Available online at www.sciencedirect.com
SCIENCE DIRECT®
Neural Networks 19 (2006) 90–106
Neural Networks
www.elsevier.com/locate/neunet

An incremental network for on-line unsupervised classification and topology learning

Shen Furao[a,*], Osamu Hasegawa[b,c]

[a]Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology R2-52, 4259 Nagatsuta, Midori-ku, Yokohama 226-8503, Japan
[b]Imaging Science and Engineering Lab., Tokyo Institute of Technology
[c]PRESTO, Japan Science and Technology Agency (JST), Japan

Received 12 December 2003; accepted 11 April 2005

Abstract

This paper presents an on-line unsupervised learning mechanism for unlabeled data that are polluted by noise. Using a similarity threshold-based and a local error-based insertion criterion, the system is able to grow incrementally and to accommodate input patterns of on-line non-stationary data distribution. A definition of a utility parameter, the error-radius, allows this system to learn the number of nodes needed to solve a task. The use of a new technique for removing nodes in low probability density regions can separate clusters with low-density overlaps and dynamically eliminate noise in the input data. The design of two-layer neural network enables this system to represent the topological structure of unsupervised on-line data, report the reasonable number of clusters, and give typical prototype patterns of every cluster without prior conditions such as a suitable number of nodes or a good initial codebook.

Keywords: On-line unsupervised learning; Stationary environment; Non-stationary environment; Clustering; Topology representation

1. Introduction

One objective of unsupervised learning is construction of decision boundaries based on unlabeled training data. Unsupervised classification is also known as data clustering and is defined as the problem of finding homogeneous groups of data points in a given multidimensional data set (Jain & Dubes, 1988). Each of these groups is called a cluster and defined as a region in which the density of objects is locally higher than in other regions.

Clustering algorithms are classifiable into hierarchical clustering, partitional clustering, fuzzy clustering, nearest-neighbor clustering, artificial neural networks for clustering, etc. (Murty, Jain, & Flynn, 1999). Hierarchical clustering algorithms such as single-link (Sneath & Sokal, 1973), complete-link (King, 1967), and CURE (Guha, Rastogi, & Shim, 1998) usually find satisfiable clustering results but suffer from computational overload and the requirement for much memory space (Murty et al., 1999). Hierarchical clustering algorithms are therefore unsuitable for large data sets or on-line data. BIRCH is an extremely efficient hierarchical clustering algorithm (Zhang, Ramakrishnan, & Livny, 1996), but is properly applicable to data sets consisting only of isotropic clusters: a two-dimensional (2D) circle, or a three-dimensional (3D) sphericity, etc. Specifically, chain-like and concentric clusters are difficult to identify using BIRCH (Guha et al., 1998; Oyang, Chen, & Yang, 2001).

Most partitional clustering algorithms run in linear time and work on large data sets (Lin & Chen, 2002). The *k*-means algorithm, a conventionally used partitional clustering algorithms, suffers from deficiencies such as dependence on initial starting conditions (Lozano, Pena, & Larranaga, 1999) and a tendency to result in local minima. Likas, Vlassis, and Verbeek (2003) proposed a global *k*-means algorithm, an incremental approach to clustering that dynamically adds one cluster center at a time through a

* Corresponding author. Tel.: +81 45 924 5180; fax: +81 45 924 5175.
E-mail addresses: furaoshen@isl.titech.ac.jp (S. Furao), hasegawa@isl.titech.ac.jp (O. Hasegawa).
URL: http://www.isl.titech.ac.jp/hasegawalab/shen/index.htm.

0893-6080/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2005.04.006

deterministic global search consisting of N (the data set size) execution of the k-means algorithm from suitable initial positions. Compared to traditional k-means, this algorithm can obtain equivalent or better results, but it suffers from high computation load. The enhanced LBG algorithm proposed by Patane and Russo (2001) defines one parameter-utility of a codeword-to overcome the drawback of LBG algorithm: the dependence on initial starting condition. The main difficulties of such methods are how to determine the number of clusters k in advance and the limited applicability to data sets consisting only of isotropic clusters.

Some clustering methods combine features of hierarchical and partitional clustering algorithms, partitioning an input data set into sub-clusters and then constructing a hierarchical structure based on these sub-clusters (Murty & Krishna, 1981). Representing a sub-cluster as only one point, however, renders the multilevel algorithm inapplicable to some cases, especially when the dimension of a sub-cluster is on the same order as the corresponding final cluster (Lin & Chen, 2002).

Another possible objective of unsupervised learning can be described as topology learning: given a high-dimensional data distribution, find a topological structure that closely reflects the topology of the data distribution. Self-organizing map (SOM) models (Willshaw & von der Malsburg, 1976; Kohonen, 1982) generate mapping from high-dimensional signal space to lower-dimensional topological structure. The predetermined structure and size of Kohonen's model imply limitations on resulting mapping (Martinetz & Schulten, 1994). Methods that identify and repair topological defects are costly (Villmann, Der, Hermann, & Martinetz, 1997). A posterior choice of class labels for prototypes of the (unsupervised) SOM causes further problems: class borders are not taken into account in SOM and several classes may share common prototypes (Villmann, Schleif, & Hammer, 2003). As an alternative, a combination of competitive Hebbian learning (CHL) (Martinetz, 1993) and neural gas (NG) (Martinetz, Berkovich, & Schulten, 1993) is effective in constructing topological structure (Martinetz & Schulten, 1994). For each input signal, CHL connects the two closest centers by an edge, and NG adapts k nearest centers whereby k is decreasing from a large initial value to a small final value. Problems arise in practical application: it requires an a priori decision about network size; it must rank all nodes in each adaptation step; furthermore, once adaptation strength has decayed, the use of adaptation parameters 'freezes' the network, which thereby becomes unable to react to subsequent changes in signal distribution. Two nearly identical algorithms are proposed to solve such problems: growing neural gas (GNG) (Fritzke, 1995) and dynamic cell structures (Bruske & Sommer, 1995). Nodes in the network compete for determining the node with the highest similarity to the input pattern. Local error measures are gathered during the learning process to determine where to insert new nodes, and new node is inserted near the node with the highest accumulated error. The major drawbacks of these methods are their permanent increase in the number of nodes and drift of the centers to capture the input probability density (Hamker, 2001). Thresholds such as a maximum number of nodes predetermined by the user, or an insertion criterion depending on overall error or on quantization error are not appropriate, because appropriate figures for these criteria are unknown a priori.

For the much more difficult problems of non-stationary data distributions, on-line learning or life-long learning tasks, the above-mentioned methods are not suitable. The fundamental issue for such problems is how a learning system can adapt to new information without corrupting or forgetting previously learned information—the Stability-Plasticity Dilemma (Carpenter & Grossberg, 1988). Using a utility-based removal criterion, GNG-U (Fritzke, 1997) deletes nodes that are located in regions of low input probability density. GNG-U uses a network of limited size to track the distribution in each moment, and the target of GNG-U is to represent the actual state. Life-long learning (Hamker, 2001) emphasizes learning through the entire lifespan of a system. For life-long learning, the 'dead nodes' removed by GNG-U can be interpreted as a kind of memory that may be useful again, for example, when the probability distribution takes on a previously held shape. Therefore, those 'dead nodes' preserve the knowledge of previous situations for future decisions and play a major role in life-long learning. The GNG-U serves to follow a non-stationary input distribution, but the previously learned prototype patterns are destroyed. For that reason, GNG-U is unsuitable for life-long learning tasks.

Lim and Harrison (1997) propose a hybrid network that combines advantages of Fuzzy ARTMAP and probabilistic neural networks for incremental learning. Hamker (2001) proposes a life-long learning cell structure (LLCS) that is able to learn the number of nodes needed to solve a current task and to dynamically adapt the learning rate of each node separately. Both methods work for supervised on-line learning or life-long learning tasks, but how to process unsupervised learning remains controversial.

The goal of the present study is to design an autonomous learning system for unsupervised classification and topology representation tasks. The objective is to develop a network that operates autonomously, on-line or life-long, and in a non-stationary environment. The network grows incrementally, learns the number of nodes needed to solve a current task, learns the number of clusters, accommodates input patterns of on-line non-stationary data distribution, and dynamically eliminates noise in input data.

Below, we describe the proposed algorithm, and then use artificial and real-world data sets to illustrate the learning process and observe details. A comparison with typical incremental networks, GNG and GNG-U, elucidates the learning behavior.

2. Proposed method

For unsupervised on-line learning tasks, we separate unlabeled non-stationary input data into different classes without prior knowledge such as how many classes exist. We also intend to learn input data topologies. Briefly, the targets of the proposed algorithm are:

- To process on-line or life-long learning non-stationary data.
- Using no prior conditions such as a suitable number of nodes or a good initial codebook or knowing how many classes exist, to conduct unsupervised learning, report a suitable number of classes and represent the topological structure of input probability density.
- To separate classes with low-density overlap and detect the main structure of clusters that are polluted by noise.

To realize these targets, we emphasize the key aspects of local representation, insertion of new nodes, similarity threshold, adaptive learning rate, and deletion of low probability density nodes.

2.1. Overview of the proposed method

In this study, we adopt a two-layer neural network structure to realize our targets. The first layer is used to generate a topological structure of input pattern. We obtain some nodes to represent the probability density of an input pattern when we finish the first-layer learning. For the second layer, we use nodes identified in the first layer as the input data set. We report the number of clusters and give typical prototype nodes of every cluster when we finish the second-layer learning.

For unsupervised classification task, we must determine if an input sample belongs to previously learned clusters or to a new cluster. Suppose we say that two samples belong to the same cluster if the Euclidean distance between them is less than threshold distance T. If T is too large, all samples will be assigned to one cluster. If T is too small, each sample will form an isolated, singleton cluster. To obtain 'natural' clusters, T must be greater than the typical within-cluster distance and less than the typical between-cluster distance (Duda, Hart & Stork, 2001).

For both layers, we must calculate the threshold distance T. In the first layer, we set the input signal as a new node (the first node of a new cluster) when the distance between the signal and the nearest node (or the second nearest node) is greater than a threshold T that is permanently adapted to the present situation. In the second layer, we calculate the typical within-cluster distances and typical between-cluster distances based on those nodes generated in the first layer, then give a constant threshold distance T_c according to the within-cluster distances and between-cluster distances.

To represent the topological structure, in on-line or life-long learning tasks, growth is an important feature for decreasing task error and adapting to changing environments while preserving old prototype patterns. Therefore, the insertion of new nodes is a very useful contribution to plasticity of the Stability-Plasticity Dilemma without interfering with previous learning (stability). Insertion must be stopped to prohibit a permanent increase in the number of nodes and to avoid overfitting. For that reason, we must decide when and how to insert a new node within one cluster and when insertion is to be stopped.

For within-cluster insertion, we adopt a scheme used by some incremental networks (such as GNG (Fritzke, 1995), GCS (Fritzke, 1994)) to insert a node between node q with maximum accumulated error and node f, which is among the neighbors of q with maximum accumulated error. Current incremental networks (GNG, GCS) have no ability to learn whether further insertion of a node is useful or not. Node insertion leads to catastrophic allocation of new nodes. Here, we suggest a strategy: when a new node is inserted, we evaluate insertion by a utility parameter, the error-radius, to judge if insertion is successful. This evaluation ensures that the insertion of a new node leads to decreasing error and controls the increment of nodes, eventually stabilizing the number of nodes.

We adopt the competitive Hebbian rule proposed by Martinetz in topology representing networks (TRN) to build connections between neural nodes (Martinetz & Schulten, 1994). The competitive Hebbian rule can be described as: for each input signal, connect the two closest nodes (measured by Euclidean distance) by an edge. It is proved that each edge of the generated graph belongs to the Delaunay triangulation corresponding to the given set of reference vectors, and that the graph is optimally topology-preserving in a very general sense. In on-line or life-long learning, the nodes change their locations slowly but permanently. Therefore, nodes that are neighboring at an early stage might not be neighboring at a more advanced stage. It thereby becomes necessary to remove connections that have not been recently refreshed.

In general, overlaps exist among clusters. To detect the number of clusters precisely, we assume that input data are separable: the probability density in the centric part of every cluster is higher than the density in intermediate parts between clusters; and overlaps between clusters have low probability density. We separate clusters by removing those nodes whose position is in a region with very low probability density. To realize this, Fritzke (1994) designed an estimation to find the low probability-density region; if the density is below threshold η, the node is removed. Here, we propose a novel strategy: if the number of input signals generated so far is an integer multiple of a parameter, remove those nodes with only one or no topological neighbor. We infer that, if the node has only one or no neighbor, during that period, the accumulated error of this node has a very low probability of becoming maximum and the insertion of new nodes near this node is difficult: the probability density of the region containing the node is very low. However, for one-dimensional input data, the nodes form a chain, the above criterion will remove the boundary nodes repeatedly. In addition, if the input data contain little noise, it is not good to delete those nodes having only one neighbor. Thus, we use another parameter, the local accumulated number of signals of the candidate-deleting node, to control the deletion behavior. If this parameter is greater than an adaptive threshold, i.e. if the node is the winner for numerous signals, the node will not be deleted because the node does not lie in a low-density area. This strategy works well for removing nodes in low-density regions without added computation load. In addition, the use of this technique periodically removes nodes caused by noise because the probability density of noise is very low.

If two nodes can be linked with a series of edges, a path exists between the two nodes. Martinetz and Schulten (1994) prove some theorems and reach the conclusion that the competitive Hebbian rule is suitable for forming a path preserving representations of a given manifold. If the number of nodes is sufficient for obtaining a dense distribution, the connectivity structure of the network corresponds to the induced Delaunay triangulation that defines both a perfectly topology-preserving map and a path-preserving representation. In TRN, neural gas (NG) is used to obtain weight vectors $W_i \in M$, $i=1,\ldots,N$, M is the given manifold. The NG is an efficient vector quantization procedure. It creates a homogeneous distribution of the weight vectors W_i on M. After learning, the probability distribution of the weight vectors of the NG is

$$\rho(W_i) \sim P(W_i)^{\alpha} \tag{1}$$

With the magnification factor $\alpha = D_{\text{eff}}/(D_{\text{eff}}+2)$, the intrinsic data dimension is D_{eff} (Villmann, 2000). Therefore, with the path-preserving representation, the competitive Hebbian rule allows the determination of which parts of a given pattern manifold are separated and form different clusters. In the proposed algorithm, we adopt a scheme like neural gas with only nearest neighbor learning for the weight vector adaptation; the competitive Hebbian rule is used to determine topology neighbors. Therefore, we can use the conclusions of TRN to identify clusters, i.e. if two nodes are linked with one path, we say the two nodes belong to one cluster.

2.2. Complete algorithm

Using the analysis presented in Section 2.1, we give the complete algorithm here. The same algorithm is used to train both the first layer and the second layer. The difference between the two layers is that the input data set of the second layer is the nodes generated by the first layer. A constant similarity threshold is used in the second layer instead of the adaptive similarity threshold used in the first layer.

Notations to be used in the algorithm

- A node set, used to store nodes
- N_A number of nodes in A
- C connection set (or edge set), used to store connections (edges) between nodes
- N_C number of edges in C
- W_i n-dimension weight vector of node i
- E_i local accumulated error of node i; it is updated when node i is the nearest node (winner) from the input signal
- M_i local accumulated number of signals of node i; the number is updated when node i is the winner
- R_i inherited error-radius of node i; the error-radius of node i is defined by the mean of accumulated error, E_i/M_i. R_i serves as memory for the error-radius of node i at the moment of insertion. It is updated at each insertion, but only for affected nodes
- C_i cluster label. This variable is used to judge which cluster node i belongs to.
- Q number of clusters.
- T_i similarity threshold. If the distance between an input pattern and node i is larger than T_i, the input pattern is a new node.
- N_i set of direct topological neighbors of node i.
- L_i number of topological neighbors of node i.
- $\text{age}_{(i,j)}$ age of the edge that connects node i and node j.
- path 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.

Algorithm 2.1: basic algorithm

(1) Initialize node set A to contain two nodes, c_1 and c_2:

$$A = \{c_1, c_2\} \tag{2}$$

with weight vectors chosen randomly from the input pattern. Initialize connection set C, $C \subset A \times A$, to the empty set

$$C = \Phi \tag{3}$$

(2) Input new pattern $\xi \in R^n$.

(3) Search node set A to determine the winner s_1, and second-nearest node (second winner) s_2 by

$$s_1 = \arg\min_{c \in A} \|\xi - W_c\| \tag{4}$$

$$s_2 = \arg\min_{c \in A \setminus \{s_1\}} |\xi - W_c|| \tag{5}$$

If the respective distances separating ξ and s_1 or s_2 are greater than similarity thresholds T_{s_1} or T_{s_2}, the input signal is a new node; add the new node to A and go to step (2) to process the next signal, i.e. if $\|\xi - W_{s_1}\| > T_{s_1}$ or $\|\xi - W_{s_2}\| > T_{s_2}$, then $A = A \cup r$ and $W_r = \xi$

(4) If a connection between s_1 and s_2 does not exist already, create it and add it to connection set C.

$$C = C \cup (s_1, s_2) \tag{6}$$

Set the age of the connection between s_1 and s_2 to zero

$$\text{age}_{(s_1,s_2)} = 0 \tag{7}$$

(5) Increase the age of all edges emanating from s_1.

$$\text{age}_{(s_1,i)} = \text{age}_{(s_1,i)} + 1 (\forall i \in N_{s_1}) \tag{8}$$

(6) Add the Euclidian distance between the input signal and the winner to local accumulated error E_{s_1}

$$E_{s_1} = E_{s_1} + \|\xi - W_{s_1}\|. \tag{9}$$

(7) Add 1 to the local accumulated number of signals M_{s_1}:

$$M_{s_1} = M_{s_1} + 1. \tag{10}$$

(8) Adapt the weight vectors of the winner and its direct topological neighbors by fraction $\varepsilon_1(t)$ and $\varepsilon_2(t)$ of the total distance to the input signal

$$\Delta W_{s_1} = \varepsilon_1(t)(\xi - W_{s_1}) \tag{11}$$

$$\Delta W_i = \varepsilon_2(t)(\xi - W_i) \quad (\forall i \in N_{s_1}) \tag{12}$$

Here, we call $\varepsilon_1(t)$ the learning rate of the winner, and $\varepsilon_2(t)$ the learning rate of the neighbor.

(9) Remove edges with an age greater than a predefined threshold age_{dead}, i.e. if $(i,j) \in C$, and $\text{age}(i,j) > age_{\text{dead}} (\forall i,j \in A)$, then $C = C \backslash \{(i,j)\}$.

(10) If the number of input signals generated so far is an integer multiple of parameter λ, insert a new node and remove nodes in low probability density as follows:

- Determine node q with maximum accumulated error E:

$$q = \arg\max_{c \in A} E_c \tag{13}$$

- Determine, among the neighbors of q, node f with maximum accumulated error:

$$f = \arg\max_{c \in N_q} E_c \tag{14}$$

- Add new node r to the network and interpolate its weight vector from q and f:

$$A = A \cup \{r\}, \quad W_r = (W_q + W_f)/2.0 \tag{15}$$

- Interpolate accumulated error E_r, accumulated number of signals M_r, and the inherited error-radius R_r from E_q, E_f, M_q, M_f, and R_q, R_f by:

$$E_r = \alpha_1(E_q + E_f) \tag{16}$$

$$M_r = \alpha_2(M_q + M_f) \tag{17}$$

$$R_r = \alpha_3(R_q + R_f) \tag{18}$$

- Decrease accumulated error variables of q and f by fraction β

$$E_q = \beta E_q, \quad E_f = \beta E_f \tag{19}$$

- Decrease the accumulated number of signal variables of q and f by fraction γ

$$M_q = \gamma M_q, \qquad M_f = \gamma M_f \tag{20}$$

- Judge whether or not insertion is successful. If the error-radius is larger than the inherited error-radius R_i ($\forall i \in \{q,r,f\}$), in other words, if insertion cannot decrease the mean error of this local area, insertion is not successful; else, update the inherited error-radius. I.e. if $E_i/M_i > R_i$ ($\forall i \in \{q,r,f\}$), insertion is not successful, new node r is removed from set A, and all parameters are restored; else, $R_q = E_q/M_q$, $R_f = E_f/M_f$, and $R_r = E_r/M_r$.
- If insertion is successful, insert edges connecting new node r with nodes q and f, and remove the original edge between q and f.

$$C = C \cup \{(r,q),(r,f)\} \tag{21}$$

$$C = C \backslash \{(q,f)\} \tag{22}$$

- For all nodes in A, search for nodes having only one neighbor, then compare the accumulated number of signals of these nodes with the average accumulated number of all nodes. If a node has only one neighbor and the accumulated number of signals is less than an adaptive threshold, remove it from the node set, i.e. if $L_i = 1$ ($\forall i \in A$) and $M_i < c\sum_{j=1}^{N_A} M_j/N_A$, then $A = A \backslash \{i\}$. Here, c is determined by the user and $1 \geq c > 0$. If much noise exists in the input data, c will be larger and vice versa.
- For all nodes in A, search for isolated nodes, then delete them, i.e. if $L_i = 0$ ($\forall i \in A$), then $A = A \backslash \{i\}$.

(11) After a long constant time period LT, report the number of clusters, output all nodes belonging to different clusters. Use the following method to classify nodes into different classes:

- Initialize all nodes as unclassified.
- Loop: Randomly choose one unclassified node i from node set A. Mark node i as classified and label it as class C_i.
- Search A to find all unclassified nodes connected to node i with a 'path.' Mark these nodes as classified and label them as the same class as node i (C_i).
- If unclassified nodes exist, go to Loop to continue the classification process until all nodes are classified.

(12) Go to step (2) to continue the on-line unsupervised learning process.

2.3. Parameter discussion

We determine parameters in Algorithm 2.1 as follows.

2.3.1. Similarity threshold T_i of node i

As discussed in Section 2.1, similarity threshold T_i is a very important variable. In step (3) of Algorithm 2.1, the threshold is used to judge if the input signal belongs to previously learned clusters or not.

For the first layer, we have no prior knowledge of input data and therefore adopt an adaptive threshold scheme. For every node i, the threshold T_i is adopted independently. First, we assume that all data come from the same cluster; thus, the initial threshold of every node will be $+\infty$. After a period of learning, the input pattern is separated into different small groups; each group comprises one node and its direct topological neighbors. Some of these groups can be linked to form a big group; the big groups are then separated from each other. We designate such big groups as clusters. The similarity threshold must be greater than within-cluster distances and less than between-cluster distances. Based on this idea, we calculate similarity threshold T_i of node i with the following algorithm.

Algorithm 2.2: Calculation of similarity threshold T for the first layer.

(1) Initialize the similarity threshold of node i to $+\infty$ when node i is generated as a new node.
(2) When node i is a winner or second winner, update similarity threshold T_i by
- If the node has direct topological neighbors ($L_i > 0$), T_i is updated as the maximum distance between node i and all of its neighbors.

$$T_i = \max_{c \in N_i} \|W_i - W_c\| \tag{23}$$

- If node i has no neighbors ($L_i = 0$), T_i is updated as the minimum distance of node i and all other nodes in A.

$$T_i = \min_{c \in A \setminus \{i\}} \|W_i - W_c\| \tag{24}$$

For the second layer, the input data set contains the results of the first layer. After learning of the first layer, we obtain coarse clustering results and a topological structure. Using this knowledge, we calculate the within-cluster distance d_w as

$$d_w = \frac{1}{N_C} \sum_{(i,j) \in C} \|W_i - W_j\|, \tag{25}$$

and calculate the between-cluster distance $d_b(C_i, C_j)$ of clusters C_i and C_j as

$$d_b(C_i, C_j) = \max_{i \in C_i, j \in C_j} \|W_i - W_j\| \tag{26}$$

That is, the within-cluster distance is the mean distance of all edges, and the between-cluster distance is the minimum distance between two clusters. With d_w and d_b, we can give a constant threshold T_c for all nodes.

The threshold must be greater than the within-cluster distance and less than the between-cluster distances. Influenced by overlap or noise, some between-cluster distances are less than within-cluster distances, so we use the following algorithm to calculate the threshold distance T_c for the second layer.

Algorithm 2.3: Calculation of similarity threshold T for the second layer.

(1) Set T_c as the minimum between-cluster distance.

$$T_c = d_b(C_{i_1}, C_{j_1}) = \min_{k,l=1,\ldots,Q, k \neq l} d_b(C_k, C_l) \tag{27}$$

(2) If T_c is less than within-cluster distance d_w, set T_c as the next minimum between-cluster distance.

$$T_c = d_b(C_{i_2}, C_{j_2}) = \min_{k,l=1,\ldots,Q, k \neq l, k \neq i_1, l \neq j_1} d_b(C_k, C_l) \tag{28}$$

(3) Go to step (2) to update T_c until T_c is greater than d_w.

2.3.2. Adaptive learning rate

In step (8) of Algorithm 2.1, the learning rate $\varepsilon(t)$ determines the extent to which the winner and the neighbors of the winner are adapted towards the input signal.

A constant learning rate is adopted by GNG and GNG-U, i.e. $\varepsilon(t) = c_0$, $1 \geq c_0 > 0$. With this scheme, each reference vector W_i represents an exponentially decaying average of those input signals for which the node i has been a winner. However, the most recent input signal always determines a fraction c_0 of the current W_i. Even after numerous iterations, the current input signal can cause a considerable change in the reference vector of the winner.

Ritter et al. (1991) proposed a decaying adaptation learning rate scheme. This scheme was adopted in the neural gas (NG) network (Martinetz et al., 1993). The exponential decay scheme is

$$\varepsilon(t) = \varepsilon_i (\varepsilon_f / \varepsilon_i)^{t/t_{\max}} \tag{29}$$

wherein ε_i and ε_f are the initial and final values of the learning rate and $t_{\max}$ is the total number of adaptation steps. This method is less susceptible to poor initialization. For many data distributions, it gives a lower mean-square error. However, we must choose parameter ε_i and ε_f by 'trial and error' (Martinetz et al., 1993). For on-line or life-long learning tasks, the total number of adaptation steps $t_{\max}$ is not available.

In this study, we adopt a scheme like k-means to adapt the learning rate over time by

$$\varepsilon_1(t) = \frac{1}{t}, \quad \varepsilon_2(t) = \frac{1}{100t} \tag{30}$$

Here, time parameter t represents the number of input signals for which this particular node has been a winner thus far, i.e. $t = M_i$. This algorithm is known as k-means. The node is always the exact arithmetic mean of the input signals it has been a winner for.

This scheme is adopted because we are hopeful of making the position of the node more stable by decreasing the learning rate when the node becomes a winner for more and more input patterns. After the network size becomes stable, the network is fine tuned by *stochastic approximation* (Wasan, 1969). This approximation denotes a number of adaptation steps with a strength $\varepsilon(t)$ decaying slowly, but not too slowly, i.e. $\sum_{t=1}^{\infty} \varepsilon(t) = \infty$, and $\sum_{t=1}^{\infty} \varepsilon^2(t) < \infty$. The harmonic series, Eq. (30), satisfies the conditions.

2.3.3. Decreasing rate of accumulated variables E (error) and M (number of signals)

When insertion between node q and f happens, how do we decrease accumulated error E_q, E_f and the accumulated number of signals M_q, M_f? How do we allocate the accumulated error E_r and the accumulated number of signals M_r to new node r?

In Fig. 1, the Voronoi regions of node q and f before insertion are shown at left, and Voronoi regions belonging to q, f, and new node r after insertion are shown at right. We assume that signals in these Voronoi regions are distributed uniformly. Comparing left to right reveals that one-fourth of the accumulated number of signals of q and f are reallocated to new node r, and that three-fourths of the accumulated number of signals remaining for q and f. For accumulated error, it is reasonable to assume that error attributable to $V1$ is double the error caused by $V2$ for node q. Consequently, after insertion, the accumulated error of q and f is two-thirds of the accumulated error before insertion. We also assume that the error to r caused by $V1$ is equal to the error to q attributable to $V2$. Consequently, the reallocated accumulated error for r is one-sixth of E_q and E_f. Based on the above analysis, the rate of decrease becomes: $\alpha_1 = 1/6$, $\alpha_2 = 1/4$, $\alpha_3 = 1/4$, $\beta = 2/3$, and $\gamma = 3/4$.

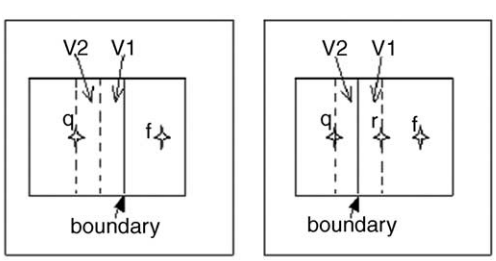

Fig. 1. Voronoi regions of nodes (Left, before insertion; Right, after insertion).

The above analysis is based on the supposition that signals are uniformly distributed over Voronoi regions. This supposition might be untenable for some tasks. For that reason, the above parameter set is not the optimal choice for such tasks. Fortunately, the choice of these parameters is not sensitive. In Section 3, we use different parameter sets to test an artificial data set and achieve nearly identical results. With the same parameter set, the system also works well for different real-world data experiments such as face recognition and vector quantization for different images.

3. Experiment

3.1. Artificial data set

We conducted our experiment on the data set shown in Fig. 2. An artificial 2-D data set is used to take advantage of its intuitive manipulation, visualization, and resulting insight into system behavior. The data set is separated into five parts: A, B, C, D, and E. Data sets A and B satisfy 2-D Gaussian distribution. The C and D data sets are a famous single-link example. Finally, E is sinusoidal and separated into E1, E2, and E3 to clarify incremental properties. We also add random noise (10% of useful data) to the data set to simulate real-world data. As shown in Fig. 2, overlaps exist among clusters; noise is distributed over the entire data set. As stated in (Guha et al., 1998), neither BIRCH nor single-link clustering can correctly partition such a data set. The CSM algorithm proposed by Lin and Chen (2002) partially solves this problem, but it cannot eliminate noise, and noise forms new clusters.

This experiment compares our proposed method with typical incremental network GNG (Fritzke, 1995) and GNG-U (Fritzke, 1997) to demonstrate the advantages of the proposed method. In all experiments, we set parameters as $\lambda = 100$, $age_{dead} = 100$, and $c = 1$. For GNG and GNG-U, the maximum number of nodes is predefined as 300;

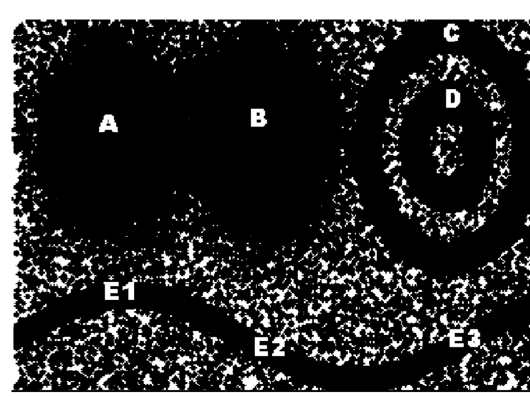

Fig. 2. 2D artificial data set used for the experiment.

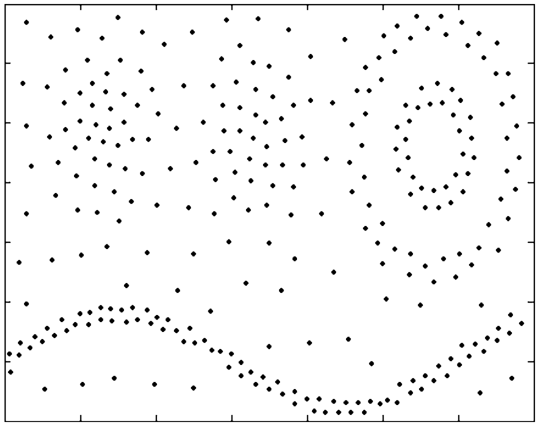

Fig. 3. Randomly input samples from a stationary environment. GNG results: 1 cluster.

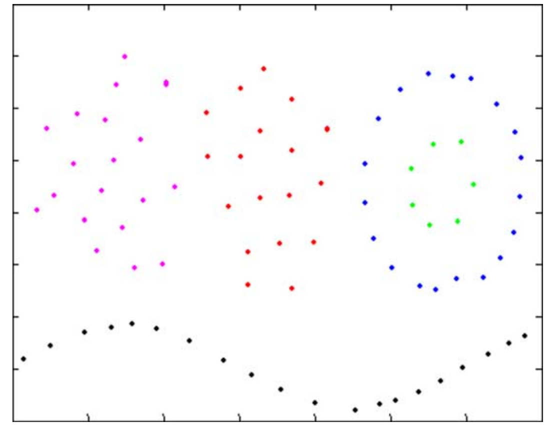

Fig. 5. Randomly input samples from a stationary environment. Results of proposed method: second layer, five clusters.

the parameter sets in (Fritzke, 1995) and (Fritzke, 1997) are adopted for other parameters.

3.1.1. Experiment in a stationary environment

First, we use Fig. 2 as a stationary data set to compare our algorithm with GNG; 100,000 patterns are chosen randomly from areas A, B, C, D, and E. Topological results of GNG and the proposed method are shown in Figs. 3–5. For stationary data, GNG can represent the topological structure, but it is affected by noise and all nodes are linked to form one cluster (Fig. 3). The first-layer of the proposed method partially eliminates the influence of noise and separates original data set to some different clusters (Fig. 4). The second layer of the proposed method efficiently represents the topology structure and gives the number of clusters and typical prototype nodes of every cluster (Fig. 5) .

3.1.2. Experiment in non-stationary environment

We simulate on-line learning through the use of the following paradigm: from step 1 to 20,000, patterns are chosen randomly from area A. At step 20,001, the environment changes and patterns from area B are chosen. At step 40,001 the environment changes again, etc. Table 1 details specifications of the test environments. The environment changes from I to VII. In each environment, areas used to generate patterns are marked as '1'; other areas are marked as '0.' For each environment, we add 10% noise to test data and noise is distributed over the whole data space.

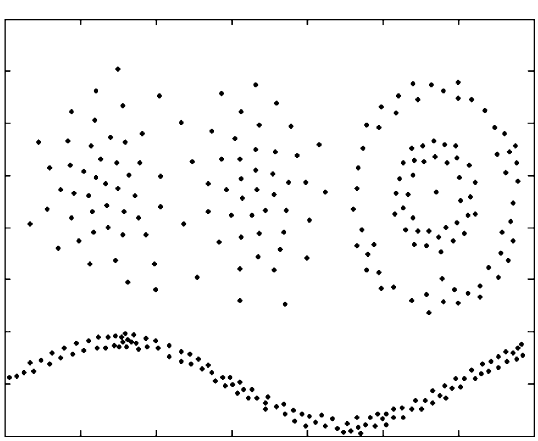

Fig. 4. Randomly input samples from a stationary environment. Results of proposed method: first layer, three clusters.

Before we test our proposed algorithm, we use GNG and GNG-U to conduct this experiment. We show the last results for GNG and GNG-U in Figs. 6 and 7, and do not report intermediate results. These results show that GNG cannot represent the topological structure of non-stationary on-line data well. GNG-U deletes all old learned patterns and only represents the structure of new input patterns. Neither method eliminates noise. In GNG-U results, for example, nodes beyond area E3 are all attributable to noise distributed over the whole space (Fig. 7).

Fig. 8 shows the first-layer results of the proposed method. After learning in one environment, we report intermediate topological structures. Environments I and II test isotropic data sets A and B, and an overlap exists between A and B. In environment I, the data set of the 2-D Gaussian distribution is tested. This system represents the Gaussian distribution structure very well. In environment II from 20,001 to 40,000 steps, probability changes to zero in area A. Remaining nodes of area A, often called 'dead nodes,' play a major role in on-line or life-long learning. They preserve the knowledge

Table 1
Experiment environments for on-line learning

Area	Environment						
	I	II	III	IV	V	VI	VII
A	1	0	1	0	0	0	0
B	0	1	0	1	0	0	0
C	0	0	1	0	0	1	0
D	0	0	0	1	1	0	0
E1	0	0	0	0	1	0	0
E2	0	0	0	0	0	1	0
E3	0	0	0	0	0	0	1

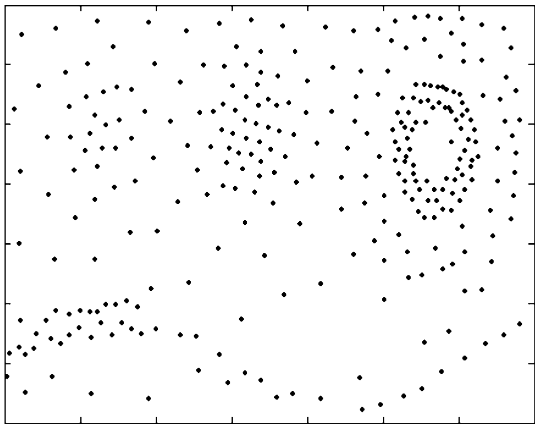

Fig. 6. Sequentially input samples from non-stationary environments. GNG results.

of previous situations for future decisions. In environment III, 40,001 to 60,000 steps, the reappearance of area A does not raise error and knowledge is preserved completely. Therefore, nearly no insertion happens and most nodes remain at their original positions. Environments III and IV test a difficult situation (data sets such as area C and area D). The system works well, removing noise between areas and separating C and D. Environments V, VI, and VII test a complicated artificial shape (area E). E is separated into three parts and data come to the system sequentially, not randomly. We find nodes of area E increasing following the change of environment from V to VI and VII, but all nodes are linked to form the same class.

Fig. 8 is used as the input data set of the second layer and a constant similarity threshold based on Fig. 8 is calculated. Fig. 9 is the result of the second layer. It reports the number of clusters and gives prototype nodes of every cluster.

Figs. 4 and 8 show that the first layer of the proposed method processes stationary and non-stationary environments well. It detects the main structure from original data, which is polluted by noise. It controls the number of nodes needed for the current task. However, some nodes generated by noise remain in results of the first layer, and the nodes are too many to represent the topology structure efficiently. Figs. 5 and 9 show that the second layer not only removes those nodes caused by noise; it also removes some redundant nodes and rearranges the position of nodes to represent the topology structure efficiently. We say that, using the second layer, the system can 'understand' original raw input data very well.

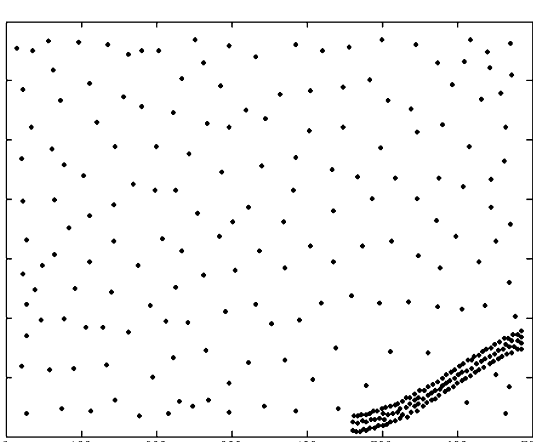

Fig. 7. Sequentially input samples from non-stationary environments. GNG-U results.

Fig. 10 shows how the number of nodes changes during on-line learning (first-layer). The number of nodes increases when an input signal comes from a new area (see Table 1, environment changes from I to VII). In the same environment, after some learning steps, the increase in nodes stops and the number of nodes converges to a constant because further insertion cannot engender decreased error. Noise leads the system to frequently insert and delete nodes. For that reason, a small fluctuation exists in the number of nodes in Fig. 10.

Finally, we perform some experiments to test the sensitivity of parameters. Four different parameter sets for $\{\alpha_1,\alpha_2,\alpha_3,\beta,\gamma\}$ are used to test the artificial data set (Fig. 2): {1/4, 1/4, 1/4, 2/3, 3/4}, {1/4, 1/4, 1/4, 2/3, 1/2}, {1/6, 1/4, 1/4, 1/2, 1/2} and {1/6, 1/4, 1/4, 1/4, 3/4}. The experiments are done in stationary and non-stationary environments. In both environments, for all four parameter sets, the system can detect the five clusters from the original data set and give prototype nodes; the last results are nearly the same as those shown in Fig. 5 for a stationary environment and those shown in Fig. 9 for a non-stationary environment. Consequently, the experiment results indicate that the choice of parameter set $\{\alpha_1,\alpha_2,\alpha_3,\beta,\gamma\}$ is not influential. In the following, we only use the original parameter set {1/6, 1/4, 1/4, 2/3, 3/4} to do different real-world experiments and to check whether or not the same parameter set is suitable to different tasks.

3.2. Real-world data set

Two real-world data sets are used to test the proposed method. For clustering, 10 classes are taken from ATT_FACE image database to train the proposed network system. For topology learning, we do vector quantization (VQ) for Lena (512×512×256) and Boat (512×512×256) to test if the proposed method can represent the topological structure well. In both experiments, we compare the proposed method with typical incremental networks GNG and GNG-U.

3.2.1. Test performance of clustering: application of face recognition

For application of clustering with the proposed method, we use some facial images as the input signal of the first layer. The input facial images are taken from the ATT_FACE database (http://www.uk.research.att.com/). The database comprises 40 distinct subjects with 10 different images of every subject. For some subjects, the images were taken at different times with various lighting,

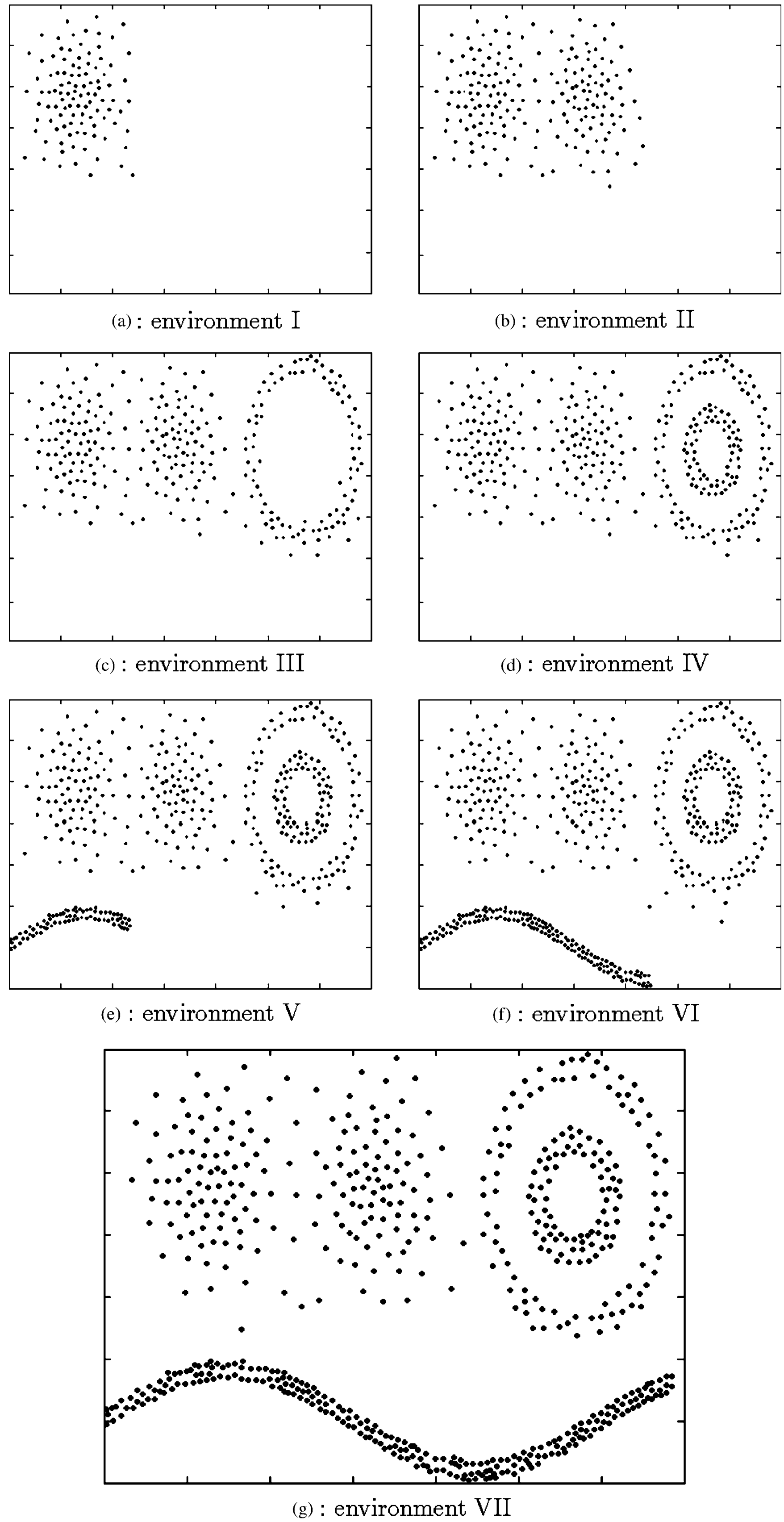

Fig. 8. Sequentially input samples from non-stationary environments. Results of the proposed method: first layer.

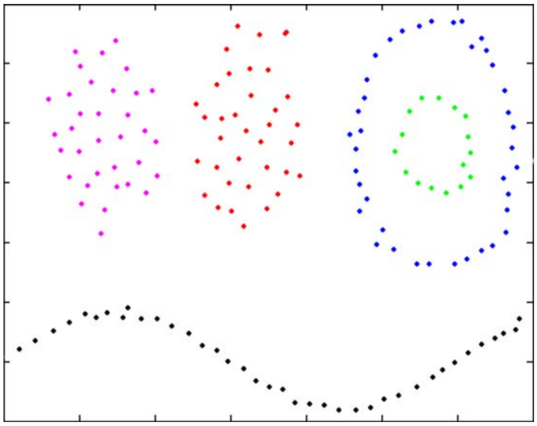

Fig. 9. Sequentially input samples from non-stationary environments. Results of the proposed method: second layer, five clusters.

facial expressions (open/closed eyes, smiling/not smiling) and facial details (glasses/no glasses). The size of each image is 92×112 pixels, with 256 grey levels per pixel. We selected 10 subjects (Fig. 11(a)) from the database to test proposed network system. Fig. 11(b) portrays the different images of the first subject in Fig. 11(a). Feature vectors of such images are taken as follows: first, the original image with size 92×112 is re-sampled to 23×28 image using the nearest neighbor interpolation method. Then Gaussian smoothing is used to smooth the 23×28 image with Gaussian width = 4, $\sigma = 2$ to obtain the 23×28 dimensional input feature vectors (Fig. 12).

The test consists of two processes: learning and evaluation. During learning, we choose vectors randomly (stationary) or sequentially (more difficult task, non-stationary on-line) from an original image vector set for the first layer, then input the results of the first layer to the second layer, and report number of clusters and prototype vectors of every cluster as the results of the second layer. In both layers, we set parameters $\lambda = 25$ and $age_{dead} = 25$. In the first layer, c is set to 1.0 because we want to delete nodes lying in overlap area; for the second layer, c is set as 0.05 to focus on clustering and to avoid deleting too many nodes. For both stationary and non-stationary environments, we do 10,000 training iterations for the first layer and 2000 training iterations for the second layer.

In the evaluation process, nearest neighbor is used to classify vectors in the original image vector set to the clusters learned through the learning process. The correct recognition ratio is reported as the performance index of learning process.

For a stationary environment, i.e. vectors are randomly chosen from the original data set, the learning process reports that 10 clusters exist in the input patterns and gives prototype vectors of every cluster. The evaluation process reports that the correct recognition ratio is 90%. In addition, GNG is tested with parameter $\lambda = 25$ and $a_{max} = 25$, maximum number of nodes is predefined as 50, other parameters are the same as in Fritzke (1995), and 10,000 samples are used to train GNG. The GNG results show four clusters in the original data set. Consequently, GNG is unsuitable for this task.

For a non-stationary on-line environment, i.e. 10,000 vectors are sequentially taken from original data set (one class by one class, 1000 samples for each class), the learning process reports 10 clusters and gives prototype vectors of every cluster (Fig. 13). Comparison of Fig. 13 with Fig. 12 reveals that the proposed algorithm reports the number of clusters correctly. It gives the prototype vectors of every cluster reasonably. The evaluation process reports that the correct recognition ratio is 86%. Furthermore, GNG-U is tested with parameter $\lambda = 25$ and $a_{max} = 25$. The maximum number of nodes is predefined as 25 and other parameters are the same as in (Fritzke, 1997); 10,000 samples are input sequentially to GNG-U (one class by one class, 1000 samples for each class). The GNG-U results show two clusters in the original data set. That result indicates that GNG-U is unsuitable for this task.

The experimental results show that the proposed method works well for unsupervised classification (learning the number of clusters and reporting prototype vectors of every cluster). Some other typical incremental networks (such as GNG, GNG-U) are unsuitable for this task.

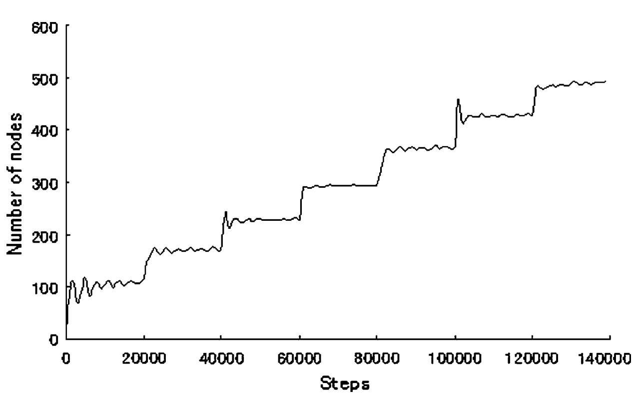

Fig. 10. Number of nodes during on-line learning (Environment I–Environment VII).

3.2.2. Test performance of topology learning: application of vector quantization

One goal of competitive learning systems is the minimization of distortion error. Vector quantization (VQ) (Linde et al., 1980) is a typical application in which error minimization is important. In vector quantization, data are transmitted over limited bandwidth channels by transmitting, for each data vector, only the index of the nearest reference vector. The set of reference vectors, the so-called codebook, is assumed to be known both to the sender and receiver. Therefore, the receiver can use the transmitted indexes to reconstruct an original vector with the corresponding reference vector. Information loss occurs because of the difference of the current data vector and

Fig. 11. Facial image (a) 10 subjects, (b) 10 images of subject one.

Fig. 12. Feature vector (a) vector of Fig. 11(a), (b) vector of Fig. 11(b).

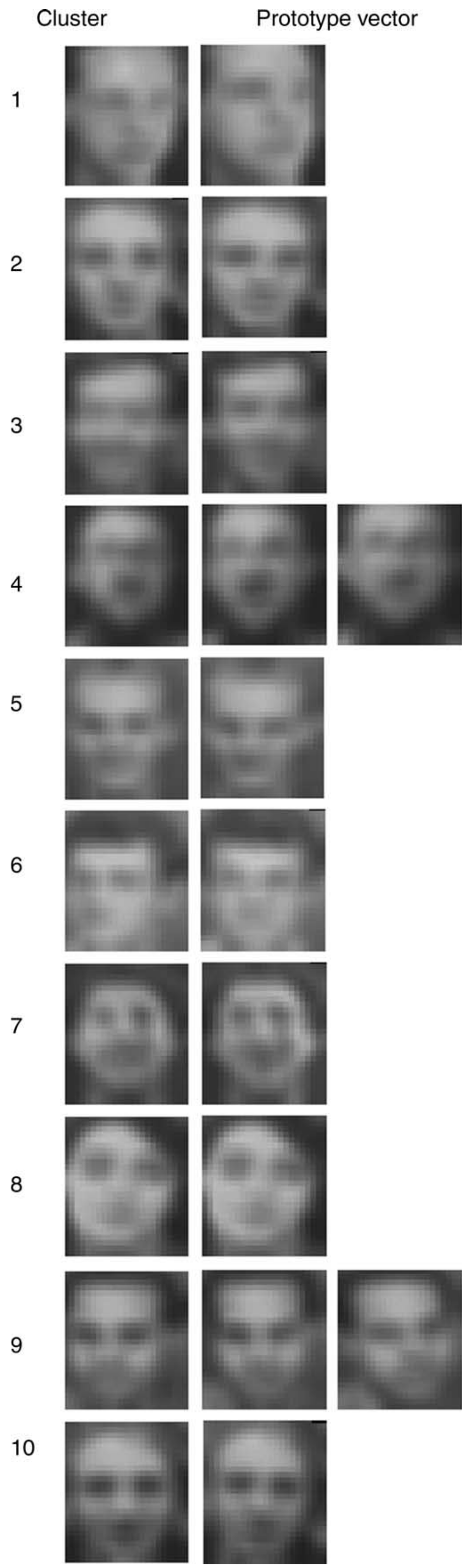

Fig. 13. Sequentially input samples from a non-stationary environment. Results of proposed method: 10 clusters.

the nearest reference vector. The main problem in VQ is how to find a codebook such that the information loss is small.

This experiment uses vector quantization to code a well-known image: Lena (512×512×256). The proposed algorithm is used to determine the codebook. Different from traditional methods, the codebook size is not predefined. Instead, it is determined by the algorithm itself. Because GNG or GNG-U is unsuitable for sequential input vectors, this experiment only compares the proposed method with GNG in a stationary environment, i.e. randomly input vectors from the original vector data set. For a non-stationary on-line environment, we only report results of the proposed method.

In image compression applications, the compression ratio and the peak signal to noise ratio (PSNR) are often used to evaluate the compression algorithm performance. Here, we use bit per pixel (bpp) to measure the compression ratio, and PSNR is defined as

$$\mathrm{PSNR} = 10\log_{10}\frac{255^2}{\sum_{i=1}^{N}(f(i)-g(i))^2/N} \quad (31)$$

where $f(i)$ is the grey level of the original image, $g(i)$ is the grey level of the reconstructed image, and N is the total number of pixels in the image. For VQ, with the same bpp, higher PSNR means less information loss. Consequently, the corresponding codebook is better.

In the experiment, first, original image Lena (Fig. 14) is separated into non-overlapping image blocks of 4×4. These blocks are input patterns (16 dimensional vectors) of the first layer. In the first layer, parameter $\lambda=50$, $\mathrm{age}_{\mathrm{dead}}=50$, and $c=0$ (for that we assume that all blocks are useful

Fig. 14. Original image of Lena (512×512×256), 8 bpp.

Fig. 15. Randomly input samples from stationary environment. Reconstructed image of Fig. 14 the first-layer codebook with 130 nodes, 0.45 bpp, PSNR = 30.79 dB.

and no block can be deleted as noise). From the original input vector data set, 200,000 samples are taken randomly. After we finish the learning of the first layer, we set the nodes generated in the first layer as the codebook (first-layer codebook) and find the index of the nearest reference vector for all original vectors. Then we use such indexes and reference vectors in the first-layer codebook to reconstruct the Lena image. Fig. 15 is the reconstructed image of the first-layer codebook.

Table 2
Comparison of the proposed method and GNG for Lena in stationary environment

	Number of nodes	bpp	PSNR
The first layer	130	0.45	30.79
GNG	130	0.45	29.98
The second layer	52	0.34	29.29
GNG	52	0.34	28.61

Then, we set the size of the first-layer codebook as the predefined maximum number of nodes of GNG and use GNG to learn the codebook. During learning, parameters used by GNG are $\lambda = 50$, $a_{max} = 50$. Other parameters are identical to those in Fritzke (1995), and 200,000 samples are processed. Fig. 16 depicts the reconstructed image of the GNG codebook.

For the second layer, the first-layer codebook is set as the input data set of the second layer. With the same parameter set as the first layer, i.e. $\lambda = 50$, $age_{dead} = 50$, and $c = 0$, 10, 000 samples are taken randomly from the first-layer codebook to train the second layer. We obtain the second-layer codebook and use this second-layer codebook to encode and reconstruct the Lena image. With the size of second-layer codebook, GNG is also used to obtain a codebook. We use this GNG codebook to encode and reconstruct the Lena image, and calculate the PSNR. Table 2 summarizes a comparison of the proposed method and GNG in a stationary environment. For both the first layer and the second layer, with the same compression ratio,

Fig. 16. Randomly input samples from a stationary environment. Reconstructed image of Fig. 14 GNG codebook with 130 nodes, 0.45 bpp, PSNR = 29.98 dB.

Fig. 17. Sequentially input samples from a non-stationary environment. Reconstructed image of Fig. 14 the first-layer codebook with 499 nodes, 0.56 bpp, PSNR = 32.91 dB.

Fig. 18. Sequentially input samples from a non-stationary environment. Reconstructed image of Fig. 14 the second-layer codebook with 64 nodes, 0.375 bpp, PSNR = 29.66 dB.

the proposed method obtains higher PSNR than GNG: the proposed method provides a better codebook than GNG.

From experiments in Sections 3.1.2 and 3.2.1, we know that GNG and GNG-U are unsuitable for an on-line non-stationary environment. Therefore, we only test the proposed method in an on-line non-stationary environment. The original vector data set is separated into five subsets. These subsets are input sequentially to the first layer, with 40,000 samples taken from each subset. The reconstructed results of

Fig. 19. Original image of Boat (512×512×256), 8 bpp.

Table 3
Comparison of the proposed method and GNG for Boat in stationary environment

	Number of nodes	bpp	PSNR
First layer	225	0.487	30.05
GNG	225	0.487	29.45
The second layer	62	0.375	28.13
GNG	62	0.375	27.59

the first layer (Fig. 17) and the second layer (Fig. 18) show that the proposed method is useful to learn the codebook of Lena well in an on-line non-stationary environment.

With the same parameter set as the Lena image, we also compare the proposed method with GNG for another well-known image-Boat (512×512×256) (Fig. 19)-in a stationary environment. Table 3 shows the results: the proposed method obtains a better codebook than GNG for the Boat image. In the on-line non-stationary environment, Fig. 20 is the reconstructed image of the first-layer codebook, and Fig. 21 is the reconstructed image of the second layer codebook. Both Lena and Boat experiments show that, for topology learning, the proposed method works well for stationary and non-stationary environments.

4. Conclusion

In this study, we proposed a new on-line learning method for unsupervised classification and topology representation. Using a similarity threshold-based and a locally accumulated error-based insertion criterion, the system can grow

Fig. 20. Sequentially input samples from a non-stationary environment. Reconstructed image of Fig. 19 the first-layer codebook with 557 nodes, 0.575 bpp, PSNR = 31.46 dB.

Fig. 21. Sequentially input samples from a non-stationary environment. Reconstructed image of Fig. 19 the second-layer codebook with 81 nodes, 0.4 bpp, PSNR = 28.54 dB.

incrementally and accommodate input patterns of on-line non-stationary data distribution. A novel on-line criterion for node removal in low probability-density regions enables this system to separate clusters with low-density overlap and to dynamically eliminate noise from input data. The utility parameter 'error-radius' is used to judge if the insertion is successful and to control the increase of nodes. Adoption of a two-layer neural network makes it possible for this system to 'understand' the original data set well. In summary, the algorithm can cope with difficulties in online or life-long unsupervised learning such as overlap, never-seen inputs, temporarily non-appearing patterns, and noise.

Other problems remain unsolved. For example, when a high-density overlap pertains between clusters, it is extremely difficult for the proposed algorithm to separate clusters from each other. Three parameters must be determined by the user: λ, age_{dead}, and c. The difficulty of automatically determining such parameters is based on the fact that, for different tasks, the optimal choice of such parameters is different: it is difficult to give a standard of such parameters for every task. Although these parameters are not so sensitive, we remain hopeful that some methods are useful to automatically deduce optimal choice of such parameters for the task. Such problems will be addressed in subsequent studies.

References

Bruske, J., & Sommer, G. (1995). Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, *7*, 845–865.

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

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley–Interscience.

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. Advances in neural information processing systems (NIPS)* pp. 625–632.

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

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An efficient clustering algorithm for large databases. *Proceeding of ACM SIGMOD Conference on Management of Data* , 73–84.

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

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Englewood Cliffs: Prentice Hall.

King, B. (1967). Step-wise clustering procedures. *Journal of American Statistical Association*, *69*, 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.

Lim, C. P., & Harrison, R. F. (1997). A incremental adaptive network for on-line supervised learning and probability estimation. *Neural Networks*, *10*, 925–939.

Lin, C. R., & Chen, M. S. (2002). A robust and efficient clustering algorithm based on cohesion self-merging. *Proceedings of ACM SIGKDD, Edmonton, Alberta, Canada* .

Linde, Y., Buzo, A., & Gray, R. M. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communication*, *COM-28*, 84–95.

Lozano, J. A., Pena, J. M., & Larranaga, P. (1999). An empirical comparison of four initialization methods for the *k*-means algorithm. *Pattern Recognition Letters*, *20*, 1027–1040.

Martinetz, T. M. (1993). Competitive Hebbian learning rule forms perfectly topology preserving maps. *ICANN* , 427–434.

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, *4*(4), 556–558.

Martinetz, T., & Schulten, K. (1994). Topology representing networks. *Neural Networks*, *7*(3), 507–552.

Murty, M. N., Jain, A. K., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Survey*, *31*(3), 232–264.

Murty, N. M., & Krishna, G. (1981). A hybrid clustering procedure for concentric and chain-like clusters. *International Journal of Computer and Information Sciences*, *10*(6), 341–397.

Oyang, Y.-J., Chen, C.-Y., & Yang, T.-W. (2001). A study on the hierarchical data clustering algorithm based on gravity theory. *Proceedings of the Fifth European Conference on Principles and Practice of Knowledge Discovery in Databases* , 350–361.

Patane, G., & Russo, M. (2001). The enhanced LBG algorithm. *Neural Networks*, *14*, 1219–1237.

Ritter, H. J., Martinetz, T. M., & Schulten, K. J. (1991). *Neuronale Netze*. Munchen: Addison-Wesley.

Sneath, H. A., & Sokal, R. R. (1973). *Numerical taxonomy*. London, UK: Freeman.

Villmann, T. (2000). Controlling strategies for the magnification factor in the neural gas network. *Neural Network World*, *10*, 739–750.

Villmann, T., Der, R., Herrmann, M., & Martinetz, T. (1997). Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, *8*(2), 226–256.

Villmann, T., Schleif, F.-M., & Hammer, B. (2003). Supervised neural gas and relevance learning in learning vector quantization. *Proceedings of the Workshop on Self-Organizing Maps (WSOM), Japan* .

Wasan, M. T. (1969). *Stochastic approximation*. Cambridge: Cambridge University Press.

Willshaw, D. J., & von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London B*, *194*, 431–445.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An efficient data clustering method for a very large database. *Proceedings of ACM SIGMOD Conference on Management of Data* , 103–114.