TASOM: A New Time Adaptive Self-Organizing Map

Hamed Shah-Hosseini and Reza Safabakhsh, *Member, IEEE*

***Abstract*—The time adaptive self-organizing map (TASOM) network is a modified self-organizing map (SOM) network with adaptive learning rates and neighborhood sizes as its learning parameters. Every neuron in the TASOM has its own learning rate and neighborhood size. For each new input vector, the neighborhood size and learning rate of the winning neuron and the learning rates of its neighboring neurons are updated. A scaling vector is also employed in the TASOM algorithm for compensation against scaling transformations. Analysis of the updating rules of the algorithm reveals that the learning parameters may increase or decrease for adaptation to a changing environment, such that the minimum increase or decrease is achieved according to a specific measure. Several versions of the TASOM-based networks are proposed in this paper for different applications, including bilevel thresholding of grey level images, tracking of moving objects and their boundaries, and adaptive clustering. Simulation results show satisfactory performance of the proposed methods in the implemented applications.**

***Index Terms*—Author, please supply your own keywords or send a blank e-mail to keywords@ieee.org to receive a list of suggested keywords.**

I. Introduction

THE self-organizing map (SOM) algorithm, which was first introduced by Kohonen [1], transforms input vectors to a discrete map in an adaptive fashion. This discrete map often resembles a one-dimensional (1-D) array or a two-dimensional (2-D) grid of neurons and is usually called a lattice. This algorithm has been used in applications such as vector quantization [2], texture segmentation [3], cloud classification [4], phonetic typewriter [5], image compression [6], and local dynamic analysis [7].

The learning (learning rate and neighborhood size) parameters of the original SOM are at their highest values at the beginning of learning. Then, they decrease with time so that the feature map stabilizes and learns the topographic map of the environment. This is, in fact, the reason why the basic SOM algorithm cannot work properly in changing environments. Even constant learning rates cannot deal with such environments [8]. The speed of change in the environment is not guaranteed to be constant. Therefore, adaptive learning parameters must be employed in the SOM algorithm.

An SOM-based network that embodies adaptive learning parameters has been introduced [9], [10]. This network, which is called time adaptive self-organizing map (TASOM), uses adaptive learning parameters whose values change based on the changes of the environment and the behavior of input vectors as time passes. It has been shown that the TASOM network is able to work in stationary as well as nonstationary environments. TASOM has been used for adaptive shape representation and adaptive segmentation [10], bilevel thresholding [11], adaptive pattern classification [12], adaptive principal components analysis [13], and automatic multilevel thresholding [14].

The TASOM algorithm is explained and discussed in Section II. In Section III, modified versions of the TASOM algorithm are proposed for bilevel thresholding, tracking of moving objects, and adaptive pattern clustering. Section IV includes some experiments with the TASOM algorithm and its modified versions. Conclusions are summarized in Section V.

Manuscript received August 8, 2001; revised January 22, 2002. This paper was recommended by Associate Editor I. Bloch.

The authors are with the Department of Computer Engineering, Amirkabir University of Technology, Tehran 15914, Iran.

Publisher Item Identifier S 1083-4419(02)06480-4.

II. Time Adaptive Self-Organizing Map Networks

The main feature of the TASOM networks is that they have adaptive learning rates and neighborhood sizes. With this feature, the weights of a TASOM network are continuously trained by the input vectors. Therefore, without any supervision, the environmental changes are reflected in the weights and learning parameters of TASOM. This enables TASOM to approximate the distribution of the input space with emphasis on the most recent input vectors, if these input vectors substantially differ from the previous ones.

This section will explain the TASOM algorithm in detail and will present the adaptation rules of the learning rates and neighborhood sizes.

A. TASOM Algorithm With Adaptive Learning Rates

The first version of the TASOM that only uses time independent learning rates was introduced in [15]. This algorithm was used for quadrature amplitude modulation (QAM) and was compared to the basic SOM having a constant learning rate. The neighborhood sizes of the two algorithms were the same; and after the initial training, they were kept larger than a constant value. Experimental results showed the superiority of this version of the TASOM algorithm [15] to the basic SOM.

B. The TASOM Algorithm With Adaptive Learning Rates and Neighborhood Sizes

The TASOM proposed in [10] uses adaptive learning rates and neighborhood sets. Like the basic SOM, it was shown that neighborhood functions could be used in the TASOM algorithm instead of neighborhood sets or radii [11]. The neurons of this TASOM use neighborhood functions instead of neighborhood radii, and the widths of these functions are controlled adaptively.

The proposed TASOM algorithm with adaptive learning rates and neighborhood sets may be specified in the following eight steps [10].

Step 1) Initialization: Choose some values for the initial weight vectors $\mathbf{w}_j(0)$, where $j = 1, 2, \ldots, N$;

and N is the number of neurons in the lattice. The learning-rate parameters $\eta_j(0)$ should be initialized with values close to unity. The constant parameters $\alpha, \beta, \alpha_{\mathbf{s}}$, and $\beta_{\mathbf{s}}$ can have any values between zero and one. The constant parameters s_f and s_g should be set to satisfy the application's needs. $R_i(0)$ should be set to include all the neurons. The components $s_k(0)$ of the scaling vector $\mathbf{s}(0) = [s_1(0), \ldots, s_p(0)]^T$ should be set to small positive values, where p is the dimension of the input and weight vectors. The parameters $E_k(0)$ and $E2_k(0)$ may be initialized with some small random values. Neighboring neurons of any neuron i in a lattice are included in the set NH_i. In this paper, for any neuron i in a 1-D lattice $N, \mathrm{NH}_i = \{i-1, i+1\}, \mathrm{NH}_N = \{N-1\}$, and $\mathrm{NH}_1 = \{2\}$. Similarly, for any neuron (i_1, i_2) in a 2-D lattice $N = M \times M, \mathrm{NH}_{(i_1,i_2)} = \{(i_1 - 1, i_2), (i_1+1, i_2), (i_1, i_2-1), (i_1, i_2+1)\}$, where $\mathrm{NH}_{(1,1)} = \{(1,2),(2,1)\}, \mathrm{NH}_{(1,M)} = \{(1, M-1), (2, M)\}, \mathrm{NH}_{(M,1)} = \{(M,2), (M-1, 1)\}$, and $\mathrm{NH}_{(M,M)} = \{(M, M-1), (M-1, M)\}$.

Step 2) Sampling: Get the next input vector $\mathbf{x}$ from the input distribution. The assumption is that the input distribution is unknown to the TASOM algorithm, and the input vectors are received in an arbitrary order.

Step 3) Similarity matching: Find the best-matching or winning neuron $i(\mathbf{x})$ at time n, using the minimum-distance Euclidean norm scaled by the scaling vector $\mathbf{s}(n)$

$$i(\mathbf{x}) = \arg\min_j \|\mathbf{x}(n) - \mathbf{w}_j(n)\|_{\mathbf{s}}, \quad j = 1, 2, \ldots, N \tag{1}$$

where $\|\mathbf{x}(n) - \mathbf{w}_j(n)\|_{\mathbf{s}} = (\sum_k ((x_k(n) - w_{j,k}(n))/s_k(n))^2)^{1/2}$..

Step 4) Updating the neighborhood size: Adjust the neighborhood set $\Lambda_i(n)$ of the winning neuron $i(\mathbf{x})$ by the following two equations:

$$\Lambda_i(n+1) = \{j \in N \mid d(i,j) \le R_i(n+1)\} \tag{2}$$

where

$$R_i(n+1) = R_i(n) + \beta \left(g \left((s_g \cdot |\mathrm{NH}_i|)^{-1} \times \sum_{j \in \mathrm{NH}_i} \|\mathbf{w}_i(n) - \mathbf{w}_j(n)\|_{\mathbf{s}} \right) - R_i(n) \right) \tag{3}$$

and β is a constant parameter between zero and one which controls how fast the neighborhood sizes should follow the local neighborhood errors $\sum_{j \in \mathrm{NH}_i} \|\mathbf{w}_i(n) - \mathbf{w}_j(n)\|_{\mathbf{s}}$. The scalar function $\lceil \cdot \rceil$ gives the cardinality of a set, and $d(i,j)$ is the distance between two neurons i and j in the lattice. The neighborhood sets of the other neurons do not change.

Function $g(\cdot)$ is a scalar function for which $(dg(z))/(dz) \ge 0$ for $z > 0$, and is used for normalization of the weight distances. For 1-D lattices of N neurons, $g(0) = 0$ and $0 \le g(z) < N$, and for 2-D lattices of $M \times M$ neurons, $0 \le g(z) < M\sqrt{2}$. Examples of this function are $g(z) = (N-1)(z/(z+1))$ and $g(z) = (M\sqrt{2} - 1)(z/(z+1))$.

Step 5) Updating the learning-rate parameters: Adjust the learning-rate parameters $\eta_j(n)$ in the neighborhood $\Lambda_{i(\mathbf{x})}(n+1)$ of the winning neuron $i(\mathbf{x})$ by

$$\eta_j(n+1) = \eta_j(n) + \alpha(f(\|\mathbf{x}(n) - \mathbf{w}_j(n)\|_{\mathbf{s}}/s_f) - \eta_j(n)) \quad \text{for } j \in \Lambda_{i(x)}(n+1). \tag{4}$$

The learning-rate parameters of the other neurons do not change.

Function $f(\cdot)$ is a monotonically increasing scalar function such that for each positive z, we have $0 < f(z) \le 1$ and $f(0) = 0$. This function is used for normalization of the distance between the weight and input vectors. An example of this function is $f(z) = z/(z+1)$.

Step 6) Updating the synaptic weights: Adjust the synaptic weight vectors of all output neurons in the neighborhood $\Lambda_{i(\mathbf{x})}(n+1)$, using the following update rule: (see (5) at the bottom of the page).

Step 7) Updating the scaling vector: Adjust the scaling vector $\mathbf{s}(n) = [s_1(n), \ldots, s_k(n), \ldots, s_p(n)]^T$ by the following equations:

$$s_k(n+1) = \sqrt{(E2_k(n+1) - E_k(n+1)^2)^+} \tag{6}$$

where $E2_k(n+1) = E2_k(n) + \alpha_{\mathbf{s}}(x_k^2(n) - E2_k(n)), E_k(n+1) = E_k(n) + \beta_{\mathbf{s}}(x_k(n) - E_k(n))$, and $(z)^+ = \max(z, 0)$.

The nonuniform scaling defined in Step 7 should be used for preserving the topological ordering of neurons in the lattice when the input distribution is nonsymmetric. For the basic SOM, a different mechanism is proposed [16]. For a uniform scaling vector, the scaling vector $\mathbf{s}(n)$ should be replaced with a scalar (1-D vector) $\mathrm{sl}(n)$, which is updated by $\mathrm{sl}(n+1) = \sqrt{\sum_k s_k(n+1)^2}$. In this case all distance calculations in the TASOM algorithm are simply scaled by the scalar $\mathrm{sl}(n)$. Specifically, $\|\mathbf{w}_i(n) - \mathbf{w}_j(n)\|_{\mathbf{s}} = \|\mathbf{w}_i(n) - \mathbf{w}_j(n)\|/\mathrm{sl}(n)$,

$$\mathbf{w}_j(n+1) = \begin{cases} \mathbf{w}_j(n) + \eta_j(n+1)(\mathbf{x}(n) - \mathbf{w}_j(n)), & j \in \Lambda_{i(\mathbf{x})}(n+1) \\ \mathbf{w}_j(n), & \text{otherwise} \end{cases} \tag{5}$$

where $\|\cdot\|$ stands for the Euclidean norm. In this case, the distance calculations in Step 3 are not required to be scaled.

Step 8) Continuous learning: go to Step 2.

When neighborhood functions are used in the TASOM algorithm instead of neighborhood sets, Step 4 of the algorithm is changed such that only the neighborhood width $\sigma_i(n)$ of the winning neuron i is updated as follows:

$$\sigma_i(n+1) = \sigma_i(n) + \beta\left(g\left((s_g \cdot |\mathrm{NH}_i|)^{-1} \times \sum_{j\in \mathrm{NH}_i} \|\mathbf{w}_i(n) - \mathbf{w}_j(n)\|_{\mathbf{s}}\right) - \sigma_i(n)\right). \quad (7)$$

Moreover, the equations in Steps 5 and 6 are replaced by

$$\eta_j(n+1) = \eta_j(n) + \alpha(f(\|\mathbf{x}(n) - \mathbf{w}_j(n)\|_{\mathbf{s}}/s_f) - \eta_j(n)) \quad \text{for all neurons } j \quad (8)$$

and

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta_j(n+1)h_{j,i}(n+1)(\mathbf{x}(n) - \mathbf{w}_j(n)) \quad \text{for all neurons } j \quad (9)$$

respectively, where index i stands for the winning neuron and index j stands for other neurons of the network. The neighborhood function $h_{j,i}(n) = \exp(-d_{j,i}^2/\sigma_i^2(n))$ is centered at the winning neuron. Using $2\sigma_i^2(n)$ instead of $\sigma_i^2(n)$ in the function $h_{j,i}(n)$ has no effect on the performance of the TASOM algorithm. The distance between neurons j and i, which is denoted by $d_{j,i}^2$, is calculated in the lattice space by $d_{j,i}^2 = \|j - i\|^2$.

C. Analysis of the Learning Rate and Neighborhood Width Updating Rules

In this section, we will have a closer look at the updating rules of (7) and (8), where the neighborhood functions are used for TASOM and thus all learning rates of the network are updated for each input vector. By replacing the discrete variable n with its continuous counterpart t, and assuming that Δt is a very small positive value, the learning rate updating rule of (8) may be written as

$$\frac{d\eta_i(t)}{dt} = -\alpha\eta_i(t) + \alpha f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f) \quad (10)$$

where $\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f = \|\mathbf{x}(t) - \mathbf{w}_i(t)\|_{\mathbf{s}}/s_f$. If we assume that the changes of $\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f$ with time are much smaller than that of the learning rate $\eta_i(t)$, we can solve the above nonhomogeneous first-order differential equation as

$$\eta_i(t) = \eta_i(0)e^{-\alpha t} + (1 - e^{-\alpha t})f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f) \quad (11)$$

where $\eta_i(0)$ is the initial learning rate of neuron i. The first term on the right-hand side (RHS) of (11) decreases exponentially to zero as time goes to infinity, as in the basic SOM. The second term on the RHS of this equation needs more careful examination. In fact, function $f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f)$ gives some sense of a normalized error of neuron i. When time goes to infinity, the second term of (11) is the dominant term. For a stationary environment, if a sufficient number of neurons has been used in the TASOM network, the normalized errors are reduced with every training step. Thus, the learning rates reduce to zero as training goes on. This, in fact, leads to the stabilization of the weights of the neurons.

For a nonstationary environment, when the characteristics of the input space change, the stabilized weights of the neurons may fail to represent the input space. At this time, the normalized errors of the corresponding neurons increase, causing the relevant learning rates to increase. Consequently, the weights of the network are forced to learn the new input data faster than before. As the training continues, the weights of the network approximate the current state of the input space with better accuracy. The normalized errors of the neurons thus decrease, resulting in the reduction of the learning rates and the stabilization of the weights.

For the neighborhood width updating rule in (7), a similar procedure leads to the following solution:

$$\sigma_i(t) = \sigma_i(0)e^{-\beta t} + (1 - e^{-\beta t})g \times \left(|\mathrm{NH}_i|^{-1} \sum_{j\in \mathrm{NH}_i} \|\mathbf{w}_i(t) - \mathbf{w}_j(t)\|_g\right). \quad (12)$$

Again, for simplicity, we have substituted $\|\mathbf{w}_i(t) - \mathbf{w}_j(t)\|_g$ for $\|\mathbf{w}_i(t) - \mathbf{w}_j(t)\|_{\mathbf{s}}/s_g$. The initial value of the neighborhood width for neuron i is denoted by $\sigma_i(0)$.

An analysis similar to the one made for (11) can be made for (12). This neighborhood width updating rule tries to keep the topological ordering of the network in spite of the changes in the weight vectors, and it helps the network to prevent underutilization of neurons.

D. Error Function Minimization

Here, the total error function e_{total} that the updating rules of (7) and (8) try to minimize is introduced. Suppose that the rule of (8) tries to minimize an error (cost) function such as $e_{i,\eta}$. The rule then equals the negative of the gradient of $e_{i,\eta}$ with respect to the learning rate η_i. Specifically,

$$\frac{de_{i,\eta}}{d\eta_i} = -\alpha(f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f) - \eta_i(t)). \quad (13)$$

By integrating both sides, ignoring constant terms, and assuming very small changes for the normalized error $f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f)$ with respect to η_i, we obtain

$$e_{i,\eta} = -\alpha f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f)\eta_i + \alpha\eta_i^2/2. \quad (14)$$

This is the learning rate error for neuron i. The learning rate error $e_\eta(t)$ at time t for all neurons is the summation of individual errors $e_{i,\eta}$. Therefore

$$e_\eta(t) = -\alpha\sum_{i=1}^{N} f(\|\mathbf{x}(t) - \mathbf{w}_i(t)\|_f)\eta_i + \frac{\alpha}{2}\sum_{i=1}^{N}\eta_i^2. \quad (15)$$

The total learning rate error $e_{\mathrm{total},\eta}$ of the network is the expectation of $e_\eta(t)$ over time. Thus

$$e_{\mathrm{total},\eta} = -\alpha\sum_i E(f(\|\mathbf{x} - \mathbf{w}_i\|_f)\eta_i) + \frac{\alpha}{2}\sum_i E\left(\eta_i^2\right). \quad (16)$$

According to (16), the total learning rate error $e_{\text{total},\eta}$ is minimized when each $E(f\|\mathbf{x}-\mathbf{w}_i\|_f)\eta_i)$ is maximized, and each $E(\eta_i^2)$ is minimized. The former is a cross correlation between the normalized learning rate error and learning rate of the same neuron. Therefore, the total learning rate error is reduced when the cross correlation increases. In other words, the learning rate modification rule tries to maximize the correlation between the normalized learning rate error and its corresponding learning rate, and thus the learning rate should follow the normalized error. On the other hand, $E(\eta_i^2)$ in (16) is the time-average of the square of learning rates which the learning rate modification rule tends to minimize. Therefore, the proposed learning rate rule tries to follow environmental changes with the least increase in the magnitude of the learning rates. This strategy leads to more stable weight vectors because the network attempts to adapt to the new conditions with the least sufficient change in its weight vectors. It may be said that the first term on the RHS of (16) plays the role of plasticity, whereas the second term plays the role of stability.

A similar analysis for the neighborhood width updating rule specified in (7) reveals that the proposed rule tries to minimize the following error:

$$e_{\text{total},\sigma} = -\beta \sum_i E\left(g\left(|\text{NH}_i|^{-1}\sum_{j\in \text{NH}_i}\|\mathbf{w}_i-\mathbf{w}_j\|_g\right)\sigma_i\right) + \frac{\beta}{2}\sum_i E\left(\sigma_i^2\right). \quad (17)$$

By looking at the RHS of (17), it can be concluded that the plasticity role for the neighborhood width adjustment is played by the first term, while the stability role is played by the second term.

The total error e_{total} that the TASOM network tries to minimize is the summation of the two errors $e_{\text{total},\eta}$ and $e_{\text{total},\sigma}$. Specifically

$$e_{\text{total}} = e_{\text{total},\eta} + e_{\text{total},\sigma}. \quad (18)$$

The learning rate updating rule in (8) tries to minimize the error $e_{\text{total},\eta}$ defined in (16), and the neighborhood updating rule in (7) minimizes the error $e_{\text{total},\sigma}$ defined in (17).

Before ending this section, we shall emphasize that there are some other SOMs that may be used for nonstationary and changing environments [17]–[20]. These SOM algorithms add or delete neurons to deal with changing environments, causing each neuron to lose its identity. For example, neuron i at the previous situation may be totally different from neuron i at the current state. On the other hand, a stable identity property exists for each neuron of the TASOM algorithm when a fixed number of neurons are used, such as in the experiments implemented in Section IV-A. The TASOM adaptation to changing environments is achieved through dynamic parameter adjustment, and thus no neuron deletion or addition is needed for this purpose.

III. TASOM-Based Algorithms for Real Applications

This section introduces several versions of the TASOM algorithm for bilevel thresholding, tracking moving objects, and adaptive clustering.

A. A TASOM-Based Algorithm for Bilevel Thresholding

Bilevel thresholding or grey level segmentation converts a grey level image to a bilevel image which consists of only black and white pixels. In this conversion, it is assumed that the image consists of two distinct regions: "Figure" and "Background." A similar process occurs in human visual system [21].

A TASOM-based bilevel thresholding algorithm was introduced in [11]. It uses two neurons which are trained by image pixels. The average of the two neuron weights is used as the desired threshold. For some cases, this threshold may not be placed correctly at the bottom of the valley between the two prominent distributions of the image histogram, especially when the two distributions are close together, and they have different variances.

To solve such problems, a modified TASOM algorithm for bilevel thresholding is proposed here. Let us call it "TASOM_BTA" for TASOM bilevel thresholding Algorithm. This algorithm begins its work with two neurons forming a 1-D lattice. The image grey level pixels form the 1-D input values of the TASOM, which are used for training the TASOM. During this network training, the number of neurons is allowed to change. The complete proposed TASOM_BTA is specified in the following steps.

- Step 1) Set the parameter θ_v to a positive constant value much less than 255, which is the maximum possible grey level value of an image. This parameter controls the accuracy of the proposed algorithm in finding the threshold value. Lower values of θ_v result in the more accurate threshold values. In this paper, we set $\theta_v = 10$.
- Step 2) Construct a TASOM network with two neurons forming a 1-D lattice.
- Step 3) The TASOM network is trained by the image pixels according to the TASOM algorithm specified in Section II-B.
- Step 4) Set the two neuron indexes: $a = 1$ and $b = 2$.
- Step 5) Add a new neuron v between the two neurons a and b, and call it the "virtual valley neuron" or simply "valley neuron" such that

$$\mathbf{w}_v = (\mathbf{w}_a + \mathbf{w}_b)/2, \quad \sigma_v = (\sigma_a + \sigma_b)/2, \quad \eta_v = (\eta_a + \eta_b)/2, \quad \text{valley} - \text{neuron} = v. \quad (19)$$

- Step 6) Train the network with the image pixels using the TASOM algorithm.
- Step 7) Find the neurons that do not win any competition during the training and delete those of them which are placed at the head or tail of the lattice.
- Step 8) If $\max(\|\mathbf{w}_v - \mathbf{w}_{v-1}\|, \|\mathbf{w}_v - \mathbf{w}_{v+1}\|) < \theta_v$, then go to Step 10.
- Step 9) The place that a new neuron is to be inserted next to the current valley neuron v is defined:

If $||\mathbf{w}_v - \mathbf{w}_{v-1}|| > ||\mathbf{w}_v - \mathbf{w}_{v+1}||$, then set $a = v$ and $b = v - 1$; else set $a = v$ and $b = v + 1$. After that, go to Step 5.

Step 10) The desired threshold is the weight of the real valley neuron. The real valley neuron as discussed below is found by a simple process using the (virtual) valley neuron v.

The winning frequency of each neuron is computed as the number of times that it wins during one epoch of training. If the winning frequency of the virtual valley neuron is less than those of its two immediate neighboring neurons, then the virtual valley neuron is the real valley neuron. Otherwise, we have to find the real valley neuron. For this purpose, we move from the virtual valley neuron to its adjacent left neuron and continue this left movement until the current left neuron has a winning frequency higher than its right neuron. The current right neuron is chosen as the left valley neuron. The same process is done for the right direction, and the right valley neuron is found. Now, the real valley neuron is the one having the least winning frequency among the two candidate valley neurons. The desired threshold is then the weight value of the real valley neuron. Now, those pixels of the image having grey levels less than the threshold are set to black, and those pixels with grey levels greater than the threshold are set to white.

B. TASOM-Based Algorithms for Tracking Moving Objects

Two algorithms based on the TASOM network are proposed here. One algorithm is used for tracking centers of and the other for modeling and tracking boundaries of moving objects.

Different methods have been suggested for motion analysis [22]–[25]. One way to track a moving object in an image sequence is to represent and follow its center. We propose to use a 2×2 TASOM network for tracking each object. The center of each object will be approximated by the average weight of the four weights of its corresponding TASOM network. A TASOM network is automatically devoted to every moving object, and each TASOM network follows its own object regardless of the movement of other objects. Here, each neuron of the TASOM network has an influence radius so that input data outside this radius are ignored by the neuron. At the beginning, there is no TASOM network. A new TASOM network is created if there are some moving pixels in the image.

The algorithm of this dynamic pool of TASOM networks for tracking moving objects, which we call "TASOM_MOTION," may be summarized as follows.

1) Set the influence radius of the neurons ρ_r to a positive constant value. This parameter should represent the maximum radius of each moving object in the image sequence. In this paper, $\rho_r = 50$.
2) For each frame of the image sequence the following process is repeated.
3) The moving pixels are identified for the current frame. These pixels may be obtained by defining a reference image having no moving object [26], and taking the difference between the current frame and this reference image. The pixels with high difference values are the candidate moving pixels. All the moving pixels are labeled as unmarked.
4) Each moving pixel is fed to the dynamic pool of TASOM networks. The neuron weight vector of the winning TASOM has the nearest distance from the current moving pixel among other TASOM networks, and this distance is lower than the parameter ρ_r. Then the weights of the winning TASOM network k are updated according to the standard learning algorithm of TASOM for 2-D lattices. The relevant moving pixel is marked with index k.
5) A new TASOM network with four neurons forming a 2×2 lattice is added to the pool of TASOM networks if there is any unmarked moving pixel.
6) A TASOM network is deleted from the pool of TASOM networks if it has won no moving pixel for two successive frames.
7) The above process is continued with the next frame by going to Step 3.

For modeling and tracking the boundary of a moving object, an active contour modeling based on TASOM is proposed in the rest of this section. An active contour modeling based on the basic SOM is introduced in [27]. However, the algorithm needs the initial contour to be fairly close to the desired contour. Moreover, the number of the control points on the contour remains fixed, and they are the pixels on the image that intersect with the initial contour. This large number of control points on the contour in fact makes the convergence of the algorithm slow.

The proposed TASOM-based algorithm, which we call "TASOM_CONTOUR," is specified in the following steps.

Step 1) The initial contour is made available by the user or some other algorithms.

Step 2) The feature points of the image are extracted. These are usually the edges of the object for which the initial contour has been defined.

Step 3) A TASOM network with a closed chain topology of neurons is constructed. Therefore, for any neuron i in the 1-D lattice N, $\mathrm{NH}_i = \{i-1, i+1\}$, $\mathrm{NH}_N = \{N-1, 1\}$, and $\mathrm{NH}_1 = \{N, 2\}$. The number of neurons N is set equal to the number of the control points on the initial contour. The 2-D weight vectors of neurons are set equal to the coordinates of the control points on the initial contour.

Step 4) The coordinates of the feature points of the image are used to train the TASOM network. Since the topology is closed, the distance $d_{j,i}$ between neurons j and i, used in the neighborhood function of (9), is modified by $d_{j,i} = \min(||j-i||, N - ||j-i||)$.

Step 5) Any unused neuron during one epoch of training on the feature points of the image is identified and is simply deleted from the TASOM network.

Step 6) Between any two adjacent neurons i and $i+1$, a new neuron j is inserted if their distance $||\mathbf{w}_i - \mathbf{w}_{i+1}||$ is higher than a predetermined parameter ϑ_h. The weight and learning parameters of the newly added neuron will be

$$\mathbf{w}_j = (\mathbf{w}_i + \mathbf{w}_{i+1})/2$$

$$\sigma_j = (\sigma_i + \sigma_{i+1})/2, \quad \eta_j = (\eta_i + \eta_{i+1})/2. \tag{20}$$

Step 7) Any two adjacent neurons i and $i+1$ with the distance $\|\mathbf{w}_i - \mathbf{w}_{i+1}\|$ lower than the predetermined parameter ϑ_l are replaced by a new neuron j with the weight and the learning parameters specified in (20).

Step 8) If no new neuron is added or deleted from the TASOM network, the desired contour is obtained by concatenating the weight values of the TASOM network; otherwise, training is repeated by going to Step 4.

The parameters ϑ_l and ϑ_h determine the accuracy of the TASOM network in approximating the boundary of the object. Lower values would result in more accurate contours. It should be mentioned that ϑ_l must be chosen less than ϑ_h, and both of them are positive.

C. A TASOM-Based Algorithm for Adaptive Clustering

In adaptive clustering, we assume that the number of clusters may change with time, and the clusters may also move in an arbitrary fashion. To keep the same level of accuracy in approximating the clusters, we propose to use a TASOM network with a variable number of neurons in a linear array topology.

The proposed clustering algorithm, which we call "TASOM_CLUSTERING," is specified in the following steps.

Step 1) A TASOM network with a linear array topology of two neurons is constructed.

Step 2) The TASOM network is trained with $K \times$NON input data vectors, where NON is the current number of neurons of the TASOM, and K is a constant positive integer. This parameter should be large enough to be sure that each neuron of the network has a chance to win in one epoch. Here, we use $K = 30$. One epoch of training is completed when $K \times$ NON input data vectors are entered to the TASOM network.

Step 3) Any unused neuron after one epoch of training is identified and deleted from the TASOM network.

Step 4) A new neuron may be added between any two adjacent neurons in the same way specified in Step 6 of the TASOM_CONTOUR algorithm of Section III-B.

Step 5) Any two adjacent neurons may be replaced by a new neuron as specified in Step 7 of the TASOM_CONTOUR algorithm of Section III-B.

Step 6) Training is continued by going to Step 2.

This algorithm is similar to the TASOM_CONTOUR algorithm of Section III-B except that the topology of neurons is a linear array, and two neurons are used at the beginning. Moreover, learning is done continuously, and it never stops.

IV. Experimental Results With TASOM and its Modified Versions

This section emphasizes on the TASOM networks with neighborhood functions [11]. It should be noted that a subtle difference between the TASOM with neighborhood function and the TASOM with neighborhood set was observed. Therefore, no comparison between the two versions of TASOM is included in the paper. This section includes applications of the TASOM algorithm in nonstationary as well as stationary environments. Different versions of TASOM are used for input distribution approximation, bilevel thresholding, tracking moving objects, and adaptive clustering.

A. Input Distribution Approximation by TASOM

In this section, we would like to examine the input distribution approximation properties of the TASOM using a uniform input distribution. The TASOM network uses a 2-D lattice having 5×5 neurons. For the first experiment, a 2-D input distribution in region [0, 1] $\times$ [0, 1] is fed to the TASOM algorithm, which uses a uniform scaling vector, as explained in Section II-B. The weights of the TASOM network after 15 000 iterations are shown in Fig. 1(a). As it was expected, the weights topologically fill the input distribution space.

The crucial constant parameters for the TASOM are s_f and s_g. For this experiment, they are set to 20 and 5, respectively. The values of the other constant parameters are not crucial to the performance of TASOM. In all the experiments of this paper, we set $\alpha = \alpha_s = \beta = \beta_s = 0.1\eta_i(0) = 1$ and $\sigma_i(0) = M/4$ for each neuron i in the $M \times M$ TASOM network. The constant parameters s_f and s_g which are the slopes of the functions $f(\cdot)$ and $g(\cdot)$, respectively, are determined experimentally using the available training data. Determination of these parameters depends on the application for which the TASOM algorithm is employed. The remaining parameters are set randomly.

Now the input distribution undergoes a uniform scaling such that the input space covers the region [0, 100] $\times$ [0, 100]. The TASOM of Fig. 1(a) has to work with the same values of weights and parameters that it had just before this change. The weights of the TASOM after 200, 1000, and 5000 iterations are shown in Fig. 1(b)–(d), respectively. These figures show that the weights gradually move toward the new enlarged input space to fill it. At first iterations, a few neurons move toward the enlarged space. As more samples are fed to the TASOM, more neurons go to the new space till finally they fill the space topologically. Similar results are obtained when the input space is scaled down to a smaller space.

The learning rate and the neighborhood width of the first neuron of the TASOM network of Fig. 1(a) versus training over each epoch of 1000 samples are shown in Fig. 2(a) and (b), respectively. The first 15 000 samples come from the input distribution in the region [0, 1] $\times$ [0, 1], and the next 15 000 samples come from the input distribution in the region [0, 100] $\times$ [0, 100]. It is seen that the learning rate and the neighborhood width decrease and increase based on the conditions of the environment. This occurs in contrast to the always decreasing learning rate and neighborhood width of SOM-based networks. The prominent change in the curves of Fig. 2(a) and (b) occurs between epochs 15 and 16. The neighborhood width obviously decreases, and the learning rate increases. This trend is, however, temporary: after a while, the previous trend follows. The quantization error of the TASOM network versus epochs is shown in Fig. 2(c). For the first 15 epochs, the quantization error is very low, indicating that the

TASOM is stable and convergent. Between epochs 15 and 16, the error rapidly increases. As the training continues, the error reaches its low level for the enlarged input distribution. The weights of the TASOM, as shown in Fig. 1(d), completely fill the current input space.

The input distributions used so far were symmetric. Assume now that the input space is a nonsymmetric uniform distribution in the region $[0, 100] \times [0, 1]$. The TASOM of Fig. 1(a) tries to learn the nonsymmetric distribution, which results in the weights shown in Fig. 3(a). The weights fill the input space, but they do not fully preserve the topological ordering of the neurons on the lattice. The priority here is to lower the quantization error, as is required in some applications. If the priority in an application is to preserve the topological ordering, the nonuniform scaling vector, as suggested in Section II-B, should be employed in the TASOM algorithm. Such a TASOM with $s_f = 30$ and $s_g = 6$, trained by 15 000 samples of a uniform distribution in the region $[0, 1] \times [0, 1]$, faces the nonsymmetric uniform distribution in region $[0, 100] \times [0, 1]$ without any new initialization of parameters or weights. The weights of the TASOM network after 4000 iterations are shown in Fig. 3(b). It is seen that the TASOM network with nonuniform scaling completely keeps its topological ordering and a quantization error of 4.2433 results. The quantization error obtained by the TASOM with uniform scaling is 1.4940. The TASOM weights for this case are shown in Fig. 3(a).

The next experiment of this section is to show that the TASOM algorithm is essentially rotation invariant. For this purpose, the uniform distribution in region $[0, 1] \times [0, 1]$ is rotated by 30° around the origin. The TASOM network of Fig. 1(a) faces this rotated region and adapts to this environment resulting in the weights of TASOM in Fig. 4, shown by asterisks. For evaluating the TASOM performance, the weights of the previous network of Fig. 1(a) are rotated by 30°, and are shown by diamonds. It is seen that the weights of TASOM automatically adapted to the rotation, and the weights of the map of Fig. 1(a), rotated by 30°, almost overlap each other. The average difference between the corresponding weights of the two maps is also calculated, which is 0.0192. This average difference is a very low value with respect to the distribution size. In summary, we can claim that the TASOM network fills the rotated region while keeping its topological ordering.

A TASOM network with 5×5 neurons, trained with 5000 samples of a uniform distribution in the region $[0, 1] \times [0, 1]$, with the same weights and parameters faces a translation in its input distribution to the region $[2, 3] \times [2, 3]$. We expect that the TASOM network adapts to the new input distribution and gradually moves its neurons toward the new input distribution. We also expect that the TASOM network converges after a while and reaches almost the same level of quantization error. We use the average of all weight differences (awd) from one iteration to another as a convergence criterion. When the awd moves toward zero, we are sure that the network weights are converging. We also use the quantization error, which is calculated for the current input distribution, as an adaptation criterion. When the quantization error moves toward its minimum, it is concluded that the network weights are approximating their current en-

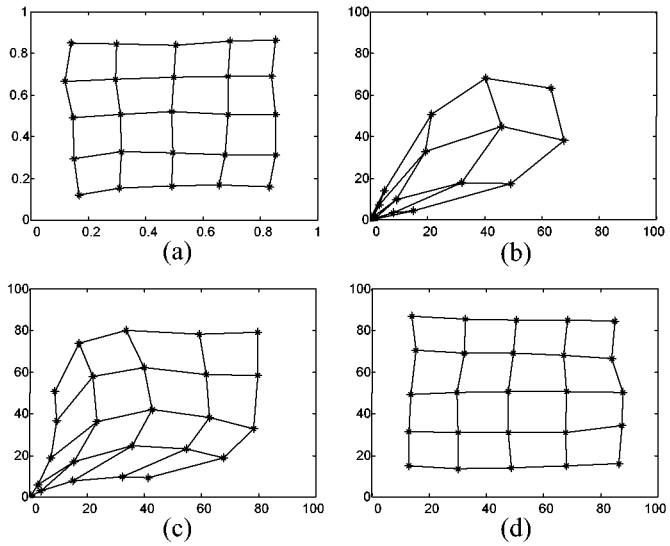

Fig. 1. (a) Converged weights of a TASOM trained by a uniform distribution. (b), (c), and (d) The weights of TASOM of (a) trained by 200, 1000, and 5000 samples of a new distribution, respectively, without reinitialization.

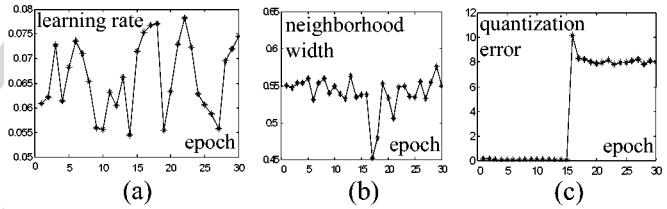

Fig. 2. (a) and (b) Variations of the learning rate and neighborhood width of neuron 1 versus epochs, respectively. Each epoch consists of 1000 samples. (c) Quantization error of the TASOM network versus epochs. In all of these figures, the first 15 epochs are for the TASOM of Fig. 1(a), and the next 15 epochs are for the TASOM of Fig. 1(b)–(d).

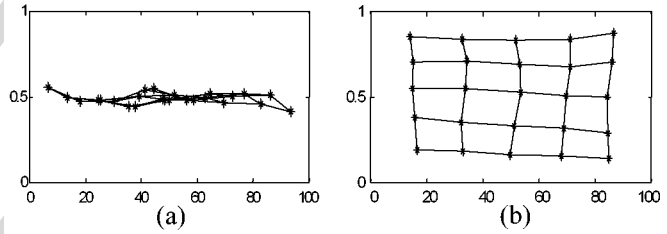

Fig. 3. (a) Converged TASOM using the uniform scaling vector, for a nonsymmetric distribution. The quantization error is 1.4940. (b) TASOM network using the nonuniform scaling vector trained by a symmetric distribution, faces a nonsymmetric distribution, without reinitialization. The quantization error is 4.2433.

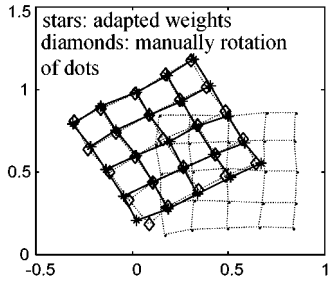

Fig. 4. TASOM of Fig. 1(a) faces a rotation by 30°° around the origin. The adapted weights are shown by asterisks.

vironment. Thus adaptation to the current distribution is being done successfully.

The TASOM weights before translation, and the converged weights after translation are shown by asterisks in Fig. 5(a). The intermediary weights trained by 500 samples of the translated distribution are also shown by dots in Fig. 5(a). The changes of the awd and the quantization error as the convergence and adaptation criteria for the TASOM network, respectively, are shown in Fig. 5(b). At the time of environmental change, adaptation speed increases to follow the moved distribution. During this period of fast adaptation, the slope of the convergence curve is high and downward. This means that the quantization error is being lowered fast. After this period of fast adaptation, adaptation and convergence curves remain almost flat. This, in fact, implies that the weights become stable and convergent.

B. The Effects of Parameters s_f and s_g on the TASOM Algorithm

The parameter s_f controls the tradeoff between generalization and memorization. When s_f is low, the TASOM weights are specialized to approximate the most recent data and forget the previous information with higher speed. On the contrary, when s_f is high, forgetting the previous information is done more slowly, and thus generalization ability of the TASOM is increased. The TASOM is tested for the input distribution in the region [0, 1] $\times$ [0, 1] for three different values of s_f, while keeping the other parameter constant at $s_g = 5$. The weights of the three TASOM networks are overlapped and shown in Fig. 6(a), where the asterisks, circles, and squares represent the TASOM weights for $s_f = 0.1, s_f = 1$, and $s_f = 10$, respectively. The weights with $s_f = 0.1$ were specialized to the most recent data. As s_f increased, the weights better filled the whole area of the distribution, confirming the relation between s_f and the forgetting speed of TASOM.

The parameter s_g controls the compactness and topological ordering of TASOM weights. Three TASOM networks are tested for the input distribution [0, 1] $\times$ [0, 1] for three different s_g values, while the other parameter remains constant at $s_f = 20$. The asterisks, circles, and squares, shown in Fig. 6(b), represent the TASOM weights for $s_g = 0.1, s_g = 1$, and $s_g = 10$, respectively. The weights with $s_g = 0.1$ remained close to each other and are placed around the center of the distribution, and the angles of the links between each two neurons are almost the same. In summary, the topological ordering is at a high level and the weights cluster around the center of the distribution. As s_g increased, the weights became looser, and better approximated the input distribution. However, the link angles are not equal to each other. In these cases, ordering is reduced for achieving lower quantization errors.

In some applications, like the one implemented in Section IV-E, low values of s_g should be used.

C. Time Complexity of the TASOM Algorithm

The time complexity of the introduced TASOM algorithm is discussed in this section. The TASOM algorithm consists of eight steps. The last step is simply a jump statement and is ignored here. The first step is the initialization step and is executed only once throughout the lifetime of a TASOM network.

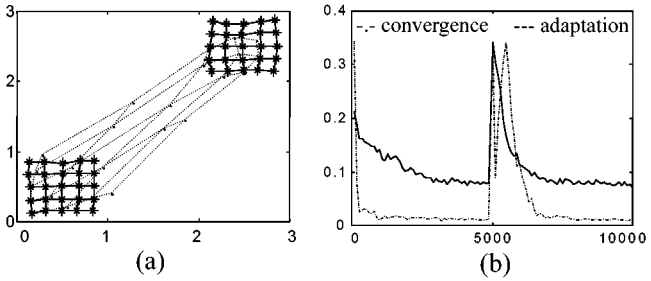

Fig. 5. (a) TASOM weights before translation and the converged weights after translation shown by asterisks. The intermediary weights are shown by dots. (b) Changes of convergence and adaptation of the TASOM of Fig. 5(a) versus the number of iterations, respectively.

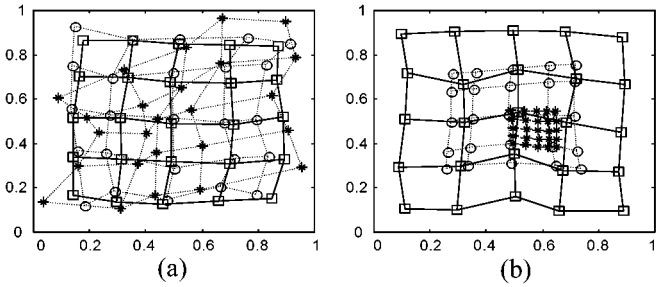

Fig. 6. (a) The asterisks, circles, and squares represent the TASOM weights for $s_f = 0.1, s_f = 1$, and $s_f = 10$, respectively; and $s_g = 5$ for the three cases. (b) The asterisks, circles, and squares represent the TASOM weights for $s_g = 0.1, s_g = 1$, and $s_g = 10$, respectively; and $s_f = 20$ for the three cases.

This step is also ignored here. In fact, Steps 2–7 of the algorithm are executed for each iteration. It can be shown that each iteration of the TASOM algorithm takes approximately $o(4N + 9)$ units of time. Consequently, for n iterations, the algorithm takes $o(n(4N + 9))$ time units to complete.

A similar analysis shows that the time complexity of the basic SOM is $o(n(3N + 3))$ units of time. For large N, we may conclude

$$o(\text{TASOM}) = (4/3)o(\text{SOM}). \tag{21}$$

The basic SOM and the TASOM algorithms are trained by a 2-D uniform distribution. For each 100 samples, the required time to complete the training with the two algorithms is measured. A linear regression analysis on the measurements shows that

$$o(\text{TASOM}) = 1.32o(\text{SOM}) + 0.12. \tag{22}$$

Comparing (21) with (22) confirms the correctness of the analytical study on the time complexities of the basic SOM and TASOM algorithms.

It has been shown that [28], although each iteration of the TASOM algorithm takes longer than the basic SOM algorithm, the convergence speed of the TASOM compensates for it, such that TASOM reaches an accepted level of quantization error faster than the basic SOM.

D. Bilevel Thresholding by TASOM

In this subsection, the TASOM_BTA algorithm, introduced in Section III-A, is tested for bilevel thresholding. Three grey level test images: the synthetic, the text, and the eagle images shown

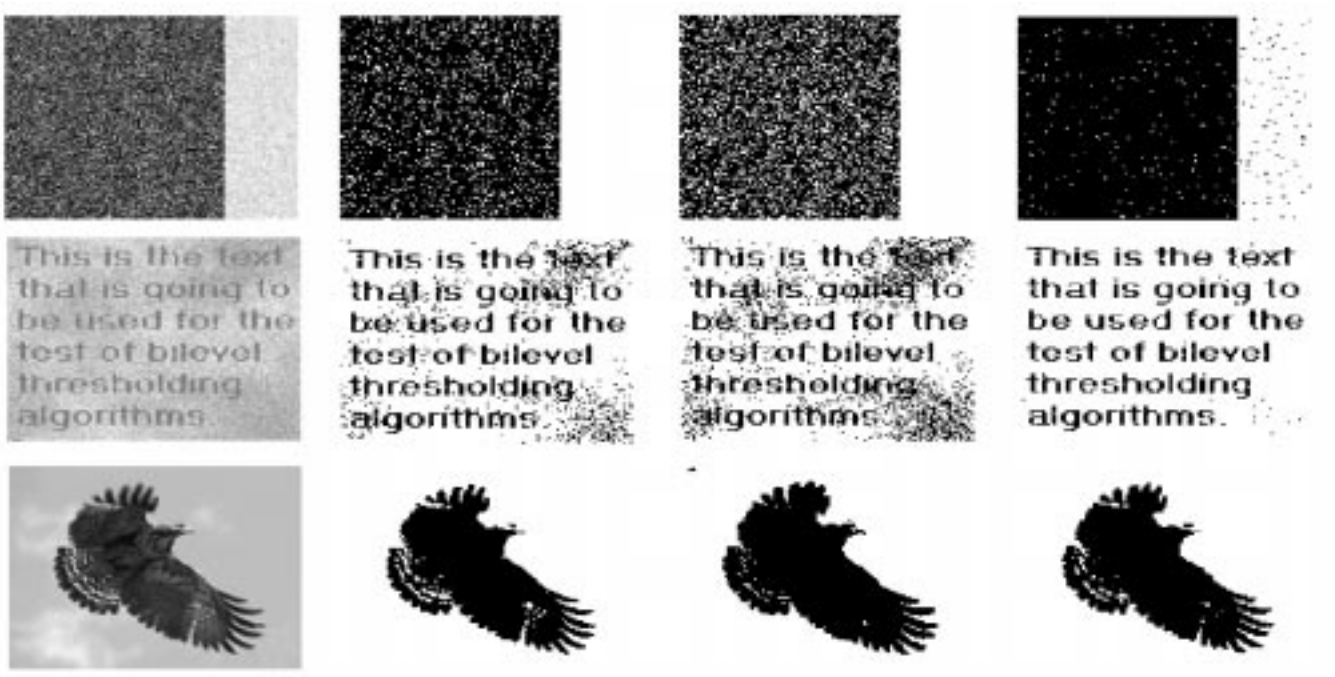

Fig. 7. Three test images: the synthetic, the text, and the eagle images are shown in the left column, from top to bottom, respectively. From the second to the last columns, the thresholded images obtained by the iterative threshold selection, the Kapur, and the TASOM_BTA methods are shown.

in the first column of Fig. 7, are employed for this purpose. For comparison, the iterative threshold selection [29] and the Kapur [30] methods are also used to threshold the images. The thresholded images obtained by the iterative threshold selection, Kapur, and TASOM_BTA methods are shown in the second to the last columns of Fig. 7. The thresholded images produced by the proposed TASOM_BTA contain more useful information than the thresholded images obtained by the other methods.

The histograms of the test images are shown in Fig. 8(a)–(c). They are composed of two completely different distributions with different variances. We expect that the number of times that a neuron of the TASOM_BTA wins on each epoch versus its weight value approximate the peaks and valleys of the histogram of the image. For the test images, the winning frequencies of the neurons of the TASOM_BTA networks versus their corresponding weight values are shown in Fig. 8(d)–(f). The weights of TASOM_BTA networks elegantly approximate the histograms of the test images. That is why the performance of the TASOM_BTA is higher than the other methods. The virtual valley neurons for the first two test images are the real valley neurons, and are shown by squares in Fig. 8(d) and (e). For the third test image, the virtual valley neuron represented by a square, is just beside the real valley neuron, represented by a diamond, as shown in Fig. 8(f).

For the synthetic test image of Fig. 7, the correct answer is known. Therefore, the misclassification error of each bilevel thresholding algorithm can be measured. To calculate this error, we simply count the number of misclassified pixels in the thresholded image and divide it by the total number of pixels. This error for the iterative threshold selection, the Kapur, and TASOM_BTA methods are 0.1052, 0.1957, and 0.0252, respectively. The error has its lowest value for TASOM_BTA. This fact confirms the superiority of TASOM_BTA for bilevel thresholding.

The estimated threshold values are rounded to their nearest integer values, and are shown in Table I. Comparing the thresholds obtained by TASOM_BTA and the histograms of the test images in Fig. 8 reveals that the thresholds are on the valley points of the histograms.

E. Tracking Moving Objects by TASOM

The TASOM_MOTION algorithm introduced in Section III-B is used here to track several airplanes that appear and move in an image sequence. As mentioned in Section IV-B, s_g should have a low value in comparison to s_f, since we need to keep the weights of each TASOM close together around the object center with low probability of self-intersection in the lattice. Here, we choose $s_g = 10$ and $s_f = 100$.

A reference background image of the sky is used to differentiate it with every frame of the image sequence. For each frame, the difference image represents the pixels of moving objects in that frame [26]. These pixels are used to train the TASOM networks of the TASOM_MOTION algorithm. The image sequence and the weights of the TASOM networks are overlapped and shown in Fig. 9(a). The weights of TASOM's, marked with plus signs, correctly follow the flying airplanes. Some other spurious moving objects are also detected in the image sequence due to changes of background in comparison to the reference image. For getting a better performance and for long image sequences, the reference image should be updated with time.

We have used four neurons for tracking the moving objects instead of one neuron. By this choice, the TASOM network of any moving object forms a closed contour which can be used

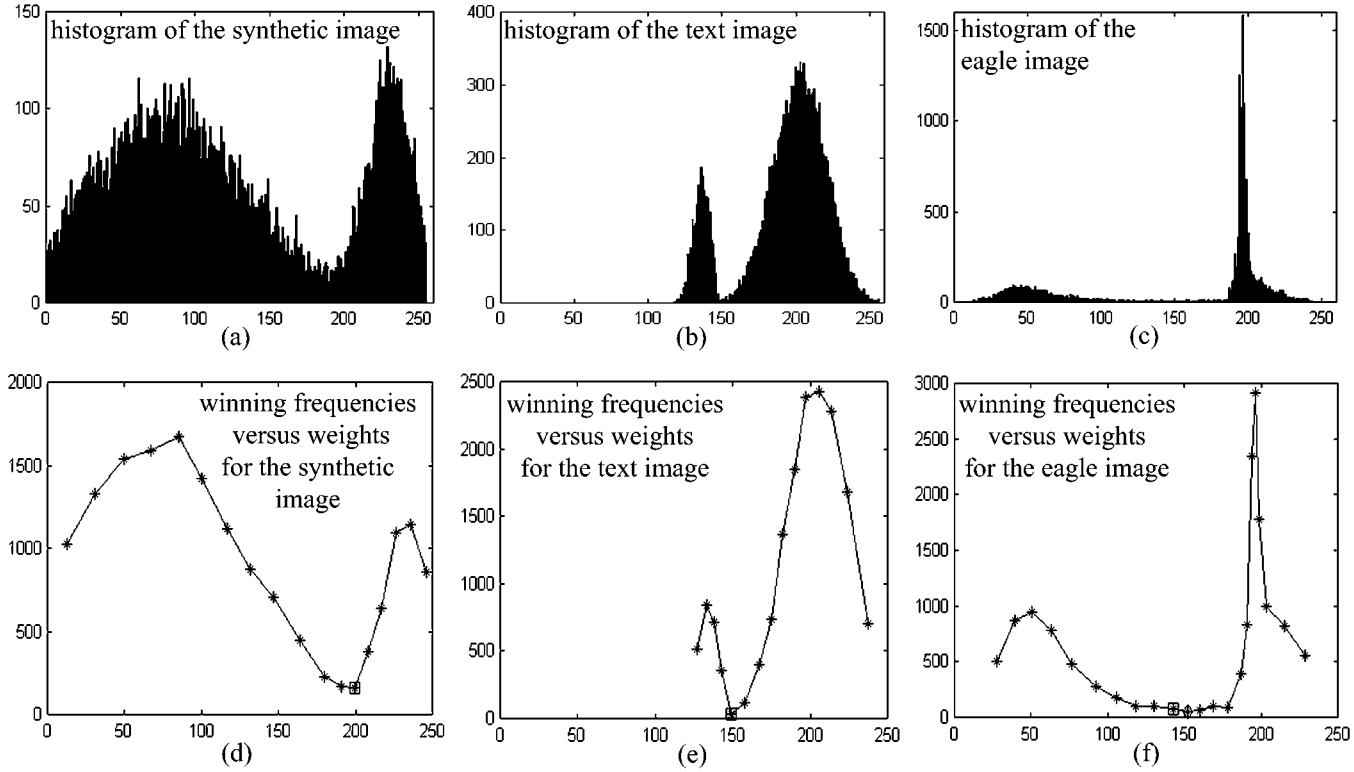

Fig. 8. Comparison between the real and approximated histograms obtained by the proposed TASOM_BTA method. (a), (b), and (c) The histograms of the test images of Fig. 7. (d), (e), and (f) The winning frequencies of the neurons of the TASOM_BTA networks versus their corresponding weight values for the test images.

TABLE I
THE THRESHOLDS OBTAINED BY THE IMPLEMENTED METHODS FOR THE TEST IMAGES OF FIG. 7

Threshold	Iterative Threshold	Kapur	TASOM_BTA
Synthetic image	143	118	199
Text image	173	181	149
Eagle image	126	186	152

as the initial contour in any active contour modeling algorithm [31].

The TASOM_CONTOUR algorithm suggested in Section III-B is used for boundary tracking. One frame of the image sequence of Fig. 9(a) is used for testing this algorithm. The weights of each TASOM network in the TASOM_MOTION algorithm is used as the initial contour for the TASOM_CONTOUR with parameters $\vartheta_l = 0.2$ and $\vartheta_h = 3$. The results are shown in Fig. 9(b), where the approximated contours are superimposed on the airplanes. Since the difference image is used as feature points, the contours do not completely cover the boundaries of the airplanes.

The second test with the TASOM_CONTOUR algorithm is conducted for tracking the boundary of a heart image at different stages of beating in an image sequence. The initial contour containing nine points is shown in Fig. 10(a). This contour is used for training a TASOM network using the TASOM_CONTOUR algorithm with parameters $\vartheta_l = 0.5$ and $\vartheta_h = 5$. The feature points are the edge points of the heart image. The resulted contour is superimposed on the image of the heart in Fig. 10(a), where the neurons are shown by asterisks. The heart boundary is correctly approximated by the weight vectors of the TASOM network.

Another frame of the heart image is used to test the algorithm. In this case, the previous values of the weight vectors form the initial contour for this frame of the heart, as shown by thin lines in Fig. 10(b). The approximated contour using the TASOM_CONTOUR for the current frame is shown by asterisks in Fig. 10(b). Again, the weight values completely cover the boundary of the heart.

F. Adaptive Clustering by TASOM

The TASOM_CLUSTERING algorithm explained in Section III-C is used here for adaptive clustering. For the tests in this section, we use 2-D Gaussian distributions, where each one is composed of two independent and identically distributed (iid) 1-D Gaussian random variables with the variances $v = 1$. The essential parameters of the TASOM_CLUSTERING here are $\vartheta_l = 0.1, \vartheta_h = 1$, and $s_f = s_g = 50$.

For the first test, a 2-D Gaussian distribution with mean $m = (0, 0)$ is simulated. The TASOM_CLUSTERING algorithm is trained with this distribution. The converged weights along with some samples of the distribution are shown in Fig. 11(a). The weight vectors, represented by asterisks, clearly cover the Gaussian distribution.

Now, the input environment suddenly changes such that two new 2-D Gaussian clusters with centers (20, 0) and (20, 20) appear in the environment. The TASOM_CLUSTERING network

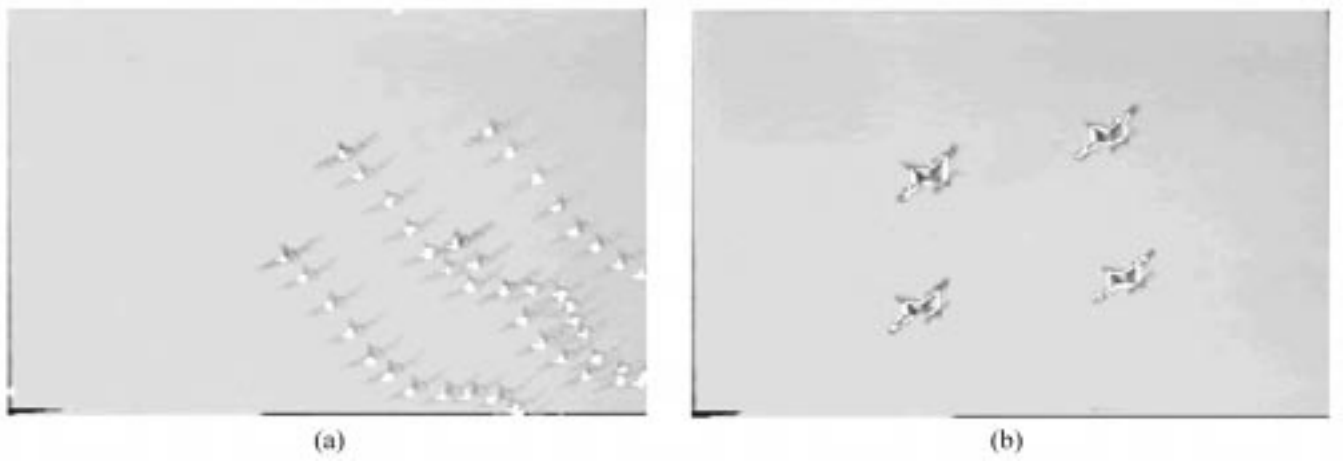

Fig. 9. (a) Image sequence of four flying airplanes, and the weights of the TASOM networks obtained by the TASOM_MOTION algorithm are overlapped and shown. The weights of TASOM networks, shown by plus signs, correctly follow the airplanes. (b) The approximated contours obtained by the TASOM_CONTOUR algorithm are superimposed on the airplanes.

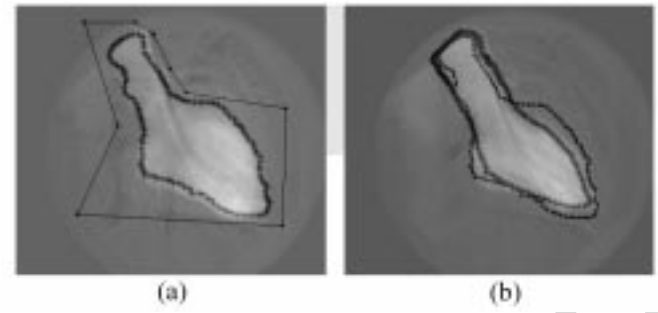

Fig. 10. (a) Initial contour containing nine points is used by the TASOM_CONTOUR algorithm resulting in the converged weights, shown by asterisks. (b) Tracking the heart boundary for the next frame without manual intervention, where the converged weights of the previous frame and those of the current frame are shown by dots and asterisks, respectively.

automatically and without manual intervention tries to adapt itself to the new conditions. The first few epochs are shown in Fig. 11(b). At the first epoch, some available neurons, which are the closest ones to the two new clusters, migrate toward the new clusters. Gradually, the neurons are added around the centers of the new clusters, and the approximation of the new environment is improved. The converged weights before Step 3 of the algorithm along with some samples of the input vectors are shown in Fig. 11(c). The unused neurons in Fig. 11(c) are shown by asterisks and other neurons are shown by dots. The interesting point is that between each two connected clusters, there is one or a few unused neurons, and there is no unused neuron inside the clusters. Therefore, with a simple algorithm, we can separate the clusters from each other.

Following the above change, the environment of Fig. 11(c) experiences another abrupt change. This time, the clusters move in different directions. The bottom two clusters move toward each other to centers $(10, -5)$ and $(10, 0)$, while the top cluster moves to the right to the center $(30, 15)$. The TASOM_CLUSTERING network of the previous test with the previous parameters and weight vectors automatically tries to follow the moved clusters. The neurons migrate toward the moved cluster centers. The converged weight vectors before Step 3 of the algorithm along with some samples of the clusters

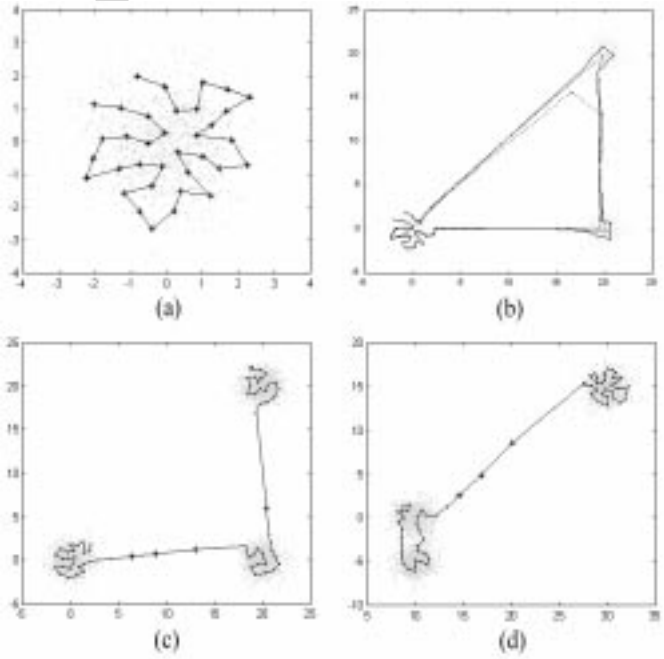

Fig. 11. (a) TASOM weights obtained by the TASOM_CLUSTERING algorithm. Some samples of the Gaussian distribution are also shown. (b) Two new clusters suddenly appear in the input distribution, and the weights are shown for the early stages of learning. (c) The converged weights of TASOM of Fig. 13(b). (d) The converged weights of TASOM of Fig. 13(c) after the clusters move in different directions.

are shown in Fig. 11(d). The weight vectors gather around the new cluster centers, and leave the old cluster centers as predicted.

The bottom two clusters are close together and can be separated as one cluster. No unused neuron is present inside the two close clusters, confirming that they should be identified as one cluster. Between the top cluster and the bottom ones, there are some unused neurons, which indicate that the top cluster should be identified as one separate cluster from the bottom ones. Therefore, the concept of unused neurons is useful in some applications, where the clusters should be separated from each other.

V. Conclusion

The TASOMs and its modified versions are efficient unsupervised neural networks for operation in stationary and nonstationary environments. This property comes from the fact that the TASOM algorithms use adaptive learning rates and neighborhood sizes in contrast to the basic SOM which uses only time-decreasing learning rate and neighborhood size. The theoretical discussion in this paper shows that the adaptation of the TASOM network to its changing environment is fulfilled with the least possible increase in the learning parameters (according to a measure) so that minimum instability is inflicted on the network. Moreover, each neuron in TASOM has its own learning rate and neighborhood size, which makes it more flexible to varied environments. A scaling vector is also used in the TASOM to enable it to work well in the presence of a scaling transformation on the input distribution. A nonuniform scaling vector is used when topological ordering is of higher priority than quantization error.

New versions of TASOM networks were introduced and used for bilevel thresholding of grey level images, tracking centers or boundaries of moving objects, and adaptive clustering. What makes TASOM interesting is that once the TASOM-based networks are trained in a specific environment, they are able to follow changes of the environment without reinitializing the weight vectors or learning parameters, and without any outside intervention.

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, pp. 59–69, 1982.

[2] T. D. Chiuch, T. T. Tang, and L. G. Chen, "Vector quantization using tree-structured self-organizing feature maps," in *Applications of Neural Networks to Telecommunications*, J. Alspector, R. Goodman, and T. X. Brown, Eds. Mahwah, NJ: Lawrence Erlbaum Assoc., 1993, pp. 259–265.

[3] E. Oja, "Self-organizing maps and computer vision," in *Neural Networks for Perception*, H. Wechsler, Ed. New York: Academic, 1992, vol. 1, pp. 368–385.

[4] C. Ambroise *et al.*, "Hierarchical clustering of self-organizing map for cloud classification," *Neurocomputing*, vol. 30, pp. 47–52, 2000.

[5] T. Kohonen, "The neural phonetic typewriter," *Computer*, vol. 21, pp. 11–22, 1988.

[6] C. Amerijckx *et al.*, "Image compression by self-organized Kohonen map," *IEEE Trans. Neural Networks*, vol. 9, pp. 503–507, June 1998.

[7] J. C. Principe, L. Wang, and M. A. Motter, "Local dynamic modeling with self-organizing maps and applications to nonlinear system identification and control," *Proc. IEEE*, vol. 86, pp. 2240–2258, Nov. 2000.

[8] T. Heskes and B. Kappen, "Neural networks learning in a changing environment," in *Proc. Int. Conf. Neural Networks*, 1991, pp. 823–828.

[9] H. Shah-Hosseini and R. Safabakhsh, "A learning rule modification in the self-organizing feature map algorithm," in *Proc. 4th Int. CSI Computer Conf.*, Tehran, Iran, 1999, pp. 1–9.

[10] ——, "TASOM: The time adaptive self-organizing map," *Proc. IEEE Int. Conf. Information Technology: Coding and Computing*, pp. 422–427, 2000.

[11] ——, "The time adaptive self-organizing map with neighborhood functions for bilevel thresholding," in *Proc. AIS'2000 Conf.*, Tucson, AZ, pp. 123–128.

[12] ——, "Pattern classification by the time adaptive self-organizing map," *Proc. IEEE ICECS*, pp. 495–498, 2000.

[13] ——, "TAPCA: Time adaptive self-organizing maps for adaptive principal components analysis," *Proc. IEEE Int. Conf. Image Processing*, pp. 509–512, 2001.

[14] ——, "Automatic multilevel thresholding for image segmentation by the growing time adaptive self-organizing map," IEEE Trans. Pattern Anal. Machine Intell.. published.

[15] ——, "Automatic adjustment of learning rates of the self-organizing feature map," *Scientia Iranica*, vol. 8, no. 4, pp. 277–286, 2001.

[16] J. A. Kangas, T. Kohonen, and J. T. Laaksonen, "Variants of self-organizing maps," *IEEE Trans. Neural Networks*, vol. 1, pp. 93–99, Feb. 1990.

[17] T. Trautmann and T. Denoeux, "Comparison of dynamic feature map models for environmental monitoring," *Proc. IEEE Int. Conf. Neural Networks*, pp. 73–78, 1995.

[18] J. Blackmore and R. Miikulainen, "Incremental grid growing: Encoding high-dimensional structures to a two-dimensional feature map," *Proc. IEEE Int. Conf. Neural Networks*, pp. 450–455, 1993.

[19] B. Fritzke, "Growing cell structures—A self-organizing network for unsupervised and supervised learning," *Neural Netw.*, vol. 7, pp. 1441–1460, 1994.

[20] T. Villmann and H.-U. Baur, "Applications of the growing self-organizing map," *Neurocomputing*, vol. 21, pp. 91–100, 1998.

[21] H. R. Schiffman, *Sensation and Perception*. New York: Wiley, 1996.

[22] S. M. Smith and J. M. Brady, "ASSET-2: Real-time motion segmentation and shape tracking," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 814–820, Aug. 1995.

[23] A. Kumar, Y. Bar-Shalom, and E. Oron, "Precision tracking based on segmentation with optimal layering for imaging sensors," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, no. Feb., pp. 182–188, 1995.

[24] A. G. Bors and I. Pitas, "Optical flow estimation and moving object segmentation based on median radial basis function network," *IEEE Trans. Image Processing*, vol. 7, pp. 693–702, May 1998.

[25] J. Heikkonen, P. Koikkalainen, and C. Schnorr, "Learning motion trajectories via self-organization," in *Proc. Int. Conf. Pattern Recognition*, 1994, pp. 554–556.

[26] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1993.

[27] Y. V. Venkatesh and N. Rishikesh, "Self-organizing neural networks based on spatial isomorphism for active contour modeling," *Pattern Recognit.*, vol. 33, no. 7, pp. 1239–1250, 2000.

[28] H. Shah-Hosseini and R. Safabakhsh, "The time adaptive self-organizing map for distribution estimation," *Int. J. Eng.*, vol. 15, no. 1, pp. 23–34, 2002.

[29] T. W. Ridler and S. Calvard, "Picture thresholding using an iterative selection method," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-8, pp. 630–632, Aug. 1978.

[30] J. N. Kapur, P. K. Sahoo, and A. K. Wong, "A new method for gray-level picture thresholding using the entropy of the histogram," *Comput, Vis. Graph. Image Process.*, vol. 29, pp. 273–285, 1985.

[31] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *Int. J. Comput. Vis.*, vol. 1, pp. 321–331, 1988.

Hamed Shah-Hosseini was born in Tehran, Iran, in 1970. He received the B.S. degree in computer engineering (with high honors) from the University of Tehran, and the M.S. degree in computer engineering (with high honors) from Amirkabir University of Technology, Tehran, in 1993 and 1996, respectively. He is currently pursuing the Ph.D. degree in computer engineering at Amirkabir University of Technology. His research interests include computer vision and neural networks.

Reza Safabakhsh (S'79–M'87) was born in Isfahan, Iran, in 1953. He received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1976, and the M.S. and Ph.D. degrees in electrical engineering from the University of Tennessee, Knoxville, in 1979 and 1986, respectively.

He was with Walters State College from 1983 to 1986, and with the Center of Excellence in Information Systems in Nashville, TN from 1986 to 1988. Since 1988, he has been with the Department of Computer Engineering, Amirkabir University of Technology, Tehran. His research interests currently include self-organizing maps, multitarget tracking, content-based image retrieval, watermarking, and cooperation and learning in multiagent systems.

Dr. Safabakhsh is a Member of several honor societies including Phi Kappa Phi and Eta Kappa Nu. He is the Founder and a Member of the Board of Executives of the Computer Society of Iran, and was the President of this society for the first four years.