Improving the Self-Organizing Feature Map Algorithm Using an Efficient Initialization Scheme

Mu-Chun Su[1], Ta-Kang Liu[2] and Hsiao-Te Chang[2]

[1]Department of Computer Science and Information Engineering
National Central University
Chung Li, Taiwan 320, R.O.C.
E-mail: muchun@csie.ncu.edu.tw
[2]Department of Electrical Engineering
Tamkang University
Tamsui, Taipei, Taiwan 251, R.O.C.

Abstract

It is often reported in the technique literature that the success of the self-organizing feature map formation is critically dependent on the initial weights and the selection of main parameters (*i.e.* the learning-rate parameter and the neighborhood set) of the algorithm. They usually have to be counteracted by the trial-and-error method; therefore, often time consuming retraining procedures have to precede before a neighborhood preserving feature amp is obtained. In this paper, we propose an efficient initialization scheme to construct an initial map. We then use the self-organizing feature map algorithm to make small subsequent adjustments so as to improve the accuracy of the initial map. Several data sets are tested to illustrate the performance of the proposed method.

***Key Words*:** Neural Networks, Self-organizing Feature Map, Unsupervised Learning, Kohonen Algorithm

1. Introduction

Recently, numerous technical reports have been written about successful applications of the self-organizing feature map algorithm developed by Kohonen [8]. These applications widely range from simulations used for the purpose of understanding and modeling of computational maps in the brain to subsystems for engineering applications such as cluster analysis, motor control, speech recognition, vector quantization, and adaptive equalization. Kohonen *et al* [12] provided partial reviews of these applications. Despite its successes in practical applications, SOMs suffer from some major deficiencies [1,9]. For example, the success of map formation is critically dependent on the initial weights and the selection of the main parameters of the algorithm, namely, the learning-rate parameter and the neighborhood set [20]. Unfortunately, a process of trial and error usually determines them. Often one realizes only at the end of a simulation that usually requires a huge amount of iterations that different selections of the parameters or initial weights would have been more appropriate. In addition, another problem associated with Kohonen self-organizing feature map (SOM) algorithm is that it tends to overrepresent regions of low input density and underrepresent regions of high input density [20]. The accuracy of the map also depends on the number of iterations of the SOM algorithm. A rule of thumb is that, for good statistical accuracy, the number of iterations should be at least 500 times large than the number of neurons [11]. A more serious problem with the SOM algorithm is that topology preserving mapping is not guaranteed even if a huge amount of iterations are used [14].

Several different approaches have been proposed to improve the conventional SOM algorithm. Some researchers used genetic algorithms to form feature maps [3,4,15,18]. Lo and Bavarian addressed the effect of neighborhood function selection on the rate of convergence of the SOM algorithm [13]. Kiang *et al* developed a "circular" training algorithm that tries to overcome some of the ineffective topological representations caused by the "boundary" effect [6]. Fritzke proposed a new self-organizing neural network model that can determine shape as well as size of the network during the simulation in an incremental fashion [2]. Jun et al proposed a self-organizing feature map learning algorithm based on incremental ordering [5]. A multilayer self-organizing feature map was proposed in [7]. Samad and Harp showed that how the Kohonen self-organizing feature map model can be extended so that partial training data can be utilized [17]. Hulle and Leuven introduced the Maximum Entropy learning rule (MER) to achieve a globally ordered map by performing local weight updates only. Hence, contrary to Kohonen's SOM algorithm, no neighborhood function is needed [21]. Extensive and good overview of some improvements can be found in the literature [10].

In this paper, we propose an efficient initialization scheme for the SOM algorithm to accelerate the learning phase of forming a topologically ordered feature map. This paper is organized into 4 sections. In the following section the initialization scheme is discussed. In Section 3 several data sets are utilized to demonstrate the effectiveness of the scheme. Finally, Section 4 concludes the paper.

2. Initialization Scheme

2.1 Backgrounds

The SOM algorithm proposed by Kohonen can be summarized as follows:

Step 1: Initialization: Choose random values for the initial weights $\underline{w}_j(0)$.

Step 2: Winner Finding: Find the winning neuron j^* at time *k*, using the minimum-distance Euclidean criterion:

$$j^* = \arg\min_j \left\| \underline{x}(k) - \underline{w}_j \right\|, j = 1, \cdots, M \times N. \quad (1)$$

where $\underline{x}(k) = [x_1(k), \cdots, x_n(k)]^T$ represents the k^{th} input pattern, $M \times N$ is the total number of neurons, and $\|\cdot\|$ indicates the Euclidean norm.

Step 3: Weights Updating: Adjust the weights of the winner and its neighbors, using the following rule.

$$\underline{w}_j(k+1) = \underline{w}_j(k) + \eta(k) N_{j^*}(k)(\underline{x}(k) - \underline{w}_j(k)) \quad (2)$$

where $\eta(k)$ is a positive constant and $N_{j^*}(k)$ is the topological neighborhood function of the winner neuron j^* at time *k*. It should be emphasized that the success of the map formation is critically dependent on how the values of the main parameters (*i.e.* $\eta(k)$ and $N_{j^*}(k)$), initial values of weight vectors, and the number of iterations are prespecified.

As we know that the initialization strongly affects the ultimate map, however, the weights of the neural array to be trained are typically initialized at small random values. To counteract the initialization problem, a conventional approach is to restart the training procedure with other random weights. Then another run of the SOM algorithm has to be completed. The price paid for this simple trial-and-error method is we have to waste substantial computational resources since a large number of iterations are usually needed for the SOM algorithm to solve the problem. In addition to the random initialization method, there are other initialization methods. One simple method is to pick the initial weight vectors, $\underline{w}_j(0)$, from the available input patterns $\underline{x}_i$. A more effective means to accelerate the learning phase is to define the initial weight vectors properly. In [10], the so-called "linear initialization" method was presented. The method first determines the two eigenvectors of the autocorrelation matrix of input vectors that have the largest eigenvalues, and then to let these eigenvectors span a two-dimensional linear subspace. A rectangular array is then defined along this subspace, its center coinciding with that of the mean of the input vectors, and the same dimensions being the same as the two largest eigenvalues. The initial values of $\underline{w}_j(0)$ are then identified with the array points. Since the initial weight vectors, $\underline{w}_j(0)$, are now already ordered and their density distribution roughly approximates the density distribution of the input vectors, it will be possible to directly start the learning with the convergence phase. One critical problem associated with the linear initialization method is that it requires a lot of computations to compute the eigenvectors of an autocorrelation matrix.

In our previous work [19], we proposed an efficient approach to forming feature maps. The method involves three stages. In the first stage we use the K-means algorithm to select $M \times N$ (*i.e.* the size of the feature map to be formed) cluster centers from a data set. Then a heuristic assignment strategy is employed to organize the $M \times N$ selected data points into an $M \times N$ neural array so as to form an initial feature map. If the initial map is not good enough then it will be fine-tuned by the traditional Kohonen self-organizing feature map (SOM) algorithm under a fast cooling regime in the third stage. By our three-stage method a topologically ordered feature map would be formed very quickly instead of requiring a huge amount of iterations to fine-tune the weights toward the density distribution of the data points which usually happened in the conventional SOM algorithm. If the size of the array is not much greater than the number of input vectors then the complexity of the method is less than the conventional SOM algorithm.

2.2 Our Initialization Method

Here we propose a simple straightforward initialization scheme to solve the initialization problem. Consider Figure 1, which depicts a two-dimensional neural array of size $M \times N$. The basic ideal is to find a large enough hyperbox to cover all the training patterns and then to squeeze the hyperbox into a plane. The scheme is described as follows:

Step 1: Initialization of the neurons on the four corners:

We first select a pair of patterns whose interpattern distance is the largest one among the training set. The coordinates of the two patterns are used to initialize the weights of the neurons on the lower left corner and the upper right corner (*i.e.* $\underline{w}_{M,1}$ and $\underline{w}_{1,N}$), respectively. From the remaining training patterns, the coordinates of the pattern which is farthest to the two selected patterns is then used to initialize the weight vector of the neuron on the upper left corner (*i.e.* $\underline{w}_{1,1}$). The initial weight vector of the neuron on the lower right corner (*i.e.* $\underline{w}_{M,N}$) is set to be the coordinates of the pattern which is farthest to the previously selected three patterns (i.e. $\underline{w}_{1,1}$, $\underline{w}_{M,1}$, and $\underline{w}_{1,N}$). Figure 2 illustrates an example for a2-dimensional case. Note that the computational complexity will increase as the number of input patterns increase since this step involves the computation of $\frac{I(I+1)}{2}$ distances (here we suppose there are total I input patterns). Besides, what if there are outliers in the input patterns? To overcome these two problems, a possible solution is first to use some vector quantization methods (e.g. the K-means algorithm) to sample the input data set and then precede this step.

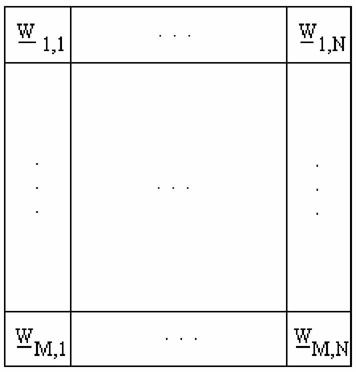


Figure 1. The arrangement of an $M \times N$ neural array

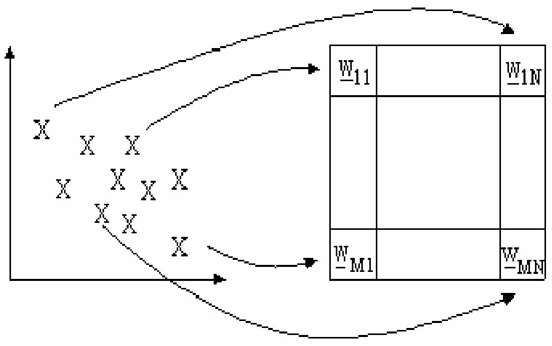


Figure 2. An example of the process occurred in step 1

Step 2: Initialization of the neurons on the four edges:

We initialize the weights of the neurons on the four edges according to the following equations:

$$\begin{aligned}\underline{w}_{1,j} &= \frac{\underline{w}_{1,N}-\underline{w}_{1,1}}{N-1}(j-1)+\underline{w}_{1,1} \qquad \text{for } j=2,\cdots,N-1 \\ &= \frac{j-1}{N-1}\underline{w}_{1,N}+\frac{N-j}{N-1}\underline{w}_{1,1}\end{aligned} \tag{3}$$

$$\begin{aligned}\underline{w}_{M,j} &= \frac{\underline{w}_{M,N}-\underline{w}_{M,1}}{N-1}(j-1)+\underline{w}_{M,1} \qquad \text{for } j=2,\cdots,N-1 \\ &= \frac{j-1}{N-1}\underline{w}_{M,N}+\frac{N-j}{N-1}\underline{w}_{M,1}\end{aligned} \tag{4}$$

$$\underline{w}_{i,1} = \frac{\underline{w}_{M,1} - \underline{w}_{1,1}}{M-1}(i-1) + \underline{w}_{1,1} \qquad \text{for } i = 2, \cdots, M-1$$
$$= \frac{i-1}{M-1}\underline{w}_{M,1} + \frac{M-i}{M-1}\underline{w}_{1,1} \tag{5}$$

$$\underline{w}_{i,N} = \frac{\underline{w}_{M,N} - \underline{w}_{1,N}}{M-1}(i-1) + \underline{w}_{1,N} \qquad \text{for } i = 2, \cdots, M-1$$
$$= \frac{i-1}{M-1}\underline{w}_{M,N} + \frac{M-i}{M-1}\underline{w}_{1,N} \tag{6}$$

The basic idea is very simple. Any two points can form a line in the input space. We then uniformly partition the line into N-1 or M-1 segments and use the coordinates of the ending points of the segments to initialize the weights of the neurons.

Step 3: Initialization of the remaining neurons:

We initialize the remaining neurons from top to bottom, and from left to right. If we change the order to be from left to right and from top to bottom, the initialization effect will be the same. It is very easy to prove this. The pseudo-code description of the initialization scheme for the remaining neurons is given as follows:

```
Begin
    For i from 2 to M-1
    Begin
        For j from 2 to N-1
        Begin
            $\underline{w}_{i,j} = \frac{\underline{w}_{i,N} - \underline{w}_{i,1}}{N-1}(j-1) + \underline{w}_{i,1} = \frac{j-1}{N-1}\underline{w}_{i,N} + \frac{N-j}{N-1}\underline{w}_{i,1}$   (7)
        End;
    End;
End;
```

One may ask why we do not directly partition the input space into hypercubes and then use the coordinates of the centers of the hypercubes to initialize the weights of the network. A direct result is that an initial map constructed by the direct method will tend to undersample high probability regions and oversample low probability ones. It is probable that we will need more iterations to refine the map if we start out from such an initial map instead of an initial map constructed by our proposed method. This can be illustrated by Figure 3. Moreover, this method will suffer from the curse of the dimensionality. Suppose we partition the *i*th input variable into S_i segments, the total number of the centers of the hypercubes will be $S_1 \times S_2 \times \cdots S_n$. The number increases exponentially as *n* increases. That is, the numbers of neurons will be very large for applications where the dimensionality of the input data is large. Another one important thing we want to point out is that compared to the "linear initialization" method presented in [10], our method is much simpler and requires mush less computations since our method does not compute the eigenvectors of the autocorrelation matrix of the input patterns.

Combining Eq. (3)-(7), one may easily find

$$\underline{w}_{i,j} = \frac{(j-1)(i-1)}{(N-1)(M-1)}\underline{w}_{M,N} + \frac{(j-1)(M-i)}{(N-1)(M-1)}\underline{w}_{1,N} + \frac{(N-j)(i-1)}{(N-1)(M-1)}\underline{w}_{M,1} + \frac{(N-j)(M-i)}{(N-1)(M-1)}\underline{w}_{1,1} \tag{8}$$

Therefore the three-step method can be downsized into a two-step method. The first step is the same. Then we use Eq.(8) to initialize the weights of the remaining neurons.

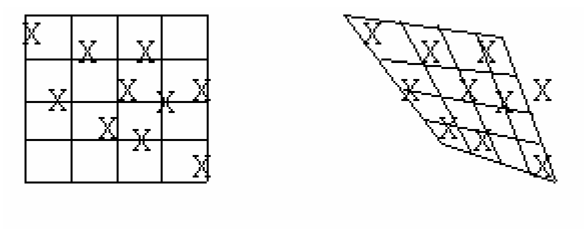


Figure 3. The difference between the grid initialization scheme and our initialization scheme: (a) the grid initialization scheme, (b) our initialization scheme

3. Simulation Results

Three data sets are used to illustrate the performance of our method. The first data set consists of 579 2D data points shown in Figure 4(a). The second data set is the well-known iris data set that consists of 150 4D data points. The third data set is a 10-D artificial data set consisting of 200 data points. The 2D configurations for the iris data set and the 10-D data set using Sammon's projection method are shown in Figure 4(b)-4(c). The patterns in Figure 4(b)-4(c) have been labeled by category to highlight the separation of the three categories. To illustrate the effectiveness of our method we use three different approaches to forming feature maps. The first method is to use the random initialization scheme. Method 2 adopts the scheme proposed in [19]. The proposed method is the third method. To further demonstrate the performance of our method two different cooling regimes were utilized. The values of the parameters were kept the same irrespective of the data used.

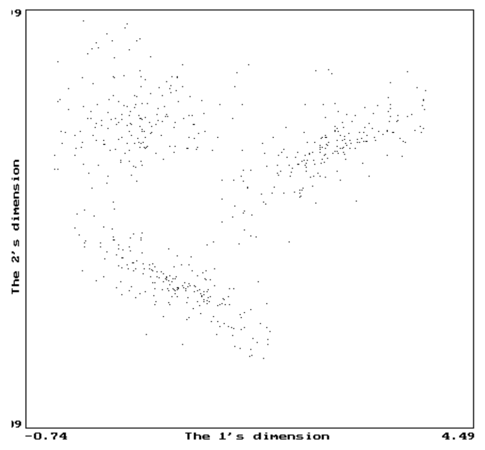
Regime 1: $N_{j^*}(k) = \exp(-\frac{d^2_{j^*,i}}{2\sigma^2(k)})$ and $\eta(k) = \max(\eta_0 (\frac{\eta_f}{\eta_0})^{\frac{k}{\tau_2}}, 0.01)$ where $\sigma(k) = \sigma_0 \exp(-\frac{k}{\tau_1})$ and $\eta(k) = \max(\eta_0 (\frac{\eta_f}{\eta_0})^{\frac{k}{\tau_2}}, 0.01)$, σ_0 = 1/3 of the edge length of the lattice (*i.e.* N), τ_1= 30, τ_2= 30, η_0=0.9, and η_f=0.01.

Regime 2: $N_{j^*}(k) = \exp(-(1+\frac{k}{\tau})d_{j,j^*})$ and $\eta(k) = \eta_0 \frac{1}{1+k}$ where η_0 =1.0, and τ = 100. In addition, we update only the winners' weight vectors for the last one-third epochs. The intention of this cooling regime is to quickly cool down the training procedure.

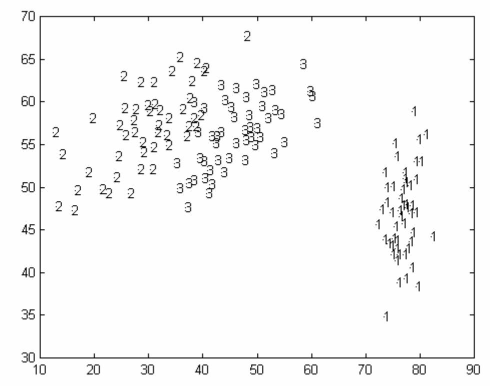
Note that the parameter k represents the number of epochs in our simulations.

3.1 Example 1: 2-D data set

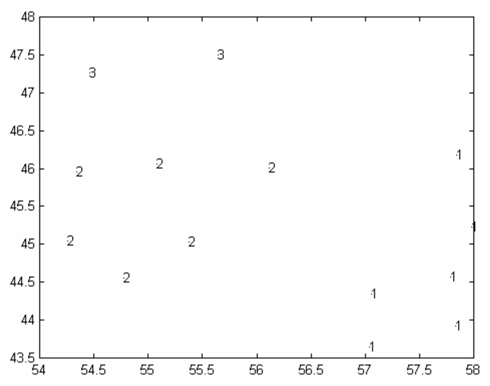
A 15x15 network is trained by the artificial data set shown in Figure 4(a). Figure 5-6 show the resultant maps constructed by the three different methods under regimes 1 and 2, respectively. The left column, the center column, and the right column of these two figures show the maps constructed by method 1, method 2, and method 3, respectively. By viewing Figure 5 we find the conventional SOM algorithm under a convenient cooling regime is insensible to initial weights. All three methods constructed topologically ordered maps. Note that although topologically ordered maps can be formed in the 10th epoch more updates were still required to fine-tune the weights toward the density distribution of the data points. On the contrary, Figure 6 tells us that the conventional SOM algorithm under a fast cooling regime is very sensible to initial weights and only method 2 and our method can form a good topologically ordered map. In the cases of method 2 and our method, the meshes remain untangled and quickly adapt in detail within 10 epochs. On the other hand, the mesh on the left columns of Figure 8 first tangled and then tried to unfold themselves. Unfortunately, even if we used 90 more epochs to continue the training process the incorrect topological ordering was not eliminated. In fact the mesh still tangled even we continued the training process for another 900 epochs.



(a)

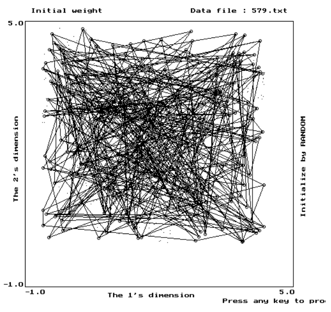


(b)

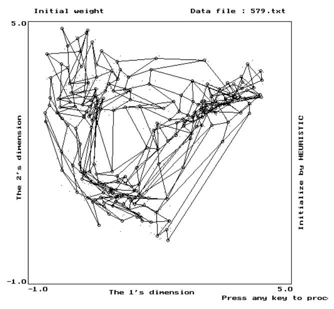


(c)

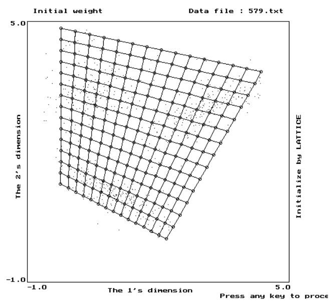
Figure 4. The three data sets used in the simulations: (a) the 2-D data set consisting of 579 data points, (b) the iris data set consisting of 150 4-D data points, (c) the animal data set consisting of 16 13-D data points



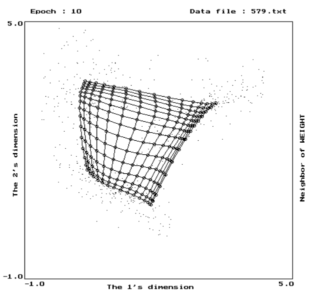
(a) the initial map



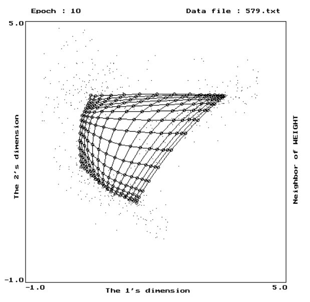
(e) the initial map



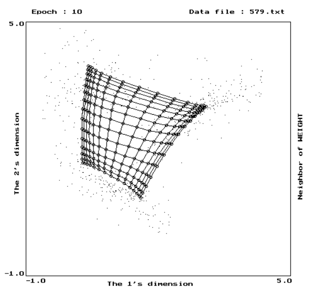
(i) the initial map



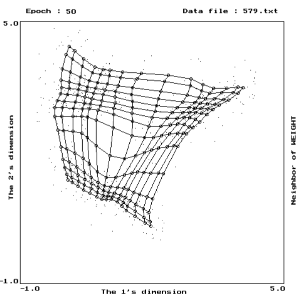
(b) after 10 epochs



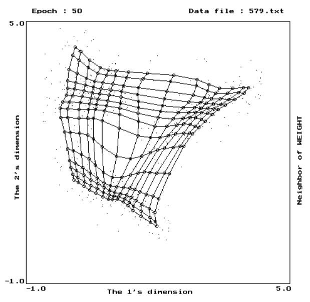
(f) after 10 epochs



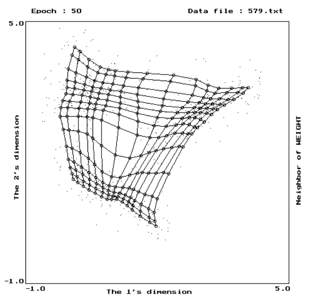
(j) after 10 epochs



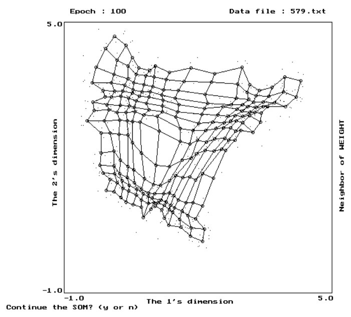
(c) after 50 epochs



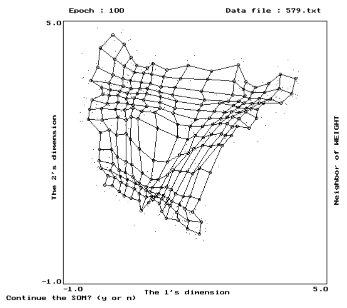
(g) after 50 epochs



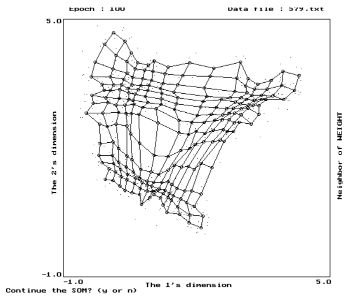
(k) after 50 epochs



(d) after 100 epochs

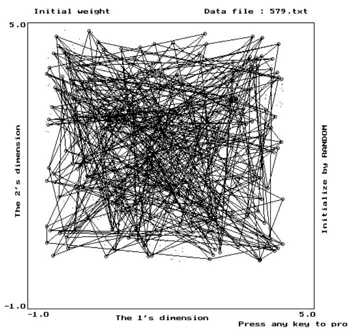


(h) after 100 epochs

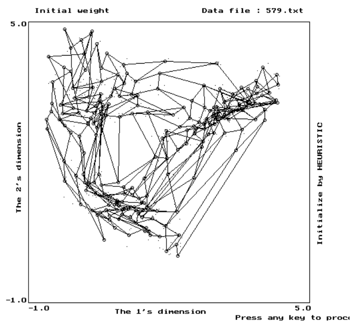


(l) after 100 epochs

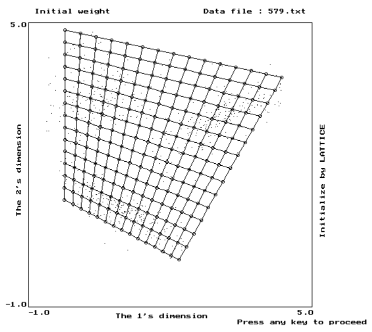
Figure 5. The reultant feature maps constructed by the three methods under the first cooling regime for the 579 data set: the left column is method 1; the center column is method 2; the right column is method 3.



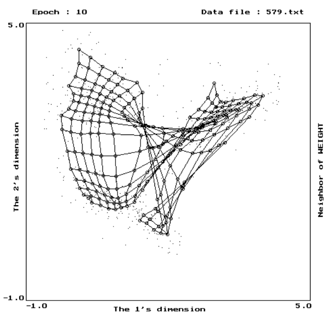
(a) the initial map



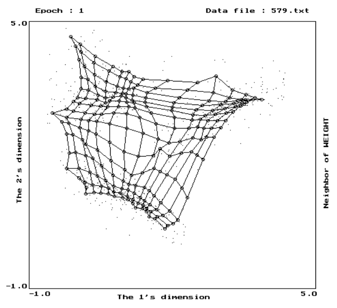
(e) the initial map



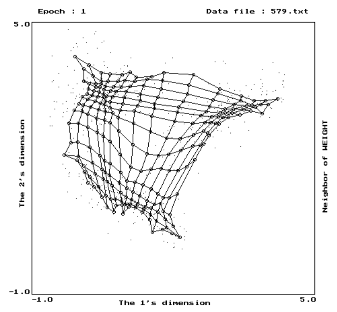
(i) the initial map



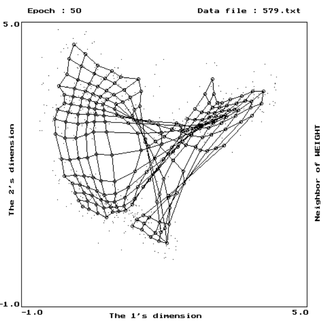
(b) after 10 epochs



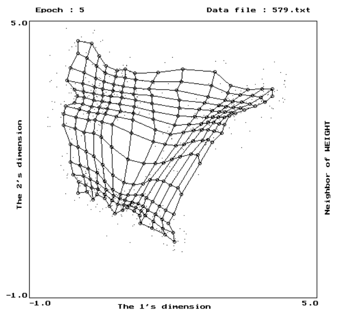
(f) after 1 epoch



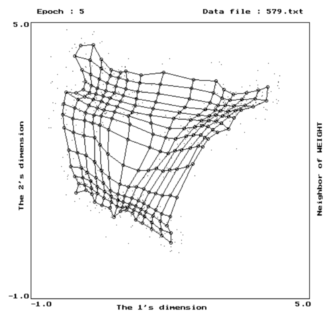
(j) after 1 epoch



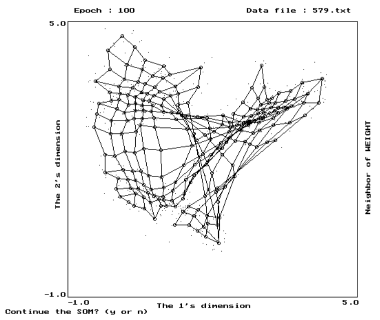
(c) after 50 epochs



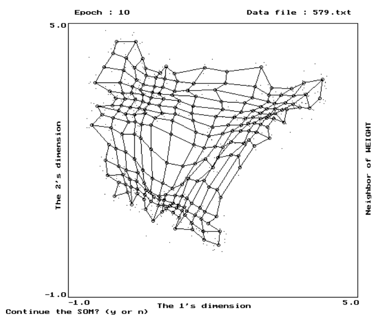
(g) after 5 epochs



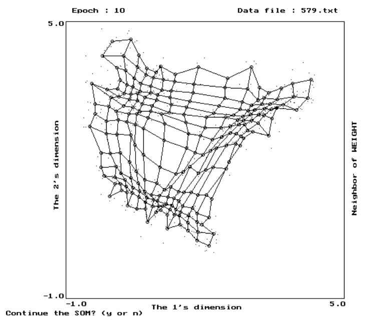
(k) after 5 epochs



(d) after 100 epochs



(h) after 10 epochs



(l) after 10 epochs

Figure 6. The resultant feature maps constructed by the three methods under the second cooling regime for the 579 data set: the left column is method 1; the center column is method 2; the right column is method 3.

3.2 Example 2: Iris Data Set

The iris data set has three subsets (i.e. iris setosa, iris versicolor, and iris virginical), two of that are overlapping. The iris data are in a four-dimensional space and there are total 150 patterns in the data set. Each class has 50 patterns. A network with 15×15 neurons was trained by the iris data set. Since it is not possible to visualize a 4-D mesh we decide to provide "calibrated maps" so that one may easily validate whether the resultant maps are topologically ordered or not. A map is calibrated if the neurons of the network are labeled according to their responses to specific known input vectors. Throughout our simulations such labeling was achieved by so-called "minimum distance method" (*i.e.* a neuron is labeled to class m if its nearest neighbor belonging to class m.) The resultant calibrated maps are shown in Figure 7-8 for regimes 1 and 2, respectively. We have manually drawn boundaries (black thin curves) between different classes on some calibrated maps in order to ease the comparisons of the three methods. Again Figure 7 confirms that the conventional SOM algorithm under a convenient cooling regime is insensible to initial weights. Although the three maps shown in Figure 7(b), 7(f), and 7(j) have no small fragment regions they have not yet matched the density distribution of the data set. As the learning procedure progressed they more matched the data distribution. From Figure 8 (d), we find that the number of neurons most responding to class 1 (i.e. iris setosa) is much less than the other two kinds of neurons, indicating the feature map is not well formed. On the contrary, classes 1, 2, and 3 shown in Figure 8(h) and 8(l) are almost separable from each other except two or three small isolated regions. Since classes 2 and 3 do overlap each other in the original 4D space it indicates this feature map matches the data structure of the iris data. Again, a good topologically-ordered map can be constructed more quickly and correctly by method 2 and our method than the SOM algorithm.

3.3 Example 3: Animal Data Set

The animal data set was originally introduced by Ritter and Kohonen [16] to illustrate the SOM for high-dimensional data set. It consists of the description of 16 animals by binary property lists tabulated in Table 1. We then group these 16 animals into three classes (1 represents bird, 2 represents carnivore, and 3 represents herbivore). Note that we find the 2D projection of the animal data set is linearly separable from each other by viewing Figure 4(c). The thirteen properties consist of the input vector to the network of 11×11 neurons. The calibrated feature maps are shown in Figures 9-10 for the regimes 1 and 2, respectively. From Figure 9, we observe that all three methods can construct topologically ordered maps. However, Figure 10 demonstrates totally different results. In Figure 10(d), classes 1, 2, and 3 span 2, 3, and 2 clusters respectively; indicating the feature map is not correctly formed because it is too fragment. On the contrary, from Figure 10(h) and 10(l) we find that the three classes are entirely enclosed by their population clusters; indicating the feature maps are well formed. Once again, method 2 and our method can construct a topologically ordered map quickly and correctly.

Table 1. Animal names and binary attributes (adapted from Ritter & Kohonen, 1989): If an attribute applies for an animal the corresponding table entry is1, otherwise 0

	small	medium	big	Two legs	Four legs	Hair	Hooves	Mane	Feathers	Hunt	Run	Fly	Swim	Class
Dove	1	0	0	1	0	0	0	0	1	0	0	1	0	1
Hen	1	0	0	1	0	0	0	0	1	0	0	0	0	1
Duck	1	0	0	1	0	0	0	0	1	0	0	0	1	1
Goose	1	0	0	1	0	0	0	0	1	0	0	1	1	1
Owe	1	0	0	1	0	0	0	0	1	1	0	1	0	1
Hawk	1	0	0	1	0	0	0	0	1	1	0	1	0	1
Eagle	0	1	0	1	0	0	0	0	1	1	0	1	0	1
Fox	0	1	0	0	1	1	0	0	0	1	0	0	0	2
Dog	0	1	0	0	1	1	0	0	0	0	1	0	0	2
Wolf	0	1	0	0	1	1	0	1	0	1	1	0	0	2
Cat	1	0	0	0	1	1	0	0	0	1	0	0	0	2
Tiger	0	0	1	0	1	1	0	0	0	1	1	0	0	2
Lion	0	0	1	0	1	1	0	1	0	1	1	0	0	2
Horse	0	0	1	0	1	1	1	1	0	0	1	0	0	3
Zebra	0	0	1	0	1	1	1	1	0	0	1	0	0	3
Cow	0	0	1	0	1	1	1	0	0	0	0	0	0	3

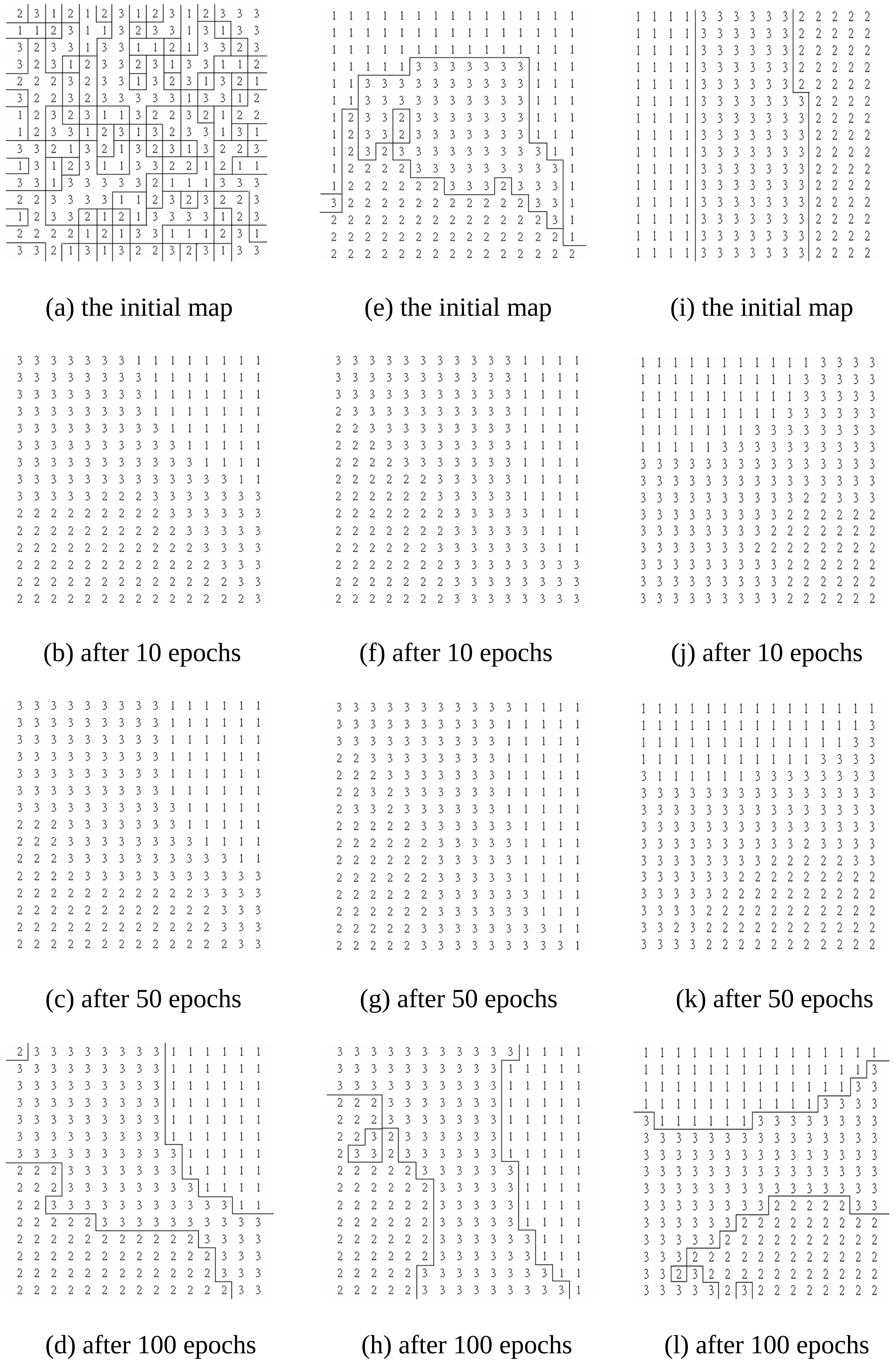
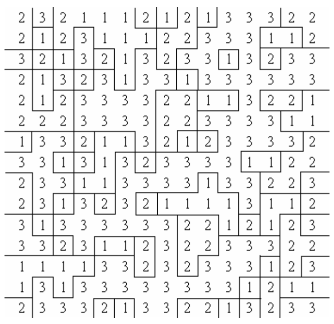
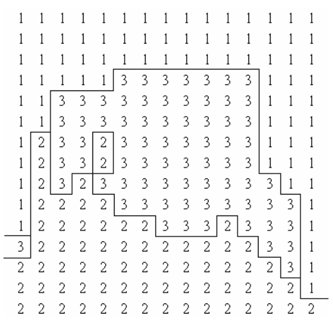


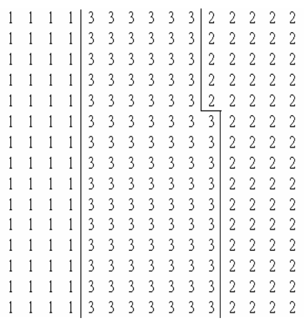
Figure 7. The resultant calibrated maps constructed by the three methods under the first cooling regime for the iris data set: the left column is method 1; the center column is method 2; the right column is method 3.



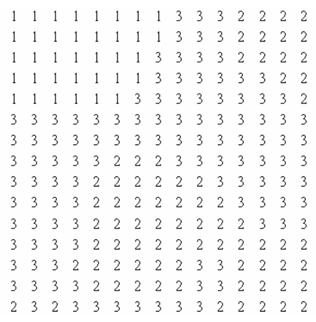
(a) the initial map



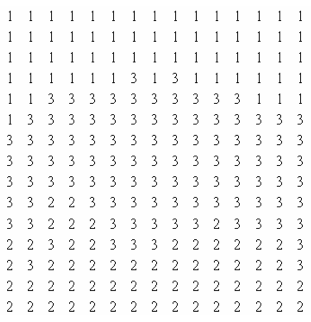
(e) the initial map



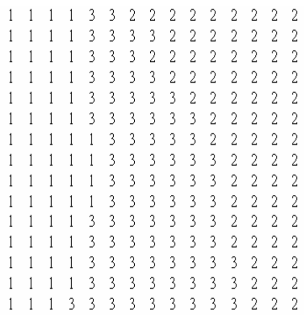
(i) the initial map



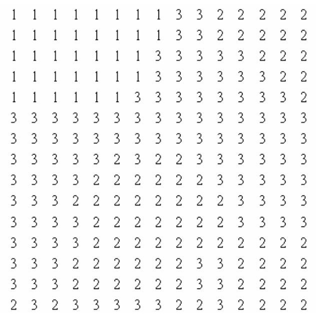
(b) after 10 epochs



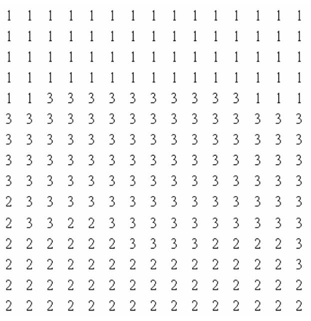
(f) after 1 epoch



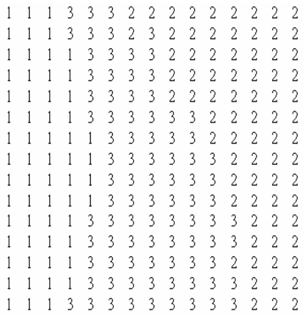
(j) after 1 epoch



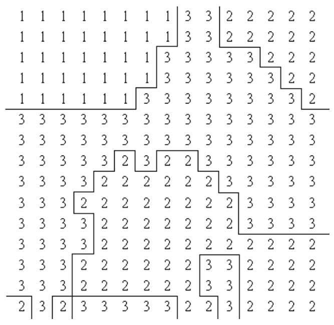
(c) after 50 epochs



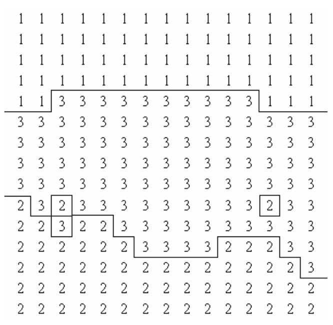
(g) after 5 epochs



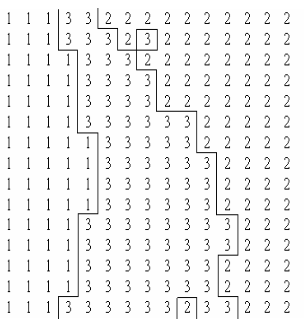
(k) after 5 epochs



(d) after 100 epochs

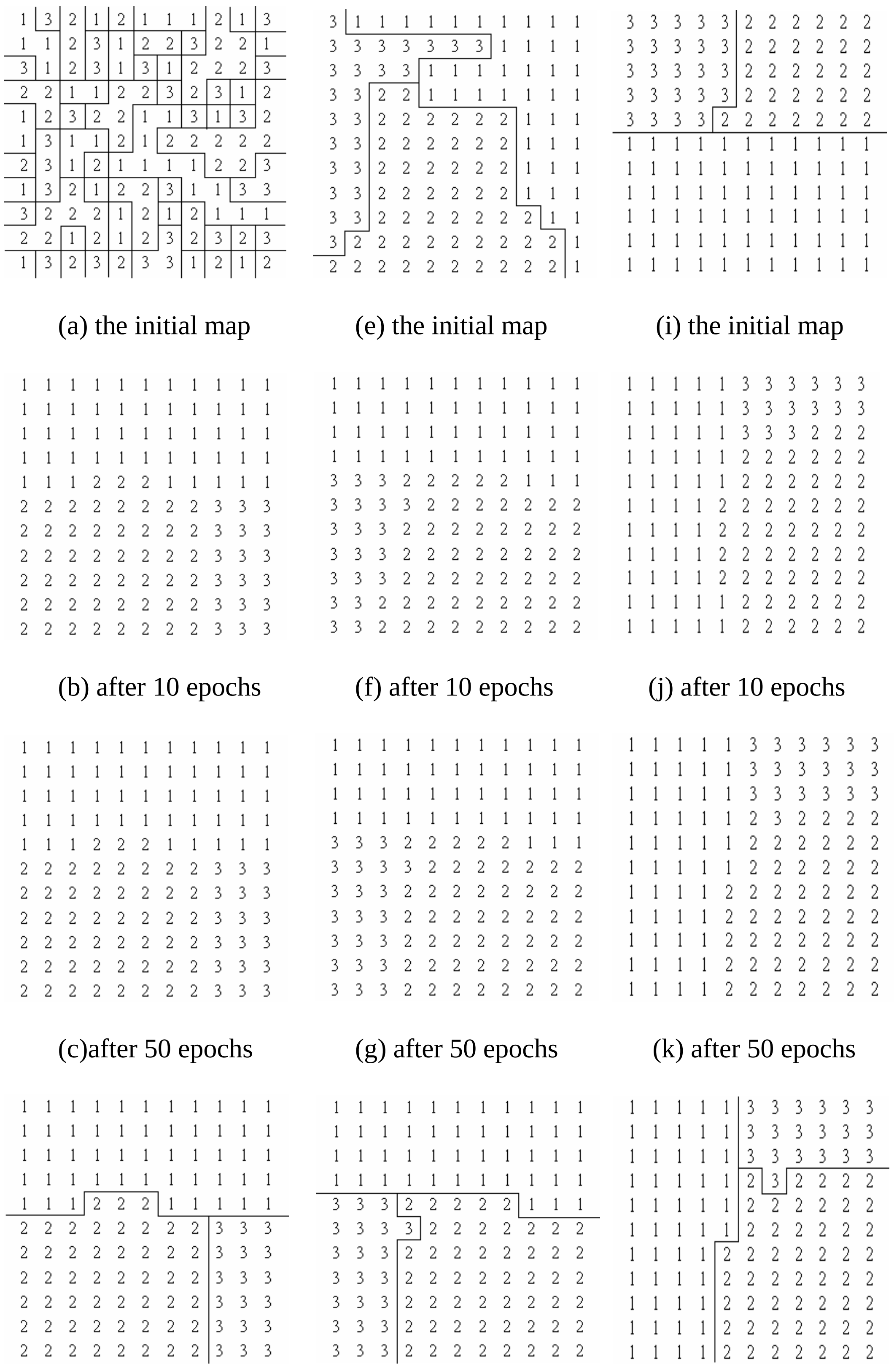


(h) after 10 epochs



(l) after 10 epochs

Figure 8. The resultant calibrated maps constructed by the three methods under the second cooling regime for the iris data set: the left column is method 1; the center column is method 2; the right column is method 3.



(a) the initial map (e) the initial map (i) the initial map

(b) after 10 epochs (f) after 10 epochs (j) after 10 epochs

(c)after 50 epochs (g) after 50 epochs (k) after 50 epochs

(d) after 100 epochs (h) after 100 epochs (l) after 100 epochs

Figure 9. The resultant calibrated maps constructed by the three methods under the first cooling regime for the animal data set: the left column is method 1; the center column is method 2; the right column is method 3.

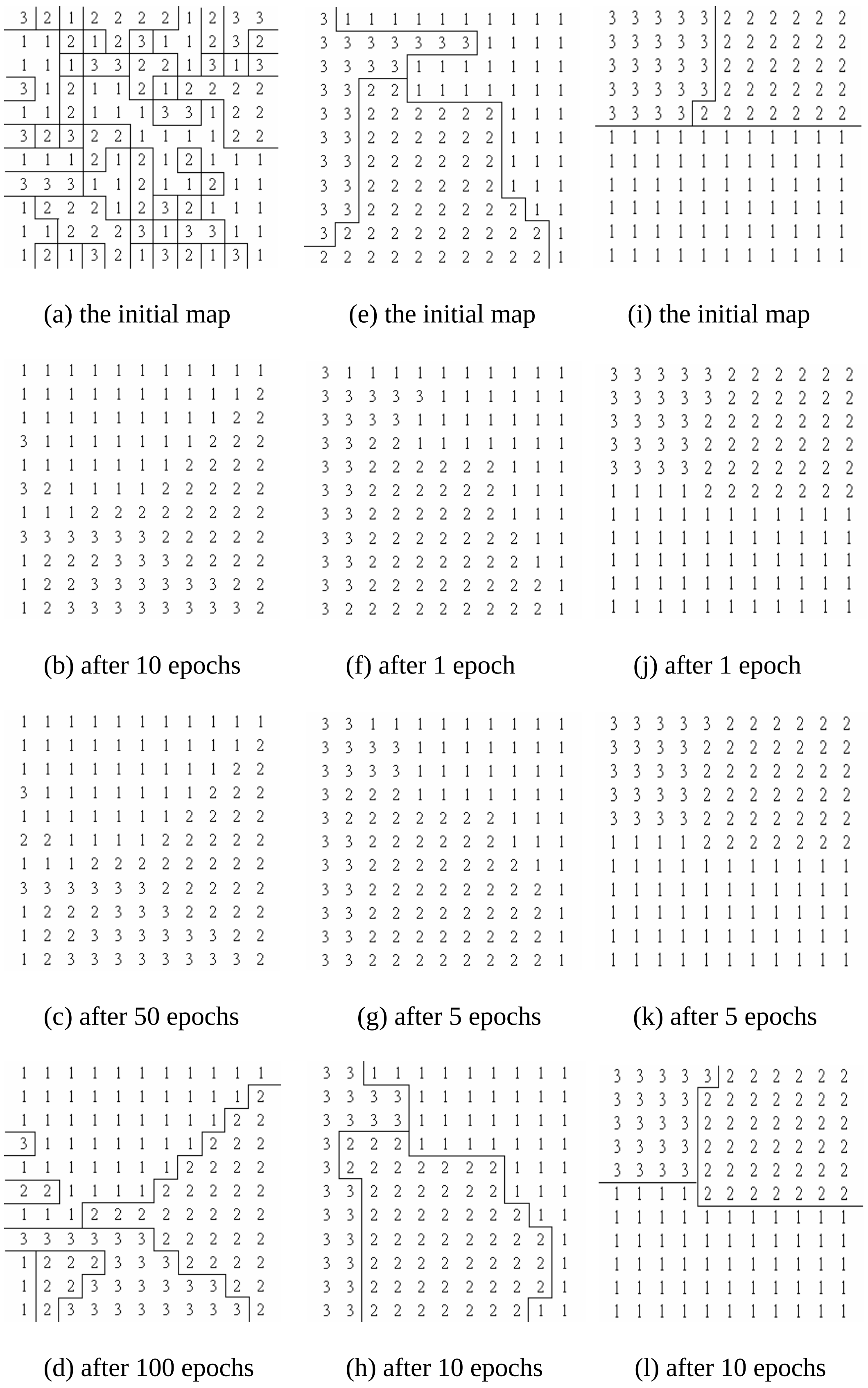


Figure 10. The resultant calibrated maps constructed by the three methods under the second cooling regime for the animal data set: the left column is method 1; the center column is method 2; the right column is method 3.

4. Conclusions

In this paper, an efficient initialization scheme for the SOM algorithm is proposed. From the simulation results, we find that it may be better to construct a good initial map and then to use the unsupervised learning to make small subsequent adjustments. Observing the simulation results we can make the following several observations:

1. The initialization scheme can greatly accelerate the training phase since we start out from a good initial map.
2. The initialization scheme is very simple and straightforward.
3. The topological relations of data can be more preserved if we incorporate the SOM algorithm with the initialization scheme.
4. Method 2 and our method both can quickly form topologically ordered maps. However, our method is simpler than method 2 because our method requires less computational resources than method 2.
5. With a proper cooling regime, the random initialization may also form topologically ordered feature maps after a lot of training epochs. However, with a wrong cooling regime, there may exist defects (e.g. a twist or a kink) in the formed map even after a large number of training epochs.

In fact, our method can be regarded as another kind of linear initialization method. However, compared to the "linear initialization" method presented in [18], our method outperforms the former one based on the comparison of computations since we do not need to compute the eigenvectors of an autocorrelation matrix of the input patterns.

References

[1] Baraldi, A., Blonda, P., Parmiggiani, F., Pasquariello, G. and Satalino, G., "Model Transitions in Descending FLVQ," *IEEE Trans. on Neural Networks*, Vol. 9, pp. 724-738 (1998).

[2] Fritzke, B., "Growing Cell Structures-a Self-Organizing Network for Unsupervised and Supervised Learning," *Neural Networks*, Vol. 7, pp. 1441-1460 (1994).

[3] Harp, S. A. and Samad, T., "Genetic Optimization of Self-Organizing Feature Maps," *Proc. Int. Conf. on Neural Networks*, pp. 341-346, (1991).

[4] Huang, S. J. and Hung, C. C., "Genetic Algorithms Enhanced Kohonen's Neural Networks," *IEEE Int. Conf. on Neural Networks*, pp. 708-712 (1995).

[5] Jun, Y. P., Yoon, H. and Cho, J. W., "L Learning: a Fast Self-Organizing Feature Map Learning Algorithm Based on Incremental Ordering," *IEICE Trans. on Information & Systems*, Vol. E76, pp. 698-706 (1993).

[6] Kiang, M. Y., Kulkarni, U. R., Goul, M., Philippakis, A., Chi, R. T. and Turban, E., "Improving the Effectiveness of Self-Organizing Map Networks Using a Circular Kohonen Layer," *Proc. of the 30th Hawaii Int. Conf. on System Sciences*, pp. 521-529 (1997).

[7] Koh, J., Suk, M. and Bhandarkar, S. M., "A Multilayer Self-Organizing Feature Map for Range Image Segmentation," *Neural Networks*, Vol. 8, pp. 67-86 (1995).

[8] Kohonen, T., *Self-Organization and Associative Memory*, 3rd ed., Springer-Verlag, Berlin, Germany (1989).

[9] Kohonen, T., *Self-Organizing Maps*, Springer-Verlag, Berlin, Germany (1995).

[10] Kohonen, T., *Self-Organizing Maps*, Springer-Verlag, New York, U.S.A. (1995).

[11] Kohonen, T., "The Self-Organizing Feature Map," *Pro. of the IEEE*, Vol. 78, pp. 1464-1480 (1990).

[12] Kohonen, T., Oja, E., Simula, O., Visa, A. and Kangas, J.," Engineering Application of the Self-Organizing Map," *Pro. of the IEEE*, Vol. 84, pp. 1358-1383 (1996).

[13] Lo, Z. P. and Bavarian, B., "On the Rate of Convergence in Topology Preserving Neural Networks," *Biological Cybernetics*, Vol. 65, pp. 55-63 (1991).

[14] Martinetz, T. M. and Schulten, K. J., "Topology Representing Networks," *Neural Networks*, Vol. 7, pp. 507-522 (1994).

[15] McInerney, M. and Dhawan, A., "Training the Self-Organizing Feature Map Using Hybrids of Genetic and Kohonen Methods," *IEEE Int. Conf. on Neural Networks*, pp. 641-644 (1994).

[16] Ritter, H. J. and Kohonen, T., "Self-Organizing Semantic Maps," *Biological Cybernetics*, Vol. 61, pp.241-254 (1989).

[17] Samad, T. and Harp, S. A., "Self-Organization with Partial Data," *Network: Computation in Neural Systems*, Vol. 3, pp. 205-212 (1992).

[18] Su, M. C. and Chang, H. T.

"Genetic-Algorithm-Based Approach to Self-Organizing Feature Map and its Application in Cluster Analysis," *IEEE Int. Joint Conf. on Neural Networks*, pp. 2116-2121 (1995).

[19] Su, M. C. and Chang, H. T., "Fast Self-Organizing Feature Map," *IEEE Trans. on Neural Networks*, Vol. 13, pp. 721-733 (2000).

[20] Tsao, E. C., Bezdek, J. C. and Pal, N. R., "Fuzzy Kohonen Clustering Network," *Pattern Recognition*, Vol. 27, pp. 757-764 (1994).

[21] Van Hulle, M. M. and Leuven, K. U., "Globally-Ordered Topology-Preserving Maps Achieved with a Learning Rule Performing Local Weight Updates Only," *IEEE Workshop of Neural Networks for Signal Processing*, pp. 95-104 (1995).

Manuscript Received: Dec. 19, 2001
and Accepted: Feb. 18, 2002