Available online at www.sciencedirect.com
ScienceDirect
Neural Networks 20 (2007) 893–903
Neural Networks
www.elsevier.com/locate/neunet

An enhanced self-organizing incremental neural network for online unsupervised learning

Shen Furao[a,b,*], Tomotaka Ogura[b], Osamu Hasegawa[b]

[a] *The State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, PR China*
[b] *Imaging Science and Engineering Laboratory, Tokyo Institute of Technology, Japan*

Received 26 April 2006; received in revised form 12 July 2007; accepted 12 July 2007

Abstract

An enhanced self-organizing incremental neural network (ESOINN) is proposed to accomplish online unsupervised learning tasks. It improves the self-organizing incremental neural network (SOINN) [Shen, F., Hasegawa, O. (2006a). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks, 19*, 90–106] in the following respects: (1) it adopts a single-layer network to take the place of the two-layer network structure of SOINN; (2) it separates clusters with high-density overlap; (3) it uses fewer parameters than SOINN; and (4) it is more stable than SOINN. The experiments for both the artificial dataset and the real-world dataset also show that ESOINN works better than SOINN.

Keywords: Self-organizing incremental neural network (SOINN); Single-layer; High-density overlap; Stability; Online; Unsupervised learning

1. Introduction

One objective of unsupervised learning is clustering. For clustering, the popular k-means method is an often-used local search procedure that depends heavily on the initial starting conditions. Some improved k-means based methods such as the global k-means algorithm (Likas, Vlassis, & Verbeek, 2003) or the adaptive incremental LBG (Shen & Hasegawa, 2006b) impart high computational loads. The main difficulty of such k-means based methods is how to determine the number of clusters k in advance. Another shortcoming of k-means based methods is that they only can process isotropic clusters such as circles, spheres, etc. The EM (Jain, Duin, & Mao, 2000) method presents the disadvantageous possibility of convergence to a point on the boundary of parameter space with unbounded likelihood. Some other clustering methods such as single-link (Sneath & Sokal, 1988), complete-link (King, 1967), and CURE (Guha, Rastogi, & Shim, 1998) require large amounts of memory and impart high calculation loads.

Another objective of unsupervised learning is topology learning, specifically, the representation of the topology structure of a high-dimension data distribution. The self-organizing map (SOM) (Kohonen, 1982) requires predetermination of the network structure and network size. The combination of "competitive Hebbian learning" (CHL) and "neural gas" (NG) (Martinetz, Berkovich, & Schulten, 1996) also requires a prior decision about the network size. The salient disadvantage of growing neural gas (GNG) (Fritzke, 1995) is the permanent increase in the number of nodes.

Incremental learning addresses the ability of repeatedly training a network using new data without destroying the old prototype patterns. Incremental learning is useful in many applications. For example, if we intend to bridge the gap between the learning capabilities of humans and machines, we must consider which circumstances allow a sequential acquisition of knowledge. The fundamental issue for incremental learning is how a learning system can adapt to new information without corrupting or forgetting previously learned information: the so-called Stability-Plasticity Dilemma (Carpenter & Grossberg, 1988). Actually, GNG-U (Fritzke, 1997) deletes nodes that are located in regions that have low-input probability density. This criterion serves to follow a non-stationary input distribution, but the old learned prototype patterns are thereby

* Corresponding address: Imaging Science and Engineering Laboratory, Tokyo Institute of Technology, R2-52, 4259, Nagatsuta 226-8503 Midori-ku, Yokohama. Tel.: +81 45 924 5180; fax: +81 45 924 5175.
E-mail address: frshen@nju.edu.cn (S. Furao).

0893-6080/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.07.008

destroyed. Hamker (2001) proposes an extension to GNG to do some supervised incremental learning tasks, but it is unsuitable for unsupervised learning.

Shen and Hasegawa (2006a) proposed an incremental learning method called the self-organizing incremental neural network (SOINN) (Shen, 2006) to realize the unsupervised incremental learning task. In fact, SOINN is useful to process online non-stationary data, report a suitable number of classes, and represent the topological structure of input probability density. In Shen and Hasegawa (2006a), the authors compared SOINN to GNG, providing analytical comparisons of SOINN to other self-organizing neural networks. Empirical results show that SOINN learns the necessary number of nodes, uses fewer nodes than GNG, and obtains better results than GNG.

There are two main problems that remain unresolved with SOINN. (1) SOINN adopts a two-layer network. During online learning, the user must determine when to stop the learning of the first layer, and when to begin the learning of the second layer. (2) SOINN can separate clusters with very-low-density overlap. If a high-density overlap exists between clusters, they cannot work appropriately, and the clusters will link together to form one cluster. Here, high-density overlap means that we can visually detect the clusters separately by the naked eye, but plenty of samples exist in the common areas between clusters.

This paper presents some designed techniques to solve problems of SOINN. The proposed method is based on SOINN. Therefore, we call the proposed method enhanced self-organizing incremental neural network (ESOINN). ESOINN inherits all functions of SOINN, uses fewer parameters than SOINN, and solves some inherent problems of SOINN.

2. Overview of SOINN

A SOINN adopts a two-layer network. The first layer learns the density distribution of the input data and uses nodes and edges to represent the distribution. The second layer separates clusters by detecting the low-density area of input data, and uses fewer nodes than the first layer to represent the topological structure of input data. When the second-layer learning is finished, SOINN reports the number of clusters and gives typical prototype nodes of every cluster. It also adopts the same learning algorithm for the first and second layers. Fig. 1 shows a flowchart of the SOINN learning process.

When an input vector is given to SOINN, it finds the nearest node (winner) and the second-nearest node (second winner) of the input vector. It subsequently judges if the input vector belongs to the same cluster of the winner or second winner using the similarity threshold criterion. The first layer of SOINN adaptively updates the similarity threshold of every node because the input data distribution is unknown. If node i has neighbor nodes, the similarity threshold T_i is calculated using the maximum distance between node i and its neighboring nodes.

$$T_i = \max_{j \in N_i} \|\boldsymbol{W}_i - \boldsymbol{W}_j\|. \tag{1}$$

Therein, N_i is the set of neighbor nodes of node i and $\boldsymbol{W}_i$ is the weight vector of node i. A similarity threshold T_i is defined as the minimum distance between node i and other nodes in the network if node i has no neighbor nodes.

$$T_i = \min_{j \in N \setminus \{i\}} \|\boldsymbol{W}_i - \boldsymbol{W}_j\|. \tag{2}$$

Here, N is the set of all nodes.

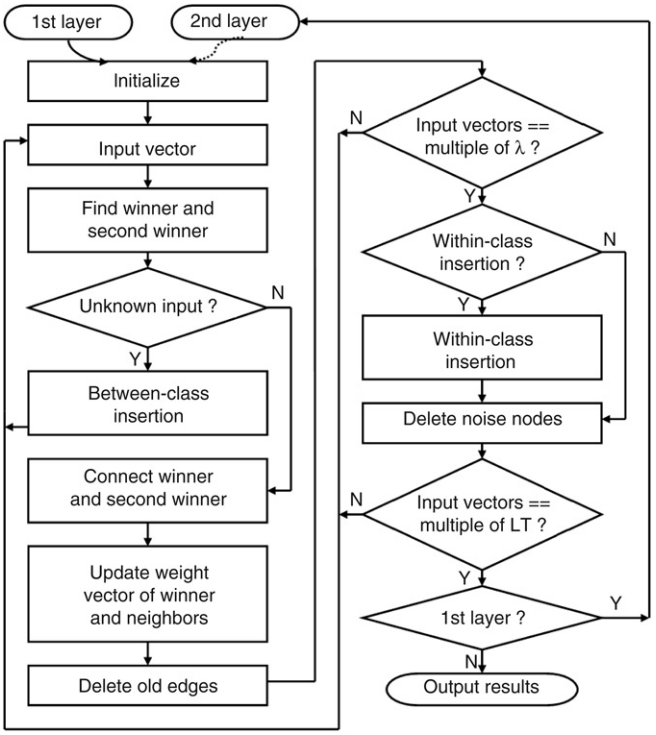

Fig. 1. Flowchart of SOINN.

The input vector will be inserted into the network as a new node to represent the first node of a new class if the distance between the input vector and the winner or second winner is greater than the similarity threshold of a winner or second winner. This insertion is called a between-class insertion because this insertion will engender the generation of a new class, even if the generated new class might be classified to some older class in the future.

If the input vector is judged as belonging to the same cluster of winner or second winner, and if no edge connects the winner and second winner, connect the winner and second winner with an edge, and set the 'age' of the edge as '0'; subsequently, increase the age of all edges linked to the winner by '1'.

Then, update the weight vector of the winner and its neighboring nodes. We use i to mark the winner node, and M_i to show the times for node i to be a winner. The change to the weight of winner $\Delta \boldsymbol{W}_i$ and change to the weight of the neighbor node $j (\in N_i)$ of i $\Delta \boldsymbol{W}_j$ are defined as $\Delta \boldsymbol{W}_i = \frac{1}{M_i}(\boldsymbol{W}_s - \boldsymbol{W}_i)$ and $\Delta \boldsymbol{W}_j = \frac{1}{100 M_i}(\boldsymbol{W}_s - \boldsymbol{W}_j)$, where $\boldsymbol{W}_s$ is the weight of the input vector.

If the age of one edge is greater than a predefined parameter $\text{age}_{\max}$, then remove that edge.

After λ learning iterations (λ is a timer), the SOINN inserts new nodes into the position where the accumulating error is extremely large. Cancel the insertion if the insertion cannot decrease the error. The insertion here is called within-class

insertion because the new inserted node is within the existing class; also, no new class will be generated during the insertion. Then SOINN finds the nodes whose neighbor is less than or equal to 1 and deletes such nodes based on the presumption that such nodes lie in the low-density area; we call such nodes "noise nodes."

In fact, because the similarity threshold of the first layer of SOINN is updated adaptively, the accumulation error will not be high. Therefore, the within-class insertion is only slightly successful. The within-class insertion for the first layer is unnecessary.

After LT learning iterations of the first layer, the learning results are used as the input for the second layer. The second layer of SOINN uses the same learning algorithm as the first layer. For the second layer, the similarity threshold is constant; it is calculated using the within-cluster distance and between-cluster distance (Shen & Hasegawa, 2006a). With a large constant similarity threshold, different from that of the first layer, the accumulation error for nodes of the second layer will be very high, and within-class insertion plays a major role in the learning process. With a large constant similarity threshold, the second layer also can delete some "noise nodes" that remain undeleted during first-layer learning. The experiment of the artificial dataset described in Shen and Hasegawa (2006a) shows details of the function of the second layer of SOINN.

3. Enhanced self-organizing incremental neural network (ESOINN)

Using the analysis described in Section 2, the two-layer SOINN presents the following shortcomings:

- It is difficult to choose when to halt first-layer learning and begin second-layer learning.
- For the second layer, if the learning results of the first layer were changed, all learned results of the second layer would be destroyed, thereby necessitating re-training of the second layer. The SOINN second layer is unsuitable for online incremental learning.
- Within-class insertion is needed for the second layer of SOINN. However, it requires many user-determined parameters.
- SOINN is not stable: it cannot separate high-density overlapped areas well.

To solve the above-mentioned shortcomings, we remove the second layer of SOINN and design some techniques to help the single-layer SOINN obtain even better clustering results than those of two-layer SOINN. We show a flowchart of ESOINN in Fig. 2.

Comparison of Fig. 2 to Fig. 1 reveals that ESOINN only adopts a single-layer network. For between-class insertion, ESOINN adopts the same scheme as SOINN. For building a connection between nodes, unlike SOINN, ESOINN adds a condition to judge if the connection is needed. After λ learning iterations, ESOINN separates nodes to different subclasses and deletes edges that lie in overlapped areas. ESOINN does not achieve within-class insertion because it adopts only a single-layer structure; within-class insertion is only slightly successful.

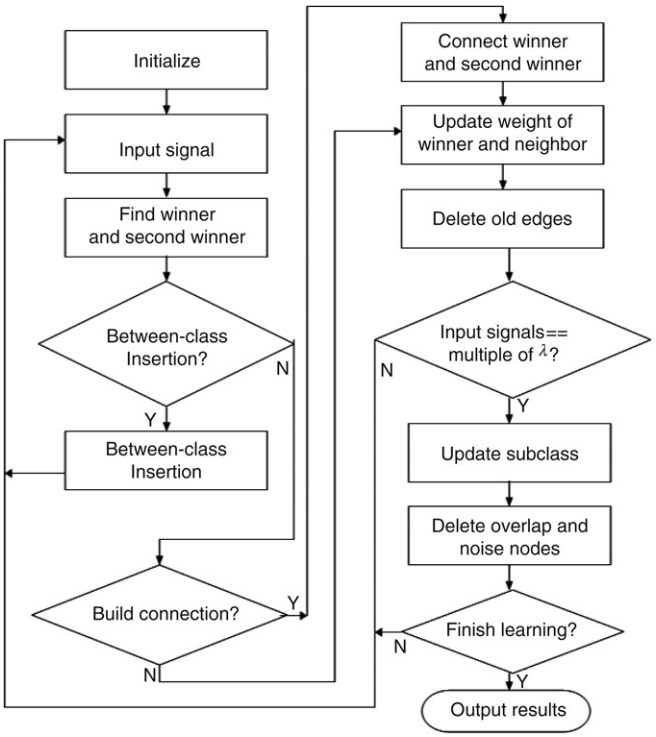

Fig. 2. Flowchart of ESOINN.

The removal of the second layer makes ESOINN more suitable for online or even life-long learning tasks than two-layer SOINN; it also obviates the difficult choice of when to stop first-layer learning and begin second-layer learning. The removal of within-class insertion eliminates five user-determined parameters. In the following section, we discuss how to process the overlapped area.

3.1. Overlap between classes

In this section, we define the node density, discuss a method to find the overlap between classes, and judge whether it is necessary to build a connection between a winner and a second winner. We note that density in the overlapped area is lower than that in the center part of a class.

3.1.1. Density of nodes

Node density is definable using the local accumulated number of samples: if many input samples are near the node, the node density is high; if few input samples are near the node, the density of this node is low. For that reason, during a period of learning, we count the times that a node has been a winner, and use this count as the node density. This "times of being a winner" definition for density is adopted by some methods such as SOINN. Although it sounds natural, this definition will engender the following problems:

(1) There will be numerous nodes that lie in the high-density area, meaning that, in the high-density area, the chance for a node to be a winner will not be considerably higher

than that in the low-density area. Consequently, we cannot simply use the "times of being winner" to measure the density.

(2) In incremental learning tasks, some nodes generated in earlier stages will not again be a winner for a long time. Using the "times of being winner" definition, such nodes might be judged as low-density nodes in a later learning stage.

In ESOINN, we use a new definition of density to solve the problems described above. The basic idea is the same as the local accumulated number of samples, but we define a "point" to take the place of "number", and use the mean of the accumulated point of a node to describe the density of that node. Unlike the "times of being a winner", which is only related to the special node itself, we consider the relationship between nodes when we calculate the point p of a node. First we calculate the mean distance $\bar{d}_i$ of node i from its neighbors.

$$\bar{d}_i = \frac{1}{m}\sum_{j=1}^{m} \|\boldsymbol{W}_i - \boldsymbol{W}_j\|. \tag{3}$$

In that equation, m is the number of neighbors of node i; $\boldsymbol{W}_i$ is the weight vector of node i.

Then calculate the "point" of node i as the following.

$$p_i = \begin{cases} \dfrac{1}{(1+\bar{d}_i)^2} & \text{if node } i \text{ is } winner \\ 0 \quad \text{if node } i \text{ is not } winner. \end{cases} \tag{4}$$

From the definition of "point" we know that if the mean distance of node i to its neighbors is large, then the number of nodes in this area is low; consequently, the distribution of nodes is sparse and the density in this area will be low. We thereby give low "points" to node i. If the mean distance $\bar{d}_i$ is small, it means that the number of nodes in this area is high, the density in this area will be high; we therefore give high "points" to node i. We add 1 in the denominator of the "point" definition to make the point value less than 1. For one iteration, we calculate only the "points" for node i when node i is the winner. The "points" of other nodes are 0 in this iteration. Therefore, for one iteration, the accumulated points for the winner will be changed, but the accumulated points for other nodes remain unchanged ('no change').

The accumulated points s_i are calculated with the sum of points for node i during a period of learning:

$$s_i = \sum_{j=1}^{n}\left(\sum_{k=1}^{\lambda} p_i\right). \tag{5}$$

Therein, λ is the number of input signals during one learning period; n indicates the learning period times (it can be calculated with LT/λ, where LT means total number of input signals).

Therefore, we give the mean accumulated point (density) of node i:

$$h_i = \bar{s_i} = \frac{1}{N}s_i = \frac{1}{N}\sum_{j=1}^{n}\left(\sum_{k=1}^{\lambda} p_i\right). \tag{6}$$

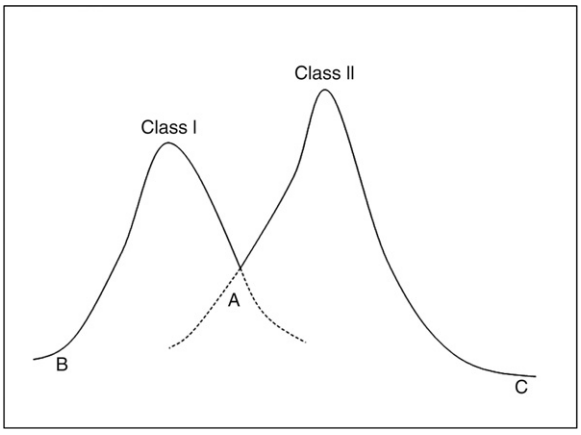

Fig. 3. Density distribution with the overlapped area.

In this equation, N represents the account of the period when accumulated points s_i are greater than 0. Note that N is not necessarily equal to n. We do not use n to take the place of N because, for incremental learning, during some periods of learning, the accumulated points s_i will be 0. If we use n to calculate the mean accumulated points, the density of some old learned nodes will decrease. Using N to calculate the mean accumulated points, even during the life-long span learning, the density of old learned nodes will remain unchanged if no new signals near the node are input to the system. However, for some applications, it is necessary that very old learned information be forgotten. In such cases, we need only to use n to take the place of N. Thereby, we can continue learning new knowledge and forget very old knowledge. In this paper, we specifically present incremental learning and hope that all learned knowledge is stored in the network. We use N to define the node density.

3.1.2. Find overlapped area between classes

Regarding the definition of density, the simplest way to find the overlapped area is to find the area with lowest density. Some methods such as GCS (Fritzke, 1994) and SOINN (Shen & Hasegawa, 2006a) adopt this technique to judge the overlapped area. However, this technique cannot ensure that the low-density area is exactly the overlapped area. For example, for some classes that follow a Gaussian distribution, at the class boundary, the density will be low. The overlap comprises some boundaries of overlapped classes. Therefore, the density of the overlap must be higher than that of the non-overlapped boundary area. For example, in Fig. 3, part A is shown as the overlapped area, but the density of part A is higher than that of either part B or part C. To solve this problem, ESOINN does not use the lowest-density rule, but rather designs a new technique to detect the overlapped area.

In SOINN, after a period of learning, if an overlap exists between classes, it is possible that all nodes of such classes link together to form one class. Our target of this section is to find the overlapped area in the composite class (which comprises many clusters), avoid building a connection between different classes, and thereby efficiently separate overlapped classes.

To detect the overlapped area, we first separate the composite class to several subclasses using the following rule:

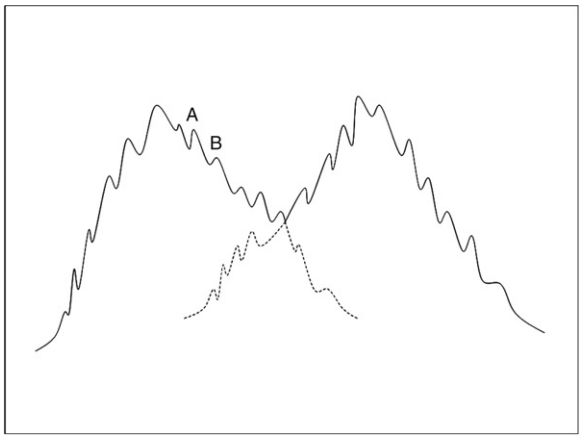

Fig. 4. Fluctuated distribution with overlapped area.

Algorithm 3.1. Separate a composite class into subclasses

(1) We call a node an apex of a subclass if the node has a local-maximum density. Find all apexes in the composite class, and give such apexes different labels.
(2) Classify all other nodes with the same subclass label as their apexes.
(3) Such nodes lie in the overlapped area if the connected nodes have different subclass labels.

For example, in Fig. 3, two classes exist: the nodes in Part A are very easily detected using Algorithm 3.1.

This method sounds reasonable, but for an actual task, it will lead to several problems. For example, two classes exist in Fig. 4, but the density distribution of nodes is not smoothing, but rather fluctuating (which might be attributable to noise or the fewer samples in the original dataset). With Algorithm 3.1, Fig. 4 will be separated into too many subclasses and many overlapped areas will be detected. We seek to smooth Fig. 4 to Fig. 3 before we separate the composite class to subclasses to solve this problem. The smoothing is realized by judging whether we need to combine two subclasses to form one united subclass.

Taking subclass A and subclass B in Fig. 4 as an example, assume that the apex density of subclass A is $A_{\max}$, and that the apex density of subclass B is $B_{\max}$. We combine subclasses A and B into one subclass if the following condition is satisfied.

If

$$\min(h_{\text{winner}}, h_{\text{secondwinner}}) > \alpha_A A_{\max} \tag{7}$$

or

$$\min(h_{\text{winner}}, h_{\text{secondwinner}}) > \alpha_B B_{\max}. \tag{8}$$

Here, the winner and second winner lie in the overlapped area between subclass A and B. Actually, α is a parameter that belongs to [0, 1] which can be calculated automatically using the threshold function:

$$\alpha_A = \begin{cases} 0.0 & \text{If } 2.0\text{mean}_A \geq A_{\max} \\ 0.5 & \text{If } 3.0\text{mean}_A \geq A_{\max} > 2.0\text{mean}_A \\ 1.0 & \text{If } A_{\max} > 3.0\text{mean}_A. \end{cases} \tag{9}$$

Therein, mean_A is the mean density of nodes in subclass A,

$$\text{mean}_A = \frac{1}{N_A} \sum_{i \in A} h_i. \tag{10}$$

N_A is the number of nodes in subclass A.

In summary, by separating the composite class into different subclasses and by combining the non-overlapped subclasses into one subclass, we can detect the overlapped area inside the composite class. After detecting the overlapped area, we remove the connection between nodes that belong to different subclasses, and separate the overlapped classes.

Algorithm 3.2. Build connection between nodes

(1) Connect the two nodes with an edge if the winner or second winner is a new node (it is not yet determined to which subclass the node belongs).
(2) Connect the two nodes with an edge if the winner and second winner belong to the same subclass.
(3) If the winner belongs to subclass A, the second winner belongs to subclass B. If (7) or (8) is satisfied, connect the two nodes, and combine subclasses A and B. Otherwise, do not connect the two nodes; if a connection exists between the two nodes, remove the connection.

Using Algorithm 3.2, if the winner and second winner belong to different subclasses, ESOINN provides a chance to connect these two nodes. We might want to do so to limit the influence of noise during separation of subclasses, and try to smooth the fluctuated subclasses. The subclasses can still be linked together (e.g., subclasses A and B in Fig. 4) if two subclasses are mistakenly separated.

Algorithm 3.2 shows that ESOINN results will be more stable than those of SOINN because, even with low-density overlap, SOINN will sometimes separate different classes correctly, and sometimes recognize different classes as one class. Using the smoothing technique, ESOINN can separate such overlapped classes stably; ESOINN also overcomes the possibility of over-separation.

3.2. Delete nodes caused by noise

To delete nodes caused by noise, SOINN removes nodes in regions with very low probability density. SOINN uses this strategy: if the number of input signals generated so far is an integer multiple of a parameter λ, remove those nodes with one or no topological neighbor. For one-dimensional input data and datasets with little noise, SOINN uses the local accumulated number of signals of the candidate-deleting nodes to control the deletion behavior. In addition, the two-layer network structure helps SOINN delete the nodes that are caused by noise.

For ESOINN, we adopt a nearly identical technique to that of SOINN to delete nodes resulting from noise: if the number of input signals generated so far is an integer multiple of a parameter λ, remove those nodes whose topological neighbors are two or fewer than two. The difference between ESOINN and SOINN is that we also delete those nodes with two topological neighbors. We use the local accumulated "point," and different

control parameters c_1 (for two-neighbor nodes) and c_2 (for one-neighbor nodes) to control the deletion behavior. We also delete some nodes with two neighbors because ESOINN only adopts a single-layer network, the adaptively changed similarity threshold renders "noise nodes" as possibly difficult to delete using the original SOINN strategy. As described in Section 2, SOINN can delete some remaining "noise nodes" using the second layer. For single-layer ESOINN, we must relax the condition for deleting nodes. We add a parameter c_1 to control the deletion process to avoid deleting some useful nodes.

3.3. Classify nodes to different classes

According to Shen and Hasegawa (2006a), if two nodes can be linked with a series of edges, we say that a path exists between the two nodes, i.e., given a series of nodes $x_i \in A, i = 1, 2, \ldots, n$, makes $(i, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n), (x_n, j) \in C$, we say that a "path" exists between node i and node j. Here, A is the node set and C is the connection set (or edge set).

Martinetz and Schulten (1994) prove some theorems and conclude that the competitive Hebbian rule is suitable for forming a path that preserves representations of a given manifold. The network connectivity structure corresponds to the induced Delaunay triangulation, which defines both a perfectly topology-preserving map and a path-preserving representation if the number of nodes is sufficient for obtaining a dense distribution. With the path-preserving representation, the competitive Hebbian rule allows determination of which parts of a given pattern manifold are separated and form different clusters. If two nodes are linked with one path, the two nodes belong to one cluster. Here, we use the same algorithm as SOINN to classify nodes.

Algorithm 3.3. Classify nodes to different classes

(1) Initialize all nodes as unclassified.
(2) Randomly choose one unclassified node i from node set A. Mark node i as classified and label it as class C_i.
(3) Search A to find all unclassified nodes that are connected to node i with a "path." Mark these nodes as classified and label them as the same class as node i.
(4) Go to Step (2) to continue the classification process until all nodes are classified if unclassified nodes exist.

3.4. Complete algorithm of ESOINN

As a summary, we give the complete algorithm of ESOINN here.

Algorithm 3.4. Enhanced self-organizing incremental neural network (ESOINN)

(1) Initialize node set A to contain two nodes with weight vectors chosen randomly from the input pattern. Initialize connection set C, $C \subset A \times A$, to the empty set.
(2) Input new pattern $\xi \in R^n$.
(3) Search the nearest node (winner) a_1, and the second-nearest node (second winner) a_2 by $a_1 = \arg\min_{a \in A} \|\xi - W_a\|$, $a_2 = \arg\min_{a \in A \setminus \{a_1\}} \|\xi - W_a\|$. If the distance between ξ and a_1 or a_2 is greater than similarity threshold T_{a_1} or T_{a_2}, the input signal is a new node, add it to A and go to Step 2 to process the next signal. Threshold T is calculated using formula (1) or (2).
(4) Increase the age of all edges linked with a_1 by 1.
(5) Use Algorithm 3.2 to judge if it is necessary to build a connection between a_1 and a_2.
   (a) If necessary: If an edge exists between a_1 and a_2, set the age of this edge 0; if no edge exists between a_1 and a_2, build a connection between a_1 and a_2, and initialize the age of this edge 0.
   (b) If that is not necessary: If an edge exists between a_1 and a_2, remove the connection between a_1 and a_2.
(6) Update the density of the winner using Eq. (6).
(7) Add 1 to a local accumulated number of signals M_{a_1}, $M_{a_1}(t+1) = M_{a_1}(t) + 1$.
(8) Adapt the weight vectors of the *winner* and its direct topological neighbors by a fraction $\epsilon_1(t)$ and $\epsilon_2(t)$ of the total distance to the input signal,

$$\Delta W_{a_1} = \epsilon_1(M_{a_1})(\xi - W_{a_1})$$
$$\Delta \boldsymbol{W}_i = \epsilon_2(M_{a_1})(\xi - \boldsymbol{W}_i) \text{ for all direct neighbors } i \text{ of } a_1.$$

   We adopt the same scheme as SOINN to adapt the learning rate over time by $\epsilon_1(t) = 1/t$ and $\epsilon_2(t) = 1/100t$.
(9) Find the edges whose ages are greater than a predefined parameter $\text{age}_{\max}$, then remove such edges.
(10) If the number of input signals generated so far is an integer multiple of parameter λ,
   (a) Update the subclass label of every node by Algorithm 3.1.
   (b) Delete nodes resulting from noise as follows:
      i. For all nodes in A, if node a has two neighbors, and $h_a < c_1 \sum_{j=1}^{N_A} h_j / N_A$, then remove the node a. N_A is the number of nodes in node set A.
      ii. For all nodes in A, if node a has one neighbor, and $h_a < c_2 \sum_{j=1}^{N_A} h_j / N_A$, then remove node a.
      iii. For all nodes in A, if node a has no neighbor, then remove node a.
(11) If the learning process is finished, classify nodes to different classes using Algorithm 3.3; then report the number of classes, output the prototype vectors of every class, and stop the learning process.
(12) Go to Step (2) to continue unsupervised online learning if the learning is not finished.

Using the algorithm shown above, we first find the winner and second winner of the input vector. Then we judge whether it is necessary to build a connection between the winner and second winner and connect them or remove the connection between them. After updating the density and weight of the winner, we update the subclass label of nodes after every λ times of learning. Then we delete nodes that are caused by noise. Here, noise is not dependent upon λ; it depends only on the input data. After learning is finished, we classify all nodes

Fig. 5. Artificial dataset I used for experimentation.

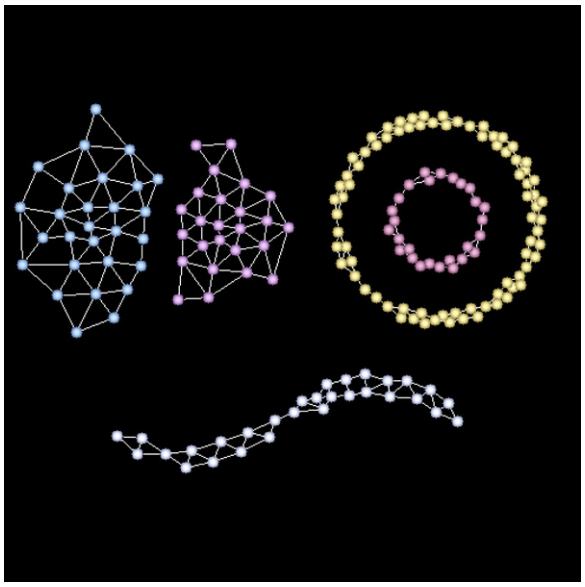

Fig. 6. ESOINN stationary results for dataset I.

to different classes. During the learning process, we need not store the learned input vectors; we say that this algorithm can realize online learning. After a period of learning, we input new data into ESOINN after ESOINN converges. The network will be grown to learn new information if the distance between new data and the winner or second winner is greater than the similarity threshold. If the distance between new data and winner or second winner is less than the similarity threshold, it means that the new data has been learned well, and no change occurs to the network. This process renders the algorithm suitable for incremental learning: we learn new information without destroying learned knowledge. Consequently, we need not retrain the network if learned knowledge is input to the system in the future.

4. Experiment

4.1. Artificial dataset

We first adopt the same artificial dataset I (Fig. 5) used in Shen and Hasegawa (2006a); it comprises two overlapped Gaussian distributions, two concentric rings, and a sinusoidal curve. There is 10% noise added to this dataset. With this dataset, under both stationary and non-stationary environments, the two-layer SOINN reports that five classes exist; it gives the topological structure of every class. The stationary environment dictates that the patterns are chosen randomly from the whole dataset to train the network online. A non-stationary environment dictates that the patterns are chosen sequentially, not randomly, from the five areas of original dataset I (see table 1 of Shen and Hasegawa (2006a)) to train the network online. A non-stationary environment is used to simulate the incremental online learning process.

We test the single-layer ESOINN using the same dataset. We set parameters as $\lambda = 100$, $\text{age}_{\max} = 100$, $c_1 = 0.001$ and $c_2 = 1.0$. Fig. 6 shows the result under a stationary environment; Fig. 7 depicts the result under a non-stationary environment.

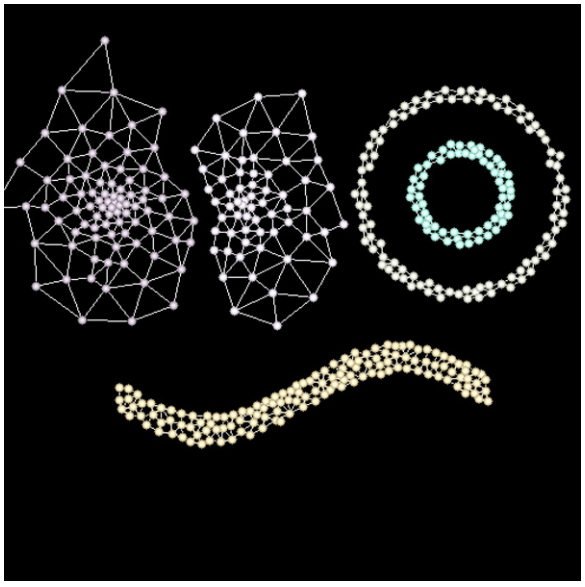

Fig. 7. ESOINN nonstationary results for dataset I.

Under both stationary and non-stationary environments, the ESOINN system reports that five classes exist and gives the topological structure of each class. The ESOINN system can realize the same functions of SOINN well for the same artificial dataset reported in Shen and Hasegawa (2006a).

Then we use artificial dataset II (Fig. 8), which comprises three overlapped Gaussian distributions, to test SOINN and ESOINN. We add 10% noise to this dataset. In the dataset, the density of the overlapped area is high, but the dataset can still be separated into three classes based solely on its appearance when plotted. Under the stationary environment, we choose samples from the original dataset randomly to train the network online. Under a non-stationary environment, the patterns are chosen sequentially, not randomly, from the three classes. At the first stage, we choose samples online from class 1 to train

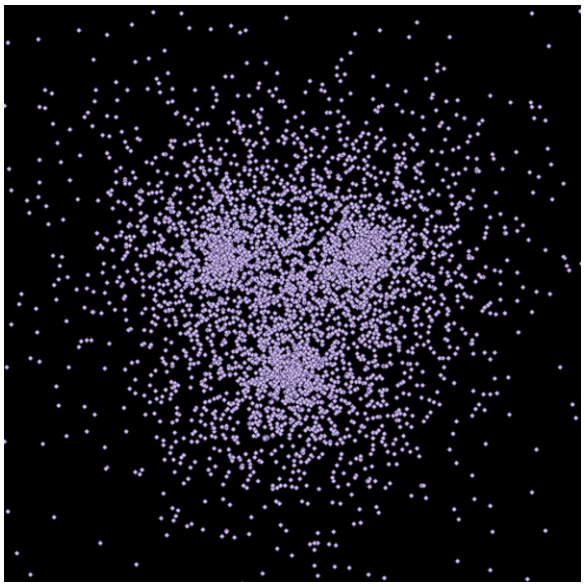

Fig. 8. Artificial dataset II used for experimentation.

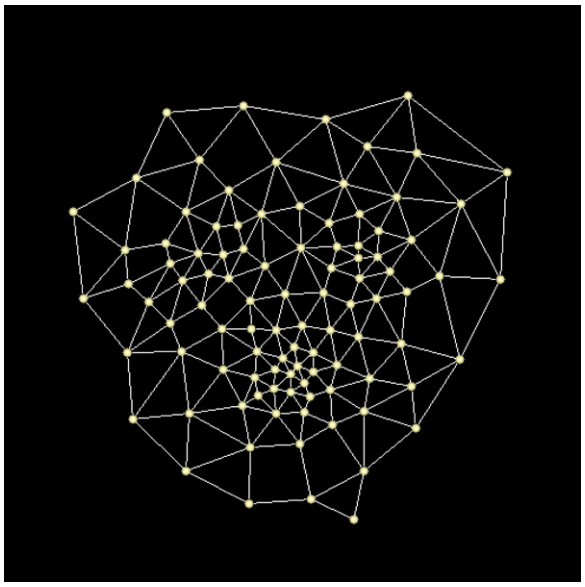

Fig. 10. SOINN nonstationary result for dataset II.

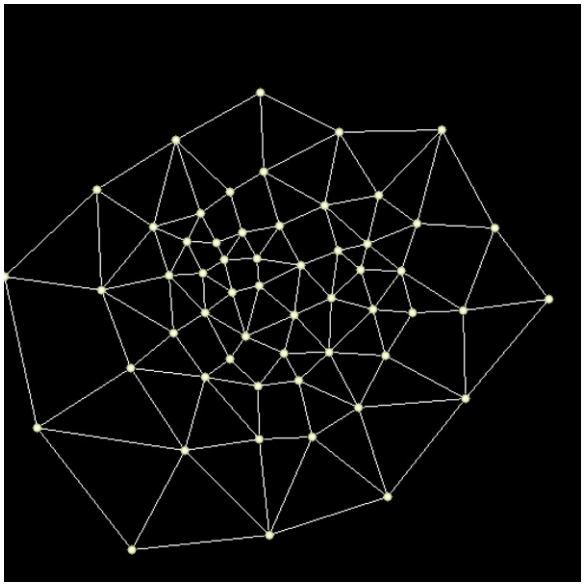

Fig. 9. SOINN stationary result for dataset II.

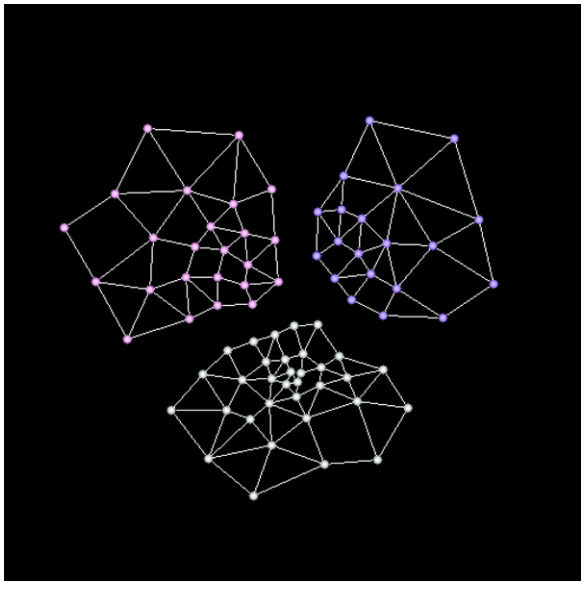

Fig. 11. ESOINN stationary result for dataset II.

the network; after learning 10,000 times, we choose samples only from class 2; and after another 10,000 times learning, we choose samples from class 3 and perform online training. After the learning process is finished, we classify nodes to different classes and report the results.

Figs. 9 and 10 depict SOINN results (The parameters are: $\lambda = 200$; $\text{age}_{\max} = 50$; $c = 1.0$; and other parameters, which have identical values to those described in Shen and Hasegawa (2006a)). For both stationary and non-stationary environments, SOINN cannot separate the three high-density overlapped classes. Figs. 11 and 12 show ESOINN results (The parameters are: $\lambda = 200$, $\text{age}_{\max} = 50$, $c_1 = 0.001$, and $c_2 = 1.0$). Using the proposed improvements for SOINN, ESOINN separated the three high-density overlapped classes. The system reports that three classes exist in the original dataset, and gives the topological structure of every class.

The experiment for artificial dataset II shows that ESOINN can separate high-density overlapped classes better than SOINN can.

4.2. Real-world data

For use as real-world data, we take 10 classes from the AT&T_FACE database (http://www.uk.research.att.com); each class includes 10 samples (Fig. 13). The size of each image is 92×112 pixels, with 256 gray levels per pixel. Feature vectors of such images are taken as follows. First, the original image, 92×112, is re-sampled to a 23×28 image using nearest-neighbor interpolation. Then the Gaussian method is used to

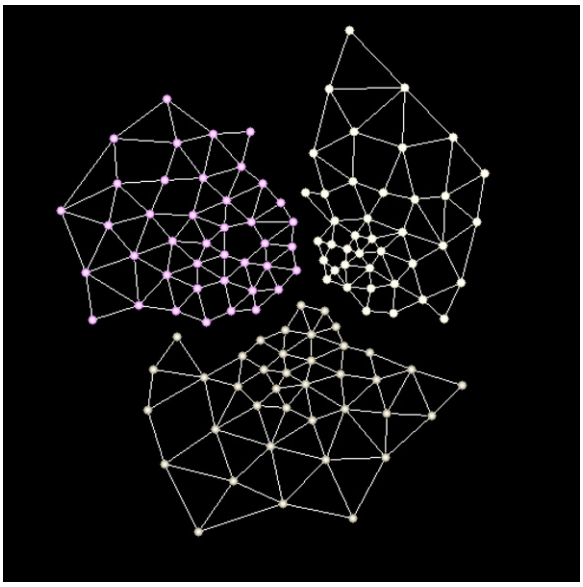

Fig. 12. ESOINN nonstationary result for dataset II.

smooth the 23 × 28 image with Gaussian width = 4, $\sigma = 2$ to obtain the 23 × 28 feature vectors (Fig. 14).

Under a stationary environment, samples are chosen randomly from the original dataset. For non-stationary environments, at the first stage, samples from class 1 are input to the system, after 1000 times training, samples from class 2 are input to the system, and so on. The parameters are set as $\lambda = 25$, $\text{age}_{\max} = 25$, $c_1 = 0.0$, and $c_2 = 1.0$. For both stationary and non-stationary environments, ESOINN reports 10 classes exist in the original dataset, and gives prototype vectors (nodes of network) of every class. We use such prototype vectors to classify the original training data to different classes, and report the recognition ratio. Compared to SOINN, we get nearly the same correct recognition ratio (90% for the stationary environment, and 86% for the non-stationary environment).

To compare the stability of SOINN and ESOINN, we performed 1000 times' training for both SOINN and ESOINN, and recorded the frequency of the number of classes. The upper panel of Fig. 15 portrays SOINN results, the lower panel of Fig. 15 depicts ESOINN results. The distribution of the number of classes for SOINN (2–16) is much wider than that of ESOINN (6–14); moreover, the frequency near number 10 of SOINN is much less than ESOINN, which reflects that ESOINN is more stable than SOINN.

For the second group of real-world data, we use the Optical Recognition of Handwritten Digits database (optdigits)

Fig. 13. Original face images.

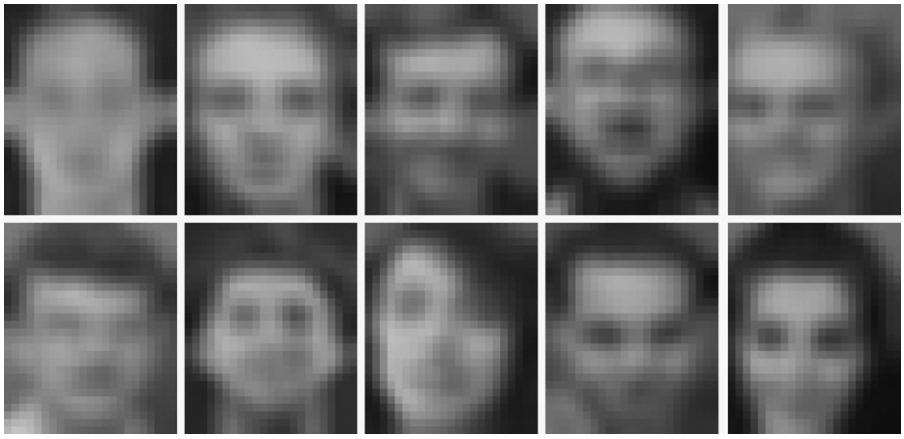

Fig. 14. Feature vectors of original face images.

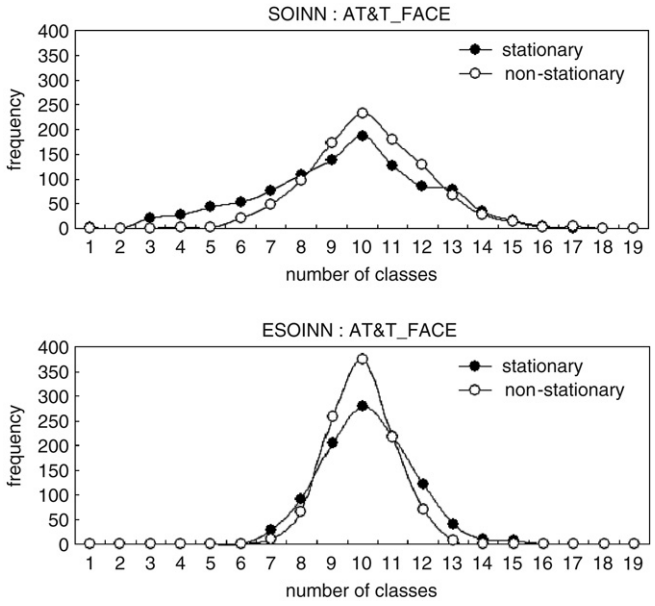

Fig. 15. Distribution for the number of classes.

(http://www.ics.uci.edu/~mlearn/MLRepository.html) to test SOINN and ESOINN. In this database, 10 classes (handwritten digits) exist among a total of 43 people, 30 contributed to the training set and a different 13 to the testing set; there are 3823 samples in all in the training set, along with a total of 1797 samples in the testing set. The sample dimensions are 64.

First, we use the training set to train SOINN (the parameters are $\lambda = 200$, $\text{age}_{\max} = 50$, and $c = 1.0$; the other parameters are the same as those in Shen and Hasegawa (2006a)). Under both a stationary environment and a non-stationary environment, SOINN reports that 10 classes exist. The upper panel of Fig. 16 portrays the typical prototype vectors of the SOINN result. Then we use the resultant SOINN prototype vectors to classify test data. The correct recognition ratio for the stationary environment is 92.2%; the correct recognition ratio for the non-stationary environment is 90.4%. We performed 100-times testing for SOINN. The number of classes is distributed from 6 to 13 for both stationary and non-stationary environments.

We used the training set to train ESOINN. Under both stationary and non-stationary environments ($\lambda = 200$, $\text{age}_{\max} = 50$, $c_1 = 0.001$, and $c_2 = 1.0$), ESOINN reports that 12 classes exist. The lower panel of Fig. 16 shows the typical prototype vectors of the ESOINN result. ESOINN separates the digit 1 into two classes, and separates the digit 9 into two classes because, from the original image of the digit, a great difference exists between samples of digit 1 and digit 1' (and a great difference exists between samples of digit 9 and digit 9'). SOINN deleted those nodes that are created by samples like digit 1' and 9'. ESOINN can separate such overlapped class digit 1 and 1' (digit 9 and 9'), the information of the original dataset is retained well.

We then use the resultant ESOINN prototype vectors to classify test data into different classes. The correct recognition ratio for the stationary environment is 94.3%, and the correct recognition ratio for the non-stationary environment is 95.8%. We performed 100-times' testing for ESOINN. The number of classes is distributed from 10 to 13 for both stationary and non-stationary environments.

From real-world data experiment of optdigits, we know that ESOINN can separate overlapped classes better than SOINN. ESOINN gets a higher recognition ratio, and works more stably than SOINN.

5. Conclusion

In this paper, we proposed an enhanced self-organizing incremental neural network (ESOINN), which is based on SOINN. The proposed method can realize all SOINN functions. Using single-layer network to take the place of two-layer network structure of SOINN, ESOINN can realize pure online incremental learning. By setting conditions for building a connection between nodes, ESOINN can separate high-density overlapped classes. In fact, ESOINN only adopts between-class insertion to realize incremental learning. For that reason, ESOINN easily realizes a solution and requires fewer parameters than SOINN; using some smoothing techniques, ESOINN is also more stable than SOINN.

Some other methods to separate overlapped areas exist, such as Learning Vector Quantization (LVQ). Such methods belong to supervised learning and must label all training samples. However, the acquisition of labeled training data is costly and time-consuming, whereas unlabelled samples are easily obtainable. Even if some supervised learning techniques can obviate the necessity of judging the overlap area, it is still very important to find some unsupervised learning method to avoid the use of labeled data. ESOINN (and SOINN) belongs to unsupervised learning, with the judging of overlap area, ESOINN can obtain satisfactory learning results. In addition, the finding of overlap area is simple. It requires merely a local maximum for determination of subclasses. That feature makes ESOINN extremely useful for some real tasks.

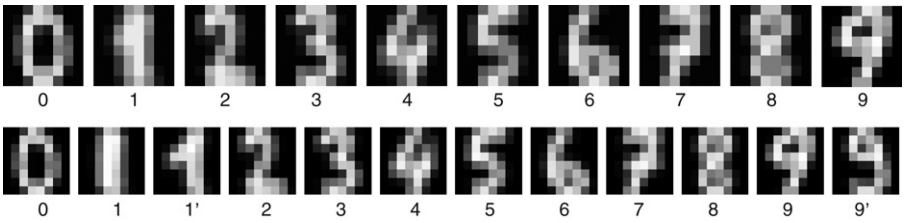

Fig. 16. Result of Optdigits dataset, the upper panel shows results of SOINN, the lower panel shows results of ESOINN.

Acknowledgment

This work was supported in part by the China NSF grant (no. 60573157).

References

Carpenter, G. A., & Grossberg, S. (1988). The art of adaptive pattern recognition by a self-organizing neural network. *IEEE Computer*, *21*, 77–88.

Fritzke, B. (1994). Growing cell structures — a self-organizing network for unsupervised and supervised learning. *Neural Networks*, *7*, 1441–1460.

Fritzke, B. (1995). A growing neural gas network learns topologies. In *Advances in neural information processing systems* (pp. 625–632).

Fritzke, B. (1997). A self-organizing network that can follow non-stationary distributions. In *Proceedings of ICANN-97* (pp. 613–618).

Guha, S., Rastogi, R., & Shim, K. (1998). Cure: An efficient clustering algorithm for large databases. In *Proceeding of ACM SIGMOD conference on management of data* (pp. 73–84).

Hamker, F. H. (2001). Life-long learning cell structures — continuously learning without catastrophic interference. *Neural Networks*, *14*, 551–573.

Jain, A. K., Duin, R., & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on PAMI*, *22*(1), 4–37.

King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association*, *69*(3), 86–101.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*, 59–69.

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global *k*-means clustering algorithm. *Pattern Recognition*, *36*, 451–461.

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1996). Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, *4*(4), 558–569.

Martinetz, T., & Schulten, K. (1994). Topology representing networks. *Neural Networks*, *7*(3), 507–522.

Shen, F. (2006). An algorithm for incremental unsupervised learning and topology representation. *Ph.D. thesis*. Tokyo Institute of Technology.

Shen, F., & Hasegawa, O. (2006a). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, *19*, 90–106.

Shen, F., & Hasegawa, O. (2006b). An adaptive incremental LBG for vector quantization. *Neural Networks*, *19*, 694–704.

Sneath, H. A., & Sokal, R. R. (1988). *Numerical Taxonomy*. London, UK: Freeman.