An Efficient Initialization Scheme for the Self-Organizing Feature Map Algorithm

Mu-Chun Su, Ta-Kang Liu and Hsiao-Te Chang
Department of Electrical Engineering, Tamkang University, Taiwan, R.O.C.
E-mail address: muchun@ee.tku.edu.tw

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 of the algorithm, namely, the learning-rate parameter and the neighborhood function. 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. Two data sets are tested to illustrate the performance of the proposed method.

I. Introduction

Recently, numerous technical reports have been written about successful applications of the self-organizing feature map algorithm developed by Kohonen [1]. 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* provided partial review [2].

It is often reported in the literature 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 function. Moreover, the accuracy of the map 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 array neurons [3]. Note that although the SOM algorithm is simple computationally, 100,000 iterations are not uncommon.

Ritter and Schulter [4] have shown that on average (for small learning rate) the SOM algorithm decreases a cost function associated with the Kohonen's feature-mapping model until the network reaches a local minimum. That is, it has problems with getting stuck in non-optimum states. Since the initialization strongly affects the ultimate solution, it is better not to initialize the weights of the neural array at small random values. In this paper, we propose an efficient initialization scheme for the SOM algorithm. The development of the initialization scheme was motivated not only by the above mathematical analysis but also by a biological assumption. It is now hard to believe that the unsupervised learning mechanism is solely responsible for the retinotopic maps found biologically because it can not reproduce the experimental phenomena of regeneration after damage. Current theories involve some degree of **chemoaffinity**, in which growing axons carry chemical markers that help to define appropriate target sites. A degree of softwiring by unsupervised learning is then invoked to refine the map [5]. Therefore, we believe that it will be a good idea to first construct a good initial map according to our proposed initialization scheme and then use the SOM algorithm to refine the map. This paper is organized into 5 sections. In the following section we will shortly describe the SOM algorithm. The initialization scheme is then discussed in Section 3. Two data sets are utilized to demonstrate the effectiveness of the scheme. The simulation results are given in Section 4. Finally, Section 5 concludes the paper.

II. Self-Organizing Feature Map Algorithm

The principal goal of self-organizing feature maps is to transform patterns of arbitrary dimensionality into the responses of one- or two-dimensional arrays of neurons, and to perform this transform adaptively in a topological ordered fashion. The transformation makes topological neighborhood relationship geometrically explicit in low-dimensional feature maps.

The training algorithm proposed by Kohonen for forming an SOM is 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:

$$j^* = \arg \min_j \|\underline{x}(k) - \underline{w}_j\|, j = 1, \cdots, N^2. \qquad (1)$$

where $\underline{x}(k) = [x_1(k), \cdots, x_n(k)]^T$ represents the k^{th} input pattern, N^2 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:

$$\underline{w}_j(k+1)=\begin{cases}\underline{w}_j(k)+\eta(k)(\underline{x}(k)-\underline{w}_j(k)) & \text{if } j\in N_{j^*}(k)\\ \underline{w}_j(k) & o.w.\end{cases} \quad (2)$$

where $\eta(k)$ is a positive constant and $N_{j^*}(k)$ is the topological neighborhood set 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 main parameters (*i.e.* $\eta(k)$ and $N_{j^*}(k)$) are selected, initial values of weight vectors, and the number of iterations.

III. Initialization Scheme

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. Instead of randomly initializing the weights of the network, we propose a simple straightforward initialization scheme to solve the problem. Consider Fig. 1, which depicts a two-dimensional neural array of size N×N. The basic ideal is to find a large enough hypercube to cover all the training patterns and then to squeeze the hypercube into a plan. The scheme is described as follows:

Step 1: Initialization of the neurons on the four corners:
We first select a pair of input 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.* $\underline{w}_{N,1}$ 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 initialized 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.* $\underline{w}_{N,N}$) is set to be the coordinates of the pattern which is farthest to the previously selected three patterns (i.e. $\underline{w}_{1,1}$, $\underline{w}_{N,1}$, and $\underline{w}_{1,N}$).

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:

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

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

$$\begin{aligned}\underline{w}_{i,1}&=\frac{\underline{w}_{N,1}-\underline{w}_{1,1}}{N-1}(i-1)+\underline{w}_{1,1} \qquad \text{for} \quad i=2,\cdots,N-1 \qquad (5)\\ &=\frac{i-1}{N-1}\underline{w}_{N,1}+\frac{N-i}{N-1}\underline{w}_{1,1}\end{aligned}$$

$$\begin{aligned}\underline{w}_{i,N}&=\frac{\underline{w}_{N,N}-\underline{w}_{1,N}}{N-1}(i-1)+\underline{w}_{1,N} \qquad \text{for} \quad i=2,\cdots,N-1 \qquad (6)\\ &=\frac{i-1}{N-1}\underline{w}_{N,N}+\frac{N-i}{N-1}\underline{w}_{1,N}\end{aligned}$$

The idea is simple. Since two points form a line in the input space, we just uniformly partition the line into N-1 segments and then 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 N-1
  Begin
   For j from 2 to N-1
    Begin
        $\underline{w}_{i,j}=\frac{\underline{w}_{i,N}-\underline{w}_{i,1}}{N-1}(j-1)+\underline{w}_{i,1}$
            $=\frac{j-1}{N-1}\underline{w}_{i,N}+\frac{N-j}{N-1}\underline{w}_{i,1}$
    End;
   End;
 End;
End;
```

One may ask why we do not directly partition the input space into N×N 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. As a result, we will probably 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 Fig. 2.

IV. Simulation Results

We use two data sets to test the proposed initialization scheme.

Example 1: 2-D artificial data set

We illustrate the behavior of the SOFM algorithm incorporated with the proposed initialization scheme by using computer simulations to study a network with 215 neurons, arranged in the form of a 2-dimensional array with 15 rows and 15 columns. The network was trained by the artificial data set shown in Fig. 3. Fig. 4 and Fig. 5 show the resulting maps constructed by the random initialization scheme and our initialization scheme, respectively. Obviously, in the case of our initialization scheme, the mesh remains untangled and quickly adapts in detail. On the other hand, in the case of the random initialization scheme, the mesh first tangled and then tried to unfold itself. Unfortunately, even if we used more iterations to continue the training process the incorrect topological ordering was not eliminated.

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. 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 resulting 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 resulting calibrated maps are shown in Figs. 6-7. Again, a topologically ordered map can be constructed more quickly and correctly by our initialization scheme than the random one.

V. Conclusions

In this paper, an efficiently initialization scheme for the SOM algorithm is proposed. From the simulation results, we find that it is better to construct a good initial map and then to use the unsupervised learning to make small subsequent adjustments.

References

[1] T. Kohonen, *Self-Organization and Associative Memory*, 3rd ed. New York, Berlin: Springer-Verlag, 1989.

[2] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas," Engineering application of the self-organizing map," Proceedings of the IEEE, vol. 84, no. 10, pp. 1358-1383, 1996.

[3] T. Kohonen, "The self-organizing feature map," Proceedings of the IEEE, vol. 78, no. 9, pp. 1464-1480, 1990.

[4] H. Ritter and K. Schulter, "Kohonen's self-organizing maps: exploring their computational capabilities," in IEEE Int. Conf. on Neural Networks, vol. 1, pp. 109-116, San Diego, 1988.

[5] J. Hertz, A. Krogh, and R. G. Palmer, Introduction to The Theory of Neural Computation, Addison-Wesley Publishing Company, 1991.

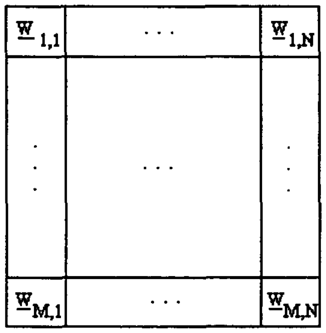


Fig.1 The arrangement of an M×N neural array.

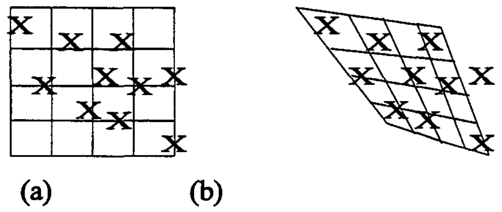


Fig.2 The difference between the random initialization scheme and our initialization scheme.

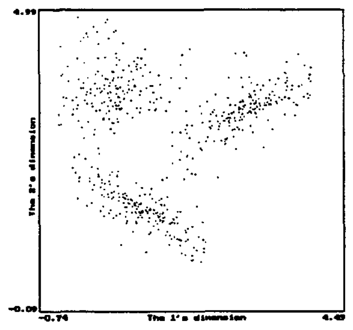
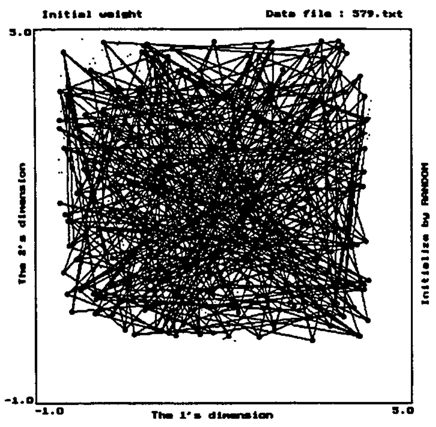
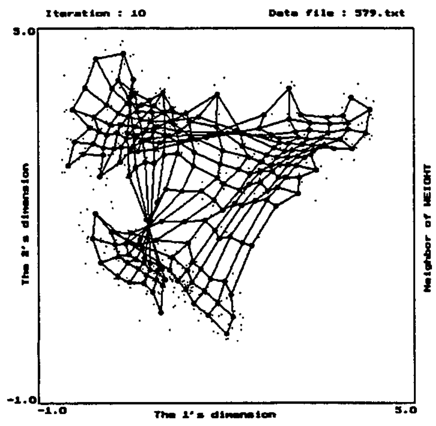


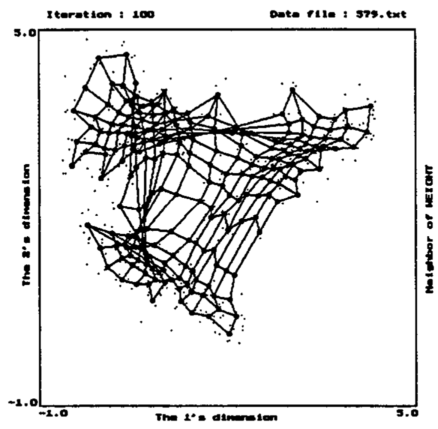
Fig. 3 The 579 artificial data points.



(a) Initial map

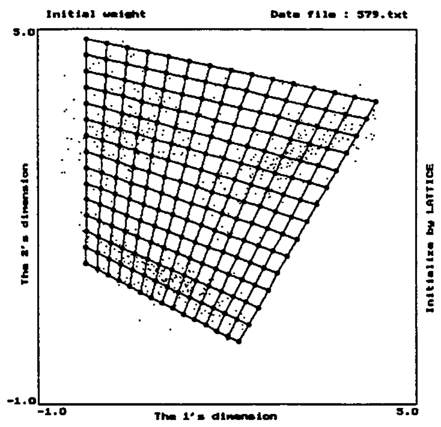


(b) After 10 iterations

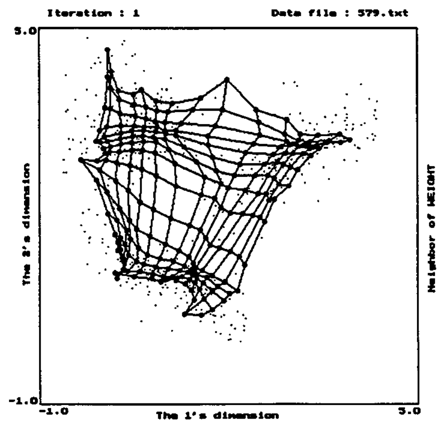


(c) After 100 iterations

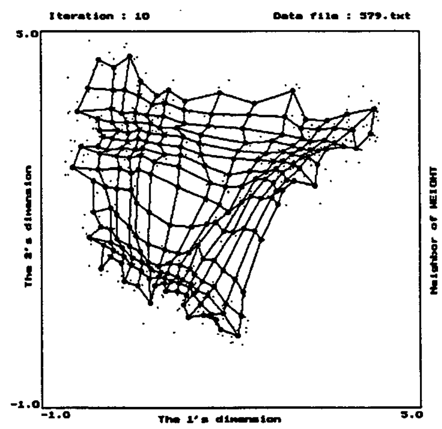
Fig. 4 The learning procedure using the random initialization scheme.



(a) Initial map

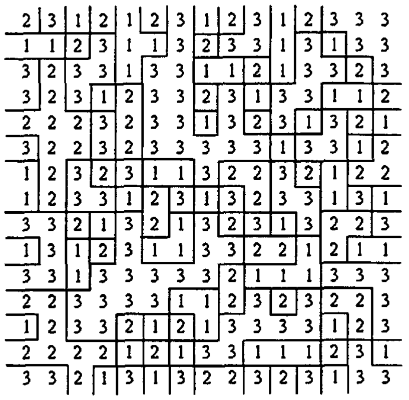


(b)After 1 iteration

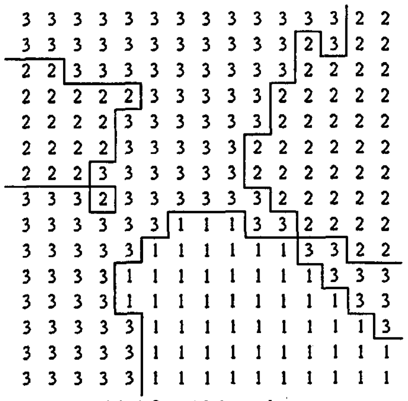


(c) After 10 iterations

Fig. 5 The learning procedure using our initialization scheme.

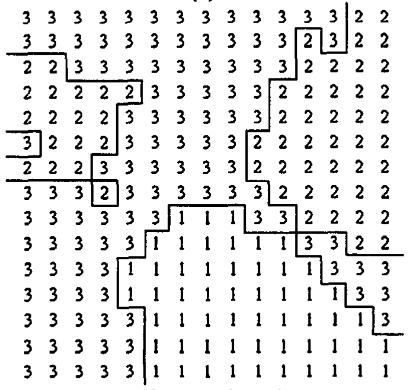


(a)The initial map



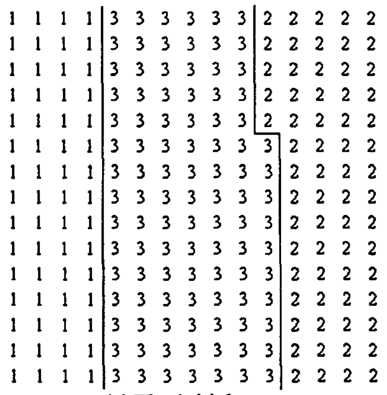
(a) After 10 iterations

(b)

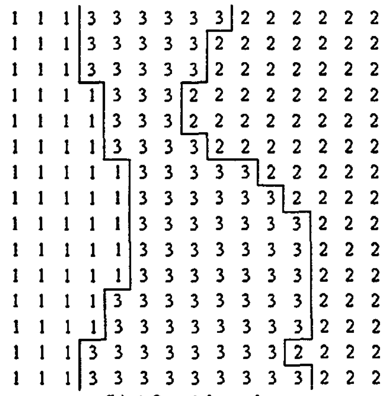


(c) After 100 iterations

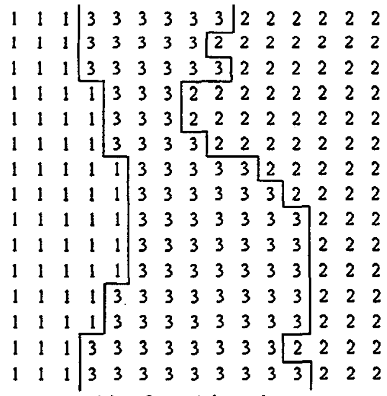
Fig. 6 The maps constructed by the random initialization scheme.



(a) The initial map



(b) After 1 itaration



(c) After 10 iterations

Fig. 7 The maps constructed by our initialization scheme.