Neurocomputing 104 (2013) 57–71

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A general associative memory based on self-organizing incremental neural network

Furao Shen [a,*], Qiubao Ouyang [a], Wataru Kasai [b], Osamu Hasegawa [b]

[a] National Key Laboratory for Novel Software Technology, Nanjing University, China
[b] Imaging Science and Engineering Lab., Tokyo Institute of Technology, Japan

ARTICLE INFO

Article history:
Received 28 September 2010
Received in revised form
28 September 2012
Accepted 4 October 2012
Communicated by G. Palm
Available online 28 November 2012

Keywords:
General associative memory
Incremental learning
Temporal sequence
Real-value data
Many-to-many association

ABSTRACT

This paper proposes a general associative memory (GAM) system that combines the functions of other typical associative memory (AM) systems. The GAM is a network consisting of three layers: an input layer, a memory layer, and an associative layer. The input layer accepts key vectors, response vectors, and the associative relationships between these vectors. The memory layer stores the input vectors incrementally to corresponding classes. The associative layer builds associative relationships between classes. The GAM can store and recall binary or non-binary information, learn key vectors and response vectors incrementally, realize many-to-many associations with no predefined conditions, store and recall both static and temporal sequence information, and recall information from incomplete or noise-polluted inputs. Experiments using binary data, real-value data, and temporal sequences show that GAM is an efficient system. The AM experiments using a humanoid robot demonstrates that GAM can accommodate real tasks and build associations between patterns with different dimensions.

1. Introduction

An associative memory (AM) stores data in a distributed fashion, which is addressed through its contents. An AM can recall information from incomplete or garbled inputs and finds applications in face recognition [1], pattern recognition [2], robotics [3], image and voice recognition, and databases [4].

When an input pattern, called a key vector, is presented, the AM is expected to return a stored memory pattern (called a response vector) associated with that key. AM system have several requirements, each putting constraints on the possible AM systems that we might use. (1) An AM system must store and recall binary or non-binary information. Traditional methods such as the Hopfield network [5], the bidirectional associative memory (BAM) [6] and their variants specifically examine the use of binary AM with a discrete state, but few reports in the literature describe generalization of the original idea to non-binary cases. Even fewer address the continuous case. However, in the real world, patterns are usually represented by gray-level feature vectors or real-valued feature vectors. It is also necessary for AM systems to store and recall non-binary information. (2) An AM system must memorize incrementally and associate new information without destroying the stored knowledge. Human memory has the ability to learn new knowledge incrementally without destroying the previously learned knowledge. Unfortunately, traditional AM systems, such as the Hopfield network and BAM, destroy learned weight vectors subsequent to network training and learned knowledge cannot be recalled accurately if we want to store new patterns [7]. (3) An AM system must be able to memorize temporal sequence information as well as static information. Humans do not memorize temporal sequences as static patterns but as patterns with a consecutive relation. (4) An AM system must not only realize one-to-one association but also one-to-many, many-to-one, or many-to-many association. (5) An AM system must be robust. It must be able to recall information from incomplete or garbled inputs.

Several neural models have been proposed to achieve one or more of these requirements. Sussner and Valle [8] designed an AM for processing non-binary cases, but ignored the other requirements. The Kohonen feature map (KFP) [9] employed competitive learning – patterns were presented sequentially to train the memory system – but it still could not realize incremental learning. When new patterns enter the system, the weight of nodes storing learned information changes to store the new pattern. Sudo et al. proposed a self-organizing incremental associative memory (SOIAM) [10] specifically to store new patterns incrementally without destroying the memorized information; however, SOIAM cannot address temporal sequences. Kosko [6] and Hattori and Hagiwara [11] processed the temporal

* Corresponding author.
E-mail addresses: frshen@nju.edu.cn (F. Shen), njuxingkong@gmail.com (Q. Ouyang), kasai.w.aa@m.titech.ac.jp (W. Kasai), hasegawa@isl.titech.ac.jp (O. Hasegawa).

0925-2312/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2012.10.003

sequences, but their methods only dealt with simple temporal sequences. Their AM systems are not effective with repeated or shared items in the temporal sequences. Barreto and Araujo [12] learned the temporal order through a time-delayed Hebbian learning rule, but the complexity of the model depends considerably on the number of context units. Sakurai et al. [13] proposed a self-organizing map based associative memory (SOM-AM) for temporal sequences, but its recall performance is affected by the initial values of the weights. Such methods only consider the binary temporal sequence without touching the real-valued sequence. Furthermore, it is difficult for such methods to implement incremental learning for temporal sequences. Multidirectional associative memory (MAM) [14] realized many-to-many association, but the association was not flexible. The number of associations must be predetermined and the number of layers must be preset with the number of associations.

In this paper, we propose an AM system that satisfies all the requirements using a three-layer network: an input layer, a memory layer, and an associative layer. The input layer accepts key and response vectors to the memory layer; the memory layer stores the information obtained from the input layer. Both key and response vector information can be stored in the memory layer. Incremental learning is available for learning of the memory layer: new key and response vectors can be memorized incrementally. The associative layer builds an associative relationship between the key vector and the response vector. This layer constructs many-to-many and temporal sequence associations.

In a way, our AM system can be considered a symbol system [15], a cognitive model of the human mind. The input layer receives the sensory representations, the memory layer encodes them as symbols, and the associative layer performs a function of symbol grounding.

Using the proposed AM system with a three-layer network, we achieve the following goals: (1) processing memory patterns by classes. Rather than memorizing all patterns in a same network, patterns belong to different classes are memorized in different subnetworks respectively; (2) storing and recalling binary or non-binary information; (3) learning the key vectors and response vectors incrementally; (4) realizing one-to-one, one-to-many, many-to-one, and many-to-many associations with no predefined condition; (5) storing and recalling both static and temporal sequence information; (6) building an association among data of different types (patterns with different dimensions); and (7) recalling information from incomplete or noise-polluted inputs. Because the proposed method satisfies all these requirements, we call it a general associative memory (GAM).

This paper is organized as follows: Section 2 discusses the GAM principles and introduces the GAM structure; Section 3 explains how GAM memorizes key vectors, response vectors, and association relationships; Section 4 discusses how it recalls or associates stored patterns; in Section 5, we perform experiments that compare GAM with other AM models, and describe the efficiency of GAM.

2. Structure of the general associative memory (GAM)

Our design decisions for GAM are based on the following six principles, such principles are very important for human intelligence [16]:

(1) *Memorize patterns by classes*: If patterns belong to one class, they are memorized with a subnetwork; different classes adopt different subnetworks to store the patterns belonging to each class. To realize this, each input pattern receives a class label during the memorization process. The patterns belonging to one class will be used to train the corresponding subnetwork.

(2) *Memorize patterns with binary or non-binary feature vectors*: It is natural for human to remember a multicolored world without transforming images into a special binary format. It is important to remember feature representation of patterns directly without transforming feature vectors to a binary format.

(3) *Memorize temporal sequences with contextual items, and do not store the complete temporal sequence as one pattern*: The human memory stores a temporal sequence, such as a song, by remembering separate context items with their time order, the way we do with musical notes. The memory system will store the contextual items with a time order rather than storing the temporal sequence as a static pattern.

(4) *Memorize patterns incrementally*: Incremental learning has two aspects: (i) class-incremental, where classes can be remembered incrementally, i.e., during system training, new classes (new subnetworks) can be added incrementally; (ii) example-incremental: within the same class, new information can be added incrementally. The system can memorize new patterns without destroying stored patterns if new patterns belonging to a trained class are input.

(5) *First, realize auto-association, and then hetero-association*: Human initially recognize or recall a class with a garbled or incomplete key vector, and then associate it with other classes according to the recognition of the key vector. Pattern recognition or pattern completion process uses auto-associative information; association between classes uses hetero-associative information.

(6) In addition to one-to-one association, one-to-many, many-to-one, and many-to-many associations are necessary, requiring an association network. In this network, one class is used for association with several other classes. The class itself is also associated by other classes. The system must be able to incrementally add new associations between the memorized classes.

Based on these six principles, we designed the three-layer network discussed in Section 1. Fig. 1 presents the three-layer GAM network structure.

The input layer accepts input patterns to the GAM. The input feature vector (either a key vector or a response vector) is input into the system with a class label. According to the class label, the GAM locates the corresponding subnetwork in the memory layer; and learns the new input information incrementally. If the input vector does not belong to an existing class in the memory layer, the GAM builds a new subnetwork in the memory layer to represent the new class. The GAM sends the class labels of subnetworks in the memory layer to the associative layer, and the associative layer builds relationships between the class of key vector (the key class) and the class of response vector (the response class) by using arrow edges. One node exists in the associative layer corresponding to one subnetwork in the memory layer. The arrow edges connecting these nodes represent the

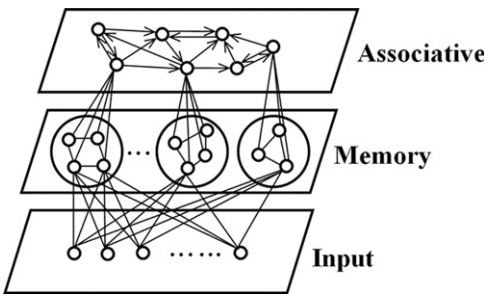

Fig. 1. Network structure of a general associative memory (GAM). The input layer accepts key vectors, response vectors, and associative relationships between these vectors. The memory layer incrementally stores the input vectors with subnetworks (classes). The associative layer incrementally builds associations between the classes.

associative relationships between the classes. The beginning of an arrow edge indicates the key class; and the end of the arrow edge indicates the corresponding response class.

This three-layer network structure is useful in realizing all six principles. The input layer accepts both binary and non-binary patterns. In the memory layer, different subnetworks memorize different classes. Contextual patterns of temporal sequences are memorized in the memory layer, and the associated time orders are memorized in the associative layer. Incremental learning is realized during the training of the memory and associative layers. The memory layer also realizes auto-association (pattern recognition or pattern completion), and the associative layer is used for hetero-association. The associative layer itself is a network for incrementally building any association: one-to-one, one-to-many, many-to-one, or many-to-many.

Section 3 provides details on training the memory and associative layers. Section 4 discusses the recall and association processes.

3. GAM learning algorithms

During training of the memory layer, it is important to memorize the input patterns by using real-value feature vectors and realize incremental learning.

During training of the associative layer, the associative relationships between the key and response vectors are memorized in the associative layer. In addition, for temporal sequences, the associative layer must memorize the associative relationship and the time order between contextual patterns. Also, the associative layer must be able to learn new associations incrementally.

3.1. Memory layer

In the memory layer, we need to store input vectors in the weight of nodes and recall the vectors with the learned weight. It means that we need to use a finite set of weight vectors to store maybe infinite input vectors. For example, we have an input data set $V \subseteq R^D$, utilizing only a set $W = (\boldsymbol{W}_1, \ldots, \boldsymbol{W}_N)$ of reference or "weight" vectors to encode V, where $\boldsymbol{W}_i \in R^D$, $i = 1, \ldots, N$.

For competitive learning, a data vector $\boldsymbol{v} \in V$ is described by the best-matching or "winning" reference vector $\boldsymbol{W}_{i(\boldsymbol{v})}$ of W for which the distortion error $d(\boldsymbol{v}, \boldsymbol{W}_{i(\boldsymbol{v})})$ is minimal. This procedure divides the data set V into a number of subregions

$$V_i = \{\boldsymbol{v} \in V \,|\, d(\boldsymbol{v}, \boldsymbol{W}_i) \le d(\boldsymbol{v}, \boldsymbol{W}_j) \forall j\} \tag{1}$$

called Voronoi polygons or Voronoi polyhedra, out of which each data vector $\boldsymbol{v}$ is described by the corresponding reference vector $\boldsymbol{W}_i$.

If the probability distribution of data vectors over the data set V is described by $P(\boldsymbol{v})$, then the average distortion or reconstruction error is determined by

$$E = \int d^D v P(\boldsymbol{v}) (\boldsymbol{v} - \boldsymbol{W}_{i(\boldsymbol{v})})^2 \tag{2}$$

and has to be minimized through an optimal choice of reference vectors $\boldsymbol{W}_i$ [17]. In general, the error surface E has many local minima, and it suits the target of associative memory for store and recalling input patterns.

The straightforward approach to minimizing (2) would be a gradient descent on E which leads to Lloyd and MacQueen's well-known K-means clustering algorithm [18,19]. In its online version, which is applied if the data point distribution $P(\boldsymbol{v})$ is not given a priori, but instead a stochastic sequence of incoming sample data points $\boldsymbol{v}(t=1), \boldsymbol{v}(t=2), \boldsymbol{v}(t=3), \ldots$ which is governed by $P(\boldsymbol{v})$ drives the adaptation procedure, the adjustment steps for the reference vectors or cluster centers $\boldsymbol{W}_i$ is determined by

$$\Delta \boldsymbol{W}_i = \varepsilon \cdot \delta_{i i(\boldsymbol{v}(t))} \cdot (\boldsymbol{v}(t) - \boldsymbol{W}_i), \quad i = 1, \ldots, N \tag{3}$$

with ε as the step size and δ_{ij} as the Kronecker delta.

Here, we build the memory layer based on a self-organizing incremental neural network (SOINN) [20]. SOINN is based on competitive learning; neural nodes represent the distribution of input data. The weights of such nodes serve as reference vectors to store the input patterns.

The memory layer stores input patterns and consists of subnetworks, each representing one class, as shown in Fig. 1. All patterns belonging to one class are memorized in the corresponding subnetwork. Building the subnetworks requires memorizing real-valued patterns and realizing incremental learning. As described in Section 1, some traditional AM systems are unable to process real-valued patterns. Some methods have been designed to memorize real-valued patterns, but it is difficult for them to perform incremental learning.

Some pioneering methods are useful in representing the distribution of input data. The well-known self-organizing map (SOM) [21] generates mapping from a high-dimensional signal space to a lower-dimensional topological structure, but the predetermined network structure and size impose limitations on the resulting mapping [17] and incremental learning [20]. The combination of "competitive Hebbian learning" (CHL) and "neural gas" (NG) [17] also requires a prior decision related to the network size. Growing neural gas (GNG) [22] presents the disadvantage of a permanent increase in the number of nodes if the number of nodes is not predetermined.

Most extensions of vector quantization cannot deal with incremental learning and often involve high computational complexity [25][26]. Therefore, these methods cannot fulfill our requirements. Using the technique of class-specific vector quantization, some algorithms such as learning vector quantization(LVQ), can partially solve the incremental learning problem with nothing predetermined. Incremental learning may be class-incremental or example-incremental. LVQ cannot handle the example-incremental learning problems.

Some noncompetitive methods, such as the support vector machine (SVM) [23] and backpropagation neural networks [24], are unsuitable for incremental learning. New inputs will require retraining of the entire system.

The SOINN and its enhanced version [27] execute topology representation and class-incremental and example-incremental learning without requiring predetermination of the network structure and size. SOINN can implement both real-valued pattern memorization and incremental learning. Self-organizing incremental associative memory (SOIAM) [10] is based on SOINN. Here, the basic idea of training the GAM memory layer is explained by SOINN. We adjust the unsupervised SOINN to a supervised mode: for each class, we adopt a SOINN to represent the distribution of that class. The input patterns (key and response vectors) are separated into different classes. For each class, one subnetwork represents the data distribution of the class.

3.1.1. 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 this distribution. The second layer separates the clusters by detecting the low-density area of input data; it uses fewer nodes than the first layer does to represent the topological structure of the input data. When the second layer learning completes, SOINN gives typical prototype nodes to every cluster; it adopts the same learning algorithm for the first and second layers.

When SOINN receives an input vector, it finds the nearest (winner) and second nearest node (runner up) of the input vector, then judges if the input vector belongs to the same cluster as the

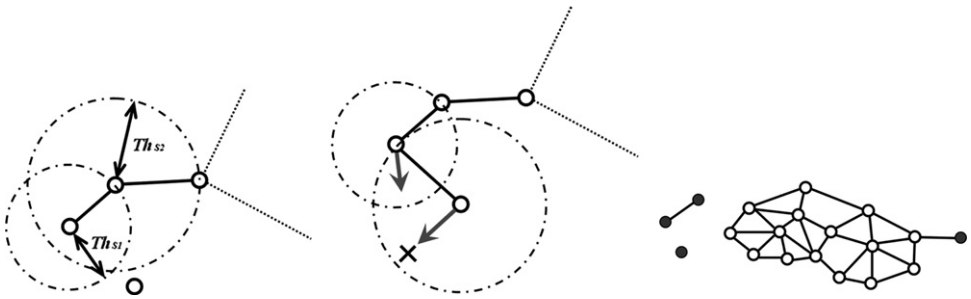

Fig. 2. Overview of SOINN: (left) if the input vector lies beyond the Voronoi region of the winner and second winner, insert it as a new node; (center) updating the weight of the winner and its neighbors by moving them toward the input vector; (right) deletion of noise nodes—marked with a solid circle.

winner or runner up by using the criterion of similarity thresholds. In the first layer, this threshold updates adaptively for every node. As shown in the left of Fig. 2, if the distance between the input vector and the winner or runner up is greater than the similarity threshold of a winner or runner up, the input vector is inserted into the network as a new node.

The similarity threshold T_i is defined as the Euclidean distance from the boundary to the center of the Voronoi region V_i of node i. During the learning process, node i changes its position to meet the input pattern distribution, and the Voronoi region V_i of the node i also changes. When a new node i is generated, the similarity threshold T_i of node i is initialized to $+\infty$. When node i becomes the winner or second winner, the similarity threshold T_i is updated. If node i has direct topological neighbor (a node linked to node i with an edge), T_i is updated to the maximum distance between node i and all its neighbors

$$T_i = \max_{c \in N_i} \|\mathbf{W}_i - \mathbf{W}_c\|, \tag{4}$$

where $\mathbf{W}_i$ is the weight vector of node i and N_i is the neighbor set of node i. $\|\mathbf{W}_i - \mathbf{W}_c\|$ means the Euclidean distance between two vectors $\mathbf{W}_i$ and $\mathbf{W}_c$. If node i has no neighbor, T_i is updated as the minimum distance of node i and all other nodes in the network A

$$T_i = \min_{c \in A \setminus \{i\}} \|\mathbf{W}_i - \mathbf{W}_c\| \tag{5}$$

and A is the node set.

If the input vector is judged as belonging to the same cluster as the winner or runner up, and if no edge connects the winner and runner up, connect the winner and runner up with an edge, and set the 'age' of the edge to '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, i.e., move the winner and its neighbors toward the input vector, as shown in the center of Fig. 2. If node i is the winner, changes $\Delta \boldsymbol{W}_i$ to the weight of winner i and $\Delta \boldsymbol{W}_j$ to the weight of the neighboring node $j (\in N_i)$ of i are defined as

$$\Delta \boldsymbol{W}_i = \frac{1}{M_i}(\boldsymbol{v} - \boldsymbol{W}_i), \tag{6}$$

$$\Delta \boldsymbol{W}_j = \frac{1}{100 M_i}(\boldsymbol{v} - \boldsymbol{W}_j), \tag{7}$$

$\boldsymbol{v}$ is the input vector, M_i is the times for which node i was the winner—it shows the frequency of node i being winner.

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

Subsequent to λ learning iterations (λ is a timer), SOINN inserts new nodes in the position where the accumulating error is extremely large. Cancel the insertion if it cannot decrease the error. Then, SOINN finds the nodes whose neighbors are less than or equal to 1 and deletes these nodes assuming that they lie in the low-density area—these nodes are called "noise nodes." The right of Fig. 2 shows the deletion of noise nodes.

Table 1
Contents of a node in the memory layer.

Notation	Meaning
c_i	Class name of node i
$\boldsymbol{W}_i$	Weight vector of node i
Th_i	Similarity threshold of node i
M_i	The number of patterns represented by node i
E_i	Set of nodes connected with node i
I_i	Index of node i in sub-network c_i

Table 2
Notation to be used in the following algorithms.

Notation	Meaning
A	Node set of the memory layer
S	Sub-network set of the memory layer
C	Connection set of the memory layer
c^i	The ith node in the sub-network c
(i,j)	Connection between node i and node j
$age_{(i,j)}$	Age of the connection between node i and node j
$\text{age}_{\max}$	Maximum age threshold

Subsequent to LT learning iterations of the first layer, the learning results are used as inputs to the second layer, which uses the same learning algorithm as the first layer. For the second layer, the similarity threshold is constant and it calculated with the help of within-cluster and between-cluster distances. With a large constant similarity threshold, the second-layer also deletes some "noise nodes" that were not deleted during first layer learning. For details about SOINN, refer to [20].

3.1.2. Learning algorithm of the memory layer

We adopt the first-layer of SOINN and adjusted the unsupervised SOINN for supervised mode: for each class we adopt one SOINN to represent the distribution of that class.

Before we assign the learning algorithm for the memory layer, we present Table 1 to define the contents of nodes in memory layer. Some other notation to be used in the learning algorithm is defined in Table 2.

Algorithm 1 shows the proposed algorithm for training of the memory layer. When a vector is input to the memory layer, if there is no sub-network named with the class name of this input vector, then set up a new sub-network with the input vector as the first node of the new sub-network. Name this sub-network with the class name of the input vector. Find the nearest node (winner) and the second nearest node (runner up) in this sub-network if there is already a sub-network with the same class name as the input vector. If the distance between input vector and

the winner is greater than the winner's similarity threshold, then insert a new node into the sub-network. The weight of the new node is set as the input vector. The similarity threshold of the new node is set as the distance from winner. Then update the winner's similarity threshold. Update the weight and similarity threshold of the winner if the distance between input vector and the winner is less than the similarity threshold of the winner. If no edge connects the winner and runner up, connect the winner and runner up with an edge, and set the 'age' of the edge to '0';subsequently, increase the age of all edges linked to the winner by '1'. At last, if the age of one edge is greater than a predefined parameter age_{max}, remove that edge. Also, isolated nodes are removed.

Algorithm 1. Learning of the memory layer.

1: Initialize the memory layer network: $A=\emptyset$, $S=\emptyset$, $C=\emptyset$.
2: Input a pattern $x \in R^D$ to the memory layer, the class name of x is c_x.
3: **if** There is no sub-network with name c_x **then**
4: Add a sub-network c_x to the memory layer by $S=S\cup\{c_x\}$, $A=A\cup\{c_x^1\}$, and $\boldsymbol{W}_{c_x^1}=x$, $Th_{c_x^1}=0$, $M_{c_x^1}=1$, $N_{c_x^1}=0$. Go to Step 2.
5: **else**
6: Find winner s_1 and runner up s_2 by
$$s_1=\arg\min_{i\in c_x}\|x-\boldsymbol{W}_{c_x^i}\|,\quad s_2=\arg\min_{i\in c_x\setminus s_1}\|x-\boldsymbol{W}_{c_x^i}\| \tag{8}$$
update M_{s_1} by $M_{s_1}\leftarrow M_{s_1}+1$.
7: **end if**
8: **if** $\|x-\boldsymbol{W}_{s_1}\|>Th_{s_1}$ **then**
9: Insert new node c_x^{new} into the sub-network c_x with $\boldsymbol{W}_{c_x^{new}}=x$, $Th_{c_x^{new}}=\|x-\boldsymbol{W}_{s_1}\|$, $M_{c_x^{new}}=1$, $N_{c_x^{new}}=0$. Update similarity threshold of s_1
$$Th_{s_1}=Th_{c_x^{new}} \tag{9}$$
10: **else**
11: Update the weight of winner and the runner up by
$$\boldsymbol{W}_{s_1}\leftarrow\boldsymbol{W}_{s_1}+\delta_{s_1}(x-\boldsymbol{W}_{s_1}) \tag{10}$$
$$\boldsymbol{W}_{s_2}\leftarrow\boldsymbol{W}_{s_2}+\delta_{s_2}(x-\boldsymbol{W}_{s_2}) \tag{11}$$
where $\delta_{s_1}=1/M_{s_1}$ and $\delta_{s_2}=1/(100M_{s_1})$. Update the threshold of s_1
$$Th_{s_1}\leftarrow(Th_{s_1}+\|x-\boldsymbol{W}_{s_1}\|)/2. \tag{12}$$
12: **end if**
13: Create a connection (s_1,s_2) and add it to C, i.e. $C=C\cup(s_1,s_2)$.
14: Set the age of the connection between s_1 and s_2 to zero, i.e. $age_{(s_1,s_2)}=0$.
15: Increase age of edges which connected with s_1 and its neighbors: $age_{(s_1,i)}\leftarrow age_{(s_1,i)}+1,(\forall i\in E_{s_1})$.
16: If the training is not finished, go to Step 2 to process the next pattern.
17: Remove old edges: If $(i,j)\in C$, and $age_{(i,j)}>age_{max}(\forall i,j\in A)$, then $C\leftarrow C\setminus\{(i,j)\}$.
18: Remove isolated nodes from A, i.e., if $E_i=\emptyset(i\in A)$, then $A=A\setminus\{i\}$.

According to [27], to build connections among neural nodes, SOINN adopts the competitive Hebbian rule [17]: for each input signal, connect the two closest nodes (winner and runner up) with an edge. This rule forms a network whose edges are in the area suggested by the input data distributions (Fig. 3b). The network represents a subgraph of the original Delaunay triangulation (Fig. 3a). Using the competitive Hebbian rule, the resultant graph approximates the shape of the input data distributions [17]. This means that SOINN can effectively represent the topological structure of input data. Hence, Algorithm 1 can represent the input data distribution. Therefore, GAM uses weights of nodes in the memory layer to represent the input pattern, and the nodes of subnetworks are the centers of Voronoi regions. Such nodes serve as attractors for a future recall phase—the Voronoi regions form basins of attraction, and the nodes similarity thresholds serve as the radii of the basins of attraction. For example, if there is a node i in the memory layer, the Voronoi region of i is V_i, the similarity threshold of i is T_i, then i serves as the attractor, V_i serves as the attraction basin, and T_i serves as the radius of V_i.

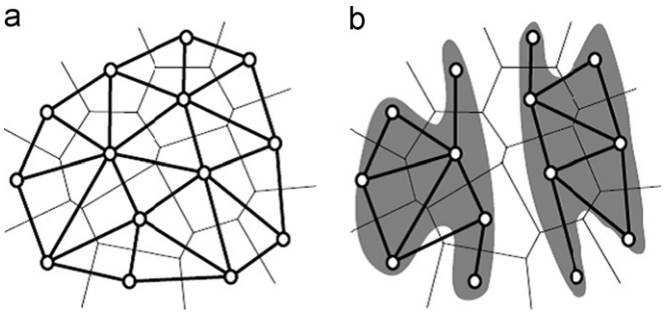

Fig. 3. Two methods for defining closeness among a set of points. (a) *Delaunay triangulation* (thick lines) connects points having neighboring *Voronoi polygons* (thin lines). (b) *Induced Delaunay triangulation* (thick lines) is obtained by masking the original Delaunay triangulation with a data distribution $P(\xi)$ (shaded).

Algorithm 1 also achieves incremental learning. For class-incremental, new classes are learned incrementally by adding new subnetworks; for example-incremental, new patterns inside one class are learned incrementally by adding new nodes to an existing subnetwork. The number of subnetworks is determined by the number of classes of input patterns. When a new class arises, the memory layer reacts to the new class without destroying old classes. Inside one class, SOINN controls the increment of nodes for learning without unlimited increase in the number of nodes.

From Algorithm 1, we know that if the similarity threshold is too small, then it is easy for a new node to be inserted into the network. If the threshold is too large, then it becomes difficult to insert new nodes, and the tuning of weights will happen easily. In Step 9 of Algorithm 1, when insertion of a node occurs, formula (9) will be used to update the threshold of the winner with a larger value to make the next insertion near the winner become difficult. In Step 11 of Algorithm 1, the tuning of weight for the winner and the runner up happens. Then formula (12) will be used to update the threshold of the winner with a smaller value to make the next insertion become easy to ensure that the memory layer can learn new knowledge. After the threshold achieving a stationary status, the insertion of new nodes and the tuning of the weight will also reach a stable status. Insertion will stop and the following input data will only engender tuning of the weight of nodes if the nodes are distributed throughout the whole feature space of the input data. It makes the memory layer avoid the permanent increase of nodes.

In Algorithm 1, each class is allocated a subnetwork. For different classes, the dimension of vectors might differ. In other words, the memory layer can memorize different data types (patterns with different dimensions).

3.2. Associative layer

The associative layer builds associations between key and response vectors. Key vectors belong to a key class and response vectors belong to a response class. In the associative layer (Fig. 1), the nodes are connected with arrow edges. Each node represents

one class—the beginning of the arrow indicates the key class and the end of the arrow indicates the corresponding response class.

During training of the associative layer, we use association pair data – the key vector and response vector – as the training data. Such data input incrementally into the system. First, Algorithm 1 is used to memorize information of both the key and the response vectors. If the key class (or response class) already exists in the memory layer, the memory layer will learn the information of the key vector (or the response vector) by adding new nodes or tuning weights of nodes in the corresponding subnetwork. If the key or response class does not exist in the memory layer, it builds a new subnetwork to memorize the new key or response class with the key or response vector as the first node of the new subnetwork.

The class name of the new class is sent to the associative layer. In the associative layer, if nodes that represent the key and response class already exist, we connect their nodes with an arrow edge. The beginning of the arrow corresponds to the key class node and the end corresponds to the response class node. This creates an associative relationship between the key class and the response class.

If no node represents the key (or response) class within the associative layer, we add a node to the associative layer and use that node to express the new class. Then, we build an arrow edge between the key class and response class. Algorithm 2 gives the details for training the associative layer with the key and response vectors as the input data.

In Table 3, we list the contents of the node in the associative layer and some notations used in the following algorithms.

Algorithm 2. Learning of the associative layer.

1: Initialize the associative layer network: initialize node set B, arrow edge set $D \subset B \times B$ to the empty set, i.e., $B=\emptyset$, $D=\emptyset$
2: Input a key vector $x \in R^D$ to GAM; the class name of x is c_x.
3: Use Algorithm 1 to memorize key vector x in the memory layer.
4: **if** No node b exists in the associative layer representing class c_x **then**
5: Insert a new node b representing class c_x into the associative layer:
$$B=B\cup\{b\},\quad c_b=c_x,\quad m_b=0,\quad \boldsymbol{W}_b=x \tag{13}$$
6: **else**
7: Increment the associative index of b: $m_b \leftarrow m_b+1$;
8: Find node i that is most frequently being winner in subnetwork c_x:
$i=\arg\max_{j\in c_x} M_{c_x^j}$
9: Update the weight of node b in associative layer:
$\boldsymbol{W}_b=\boldsymbol{W}_{c_x^i}$.
10: **end if**
11: Input the response vector $y \in R^m$ to GAM, the class name of y is c_y.
12: Use Algorithm 1 to memorize the response vector y in the memory layer.
13: **if** No node d representing class c_y in the associative layer **then**
14: Insert a new node d representing class c_y into the associative layer:
$B=B\cup\{d\}$, $c_d=c_y$, $m_d=0$, $\boldsymbol{W}_d=y$.
15: **else**
16: Find node i which is most frequently being winner in subnetwork c_y:
$i=\arg\max_{j\in c_y} M_{c_y^j}$
17: Update the weight of node d in associative layer:
$\boldsymbol{W}_d=\boldsymbol{W}_{c_y^i}$.
18: **end if**
19: **if** There is no arrow between node b and d **then**
20: Connect node b and d with an arrow edge. The beginning of the arrow is node b, the end of the arrow is node d.
21: Add arrow (b,d) to connection set D: $D=D\cup\{(b,d)\}$,
22: Set the m_bth response class of b as c_d: $RC_b[m_b]=c_d$,
23: Set the weight of arrow (b,d) as 1: $W_{(b,d)}=1$.
24: **else**
25: Set the m_bth response class of b as c_d: $RC_b[m_b]=c_d$,
26: Increment the weight of arrow (b,d) with 1:
$W_{(b,d)} \leftarrow W_{(b,d)}+1$.
27: **end if**

Table 3
Contents of node in the associative layer and some notations to be used in the following algorithms.

Notation	Meaning
c_i	Class name of node i
m_i	Associative index of node i
$\boldsymbol{W}_i$	Weight of node i
RC_i	Response classes of node i
T_i	Time order of current node i in a temporal sequence
TL_i	Time order of the latter item of node i in a temporal sequence
TF_i	Time order of the former item of node i in a temporal sequence
B	Node set of the associative layer
D	Connection (arrow edge) set of the associative layer
(i,j)	Connection (arrow edge) from node i to node j
$W_(i,j)$	Weight of connection (i,j)

In the associative layer, the weight vector of every node is selected from the corresponding subnetwork of the memory layer. Steps 8, 9, 16, and 17 in Algorithm 2 show that the node that is most frequently the winner is chosen as the typical node of the corresponding subnetwork in the memory layer, and the weight vector of the typical node is set to the weight of that class node in the associative layer.

Algorithm 2 can realize incremental learning. For example, we presume that Algorithm 2 has built the association of $x_1 \rightarrow y_1$. We want to build the association $x_2 \rightarrow y_2$ incrementally. If c_{x_2} and c_{y_2} differ from class c_{x_1} and c_{y_1}, we need to only build a new arrow edge from class c_{x_2} to class c_{y_2}. This new arrow edge does not influence the arrow edge (c_{x_1},c_{y_1}). If either c_{x_2} or c_{y_2} is the same as c_{x_1} or c_{y_1}, for example, $c_{x_2}=c_{x_1}$, and $c_{y_2}\neq c_{y_1}$, then Algorithm 1 incrementally memorizes the pattern x_2 in subnetwork c_{x_1}, and Algorithm 2 updates the weight and associative index of node c_{x_1} in the associative layer. Then, Algorithm 2 finds or generates a node c_{y_2} in the associative layer and builds an arrow edge from c_{x_1} to c_{y_2}, which differs from the arrow edge (c_{x_1},c_{y_1}). In this situation, the pair $x_2 \rightarrow y_2$ is learned incrementally. The situation $c_{x_2}\neq c_{x_1}$, $c_{y_2}=c_{y_1}$ can be analyzed similarly.

From Step 26 of Algorithm 2 we also find that, if an associative relationship exists between the key class and response class, the weight of the arrow edge between the two classes will be incremented. Using this weight of the arrow edge, during the recall, if one key vector is associated with several response classes, we can recall the stored response classes in the order of their weights—the response class with the highest weight is recalled first. Thus, we can easily recall the stored responses associated with the key most frequently.

Because we adopt a separated layer to memorize the associative relationship between classes, building the associative layer with Algorithm 2 also demonstrates that GAM can implement

one-to-one, one-to-many, many-to-one, or many-to-many associations. The third layer of Fig. 1 presents an example of a many-to-many association network.

Furthermore, the adoption of the arrow edge between the classes is very useful for real tasks. For example, for some robotic intelligence tasks, the arrow edge can be set as an instruction for the robot to complete a specific action, as shown in the experiment in Section 5.5.

3.3. Temporal sequence

For temporal sequence associations, the question is, given a key vector, how to associate the complete temporal sequence. This key vector is usually one element chosen randomly from the complete temporal sequence.

To build associations between context elements with time order, we take all elements in the temporal sequence as both key vectors and response vectors, i.e., the former element is a key vector and the following element is the corresponding response vector, and on the contrary, the latter element is also set as a key vector and the former one is set as the corresponding response vector, as shown in Fig. 4. This was done to achieve the goal: randomly choosing one item from the temporal sequence as the key vector, we are able to associate the complete temporal sequence.

We use Algorithm 2 to build an associative relationship between key vectors and their context vectors. At the same time, we store the time order information in the nodes of the associative layer. Algorithm 3 gives details of the temporal sequence training process.

Algorithm 3. Learning of the temporal sequence.

1: Input a temporal sequence $X=x_1,x_2,\ldots,x_n$ with time order $t_1,t_2,\ldots,t_n$. The class names of the sequence items are $c_{x_1},c_{x_2},\ldots,c_{x_n}$.
2: **for** $k=1,2,\ldots,n$ **do**
3: **if** $k<n$ **then**
4: Set x_k as the key vector, x_{k+1} as the response vector.
5: Use Algorithm 2 to build an associative connection between x_k and x_{k+1}. The corresponding nodes in the associative layer are b_{x_k} and $b_{x_{k+1}}$.
6: **end if**
7: **if** $k>1$ **then**
8: Set x_k as the key vector, x_{k-1} as the response vector.
9: Use Algorithm 2 to build an associative connection between x_k and x_{k-1}. The corresponding nodes in the associative layer are b_{x_k} and $b_{x_{k-1}}$.
10: **end if**
11: Update the time order of b_{x_k} with:

$$TF_{b_{x_k}}[m_{b_{x_k}}]=t_{k-1} \quad (14)$$

$$T_{b_{x_k}}[m_{b_{x_k}}]=t_k \quad (15)$$

$$TL_{b_{x_k}}[m_{b_{x_k}}]=t_{k+1} \quad (16)$$

12: **end for**

Algorithm 3 builds an associative relationship between context patterns in the temporal sequence. With the associative relationship described here, randomly given one item in any position of the temporal sequence as a key vector, it is possible for the GAM to recall the complete temporal sequence. In addition, Algorithm 3 is suitable for incremental learning. For example, if we want to add items $x_{n+1},x_{n+2},\ldots,x_{n+m}$ to the temporal sequence X with time order $t_{n+1},t_{n+2},\ldots,t_{n+m}$, we only need to repeat Steps 2–12 of Algorithm 3 for $k=n,n+1,..n+m$. This lets it to learn the new items incrementally, without destroying the learned associative relationship.

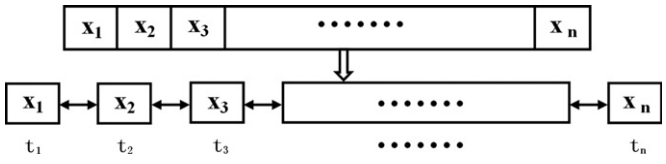

Fig. 4. Temporal sequence is separated into a series of patterns with time order. Every element of a temporal sequence serves as both key vector and response vector.

If we want to learn a new temporal sequence Y that is different from X, we use Algorithm 3 to train Y and build an associative relationship between items of Y in the associative layer. We increment the associative index m_i of the repeated class i to store the corresponding response class and time order if some items of the Y sequence are repeated with some items of the X sequence. Consequently, the learning results of sequence Y do not influence the learned results of sequence X.

Using the associative index m_i for response class RC_i, time order T_i, the time order of the latter pattern TL_i, and the time order of the former pattern TF_i ensures that even if several repeated or shared items exist in a temporal sequence, the GAM can recall the complete temporal sequence accurately. Temporal sequences with repeated or shared items are difficult for some traditional AM systems, as described in Section 1.

4. Recall and associate

In Section 2, we describe the GAM network structure. Section 3 discusses GAM learning algorithms. This section shows the recall and associating algorithms.

When a key vector is presented, the AM is expected to return a stored memory pattern corresponding to that key. Typical AM models use both auto-associative and hetero-associative mechanisms [28]. Auto-associative information supports the processes of recognition and pattern completion. Hetero-associative information supports the processes of paired-associate learning. In this section, we explain the recall algorithm for auto-associative tasks, and subsequently discuss the associating algorithm of hetero-associative tasks; the hetero-associative tasks include one-to-one, one-to-many, many-to-one, and many-to-many associations. We also describe the recall algorithm of the temporal sequence.

4.1. Recall in auto-associative mode

For auto-associative tasks, the AM is expected to recall a stored pattern resembling the key vector such that noise-polluted or incomplete inputs can also be recognized. Fig. 5 shows the basic idea for auto-associative tasks. Some attractors exist in the vector space, and every attractor has an attraction basin. If the input

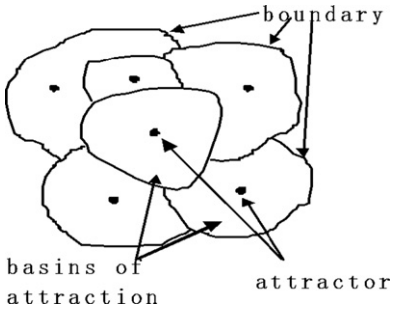

Fig. 5. Every attractor has an attraction basin. If the input pattern (key vector) is located in an attraction basin, the corresponding attractor is the associated result. If the input key vector lies outside all attraction basins, the key vector fails to recall the memorized pattern.

pattern (key vector) is located in an attraction basin, the corresponding attractor will be the associated result. If the input key vector lies outside all attraction basins, the key vector fails to recall the memorized pattern.

As shown in Fig. 3, SOINN separates input patterns into different Voronoi regions; each Voronoi region acts as an attraction basin for an associative process, and the node in the Voronoi region acts as an attractor. For the associative process, if an input key vector lies in one Voronoi region V_i, we assign the weight vector $\boldsymbol{W}_i$ of the corresponding node i as the associative result. If the distance between the input vector and one node is greater than the similarity threshold that is the radius of the Voronoi region, it means that the input key vector is beyond the attraction basin. If the input key vector is beyond the corresponding Voronoi regions for all nodes, then the key vector fails to recall the memorized pattern. Algorithm 4 gives the detail for the auto-associative recall process.

Algorithm 4. Auto-associative: recall the stored pattern with a key vector.

1: Assume there are n nodes in the memory layer, input a key vector x.
2: **for** $i=1,2,\ldots,n$ **do**
3: Calculate the weight sum of input vector, and $\frac{1}{2}\|\boldsymbol{W}_i\|^2$: is a bias.
$$g_i(x)=\boldsymbol{W}_i^T x-\tfrac{1}{2}\|\boldsymbol{W}_i\|^2 \quad (17)$$
4: **end for**
5: Find the maximum $g_k(x)=\max_{i=1,2,\ldots,n} g_i(x)$
6: **if** $\|x\|^2-2g_k(x)>T_k$ **then**
7: Output message: x failed to recall the memorized pattern.
8: **else**
9: Output $\boldsymbol{W}_k$ as the recalling pattern.
10: Output the class of node k as the class of x.
11: **end if**

Through Steps 2–5, Algorithm 4 judges which Voronoi region the input x most likely belongs to. This is because

$$\|x-\boldsymbol{W}_i\|^2=\|x\|^2-2\boldsymbol{W}_i^T x+\|\boldsymbol{W}_i\|^2, \quad (18)$$

$\|x\|$ is the common item for all nodes; thus, minimizing $\|x-\boldsymbol{W}_i\|^2$ is equivalent to maximizing $\boldsymbol{W}_i^T x-\frac{1}{2}\|\boldsymbol{W}_i\|^2$, which is calculated in Step 3 of Algorithm 4. In Step 6, Algorithm 4 judges if the distance between the input vector and $\boldsymbol{W}_k$ (which is the center of the Voronoi region v_k) is larger than the similarity threshold T_k (which is the radius of the Voronoi region V_k). If the distance is larger than T_k, it means that x is beyond the Voronoi region V_k. Because x most likely belongs to V_k; therefore, we conclude that x fails to recall the memorizing pattern. If the distance is less than T_k, it means that x is located in the attraction basin of attractor k, we output the $\boldsymbol{W}_k$ as the associative result, and x belongs to the same class of $\boldsymbol{W}_k$.

4.2. Association in the hetero-associative mode

The paired-associate learning task is a standard evaluation of human episodic memory [6]. Typically, the subjects are presented with randomly paired items (e.g., words, letter strings, and pictures) and are asked to remember each $x\rightarrow y$ pair for a subsequent memory test. During the testing, the x items are presented as cues, and the subjects attempt to recall the appropriate y items.

The GAM memorizes the $x\rightarrow y$ pair with Algorithm 2. To associate y with x, first we use Algorithm 4 to recall the stored key class c_x of the key vector x; the corresponding node for class c_x in the associative layer is b_x. Then, we use $RC_{b_x}[k]$, $k=1,\ldots,m_{b_x}$ to obtain the response class c_y and corresponding node b_y. Finally, we output $\boldsymbol{W}_{b_y}$ as the hetero-associative results for the key vector x. Algorithm 5 shows the details of associating y with key vector x.

Using Algorithm 5, we associate a pattern y from the key vector x. If more than one class is associated with a key, we output all associated patterns, which are typical prototypes of response classes. These recalled results are sorted in order of the weights of the associative connection arrows. That is, Algorithm 5 recalls the one-to-many associated patterns in their weighted order. Combined with the learning process Algorithm 2, we know that the GAM is able to realize one-to-one, one-to-many, many-to-one, and many-to-many associations.

Algorithm 5. Hetero-associative: associate stored patterns with key vector x.

1: Input a key vector x.
2: Using Algorithm 4 to classify x to class c_x.
3: In associative layer, find node b_x corresponding to subnetwork c_x.
4: **for** $k=1,2,\ldots,m_{b_x}$ **do**
5: Find the response classes $c_y[k]$: $c_y[k]=RC_{b_x}[k]$.
6: Sort $c_y[k]$ with the order of $W_{(c_x,c_y[k])}$.
7: **end for**
8: **for** $k=1,2,\ldots,m_{b_x}$ **do**
9: Find node $b_y[k]$ in the associative layer corresponding to subnetwork $c_y[k]$.
10: Output weight $\boldsymbol{W}_{b_y[k]}$ as the associated result with key vector x.
11: **end for**

Algorithm 6. Recall temporal sequence with a key vector x.

1: Input a key vector x.
2: Using Algorithm 4 to classify x to class c_x.
3: In associative layer, find node b_x corresponding to subnetwork c_x.
4: **for** $k=1,2,\ldots,m_{b_x}$ **do**
5: Find the corresponding time order t_s^k by $t_s^k=T_{b_x}[k]$, $k=1,2,\ldots,m_{b_x}$.
6: Find the minimal time order t_s^* from t_s^k, $k=1,2,\ldots,m_{b_x}$. The corresponding index is k_s^*.
7: **end for**
8: Output the weight $\boldsymbol{W}_{b_x}$ of node b_x as the recall item for key vector x, the corresponding time order of x is t_s^*.
9: Set $k_L=k_s^*$, node $b=b_x$.
10: **while** Any latter items of the key vector are not recalled **do**
11: Find the time order of the latter pattern by $t_{latter}=TL_b[k_L]$.
12: Find the response class c_y by $c_y=RC_b[k_L]$. For c_y, the corresponding node of c_y in the associative layer is b_y.
13: Output the weight $\boldsymbol{W}_{b_y}$ of b_y as the recalled next item. Output t_{latter} as the time order of the next item.
14: Find index k in node b_y with $T_{b_y}[k]=t_{latter}$, update parameters by $k_L=k$, $b=b_y$.
15: **end while**
16: Set $k_F=k_s^*$, $b=b_x$.
17: **while** Any former items of the key vector are not recalled **do**
18: Find the time order of former item by $t_{former}=TF_b[k_F]$.
19: Find the response class c_y by $c_y=AC_b[k_F]$. For c_y, the corresponding node of c_y in the associative layer is b_y.
20: Output the weight $\boldsymbol{W}_{b_y}$ of b_y as the former item, and output t_{former} as the time order of the former item.

21: Find the index k in node b_y with $T_{b_y}[k] = t_{former}$, update parameters by $k_F = k$, $b = b_y$.
22: **end while**

4.3. Recall temporal sequences

Section 3.3 explains the learning algorithm for temporal sequences. All elements of a temporal sequences are trained as key vectors and response vectors. The time order of every item is memorized in the node of the associative layer. To recall the temporal sequence when a key vector is presented, we first perform auto-association for the key vector with Algorithm 4 and recall the key class. Then, we associate the former and the latter items with the help of the recalled time order of the current item. Finally, we set the associated items as the key vector and repeat the above steps to recall the complete temporal sequence. Algorithm 6 gives details of recalling a temporal sequence from a key vector.

In Algorithm 6, we first recall the item corresponding to the key vector; then, we recall the latter items and the former items of the recalled key with the help of the time order stored in the associative layer. Because only one time order corresponds to one item of the temporal sequence, even if several repeated or shared items exist in the temporal sequence, Algorithm 6 can recall the complete temporal sequence accurately. With the learning process in Algorithm 3 and recall process in Algorithm 6, the GAM accurately associates temporal sequences.

In fact, without using of the time order, the recall and associating processes in Algorithm 6 are the same as those in Algorithm 5. Algorithm 4 forms the basis of Algorithms 5 and 6. The recall processes of auto-associative information, hetero-associative information, and temporal sequences are, therefore, uniform. If we define a pattern as a temporal sequence with only one item and define a paired pattern as a temporal sequence with two items, Algorithm 6 can be used for recalling auto-associative and hetero-associative information.

5. Experiment

In this section, we perform experiments to test the GAM. Initially, we use binary (bipolar) and real-value data to test its memory and association efficiency, and then, we use temporal sequential data to test the GAM and compare it with some other methods. Finally, an experiment with a humanoid robot is used to test the GAM for real tasks.

5.1. Binary (bipolar) data

Several traditional AM systems process only binary data. In this experiment, we use a binary text character dataset taken from the IBM PC CGA character font to test the GAM. This dataset has been adopted by methods, such as SOIAM [10] and the Kohonen feature map associative memory (KFMAM) [9], to test their performance. Fig. 6 shows the training data, which consists of 26 capital letters and 26 small letters. Each letter is a 7×7 pixel image, and each pixel can have the values of either -1 (black) or 1 (white). During the memorization, capital letters are used as the key vectors and small letters are used as the response vectors, i.e., $A \rightarrow a$, $B \rightarrow b, \ldots, Z \rightarrow z$.

In [10], with the dataset presented in Fig. 6, Sudo et al. compare the results of their SOIAM with bidirectional AM with the pseudo-relaxation learning algorithm for BAM (PRLAB) [29], KFMAM [9], and KFMAM with weights fixed and semi-fixed neurons (KFMAM-FW) [30]. Here, we compare the GAM with other methods by using the same dataset. For the GAM, the two parameters of SOINN are set as $age_{\max} = 50$ and $\lambda = 50$. For other methods, we adopt the same parameters as those reported in Table 1 of [10].

For the GAM, every letter is considered to be one class; thus, there are 52 classes for this task. For each class, the original training set comprises one pattern (7×7 binary image). To expand the training set, we randomly add 5–20% noise to the original pattern and repeat this process 100 times to obtain 100 training patterns for each class. The noise is generated by the following method: randomly choose some pixels (e.g., 10% of total pixels) and transform their value from 1 to -1 or from -1 to 1. Fig. 7 shows an example. Only the GAM is able to memorize patterns with the class; thus, newly generated patterns are used only for training the GAM. For other methods, original patterns are used as the training set.

First, we test GAM, SOIAM, BAM with PRLAB, KFMAM, and KFMAM-FW under a stationary environment, which means that all pairs $A \rightarrow a$, $B \rightarrow b, \ldots, Z \rightarrow z$ are used to train the systems without changing the data distribution. For the GAM, 90 nodes are automatically generated to memorize the input patterns in the memory layer, and 52 nodes exist in the associative layer to represent the 52 classes. An associative relationship is also built between capital letters and small letters. During the recall process, capital letters (without noise) serve as key vectors. With Algorithm 5, all associated letters are recalled, the accurate recall rate is 100%. SOIAM clustered the combined vectors $A+a, B+b, \ldots, Z+z$, and generated 93 nodes to represent the 26 clusters. When a capital letter serves as the key vector, the letter

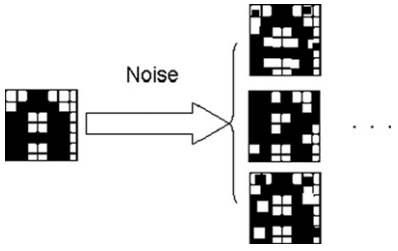

Fig. 7. Generate noise patterns from the original pattern.

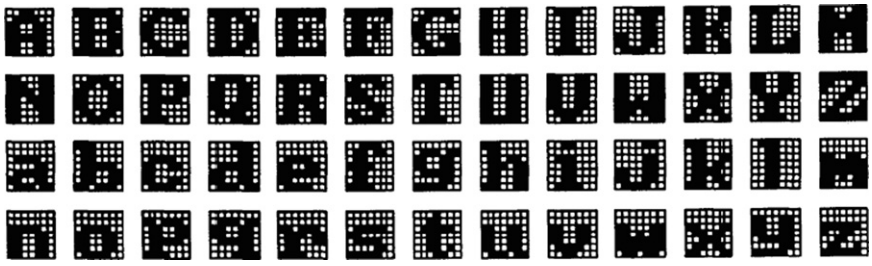

Fig. 6. Binary text character dataset.

is compared with the former part of every node and the nearest one is found; then, the latter part is reported as the associated results. SOIAM also achieved a 100% recall ratio. For BAM with PRLAB, KFMAM, and KFMAM-FW, the training data consist of 26 pairs, $A \rightarrow a$, $B \rightarrow b, \ldots, Z \rightarrow z$. Under this stationary environment, BAM with PRLAB and KFMAM-FW obtained perfect recall results (100%), but KFMAM showed a poor recall ratio of only 63%.

Next, we consider incremental learning. The patterns of $A \rightarrow a$, $B \rightarrow b, \ldots, Z \rightarrow z$ are input into the system sequentially. In the first stage, only $A \rightarrow a$ are memorized, and then $B \rightarrow b$ are input into the system and memorized. This environment is non-stationary—new patterns and new classes are input incrementally into the system. Table 4 compares the results for the GAM with other methods. GAM requires 94 nodes to memorize all 52 classes in the memory layer. One node represents one class in the associative layer; therefore, there are 52 nodes in the associative layer. This gives GAM an accurate recall rate of 100%. It is difficult for BAM and KFMAM to implement incremental learning. New input patterns destroy the memorized patterns. SOIAM requires 99 nodes to represent the association pairs; it recalls the associated patterns with a 100% accurate recall rate. For KFMAM-FW, if we adopt a sufficient number of nodes (more than 36), it can achieve perfect recall results; however, if the maximum number of patterns to be learned is not known in advance, we do not know how to assign the total number of nodes for KFMAM-FW [10].

Third, we consider the many-to-many association. The BAM- and KFMAM-based methods are unsuitable for this task. SOIAM can realize many-to-many associations; however, if it learns a new association pair incrementally, it puts the key and response vectors of a new pair together as one combination vector and sends it to SOIAM for clustering. Consequently, even if we build new associations within the learned letters, SOIAM requires the addition of new nodes to represent the new associations. In [10], pairs such as (A, a), (A, b), (C, c), (C, d) (C, e), (F, f), (F, g), (F, h), and (F, i) as shown in Fig. 8 are used to test the one-to-many association.

To achieve this target, SOIAM combines A and b to produce vector $A+b$, C and d to produce vector $C+d$, and so on, and then clusters such combination vectors with new nodes. New nodes, different from $A+a$, $C+c$, and $F+f$, are added into the system to represent the associative relationship between $A \rightarrow b$, $C \rightarrow d$, etc. To realize such one-to-many associations, SOIAM adds 81 nodes to store the new associative relationship. GAM only adds new associative relationships (arrow edges) between the nodes in the associative layer—no new nodes are added in the memory or associative layer. For example, to realize $A \rightarrow b$ association, we only need to add an arrow edge from node A to node b in the associative layer—no new nodes are generated. Both GAM and SOIAM can effectively recall old associated patterns and new added response vectors (100% accurate recall rate); however, SOIAM requires additional storage and computation time to cluster new association pairs and adds 81 new nodes. On the other hand, GAM requires no new storage and nearly no additional computation time for building new associations. If we want to build much more association between patterns such as $A \rightarrow c$, $A \rightarrow d, \ldots, A \rightarrow z$, and even much more complicated many-to-many association among all capital or small letters, SOIAM must add a large number of new nodes. Whereas, by using GAM, it is easy to build an associative relationship in the associative layer—no additional nodes are needed, and GAM saves significant storage and computation time compared to SOIAM.

Finally, we test how noise influences these methods. Along with our memorized results, we add noise to the capital letters and use the noisy capital letters as key vectors for recalling the memorized small letters. We generated 100 noisy key vectors for each capital letter; 2600 noisy patterns serve as key vectors. Fig. 9 shows the comparison. Fig. 9 shows that GAM and SOIAM are robust with respect to noise, whereas, other methods are sensitive to noise. For example, when the noise level reaches 20%, the recall rate for GAM and SOIAM reached 91.3%, but it was less than 70% for other methods. A noise level of 24% produced a recall rate of greater than 85% for GAM and SOIAM; whereas, for other methods it was less than 55%.

Table 4
Comparison: recalling results of GAM and other methods under an incremental environment.

Method	Number of nodes	Recall rate
GAM	94	100%
SOIAM	99	100%
BAM with PRLAB	–	3.8%
KFMAM	64	31%
	81	38%
	100	42%
KFMAM-FW	16	Infinite loop
	25	Infinite loop
	36	100%
	64	100%

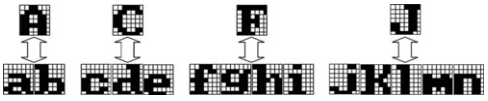

Fig. 8. One-to-many association examples.

5.2. Real-value data

In this experiment, we adopt the AT&T face database, which includes 40 distinct subjects and 10 different images per subject. These subjects are of different genders, ages, and races. For some subjects, the images are taken at different times. There are variations in facial expressions (open/closed eyes, smiling/non-smiling) and facial details (glasses/no glasses). All images are taken against a dark homogeneous background with subjects in an upright frontal position, with tolerance for tilting and rotation of up to about 20°. There is a variation of up to about 10% in the scale.

The original images are grayscale, with a resolution of 112×92. Before presenting them to the GAM, we normalize the value of each pixel to the range $[-1, 1]$. Fig. 10(a) shows 10 images of the same person, and Fig. 10(b) shows the 10 different people to be memorized. To memorize each person, five images are used, and the remaining five images of such persons will be

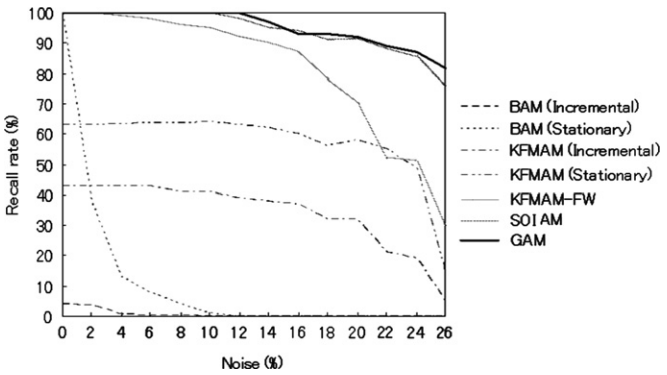

Fig. 9. Comparison: how noisy key vectors influence GAM and other methods.

Fig. 10. Facial image (a) 10 images of one person and (b) 10 different persons.

used to test the memorized efficiency. No overlap exists between the training and test sets.

Under a non-stationary incremental environment, 50 patterns belonging to 10 classes are input sequentially into the system. During the training, SOIAM defines a pattern as a response vector, then combines key vector and the response vector, and sends it to SOIAM for memorization. There are 50 combination vectors, for which SOIAM generates 101 nodes to store those associative pairs. For GAM, the key vectors are memorized in the auto-associative mode—its memory layer will memorize the patterns incrementally and learn the number of nodes automatically. With $age_{\max}=50$ and $\lambda=50$ for GAM, 22 nodes are generated to store the input patterns. The associative layer has 10 nodes representing 10 classes. No association is produced between classes (auto-associative mode). During the recall process, the remaining test data for the same person serve as key vectors. Because the recall performance is affected by the selection of training images, training is repeated 20 times and we report the average recall rate as the recall performance. For each training time, we adopt different training examples (random selection of five images from 10 per subject). SOIAM yields different results with different parameters. Its best recall rate is 95.3%. Using Algorithm 4, we recall the memorized pattern according to the key vectors and the recall rate is 96.1%, which is slightly better than for SOIAM.

Under a stationary environment, both GAM and SOIAM obtained nearly the same results as for the incremental environment.

As described in Section 5.2, if we want to add a new associative relationship between different classes (e.g., use the face of a person as the key vector, associated with a face belongings to another person), SOIAM must add new combination vectors for training and add new nodes to memorize the new knowledge. However, for GAM, if no new class is created, we only need to add a new associative relationship between the nodes in the associative layer, and no new nodes are generated. This is a benefit that is derived from the properties of memorization in classes and association with classes.

In this experiment, we only compared GAM with SOIAM under a non-stationary incremental environment, with no comparison with other methods, because no other methods are suitable for non-stationary incremental learning with real-value data.

5.3. Temporal sequences

In this section, we test the ability of GAM to store and recall temporal sequences. In [13], Sakurai et al. compared their proposed SOM-AM with temporal associative memory (TAM) [6], and conventional SOM [30]. According to [13], two open temporal sequences (Fig. 11) are used. Here, sequence $A\rightarrow B\rightarrow C\rightarrow D\rightarrow E$ is first learned using each method; $F\rightarrow G\rightarrow C\rightarrow H\rightarrow I$ is then learned incrementally as new information. The two temporal sequences have a shared item—C. Subsequent to the training, patterns A and F are used as the key vectors to recall the temporal sequences. In fact, TAM cannot

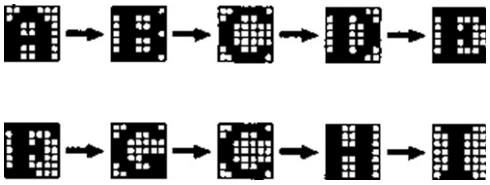

Fig. 11. Two open temporal sequences: C is shared by both sequences.

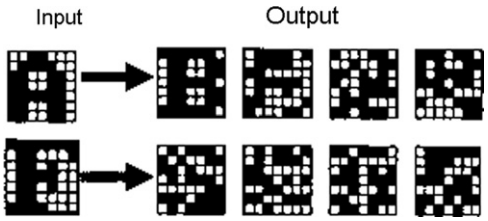

Fig. 12. Recall results of temporal associative memory (TAM), shared item leads to failure.

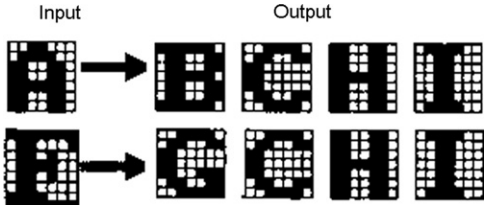

Fig. 13. Recall results of self-organizing map (SOM), shared item leads to failure.

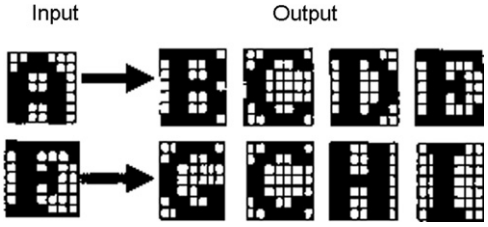

Fig. 14. Recall results of SOM associative memory (SOM-AM), the first item has to serve as key vector.

store one-to-many associations. TAM learning does not converge; it fails to recall the sequences (Fig. 12). For conventional SOM, because the contextual information of temporal sequences is not considered in the learning or recall process, the correct sequence is not recalled (Fig. 13). SOM-AM resolves the ambiguity by using recurrent difference vectors and recalled both temporal sequences accurately (Fig. 14).

For the proposed GAM, the sequence items are first memorized in the memory layer as different classes, and then the associative relationships are built in the associative layer. Using Algorithm 6, GAM can recall sequences with any item as the key vector. While SOM-AM can recall the sequence only if the first item serves as the key vector. For the temporal sequences in Fig. 11, it can recall the first sequence only if pattern A serves as the key vector, and can recall the second sequence only if pattern F serves as the key vector. For GAM, if A, B, D, or E serves as the key vector, the first sequence is recalled. If F, G, H, or I serves as the key vector, the second sequence is recalled. If C serves as the key vector, both the first and the second sequences are recalled. Fig. 15 shows the recall results.

Then, we test the performance of GAM with a real audio data. The data were captured from a recording of the following passage:

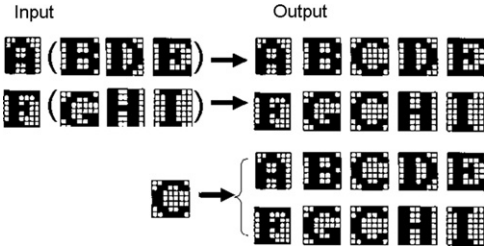

Fig. 15. Recall results of GAM, any item is able to serve as key vector.

Fig. 16. Each image is divided into five image blocks to represent marked feature areas.

We hold these truths to be self-evident that all men are created equal, that they are endowed by the creator with certain unalienable right that among these are life, liberty and the pursuit of happiness.

This paragraph was pronounced slowly such that there was a pause after each word was uttered. We then split this audio file into 35 segments, each containing the audio data of one word. Then, we extracted the 15-dimensional spectrum feature of these 35 segments. Then, we used GAM to memorize the sequence of the audio segments. For testing, we selected an arbitrary word such as "that" or "happiness", as the key vector. The results show that the entire passage was recalled perfectly. Subsequent to the 50 trials with GAM, the recall rate was 100%.

Finally, we compare the experimental results of GAM and Hidden Markov Model (HMM) on the Yale Face Database [31]. This database contains 165 gray-scale images in Graphics Interchange Format (GIF) of 15 individuals. There are 11 images per subject, one per different facial expression, illumination or configuration. They are divided into two parts, 10 or five images for training, and the others for testing.

Each image is partitioned into five overlapped image blocks (Fig. 16), representing five marked feature areas (hair, forehead, eyes, nose and mouth). Instead of using the pixel intensities within each image block, we form an observation vector with the coefficients of the two-dimensional Discrete Cosine Transform (2D-DCT) for each image block. This extracted feature (observation vector) leads to a decrease in the computational complexity and tend to be less sensitive to noise and changes in illumination. In our experiment, the DCT size is 12×12, and at last we get nine DCT coefficients (a 3×3 array of low-frequency coefficients) as observation vectors. After 2D-DCT, each image is described as a sequence of five 9-dimension vectors.

At training phase, we train one Gaussian Mixture Hidden Markov Model (GMHMM) for per individual. Following the model initialization (five hidden states), the model parameters are re-estimated using an expectation maximization (EM) algorithm to maximize the probability of observing data. At the M-step, we adopt Gaussian

probability density function to estimate the emission probability

$$b_j(o) = \sum_{s=1}^{R} c_{j,s} G(o; u_{j,s}, U_{j,s}) \quad \text{for } j = 1,2\ldots,N, \tag{19}$$

where o is the observation vector, G is Gaussian distribution, $u_{j,s}$ and $U_{j,s}$ is the expectation and variance of the sth Gaussian distribution of the jth state. The parameters $u_{j,s}$ and $U_{j,s}$ are re-estimated by the weighted expectation and covariance matrix of the observations.

Similar to HMM, one GAM is trained for one individual. We initialize the network with five nodes in a supervised way. Then the observation vectors are fed into the network to activate the winner neuron and adjust its weights. At testing phase, we use the following score function to estimate the similarity between testing images and GAMs

$$score(j) = \sum_{i=1}^{5} \|v_i - v_i^j\|^2 \quad \text{for } j = 1,2\ldots,15, \tag{20}$$

where v_i is the ith block images of testing image and v_i^j is the ith neuron of the j th GAM.

Table 5 shows the comparison results of HMM and GAM. It shows that our model and HMM can accurately recognize all test images when ten images of per individual participate the training. When only five images of one individual is trained, the recognition rate of GAM is 94%, which is slightly better then HMM.

5.4. Real task for robot with GAM

This experiment uses a humanoid robot with an image sensor and sound sensor to test whether GAM is applicable to real tasks. A humanoid robot, HOAP-3 (shown in Fig. 18) (Fujitsu Ltd.), is adopted for this experiment. Fig. 17 shows four objects used for testing GAM: an apple, an orange, a bell, and a drum.

Table 5
Comparison: recalling results of GAM and HMM.

Method	Number of training images	Recall rate (%)
HMM	5	92
	10	100
GAM	5	94
	10	100

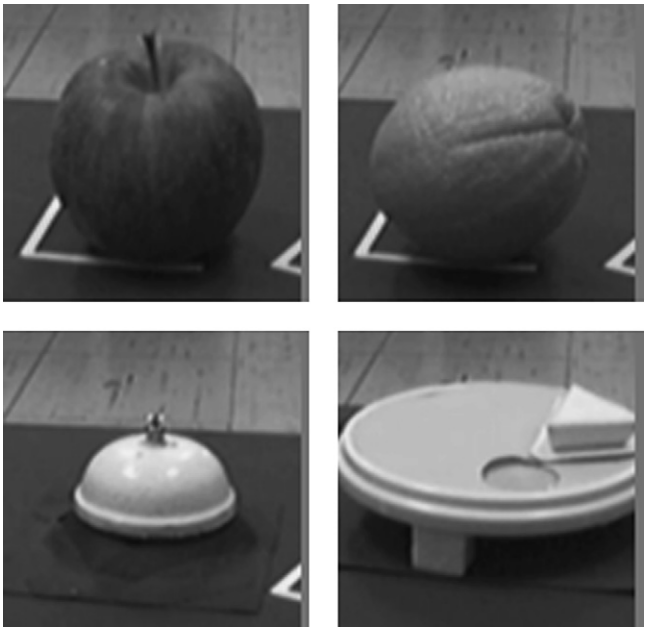

Fig. 17. Four objects memorized by a humanoid robot: an apple, an orange, a bell, and a drum.

For the apple and orange, we merely show them to HOAP-3 and make HOAP-3 memorize their images in the GAM memory layer. We set the associative action as the instruction for HOAP-3 to point its finger at the objects. This task takes place in the auto-associative mode. During the testing, if the image of the apple or orange is shown to HOAP-3, it must remember what the image is and point its finger at the object.

For the bell, we show it to HOAP-3 and then push the button of the bell so that the robot can sense the bell sound. Image of the bell serves as the key vector and sound of the bell serves as the response vector. The associative action is set as an instruction for HOAP-3 to use its finger to push the button. On the other hand, sound of the bell serves as the key vector and image of the bell serves as the response vectors. The associative action is set as an instruction for HOAP-3 to point its finger at the bell. This task takes place in the hetero-associative mode. During the testing, if HOAP-3 is shown a bell, it must recall the sound of the bell, and use its finger to push the bell button. On the other hand, if HOAP-3 hears the sound of the bell, it should points its finger at the bell.

Similarly, image and sound of the drum serve as the key and response vectors, respectively. The corresponding associative action is also set as an instruction for HOAP-3 to hit the drum with its hand or point its finger at the drum.

The patterns and association pairs described above are presented to a GAM built in HOAP-3's brain for storing objects and building associations between key-response pair vectors. For features of images collected using the HOAP-3 image sensor, we perform grayscale transformation and adopt 36-dimensional low-frequency DCT coefficients as the feature vectors. For features of the sounds collected using the HOAP-3 sound sensor, we extract the 15-dimensional spectrum feature at a 20 kHz, 50 ms sampling rate.

Subsequent to training the memory system (GAM) of HOAP-3, we test HOAP-3 with some key vectors. First, we show an apple to HOAP-3. HOAP-3 recalls the apple's image, turns its head to the apple, and points its finger at the apple according to the memorized associative action (instruction). Similarly, when we show a orange, a bell or a drum to the HOAP-3, it can recall the correct actions. Fig. 18 shows the HOAP-3 points its finger at the sound source(bell) when it hear the sound of the bell.

We repeat the above experiment 10 times, and HOAP-3 successfully completes the task every time. Table 6 is the statistical results for hetero-associative of bell (image and sound). For drum, the results are same as for bell.

This experiment demonstrates that GAM can perform a real task efficiently. It recalls the objects (apple, orange, bell, and

Fig. 18. Humanoid robot HOAP-3 (Fujitsu Ltd.). The top-right image is obtained from HOAP-3 image sensor. After hearing the bell, HOAP-3 turns its head to the bell and watches it; it then points its finger at the bell.

Table 6
Result of hetero-associative for robot.

Key vector	Response vector	Number of experiments	Recall rate (%)
Sound of bell	Image of bell	10	100
Image of bell	Sound of bell	10	100

Table 7
Comparison of GAM with other typical AM systems: which target can it achieve? Mark ○ for best systems; mark △ for systems with lower performance than ○; mark × for systems unable to realize the target.

Targets \ AM systems	Hopfield net	BAM	KFM AM	SOM-AM	MAM	SOIAM	GAM
Memory with class	×	×	×	×	×	×	○
Non-binary data	×	×	×	×	×	○	○
Incremental learning	×	×	×	×	×	△	○
Temporal sequence	×	×	×	△	×	×	○
Many-to-many association	×	×	×	×	△	△	○
Patterns with different dimensions	×	×	×	×	×	×	○

drum) effectively in auto-associative mode. It also builds effective association between objects (bell and drum) and their sounds, and the association is represented by actions of the robot. With the stimuli of key vectors, GAM recalls the response patterns effectively in a hetero-associative mode. Also note that, in this experiment, although the dimensions of image and sound are different, GAM builds an effective association between vectors with different dimensions. Thus, GAM is capable of process different data types.

6. Conclusion

This paper proposes a GAM system constructed using a network consisting of three layers: input, memory, and associative. The input vectors are memorized in the memory layer. Associative relationships are built in the associative layer. Patterns are memorized with classes. New information (new patterns or new classes) is stored incrementally. Patterns with binary or non-binary data can be stored in the memory layer. The associative layer accommodates one-to-one, one-to-many, many-to-one, and many-to-many associations. New associations can be added incrementally between the stored items. The special use of arrow edges in the network makes GAM adaptable to real tasks. The GAM can also store and recall temporal sequences.

Based on experiments reported in Section 5, Table 7 facilitates the comparison of GAM with other typical AM systems from the viewpoint of functionalities. The first column of Table 7 presents a list of targets, which are very important for AM systems. If a system can realize a target effectively, it is marked with ○, and if a system cannot realize the target, the mark × is assigned to it. △ indicates that a system can realize the target but the performance is below that of systems marked with a ○. Table 7 shows that GAM realizes all the targets well. Among the other systems, few can deal with one or more targets, but none of them can deal adequately with all targets.

In conclusion, GAM combines several functions from different AM systems. Experiments with binary data, real-value data, temporal sequences, and real-world robot tasks show that GAM is effective for all the targets listed in Table 7.

In the future, we will further analyze true intelligence using GAM. Based on the symbol grounding theory, the subnetwork of the memory layer will be considered as primitive atomic symbol tokens, and the associative layer as the set of rules in order to construct symbol-token strings to realize an infinite variety of complicated human brain behavior.

Acknowledgements

This work was supported in part by the 973 Program 2010CB327903, the Fund of the National Natural Science Foundation of China 60975047, and Jiangsu NSF grant BK2009080, BK2011567.

References

[1] Bai ling Zhang, P. Cerone, Robust face recognition by hierarchical kernel associative memory models based on spatial domain gabor transforms, J. Multimedia 1 (4) (2006) 1–10.
[2] Mustafa C. Ozturk, Jose C. Principe, An associative memory readout for esns with applications to dynamical pattern recognition, Neural Networks 20 (2007) 377–390.
[3] Kazuko Itoha, Hiroyasusnm Miwab, Hideakisnm Takanobud, Atsuosnm Takanishi, Application of neural network to humanoid robot-sdevelopment of co-associative memory model, Neural Networks 18 (2005) 666–673.
[4] N. Ikeda, P. Watta, M. Artiklar, M. Hassoun, A two-level hamming network for high performance associative memory, Neural Networks 14 (9) (2001) 1189–1200.
[5] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. U.S.A. 79 (1982) 2554–2588.
[6] B. Kosko, Bidirectional associative memories, IEEE Trans. Systems, Man Cybernet. 18 (1) (1988) 49–60.
[7] R.M. French, Using semi-distributed representation to overcome catastrophic forgetting in connectionist networks, in: Proceeding of the 13th Annual Cognitive Science Society Conference, 1991, pp. 173–178.
[8] Peter Sussner, Marcos Eduardo Valle, Gray-scale morphological associative memories, IEEE Trans. Neural Networks 17 (3) (2006) 559–570.
[9] T. Kohonon, Self-Organization and Associative Memory, Springer-Verlag, Berlin, 1984.
[10] Akihito Sudo, Akihitosnm Sato, Osamu Hasegawa, Associative memory for online learning in noisy environments using self-organizing incremental neural network, IEEE Trans. Neural Networks 20 (6) (2009) 964–972.
[11] M. Hattori, M. Hagiwara, Episodic associative memory, Neurocomputing 12 (1996) 1–18.
[12] G. de A. Barreto, A.F.R. Ara ujo, Storage and recall of complex temporal sequences through a contextually guided self-organizing neural network, in: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, 2000.
[13] Naoaki Sakurai, Motonobu Hattori, Hiroshi Ito, Som associative memory for temporal sequences, in: Proceedings of the 2002 International Joint Conference on Neural Networks, 2002, pp. 950–955.
[14] M. Hagiwara, Multidirectional associative memory, in: Proceedings of the 1990 International Joint Conference on Neural Networks, 1990, pp. 3–6.
[15] Stevan Harnad, The symbol grounding problem, Physica D 42 (1990) 335–346.
[16] Rolf Pfeifer, Christian Scheier, Understanding Intelligence, The MIT Press, 2001.
[17] T.M. Martinetz, S.G. Berkovich, K.J. Schulten, "Neural-gas" network for vector quantization and its application to time-series prediction, IEEE Trans. Neural Networks 4 (4) (1996) 558–569.
[18] S.P. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory IT-28 (1982) 2.
[19] J. MacQueen, Some methods for c1assification and analysis of multivariate observations, in: L.M. LeCam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics, and Probability, 1967, pp. 281–297.
[20] F. Shen, O. Hasegawa, An incremental network for on-line unsupervised classification and topology learning, Neural Networks 19 (2006) 90–106.
[21] T. Kohonen, Self-organized formation of topologically correct feature maps, Biol. Cybernet. 43 (1982) 59–69.
[22] B. Fritzke, A growing neural gas network learns topologies, Adv. Neural Inf. Proces. Syst. 7 (1995) 625–632.
[23] Danielesnm Casali, Giovannisnm Costantini, Renzo Perfetti, Elisa Ricci, Associative memory design using support vector machines, IEEE Trans. Neural Networks 17 (5) (2006) 1165–1174.
[24] Simon Haykin, Neural Networks: A Comprehensive Foundation, Macmillan Coll Div, 1994.
[25] Linde, Buzo, Gray. An algorithm for vector quantizer design. IEEE Transactions on Communication, COM-28 (1980) 84–95.
[26] F. Shen, O. Hasegawa, An adaptive incremental LBG for vector quantization, Neural Network 19 (2006) 694–704.

[27] F. Shen, O. Hasegawa, An enhanced self-organizing incremental neural network for online unsupervised learning, Neural Networks 20 (2007) 893–903.
[28] D.S. Rizzuto, M.J. Kahana, An autoassociative neural networkmodel of paired-associate learning, Neural Comput. 13 (2001) 2075–2092.
[29] H. Oh, S.C. Kothari, Adaptation of the relaxation method for learning in bidirectional associative memory, IEEE Trans. Neural Networks 5 (4) (1994) 576–583.
[30] T. Yamada, M. Hattori, M. Morisawa, H. Ito, Sequential learning for associative memory using kohonen feature map, in: Proceedings of the 1999 International Joint Conference on Neural Networks, 1999, pp. 1920–1923.
[31] A.S. Georghiades, P.N. Belhumeur, D.J. Kriegman, From few to many: illumination cone models for face recognition under variable lighting and pose, IEEE Trans. Pattern Anal. Mach. Intell. 23 (6) (2001) 643–660.

Furao Shen received the Engineering degree from Tokyo Institute of Technology, Tokyo, Japan, in 2006. Currently he is an associate professor at Nanjing University. His research interests include neural computing and robotic intelligence.

Qiubao Ouyang received the BS degree in mathematics in 2009 and the MS degree in computer science in 2012 from Nanjing university, Nanjing, China. His current research interests include neural networks, artificial intelligence, and image processing.

Wataru Kasai received the Engineering degree in intelligence system from Tokyo Institute of Technology, Tokyo, Japan. His research interests include pattern recognition and machine learning.

Osamu Hasegawa received the Engineering degree in electronic engineering from the University of Tokyo, Tokyo, Japan, in 1993. He was a Research Scientist with the Electrotechnical Laboratory from 1993 to 1999 and with the National Institute of Advanced Industrial Science and Technology, Tokyo, from 2000 to 2002. From 1999 to 2000, he was a Visiting Scientist with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. In 2002, he became a faculty member with the Imaging Science and Engineering Laboratory, Tokyo Institute of Technology, Yokohama, Japan. In 2002, he was jointly appointed as a Researcher at PRESTO, Japan Science and Technology Agency. He is a member of the IEEE Computer Society, Institute of Electronics, Information and Communication Engineers, and Information Processing Society of Japan.