)$	Variance	$(\beta_0, \beta_1, \beta_2)$	Variance
Cluster1	(973.1, 0.00, 0.024)	8.3	(976.5, 0.12, -0.09)	8.12	(976.4, 0.10)	7.20
Cluster2	(908.4, -0.01, -0.01)	20.4	(908.4, -0.01, -0.01)	20.03	(908.2, 0.00, 0.00)	4.12
Cluster3	(750.0, 1.21, -0.76)	43.2	(750.0, 1.21, -0.76)	39.05	(750.0, 1.21, -0.76)	5.15
Cluster4	(1227, 2.16, 0.093)	8.27	(1227, 2.16, 0.093)	8.24	(1227, 2.16, 0.093)	23.01
Cluster5	(1096, 1.10, -0.02)	9.01	(1096, 1.10, -0.02)	9.07	(1096, 1.10, -0.02)	4.90
Cluster6	(1056, 0.85, 0.084)	13.67	(1056, 0.85, 0.084)	17.5	(1056, 0.85, 0.084)	11.02
Cluster7	(876.5, -0.27, -0.34)	3.81	(876.8, -0.07, -0.43)	3.4	(876.3,-0.31, -0.32)	3.44
Cluster8	(1209, 2.07, 0.111)	6.22	(1209, 2.07, 0.111)	6.22	(1209, 2.07, 0.111)	5.23
Cluster9	(1155, 1.77, 0.163)	6.24	(1155, 1.77, 0.163)	6.24	(1155, 1.77, 0.163)	5.45
Cluster10	(1020, 0.50, 0.121)	12.20	(1020, 0.50, 0.119)	11.85	(1020, 0.50, 0.119)	4.32
Cluster11	(802, 1.19, -0.71)	6.12	(802.5, 1.19, -0.71)	3.24	(802.5, 1.19, -0.71)	2.10
Cluster12	(832, 0.31, -0.54)	5.38	(832.0, 0.30, -0.54)	5.32	(832.1, 0.31, -0.54)	6.5
Cluster13	(1183, 1.95, 0.112)	5.75	(1183, 1.95, 0.112)	5.6	(1183, 1.95, 0.112)	2.09
Cluster14	(1130, 1.49, 0.161)	12.80	(1130, 1.49, 0.161)	9.35	(1130, 1.49, 0.161)	2.97
Cluster15	(983.6, -0.53, -0.40)	11.08	(979.3, 0.00, 0.00)	4.02	(979.3, 0.00, 0.00)	4.00
Cluster16	N/A	N/A	(964.6, 0.01, 0.004)	3.24	(964.7, 0.00, 0.016)	2.82
Cluster17	N/A	N/A	(1040, 0.132, 0.71)	7.43	(1040, 0.71, 0.132)	4.36
Cluster18	N/A	N/A	N/A	N/A	(923.0, 0.44, 2.439)	6.21
Cluster19	N/A	N/A	N/A	N/A	(937.2, 1.13, 1.875)	1.77
Cluster20	N/A	N/A	N/A	N/A	(947.9, -0.03, 0.057)	1.03

Thus, we transform each input sample pixel, $\vec{X} = (x_1, x_2)$, into $\vec{Z} = (z_1, z_2)$ (which can be treated as a "*virtual*" data point in the feature space) according to (24). Hence, the learning process is performed in the feature space with $\vec{Z}$ as the input stimulus.

As shown in Fig. 8(a), our data consists of the "on" pixels in an image containing four line segments which form a diamond. The mathematical equations of the four lines are, L_1: $x_1 + x_2 = 1$, L_2: $x_1 - x_2 = 1$, L_3: $-x_1 - x_2 = 1$, and L_4: $-x_1 + x_2 = 1$. Each line segment in the actual digital binary image consists of 100 pixels. The problem is to detect the parametric vectors (1, 1), (1, -1), (-1, -1) and (-1, 1) so that each line can be labeled according to the mapping rule defined in (21). For simplicity, we applied (17) as the update scheme for

DPV's. Fig. 8(b) shows the SSCL learning trajectories, splitting processes and the transformed feature points. Fig. 8(c) shows that the four lines have been detected accurately, each one is identified by a prototype as its parametric vector.

Fig. 9(a) is another image of 200×200 pixels. There are three overlapping discs. The image has significant noise and the disc boundaries are not well defined. In the two-dimensional space, the circle equation can be written in the mathematical form as $F(\vec{a}, \vec{X}) = (x_1 - a_1)^2 + (x_2 - a_2)^2 - r^2 = 0$. Therefore, the parametric vector consists of three parameters (a_1, a_2, r), where (a_1, a_2) represents the circle center and r is the radius of the circle. According to (24), for each input pixel, we may compute its corresponding feature point which is actually used for the real input stimulus. In Fig. 9(b), we may observe the learning trajectories, splitting processes, and feature points in the two-dimensional space. Fig. 9(c) shows the final parametric vectors obtained by SSCL and their corresponding circles. We can see that the estimates are very accurate: each edge circle has been labeled by the prototype with the least distortion indicated by (22) and its mathematical formula mentioned above.

V. Range Image Segmentation

As a final demonstration, we applied our SSCL algorithm to range image segmentation, which involves three dimensional feature sets. Fig. 10(a) shows the popular ABW training image (512×512 pixels)[1] used to study the behavior of SSCL under different ϵ. We used the planar facet model, $z = \beta_0 + \beta_1\mu + \beta_2\nu$. For each point, we computed the planar equation using three different windows ($(2L+1) \times (2L+1)$, $L = 3, 4, 5$). The $(\beta_0, \beta_1, \beta_2)$ with the smallest variance is taken as the feature vector for this point. The experimental results are summarized in Table I. First, we set the split constant ϵ to 12. It detected 15 clusters with an average variance of 13.7. The labeled image, which is the output of the SSCL algorithm, is shown in Fig. 10(b). Fig. 10(c) shows the segmented image with the split constant $\epsilon = 8$ for which 17 clusters were detected with an average variance of 9.3. In the third experiment [see Fig. 10(d)], we set ϵ to four and discovered 20 clusters. The average variance reduced to 4.6.

VI. Conclusion

In this paper, we have presented the SSCL algorithm as a solution to the two long standing critical problems in clustering, namely, 1) the difficulty in determining the number of clusters, and 2) the sensitivity to prototype initialization.

Our SSCL algorithm is based on a new concept, OPTOC, and a split validity criterion. Using SSCL for clustering data, we need to randomly initialize only one prototype in the feature space. During the learning process, according to the split validity criterion, one prototype is chosen to split into two prototypes. This splitting process terminates only when SSCL achieves an appropriate number of clusters given the input data. We have conducted extensive experiments on a variety of data types and demonstrated that the SSCL algorithm is indeed a powerful, effective, and flexible technique in classifying clusters. Recently we have successfully applied SSCL to web data analysis [48], [49]. Since web databases are highly dynamic, SSCL is able to adaptively split according the actual datasets present. In addition, features in web information are usually high dimensional, SSCL has demonstrated its ability in dealing with such data.

The SSCL algorithm presented in this paper is very general and can be used in any practical tasks where competitive learning is applicable. With the growing popularity of competitive learning, we expect that our new approach will find many applications, particularly in intelligent software agents and web/data mining.

[1]This image was obtained from Michigan State University via a free, public ftp site.

References

[1] H. B. Barlow, "Unsupervised learning," *Neural Comput.*, vol. 1, no. 2, pp. 295–311, 1989.

[2] C. Dacaestecker, "Competitive clustering," in *Proc. IEEE Int. Neural Networks Conf.*, vol. 2, 1988, p. 833.

[3] J. M. Jolion, P. Meer, and S. Bataouche, "Robust clustering with applications in computer vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, pp. 791–802, Aug. 1991.

[4] M. Figueiredo, J. Leitão, and A. K. Jain, "On fitting mixture models," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, E. R. Hancock and M. Pellilo, Eds. Berlin, Germany: Springer-Verlag, 1999, pp. 54–69.

[5] G. McLachlan and K. Basford, *Mixture Models: Inference and Application to Clustering*. New York: Marcel Dekker, 1988.

[6] S. J. Roberts, D. Husmeier, I. Rezek, and W. Penny, "Bayesian approaches to Gaussian mixture modeling," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 20, pp. 1133–1142, Nov. 1998.

[7] J. Buhmann and M. Held, "Model selection in clustering by uniform convergence bounds," in *Advances in Neural Information Processing Systems 12*, T. L. S. Slla and K.-R. Müller, Eds. Cambridge, MA: MIT Press, 2000.

[8] T. Kohonen, *Self-Organizing Maps*, 2nd ed. Berlin, Germany: Springer-Verlag, 1997.

[9] N. R. Pal, J. C. Bezdek, and E. C. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme," *IEEE Trans. Neural Networks*, vol. 4, pp. 549–557, July 1993.

[10] J. Mao and A. K. Jain, "A self-organizing network for hyperellipsoidal clustering (HEC)," *IEEE Trans. Neural Networks*, vol. 7, pp. 16–29, Jan. 1996.

[11] T. Huntsberger and P. Ajjimarangsee, "Parallel self-organizing feature maps for unsupervised pattern recognition," *Int. J. General Syst.*, vol. 16, pp. 357–372, 1990.

[12] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[13] S. Grossberg, "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biol. Cybern.*, vol. 23, pp. 121–134, 1976.

[14] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Comput. Vision, Graphics, Image Processing*, vol. 37, pp. 54–115, 1987.

[15] ——, "ART2: Self-organization of stable category recognition codes for analog output patterns," *Appl. Opt.*, vol. 26, pp. 4919–4930, 1987.

[16] ——, "ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures," *Neural Networks*, vol. 3, pp. 129–152, 1990.

[17] G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," *Neural Networks*, vol. 3, pp. 129–152, 1990.

[18] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[19] J.-Q. Chen and Y.-G. Xi, "Nonlinear system modeling by competitive learning and adaptive fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, pp. 231–238, May 1998.

[20] F. L. Chung and T. Lee, "Fuzzy competitive learning," *Neural Networks*, vol. 7, no. 3, pp. 539–551, 1994.

[21] H. Frigui and R. Krishnapuram, "A robust competitive clustering algorithm with applications in computer vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 21, pp. 450–465, May 1999.

[22] I. Gath and A. Geva, "Unsupervised optimal fuzzy clustering," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 773–781, July 1989.
[23] L. Jouffe, "Fuzzy inference system learning by reinforcement methods," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, pp. 338–355, Aug. 1998.
[24] R. Krishnapuram, O. Nasraoui, and H. Frigui, "The fuzzy C spherical shells algorithm: A new approach," *IEEE Trans. Neural Networks*, vol. 3, pp. 663–671, Sept. 1992.
[25] R. Krishnapuram and J. M. Keller, "A possibilistic approach to clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 98–110, May 1993.
[26] ——, "The possibilistic C-Means algorithm: Insight and recommendations," *IEEE Trans. Fuzzy Syst.*, vol. 4, pp. 385–396, Aug. 1996.
[27] J. M. Buhmann and H. Kühnel, "Complexity optimized data clustering by competitive neural networks," *Neural Comput.*, vol. 5, pp. 75–88, 1993.
[28] Z.-Q. Liu, M. Glickman, and Y.-J. Zhang, "Soft-competitive learning paradigms," in *Soft Computing and Human-Centered Machines*, Z.-Q. Liu and S. Miyamoto, Eds. New York: Springer-Verlag, 2000, pp. 131–161.
[29] L. Xu, A. Krzyzak, and E. Oja, "Rival penalized competitive learning for clustering analysis, RBF Net, and curve detection," *IEEE Trans. Neural Networks*, vol. 4, pp. 636–649, July 1993.
[30] N. Ueda and R. Nakano, "A new competitive learning approach based on an equidistortional principle for designing optimal vector quantizers," *Neural Networks*, vol. 7, no. 8, pp. 1211–1227, 1994.
[31] E. Yair, K. Zeger, and A. Gersho, "Competitive learning and soft competition for vector quantizer design," *IEEE Trans. Signal Processing*, vol. 40, pp. 294–309, Feb. 1992.
[32] T. Hofmann and J. M. Buhmann, "Competitive learning for robust vector quantization," *IEEE Trans. Signal Processing*, vol. 46, no. 6, pp. 1665–1675, 1998.
[33] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. New York: Addison-Wesley, 1991.
[34] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, pp. 84–95, 1980.
[35] E. Forgy, "Cluster Analysis of multivariate data: Efficiency vs. interpretanility of classifications," *Biometrics*, vol. 21, p. 768, 1965.
[36] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inform. Theory*, vol. 28, pp. 129–137, 1982.
[37] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probability*. Berkeley, CA: Univ. California Press, 1967, pp. 281–297.
[38] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Sci.*, vol. 9, pp. 75–112, 1985.
[39] S. Grossberg, "Competitive learning: From iterative activation to adaptive resonance," *Cognitive Sci.*, vol. 11, pp. 23–63, 1987.
[40] R. Hecht-Nielsen, "Counterpropopagation networks," *Appl. Opt.*, vol. 26, pp. 4979–4984, 1987.
[41] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, "Neural-gas network for vector quantization and its application to time-series prediction," *IEEE Trans. Neural Networks*, vol. 4, pp. 558–568, July 1993.
[42] D. deSieno, "Adding consciousness to competitive learning," in *Proc. 2nd IEEE Int. Conf. Neural Networks*, vol. 1, 1988, pp. 117–124.
[43] S. C. Ahalt, A. K. Krishnamurty, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, no. 3, pp. 277–291, 1990.
[44] A. Joshi and R. Krishnapuram, "Robust fuzzy clustering methods to support web mining," in *Proc. ACM SIGMOD Workshop Data Mining Knowledge Discovery*, Aug. 1998.
[45] B. Fritzke, "Growing cell structures—A self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441–1460, 1994.
[46] ——, "Fast learning with incremental RNF networks," *Neural Processing Lett.*, vol. 1, no. 1, pp. 2–5, 1994.
[47] V. Cherkassky and F. Mulier, *Learning From Data*. New York: Wiley, 1998.
[48] Z. Q. Liu and Y. J. Zhang, "Refine web search engine results using incremental clustering," Int. J. Intell. Syst., 2002, to be published.
[49] Y. J. Zhang and Z. Q. Liu, "An efficient neural solution for web pages categorization," Int. J. Uncertainty, Fuzziness, Knowledge-Based Syst., 2002, to be published.

Ya-Jun Zhang received the B.S. and M.S. degrees, both in computer science, in 1995 and 1997, respectively, both from Tsinghua University, Beijing, P.R. China. He received the Ph.D. degree in computer science from the University of Melbourne, Australia, in 2001.

He is currently a Senior R&D Engineer in the Silicon Valley. His research interests include neural networks, fuzzy systems and applications, soft computing algorithms, image processing and web mining, broadband networks, and communication systems.

Zhi-Qiang Liu (S'82–M'86–SM'91) received the M.A.Sc. degree in aerospace engineering from the Institute for Aerospace Studies, The University of Toronto, Toronto, ON, Canada, and the Ph.D. degree in electrical engineering from The University of Alberta, Edmonton, AB, Canada.

He is currently a Professor with School of Creative Media, City University of Hong Kong, and Department of Computer Science and Software Engineering, The University of Melbourne, Australia. He has taught computer architecture, computer networks, artificial intelligence, programming languages, machine learning, and pattern recognition. His research and development interests include machine intelligence, fuzzy-neural systems, visual communications, biometric systems and applications, and web/data-mining.