Evolutionary Self-Organizing Map

Maiga Chang, Horng-Jyh Yu and Jia-Sheng Heh
Department of Information and Computer Engineering, C.Y.C.U., Chungli, Taiwan

ABSTRACT

Extending Kohonen's SOM, this paper proposes one kind of dynamically growing neural network, called Evolutionary SOM (ESOM). Firstly, the output layer of SOM is represented by a so-called neighborhood graph, where nodes are neurons' weights and edges are the neighborhood relationships of SOM. Then two basic differentiation operations, node differentiation and edge differentiation, are proposed for network differentiation. As Nature's evolution, each generation of ESOM includes several species of neural nets and the survivors of competition will differentiate to the next generation.

This kind of evolution is implemented as two new modules of ESOM, in addition to Kohonen's SOM toolbox in Matlab. A cross pattern with 1000 data points is taken as example. The results show that there are a large quantity of unnecessary neurons in Kohonen's SOMs; whereas, the resultant ESOM has much less size and better fitness to training input.

keywords: *Self-Organizing Map, neighborhood, evolution, differentiation*

1. Introduction

In 1973, Kohonen proposed a new generation of neural network model, called Self-Organizing Map, then opened a new research topic in artificial neural network [1][2]. In this Kohonen's SOM, there is no hidden layer and based on its learning strategy, those neurons in output layer form a specific structure. When a neuron wins a chance of updating weights, its neighbors within some specific range of the output layer will follow this learning. This neighborhood relationship in the output layer makes the most significant feature of SOM. And for those applications with some kinds of relation among outputs, Kohonen's SOM provides a last resort. [3][4][5]

In Kohonen's SOM, the structure of neighborhood relationship is fixed during the learning phase. This makes the architectures of resultant neural network be limited to some special structures, such as mesh or linear structure, to fit training data. [6] Recently, many researchers modify Kohonen's neighborhood relationship to propose some special architectures of SOM for some application fields. [7][8] However, these extensions still rely on fixed neighborhood relationship, thus new architectures of SOM should be sophisticatedly designed for achieving special requirements of new applications. [9][10] Therefore, the investigation of SOM neighborhood relationship to generate automatically growing neural network becomes more and more important. [11][12]

This paper proposes a dynamic neighborhood relationship to improve Kohonen's SOM, called *Evolutionary SOM* (*ESOM*). Two basic differentiation operations of graph are developed, analyzed and implemented. Similar to Nature's evolution, these two operations make neighborhood relationship grow up dynamically, then the resultant network of ESOM can track input data well.

Section 2 defines several basic concepts of Kohonen's SOM and discusses ts training model and evaluation criteria. The evolution of graph is studied in Section 3, and two differentiation operations are developed and analyzed. Section 4 designs several modules for the implementation of ESOM. One example is given and the resultant network is compared with those from Kohonen's SOM in Section 5. Finally, Section 6 makes a brief conclusion.

2. SOM and its training model

This paper extends the neighborhood relationship and the dynamic architecture of Kohonen's Self-Organizing Map (SOM) to propose our Evolutionary SOM (ESOM). Here, the definitions of SOM and its structures have to be introduced first. And then a training model of SOM is explained.

The basic structure of SOM is shown in Figure 1, where the i^{th} *data vector* x^i is one of the training patterns, the j^{th} *weight vector* w^j represents the connection weights from the input layer to neuron j in the output layer and the j^{th} *output* o^j is the output of neuron

neuron j in the output layer is

$$o^j = f(\mathrm{x}^i \cdot \mathrm{w}^{jT}), \qquad (1)$$

where $f(\cdot)$ is the *threshold function.* As $f(\cdot)$ is continuous, $f: R \rightarrow [0,1]$; whereas, $f: R \rightarrow \{0,1\}$ when $f(\cdot)$ is discrete. These outputs o^j's will compete through lateral connections in the output layer, and the output with the largest value of Eq.(1), i.e.

$$j^* = \arg\max_j o^j = \arg\max_j (\mathrm{x}^i \cdot \mathrm{w}^{jT}), \qquad (2)$$

will increase to value 1 and others will fade out. This is one kind of *winner-take-all.*

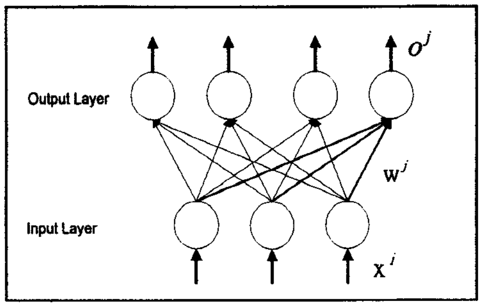

Figure 1. Basic structure of Kohonen's SOM

A special feature of SOM is its neighborhood. The *neighborhood function* $Nbd(\cdot)$ of SOM defines the adjacent relationships of the connections among neurons. When neuron j^* win the competition, its weights and the weights of its neighboring neurons $Nbd(j^*)$ will be modified through

$$\mathrm{w}^j(t+1) = \mathrm{w}^j(t) + \alpha(t)\,(\mathrm{x}^j - \mathrm{w}^j(t)), \qquad (3)$$

where $j \in Nbd(j^*)$ and $\alpha(t)$ is a learning factor. Since there is no output is needed in the learning process, such learning is one kind of unsupervised learning. The most general neighborhood function is *Mexican-hat function,* with which the excitation strength of a neuron decreases as its distance to the activated neuron increases.

As other neural network, Kohonen's SOM is used in two phases: training phase and production phase. In the beginning of *training phase,* weight vectors w^j's are initialized with small random numbers. As one training pattern x^i is given in the input layer, all the outputs o^j's in the output layer appear as Eq.(1) and one output, say j^*, wins the competition by Eq.(2). Then the winning neuron j^* and its neighbors $Nbd(j^*)$ will modify their weights through Eq.(3). Such training process is repeated, as shown in Figure 2. After a long-term training, the

j in the output layer. The *output function* of

whole net becomes stable and can be applied to solve other testing patterns. In *production phase,* some new data, also called x^i, are given as testing patterns in input layer and the corresponding output o^j is found out through Eqs.(1) and (2).

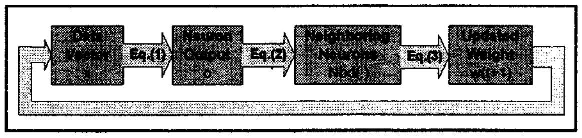

Figure 2 Training process of Kohonen's SOM

Within the four-step training process: data vector x^i, neuron output o^j, neighboring neurons $Nbd(j^*)$ and updated weights $\mathrm{w}^j(t+1)$, two factors are very influential in the formation of resultant neural net: data vector and neighborhood function. When a data vector x^i is given in the input layer, the neuron output o^j's will be activated and then the neuron j^* with the largest $\mathrm{x}^i \cdot \mathrm{w}^j$ will be elected as the winner of the competition. Let θ be the intersection angle between x^i and w^j, $\mathrm{x}^i \cdot \mathrm{w}^{jT} = |x^i||w^j|\cos\theta$ indicates the similarity measure between x^i and w^j. Therefore j^* (w^{j^*}) is the neuron (weight) that is the closest to the training input x^i. This means the production phase of SOM is one kind of classification problem, called *data classification problem.* In such problem, several data vector x^i's will correspond to one w^j and this w^j represents a *data center* of these *data cluster* x^i's. This relationship is shown in the upper part of Figure 3.

In data classification problem, the winning neuron j^* is defined a function of input vector x^i, called *classification index*:

$$C_{index}(\mathrm{x}^i) = j^* = \arg\max(\mathrm{x}^i \cdot \mathrm{w}^{jT}), \qquad (4)$$

which also indicates the cluster of x^i. For each cluster, the *classification error* can be defined as the accumulated distance between data vectors and its data center:

$$d_c(j^*) = \sum_{C_{index}(\mathrm{x}^i)=j^*} d(\mathrm{x}^i, \mathrm{w}^{j^*}), \qquad (5)$$

Moreover, the whole *classification error* of SOM is composed by those errors of all data centers:

$$\varepsilon_C = \sum_j \sum_{C_{index}(\mathrm{x}^i)=j} d(\mathrm{x}^i, \mathrm{w}^j) = \sum_j d_c(j), \qquad (6)$$

which is a performance index of the training results.

On the other hand, different formations of SOM network with same data vector can be generated by different neighborhood functions. This is the *network structurization problem* in Figure 3. Besides the classification error discussed previously, Figure 4 presents another important evaluation criteria of SOM. In Figure 4, both neurons have the same weights but the left one will be more efficient than the right one. Different values of weights come from their different initial values of weights. Although these two resultant graphs are isomorphic, an energy function have to be provided to evaluate such structure differences of training results.

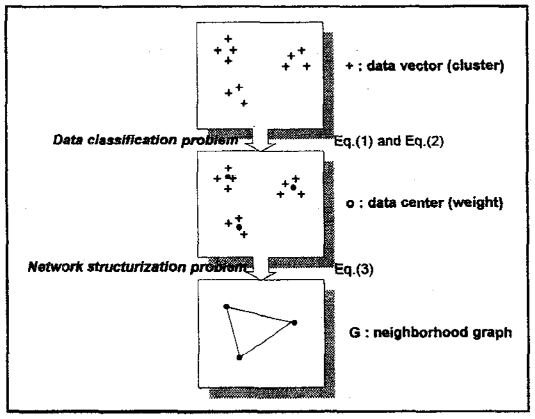
+ : data vector (cluster)
Data classification problem
Eq.(1) and Eq.(2)
o : data center (weight)
Network structurization problem
Eq.(3)
G : neighborhood graph

Figure 3. Two-problem model of SOM's training

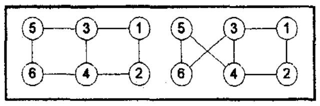

Figure 4. Two SOMs with the same neurons

Applying the neighborhood concept to the above classification error, we define the *structuralization error* of a neuron as the accumulated distance between its weights and those weights of its neighbors:

$$d_s(j) = \sum_{k \in Nbd(j)} d(\mathrm{w}^j, \mathrm{w}^k), \tag{7}$$

Therefore, the whole *structurization error* of SOM is composed by those errors of all neurons:

$$\varepsilon_S = \sum_j \sum_{k \in Nbd(j)} d(\mathrm{w}^j, \mathrm{w}^k) = \sum_j d_s(j), \tag{8}$$

Traditionally, the training of SOM is focused on the minimization of the classification error, which then gives the learning rule Eq.(3) as one kind of delta rule. For obtaining more suitable SOM, the structurization error has to be taken into consideration. In the following, an evolutionary SOM combining learning algorithm with evolution algorithm is proposed based on the optimization of structurization error ε_S.

3. Evolution of graph

Through graph theory, weight vectors in the training model of Figure 3 can be translated into several simple graphs. First of all, the data centers (weights, clusters) $W = \{\mathrm{w}^1, \mathrm{w}^2, \ldots\}$ obtained from solving data classification problem can be treated as a *null graph* $G_0(W, \phi)$, where node set is W and dge set is empty. And all the input data vectors have been classified into these clusters. Therefore, a null graph $G_0(W, \phi)$ is formed before the weight update in network structurization problem. Moreover, $G_0(W, \phi)$ is possible to be extended to combine the neighborhood function $Nbd(\cdot)$ to form a *neighborhood graph* $G_N(W, N)$, where

$$N = \{edge(\mathrm{w}^j, \mathrm{w}^k) : \mathrm{w}^k \in Nbd(\mathrm{w}^j)\}, \tag{9}$$

during the learning of weight vectors. The edge set of a graph represents the internal structure of this graph; therefore, the edge set N of an SOM includes all the internal structure of this SOM. When such structure N is fixed during training process, the structurization error Eq.(8) can be optimized through

$$\varepsilon_S = \sum_{edge(\mathrm{w}^j, \mathrm{w}^k) \in N} d(\mathrm{w}^j, \mathrm{w}^k), \tag{10}$$

if the energy function combined with ε_C and ε_S is suitably defined.

When the neighborhood structure N of SOM is fixed, the optimization of network can only adjust the values of connection weights. Only when the weight vectors can be dynamically changed, it is possible to obtain optimal SOM structure to fit the training inputs. This is the motivation of our Evolutionary SOM (ESOM).

Since the network structure is represented by neighborhood graph, the evolution of SOM is also based on the evolution of simple graph. Here, two basic *evolution operations* of graph are designed to process graph evolution: node differentiation and edge differentiation. *Node differentiation* means the process of dividing a node into two nodes; whereas, *edge differentiation* is the splitting of an edge into two edges.

(1). Node differentiation

In a simple graph, one node is selected as the differentiation point, then a new node is generated with a new edge connected to the origin node, as Figure 5 shown. When the original node has several edges connected to other node, the differentiation is more complex. In this condition, these edges are distributed to both nodes. Figure 6 shows some cases of such edge distribution. A *node differentiation operator* $NodeDiff(\cdot, \cdot, \cdot)$ is defined as

$$G_2 = NodeDiff(G_1, V_i, \{e_k\}), \quad (11)$$

where $G_2(\cdot,\cdot)$ is the transformed graph and $G_1(\cdot,\cdot)$ is the original graph with selected node V_i and adjacent edge set $\{e_k\}$, $k \in \{1, 2, ..., \rho(V_i)\}$, $\rho(V_i)$ = the connection degree of V_i in G_1.

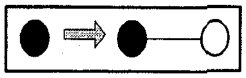

Figure 5. Node differentiation

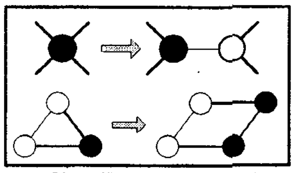

Figure 6. Edge distribution of node differentiation

(2). Edge differentiation

This type of differentiation is just to split one selected edge into two edges. These two edges own the same two nodes, as Figure 7 shown. Similar to node differentiation operator, an *edge differentiation operator* $EdgeDiff(\cdot,\cdot)$ is defined as:

$$G_2 = EdgeDiff(G_1, e_j), \quad (12)$$

where $G_2(\cdot,\cdot)$ is the transformed graph and $G_1(\cdot,\cdot)$ is the original graph with selected edge e_j.

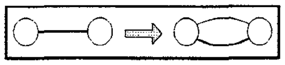

Figure 7. Edge differentiation

Figure 8 illustrates a complete evolution process of two kinds of basic differentiation operators. With such procedure, any connected graph can be evolved through these two differentiation operators.

For neighborhood graphs, two edges with the same nodes represent only one neighboring relationship, then make redundancy. However, any edge differentiation of SOM will be followed with a node differentiation, as Figure 9 shown. Therefore, these two basic differentiation operations constitute a *modified edge differentiation operator* :

$$G_2 = ModEdgeDiff(G_1, e_j, V_i, \{e_k\}) \overset{\Delta}{=} NodeDiff(EdgeDiff(G_1, e_j), V_i, \{e_k\}), \quad (13)$$

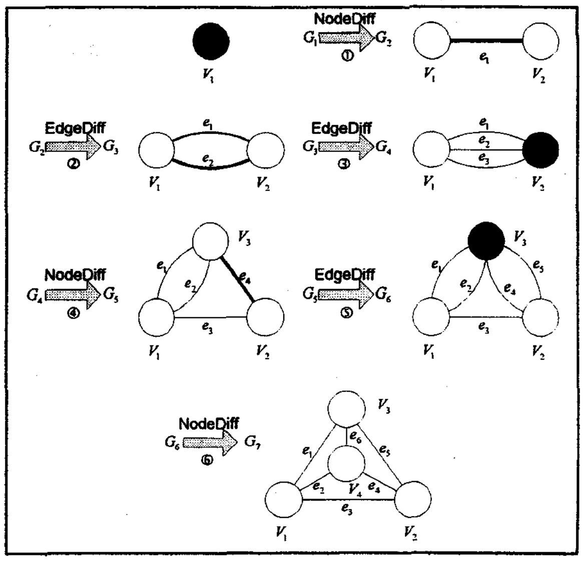

Figure 8. A graph evolution example

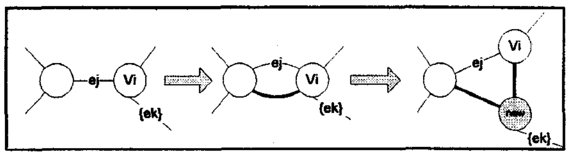

Figure 9. Modified edge differentiation operator

Figure 9 reveals the possibility that more than one differentiation operator can be used to compose *high-level differentiation operator*. Such combination may be very useful. The following demonstrates two examples, *mesh differentiation* and *branch differentiation* :

$$G_2 = MeshDiff(G_1, V_1, V_2) \overset{\Delta}{=} NodeDiff(NodeDiff(EdgeDiff(G_1, e_1), V_1, \{e_2\}), V_2, \{e_3\}) \quad (14)$$

and

$$G_2 = BranchDiff(G_1, V_1) \overset{\Delta}{=} NodeDiff(NodeDiff(G_1, V_1, \phi), V_1, \phi), \quad (15)$$

as shown in Figures 10 and 11. The former example is used in traditional SOM training and the latter is recently used in image processing. [CTC94]

After analyzing the basic differentiation operations of graph and depicting how to compose high-level differentiation operators, we will utilize these differentiation operators to design our Evolutionary SOM in next section.

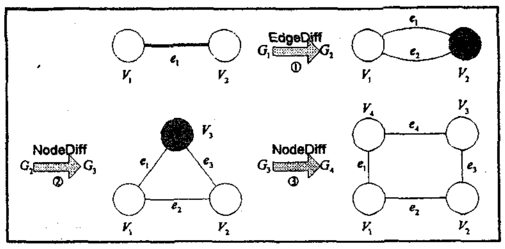

Figure 10. Mesh differentiation

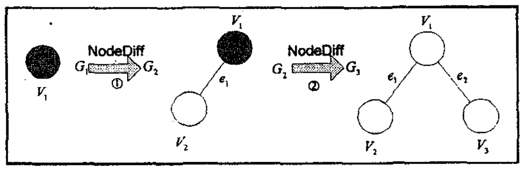

Figure 11. Branch differentiation

4. Design of Evolutionary SOM

Before designing the modules of ESOM, we first look at the modules in Kohonen's SOM. As mentioned above, the operation of SOM is divided into two phases: training phrase and production phrase, each of which is executed by dedicated modules as follows:

- **Initialization module**
- **Training module**
- **Production module**

Evolutionary SOM (ESOM) is an extension of Kohonen's SOM basically. The major difference of ESOM's modules is the evolution issue. In Nature's evolution, different species with different adaptability coexist in the same space. Those species having good performance are able to proliferate. In the implementation of ESOM, training module will generate many kinds of weights of SOMs (species); selection module controls the size of evolutionary SOM group and decides which species of neural net should stay in the next generation. Then these living SOMs get the chance to proliferate in the differentiation module. The whole operations of ESOM are shown in Figure 12.

Except that the training module of ESOM will produce more than one weight as species for evolution, the operations of initialization module, training module and production module in ESOM are just the same as those in Kohonen's SOM. There are two more modules in ESOM: selection module and differentiation module, whose operations are described in the following:

- **Selection module**
- **Differentiation module**

The new two modules form an *evolution phase* and the training module, selection module and differentiation constitute a learning cycle.

5. Example and Evaluation

The Neural Network Toolbox of Matlab is used as the foundation for implementing ESOM in this paper. As mentioned, the training module needs to be modified to train all species of the present generation. At the same time, another two additional modules should be included in the operation of ESOM.

With these functions, both traditional Kohonen's SOM and Evolutionary SOM are established. 1000 data points scattering around a cross are taken as our training data, as Figure 13a shown. The training results of two kinds of Kohonen's SOMs, Mesh-SOM and Linear-SOM, are shown in Figures 13b and 13c. On the other hand, the resultant ESOM is shown in Figure 13d.

Since the input patterns of training data (cross shape) are far from those in assigned Kohonen's SOMs (mesh and linear shapes), many neurons (18 and 17 neurons for each, SOM near 20%) fail to be data center of any cluster, therefore the overall structures of resultant SOMs fail to follow the training data pattern. On the other hand, the training result of ESOM shows much better and there is no unnecessary neuron.

Furthermore, the comparison between classification errors and the sizes of neural nets is made in Figure 14. Note that Kohonen's SOMs, no matter whether Mesh or Linear, are trained with different number of neurons, so the resultant curves are discrete. However, the training process of ESOM is continuous and then can be stopped when suitable energy function (error) is obtained.

6. Conclusion

Through analyzing the differentiation mechanism of graph, this paper proposes an algorithm of Evolutionary SOM. In such ESOM, both classification error and structurization error are taken into consideration. Two basic differentiation operations are developed for the evolution of graph. Similar to Nature's evolution, each generation of ESOM includes several species of neural nets and the survivors of competition will differentiate to the next generation. In addition to Kohonen's SOM toolbox in Matlab, two new modules are designed in the implementation of ESOM.

From the example in last section, Figure 13d shows that the resultant network of ESOM can track input data properly. Such advantage is not able to be depicted from the classification error in Figure 14. This is because the classification error does not include those neurons not representing any data cluster. Such phenomenon will be avoided by improving the definition of classification error in the near future. Moreover, the present evolution scheme only considers the proliferation of neurons. A pruning algorithm has to be added to eliminate some less representative neurons.

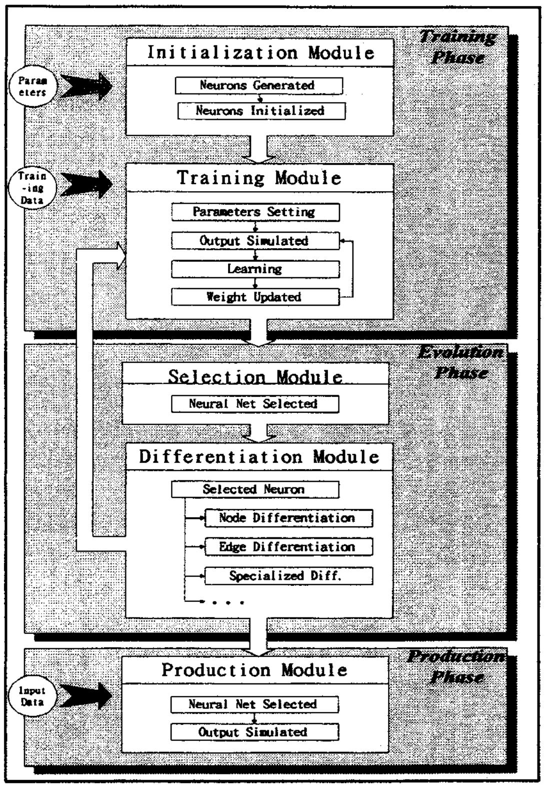

Figure 12. Detailed internal design of ESOM's modules

References

[1] T. Kohonen, *Self-Organization and Associative Memory*, Vol. 8, *Springer Series in Information Sciences.* Springer-Verlag, New York, 1984

[2] James A. Freeman, David M. Skapura, *Neural Networks algorithms, Applications and Programming Techniques*, Addison Wesley Pub. Co., 1991

[3] Willian Y. Huang and Richard P. Lippmann, "Neural net and traditional classifier", *Conf. Neural Information Processing Systems*, Denver, CO, November 1987

[4] H. Ritter and K. Schulten, "Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements", In R. Eckmiller and Christoph v.d. Malsburg, editors, *Neural Computers*. Springer-Verlag, Heidelberg, 1987, pp.393-406

[5] H. J. Ritter, T. M. Martinetz and K. J. Schulten, "Topology conserving maps for learning visuo-motor-coordination", *Neural Networks*, Vol.2, No.3, 1989, pp.159-168

[6] Tzi-Dar Chiueh, Tser-Tzi Tang, and Liang-Gee Chen, "Vector quantization using tree-structured self-organizing feature maps", *IEEE Journal on Selected Areas in Communications*, Vol.12, No.9, December 1994, pp1594-1599

[7] J. A. Kangas and T. K. Kohonen, "Variants of self-organizing maps", *IEEE Trans. Neural Networks*, Vol.1, 1990, pp.93-99

[8] B. Fritzke, "Let it grow-Self-organizing feature maps with problem dependent cell structure", in *Proc. Int. Conf. Artificial Neural Networks*, Vol 1, 1991, pp.403-408

[9] C.-X. Zhang, and D. A. Mlynski, "Mapping and hierarchical self-organizing neural network for VLSI placement", *IEEE Trans. Neural Networks*, Vol.8, No.2, March 1997, pp.299-314

[10] T. M. Martinetz, H. Ritter, and K. Schulten, "Three-dimensional neural net for learning visuomotor-coordination of a robot arm", *IEEE Trans. Neural Networks*, Vol.1, 1990, pp.131-136

[11] E. A. Riskin and R. M. Gray, "A greedy tree-growing algorithm for the design of variable rate vector quantizer", *IEEE Trans. Signal Processing*, Vol.39, 1991, pp.2500-2507

[12] Hans-Ulrich Bauer and Thomas Villmann, "Growing a hypercubical output space in a self-organizing feature map", *IEEE Trans. on Neural Networks*, Vol.8, No.2, March 1997, pp.218-226

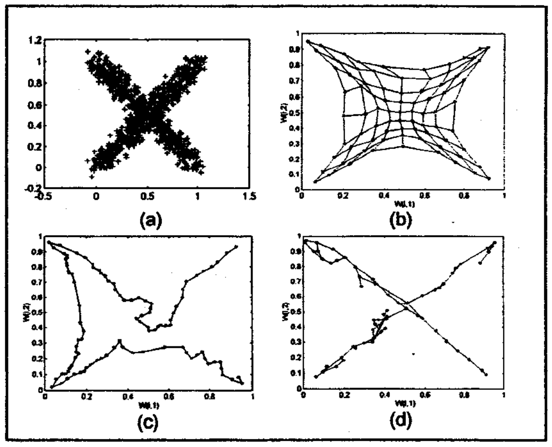

Figure 13. (a). Training data of 1000 points (b). Kohonen's SOM with 100-mesh structure (18 neurons have no data) (c). Kohonen's SOM with 100 linear structure (17 neurons have no data) (d). The 100th generation of ESOM

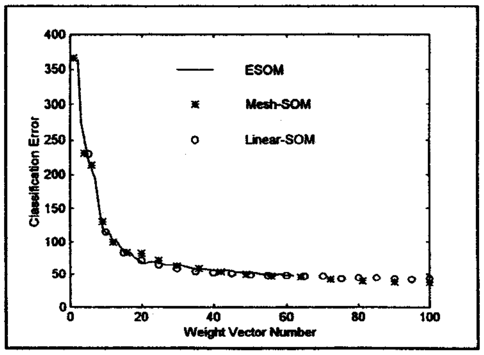

Figure 15 Classification errors of Kohonen's SOM and ESOM