An Associative Memory System for Incremental Learning and Temporal Sequence

Furao Shen, *Member, IEEE*, Hui Yu, Wataru Kasai and Osamu Hasegawa, *Member, IEEE*

Abstract— **An associative memory (AM) system is proposed to realize incremental learning and temporal sequence learning. The proposed system is constructed with three layer networks: The input layer inputs key vectors, response vectors, and the associative relation between vectors. The memory layer stores input vectors incrementally to corresponding classes. The associative layer builds associative relations between classes. The proposed method can incrementally learn key vectors and response vectors; store and recall both static information 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 the proposed method works well.**

I. Introduction

An associative memory (AM) is a memory that stores data in a distributed fashion and which is addressed through its contents. They can recall information from incomplete or garbled inputs. Traditionally, when an input pattern, called a key vector, is presented, the associative memory is expected to return a stored memory pattern (called a response vector) associated with a key. However, Human memory can learn new knowledge incrementally and not destroy old learned knowledge. Also, for human beings, temporal sequences are not memorized as a static pattern, but memorized as patterns with a consecutive relation. It means that AM systems must incrementally memorize and associate new information without destroying stored knowledge. Also, systems must not only memorize static information, but also store temporal sequence information.

For incremental learning, Sudo et al. proposed a self-organizing incremental associative memory (SOIAM) [1] specifically to examine incrementally stored new patterns without destruction of memorized information, but that memory system is unable to address temporal sequences. Kosko [2] and Hattori and Hagiwara [3] processed temporal sequences, but their method can deal only with simple temporal sequences. Presented with some repeated or shared items existing in the temporal sequences, they cannot work well. Barreto and Araujo [4] learned the temporal order through a time-delayed Hebbian learning rule, but the complexity of the model depends highly on the number of context units. Sakurai et al. [5] proposed a self-organizing map based associative memory (SOM-AM) for temporal sequences, but its recall performance is affected by the initial values of weights. Such methods only considered the binary temporal sequence without touching on the real-valued sequence. Furthermore, it is difficult for such methods to realize incremental learning for temporal sequences.

F. Shen (frshen@nju.edu.cn) and H. Yu (ukei.yu@gmail.com) are with the State Key Laboratory for Novel Software Technology, and Jiangyin Information Technology Research Institute, Nanjing University, Nanjing, 210093, P.R. China.

W. Kasai is with the Department of Computer Intelligence and Systems Science, Tokyo Institute of Technology, Yokohama, 226-8503, Japan. E-mail: kasai.w.aa@m.titech.ac.jp

O. Hasegawa is with the Imaging Science and Engineering Lab., Tokyo Institute of Technology, R2-52, 4259 Nagatsuda, Midori-ku, Yokohama, 226-8503, Japan. E-mail: hasegawa@isl.titech.ac.jp

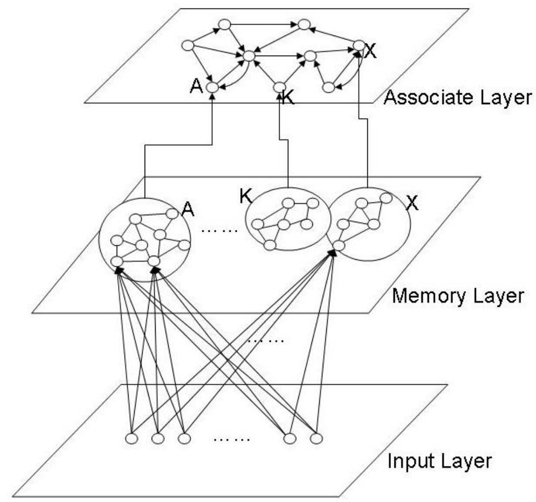

Fig. 1. Network structure of proposed Associative Memory.

In this paper, we propose an associative memory system to realizing incremental learning and temporal sequence learning. We construct a three-layer network to realize our targets, with an input layer, a memory layer, and an associative layer. The input layer is used to input key vector and response vector to memory layer. The memory layer is used to store information from the input layer. Both key vector information and response vector information can be stored in the memory layer. For learning of the memory layer, incremental learning is available: new key vectors and response vectors can be memorized incrementally. The associative layer will be used to build the association relation between the key vector and the response vector. In this layer, we will construct temporal sequence associations.

II. Structure of proposed Associative Memory

We designed a three-layer network to realize our design targets discussed in section I. Figure 1 presents the three-layer network structure.

The input layer is used to input patterns. The input feature vector (key vector or response vector) is input into the system

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

with a class label. According to the class label, the proposed method finds the corresponding sub-network in the memory layer, and incrementally learns the new input information. If the input key vector or response vector does not belong to any class existing in the memory layer, then the new input vector will become the first node of a new class (new sub-network) and the new class will be added to the memory layer. The class label of memory layer will be sent to the associative layer; the associative layer will build a relation between the key vector's class (called key class) with the response vector's class (called response class) with arrow edges. One node exists in the associative layer corresponding to one sub-network in the memory layer. Arrow edges connect such nodes to build an association relation. The beginning of a connection is the key class. The end of the connection is the corresponding response class.

The input layer will input both binary patterns and non-binary patterns. In the memory layer, the proposed method uses different sub-networks to memorize different classes. Context patterns of temporal sequence can be memorized in the memory layer, and the association time order can be memorized in the associative layer. Incremental learning can be realized during the training of the memory layer and associative layer.

III. Learning algorithms

During memory layer training, how to realize incremental learning is important. When new patterns are input, we must memorize such new patterns without destroying the stored patterns.

During training of the associative layer, we input the key-response pair to the system. The association relation between the key vector and response vector will be memorized in the associative layer. In addition, the context relation between temporal sequences can be input to the associative layer as input data; the associative layer must be able to memorize the associative relation and the time order between context patterns. The associative layer must be able to learn the new association incrementally if a new association between memorized classes occurs.

A. Memory layer

The memory layer (see Fig. 1) comprises some sub-networks; each sub-network is used to represent one class. All patterns belonging to one class will be memorized in the corresponding sub-network.

Herein, we adopt a self-organizing incremental neural network (SOINN) [6] to build the memory layer. It is based on competitive learning. Neural nodes are used to represent the data distribution of input data. The weights of such nodes are used to store the input patterns.

Self-organizing incremental neural network (SOINN) [6] and its enhanced version [7] execute topology representation and incremental learning without predetermination of network structure and size; SOINN is able to realize both real-valued pattern memorization and incremental learning. Self-organizing incremental associative memory (SOIAM) [1] is based on SOINN. Here, the basic idea of training memory layer for the proposed method is also enlightened by SOINN. We adjust the unsupervised SOINN for supervised mode: for each class we adopt one SOINN to represent the distribution of that class. The input patterns (key vectors and response vectors) are separated to different classes. For each class, we use one sub-network to represent the data distribution of the class.

Algorithm 3.1 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. If there is already a sub-network with the same class name as the input vector, we use training algorithm of SOINN to update the sub-network with the input vector.

Algorithm 3.1: Learning of the memory layer

1. Initialize the memory layer network: node set A, sub-network set S, and connection set C, $C \subset A \times A$ to the empty set.

$$A = \emptyset, S = \emptyset, C = \emptyset \tag{1}$$

2. Input a pattern $x \in R^n$ to the memory layer, the class name of x is c_x.
3. If there is no sub-network with name c_x, then add a sub-network c_x to the memory layer. This sub-network has a node c_x^1, and the node c_x^1 is added to the node set A, i.e., $S = S \cup \{c_x\}$, $A = A \cup \{c_x^1\}$.
4. If there already exists a sub-network c_x in memory layer, then update the sub-network with SOINN.

According to [7], SOINN is able to represent the topology structure of input data. For that reason, Algorithm 3.1 can represent the topology structure of input patterns. It uses weights of nodes in the memory layer to represent the input pattern. It can also realize incremental learning. New classes are learned incrementally by adding new sub-networks. New patterns inside one class are learned incrementally by adding new nodes to the sub-network. The number of sub-networks is determined by the number of classes in the input patterns. When a new class arises, the memory layer can react for the new class without destroying other old classes. Inside one class, SOINN controls the increment of nodes to learn new knowledge without unlimited increase of number of nodes.

B. associative layer

Associative layer will be used to build association between key vectors and response vectors. We designate the class a "key class", to which the key vector belongs, and call the class the "response class", to which the response vector belongs. In the associative layer (see Fig. 1), nodes are connected with arrow edges. Each node represents one class: the beginning of the arrow means the key class; the end of the arrow means the 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 can be input incrementally into the system. First, Algorithm 3.1 is used to memorize the information of both the key vector and the response vector. Then, the class name of the key class and associative class will be sent to the associative layer. In the associative layer, if there already exist nodes representing the key class and response class, then we connect the nodes of the key class and response class with an arrow edge. If no node represents the key class (or response class) within the associative layer, then add a node to the associative layer and use that node to express the new class. Then build an arrow edge between the key class and response class. Algorithm 3.2 gives details of how to build an association between the key vector and response vector.

Algorithm 3.2: Learning of the associative layer

1. Initialize the associative layer network: node set B, arrow edge set D, and $D \subset B \times B$ to the empty set.

$$B = \emptyset, D = \emptyset \tag{2}$$

2. Input a key vector $x \in R^n$; the class name of x is c_x. The typical prototype of class c_x is p_{c_x}.
3. Use Algorithm 3.1 to memorize key vector x in the memory layer.
4. If no node exists in the associative layer representing class c_x, then insert a new node b, representing class c_x, into the associative layer ($B = B \cup \{b\}$) with

$$c_b = c_x \tag{3}$$

$$W_b = W_{p_{c_x}} \tag{4}$$

$$m_b = 0 \tag{5}$$

If there already exists a node b representing class c_x, then

$$m_b \leftarrow m_b + 1 \tag{6}$$

5. Input the response vector $y \in R^m$, the class name of y is c_y. The typical prototype of class c_y is p_{c_y}.
6. Use Algorithm 3.1 to memorize the response vector y in the memory layer.
7. Find node d in the associative layer representing class c_y. If there is no node representing class c_y, then insert a new node d, representing class c_y, into the associative layer with

$$c_d = c_y \tag{7}$$

$$m_d = 0 \tag{8}$$

$$W_d = W_{p_{c_y}} \tag{9}$$

If there already exists a node d representing class c_y, then do

$$m_d \leftarrow m_d + 1 \tag{10}$$

8. If there is no arrow between node b and d, 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.

$$D = D \cup \{(b, d)\} \tag{11}$$

Set the m_bth response class of b as c_d,

$$AC_b[m_b] = c_d \tag{12}$$

Set the weight of arrow (b, d) as 1,

$$W_{(b,d)} = 1 \tag{13}$$

In Step2 and Step4 of Algorithm 3.2, a typical prototype of class c_x or c_y is mentioned. This typical prototype is predefined for the corresponding class. It is user defined pattern and it represents the class.

Algorithm 3.2 can realize incremental learning. For example, we presume that Algorithm 3.2 has built the association of $x_1 \rightarrow y_1$. We want to build $x_2 \rightarrow y_2$ association incrementally. If c_{x_2} and c_{y_2} differ from class c_{x_1} and c_{y_1}, we need only build a new arrow edge from class c_{x_2} to class c_{y_2} with Algorithm 3.2. This new edge has no influence to the edge (c_{x_1}, c_{y_1}). If one of c_{x_2} and 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 3.1 will memorize the pattern x_2 in sub-network c_{x_1} incrementally, and Step 4 of Algorithm 3.2 will be used to update the account index of node c_{x_1} in the associative layer. Then Step 6, Step 7 and Step 8 of Algorithm 3.2 are used to generate a node c_{y_2} in the associative layer and build an arrow edge from c_{x_1} to c_{y_2}, which differs from edge (c_{x_1}, c_{y_1}). In this situation, the pair $x_2 \rightarrow y_2$ is learned incrementally. For the situation $c_{x_2} \neq c_{x_1}$, $c_{y_2} = c_{y_1}$, we can give a similar analysis.

C. Temporal Sequence

For the temporal sequence association, the question is, given a key vector, how to associate the full temporal sequence. This key vector might be one pattern chosen randomly from the whole temporal sequence. The chosen pattern might be polluted by noise.

To build associations between context patterns with time order, we will take all patterns in the temporal sequence as both key vectors and response vectors, i.e., the former pattern is a key vector and the following pattern is the corresponding response vector, and on the contrary, the latter pattern is also set as a key vector and the former one is set as the corresponding response vector. We do this because we want to realize this goal: randomly choosing one part from the temporal sequence as the key vector, we can associate the full temporal sequence. The key vector and its context vectors build the training pairs. We use Algorithm 3.2 to build an association relation between key vectors and their context vectors. At the same time, we save the time order information in the contents of nodes in the associative layer. Algorithm 3.3 includes details of the temporal sequence training process.

Algorithm 3.3: Learning of the temporal sequence

1. Input a temporal sequence $X = x_1, x_2, ..., x_n$ with time order $t_1, t_2, ..., t_n$. The class names of the sequence items are $c_{x_1}, c_{x_2}, ..., c_{x_n}$.
2. For $k = 1, 2, ..., n$, do the following Step 3 – Step 4.

3. If $k < n$, then set x_k as the key vector, x_{k+1} as the response vector, then use Algorithm 3.2 to build an association connection between x_k and x_{k+1}. The corresponding nodes in the associative layer are b_{x_k} and $b_{x_{k+1}}$.
4. If $k > 1$, set x_k as the key vector, x_{k-1} as the response vector, then use Algorithm 3.2 to build an association connection between x_k and x_{k-1}. The corresponding nodes in the associative layer are b_{x_k} and $b_{x_{k-1}}$.
5. Update the time order of b_{x_k} with the following.

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

$$T_{b_{x_k}}[m_{b_{x_k}}] = t_k \tag{15}$$

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

Algorithm 3.3 can build an association relation between context patterns in the temporal sequence. With the association relation described here, randomly given one pattern in any position of the temporal sequence as a key vector, it is possible for the proposed method to recall the full temporal sequence. In addition, Algorithm 3.3 is suitable for incremental learning. For example, if we want to add some new items $x_{n+1}, x_{n+2}, ..., x_{n+m}$ to the temporal sequence X with time order $t_{n+1}, t_{n+2}, ..., t_{n+m}$, then we need only repeat Step 3 – Step 5 of Algorithm 3.3 for $k = n, n+1, ..n+m$, we can incrementally learn the new items without destroying the learned association relation. If the new item is not added behind the temporal sequence but inserted into the sequence in any position, that is, if x_{new} between x_k and x_{k+1} with time order t_{new} is inserted into the temporal sequence X, then we need only remove the association between x_k and x_{k+1}, then repeat Step 3 – Step 5 of Algorithm 3.3 for x_k, x_{new}, and x_{k+1}. Here, t_{new} is a different value from $t_1, t_2, ..., t_n$ with any value.

We use Algorithm 3.3 to train Y and build an association relation between items of Y in the associative layer if we want to learn a new temporal sequence Y that is different from X. We increment the account index m_i of those 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.

The use of account index m_i for response class AC_i, time order T_i, time order of the latter pattern TL_i, and time order of the former pattern TF_i ensures that even if plenty of repeated or shared items exist in a temporal sequence, then the proposed method is able to recall the whole temporal sequence correctly. Such temporal sequences with repeated or shared items are difficult for some traditional associative memory systems, as we described in section I.

IV. Recall and associate

When a key vector is presented, the associative memory is expected to return a stored memory pattern that is coincident with the key. Typical associative memory models use both auto-associative and hetero-associative mechanisms [8]. 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 recalling algorithm for auto-associative tasks, and subsequently discuss the associating algorithm of hetero-associative tasks; We also describe the recalling algorithm of the temporal sequence in this section.

A. *Recall in auto-associative mode*

For auto-associative tasks, the associative memory is expected to be able to recall a stored pattern resembling the key vector such that noise-polluted or incomplete inputs can also be recognized.

When a key vector is presented, if the class name of the key vector is available, then we will find the corresponding node in the associative layer. The weight of the node will be the recall result. We will use the k-nearest neighbor rule to determine which class the key vector belongs to if the key class is unavailable. Assuming that the determined class is c, we output the weight of the corresponding node of class c in the associative layer as the recalling result. Algorithm 4.1 gives details related to the recall process.

Algorithm 4.1: Recall the stored pattern with a key vector

1. Input a key vector x.
2. Find the corresponding node b with class name c_x in the associative layer if the class name c_x of x is available.
3. If the class name c_x of x is unavailable, then do the following Step 4 – Step 6.
   4. Find the first k-nearest nodes to key vector x in the whole network of memory layer as

$$s_1 = \underset{i \in A}{\operatorname{argmin}} ||x - W_i||_d \tag{17}$$

$$s_2 = \underset{i \in A \setminus s_1}{\operatorname{argmin}} ||x - W_i||_d \tag{18}$$

$$\vdots$$

$$s_k = \underset{i \in A \setminus s_1, s_2, ..., s_{k-1}}{\operatorname{argmin}} ||x - W_i||_d \tag{19}$$

   the class names of found nodes $s_1, s_2, ..., s_k$ are $c_1, c_2, ..., c_k$.
   5. Do major voting for classes $c_1, c_2, ..., c_k$; obtain the major repeated class c.
   6. Find the corresponding node b with class name c
7. Output W_b as the recall result for key vector x.

In Algorithm 4.1, one parameter k is needed. We can tune this parameter using some methods such as cross-validation.

B. *Associate in hetero-associative mode*

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

With Algorithm 3.2, the proposed method can memorize the $x \rightarrow y$ pair. To associate y from x. First, we use Algorithm 4.1 to recall the stored key class c_x of key vector x, the corresponding node for class c_x in the associative layer is b_x; then, we use $AC_{b_x}[k], k = 1, ..., m_{b_x}$ to obtain the response class c_y and corresponding node b_y. Finally, we output W_{b_y} as the associating results for key vector x. Algorithm 4.2 shows details of associating y from key vector x.

Algorithm 4.2: Associate the stored y pattern from key vector x

1. Input a key vector x.
2. Using Algorithm 4.1, find the class name c_x of x and the corresponding node b_x in the associative layer.
3. For $k = 1, 2, ..., m_{b_x}$, do the following Step 4 – Step 5.
    4. Find the response class c_y:
    $$c_y = AC_{b_x}[k]. \tag{20}$$
    5. Find node b_y in the associative layer corresponding to class c_y. Then output weight W_{b_y} as the associated result of key vector x

Using Algorithm 4.2, we can associate a pattern y from key vector x. If more than one class associated from a key class exists, then we output all associated patterns, which are typical prototypes of response classes.

C. Recall temporal sequence

Section III-C explains the learning algorithm for temporal sequences. All elements of 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 do auto-association for the key vector with Algorithm 4.1 and recall the key class. Then, with the recalled time order of the current item, former item, and latter item to associate the former and next items, we set the associated items as the key vector and repeat the steps listed above to recall the full temporal sequence. Algorithm 4.3 gives details of recalling a temporal sequence from a key vector.

Algorithm 4.3: Recall temporal sequence from a key vector

1. Input a key vector x.
2. Using Algorithm 4.1 to find the class name c_x of x, and find the corresponding node b_x for class c_x in the associative layer.
3. For $k = 1, 2, ..., m_{b_x}$, find the corresponding time order t_s^k by
$$t_s^k = T_{b_x}[k], k = 1, 2, ..., m_{b_x}. \tag{21}$$
Find the minimal time order t_s^* from $t_s^k, k = 1, 2, ..., m_{b_x}$. The corresponding index is k_s^*.
4. Output the weight of node b_x as the recall item for key vector x, the corresponding time order of x is t_s^*.
5. To recall the latter items of the current recalled item, set $k_L = k_s^*$, node $b = b_x$, and do Step 6 – Step 8.

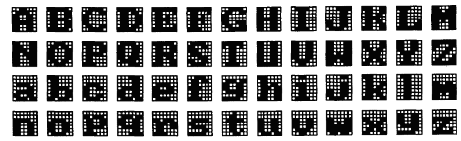

Fig. 2. Binary text character dataset

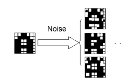

Fig. 3. Generate noise patterns from the original pattern

    6. Find the time order of the latter pattern by
    $$t_{latter} = TL_b[k_L]. \tag{22}$$
    Find the response class c_y using
    $$c_y = AC_b[k_L]. \tag{23}$$
    The corresponding node to c_y in the associative layer is b_y.
    7. Output the weight W_{b_y} of b_y as the recalled next item. Output t_{latter} as the time order of the next item.
    8. Find index k in node b_y with $T_{b_y}[k] = t_{latter}$, update parameters using $k_L = k, b = b_y$, go to Step 6 to recall the next item until all latter items of the key vector are recalled.
9. To recall the former items of the current item, set $k_F = k_s^*$, $b = b_x$, and do Step 10 – Step 12.
    10. Find the time order of former item by $t_{former} = TF_b[k_F]$. Find the response class c_y by $c_y = AC_b[k_F]$. The corresponding node to c_y in the associative layer is b_y.
    11. Output the weight W_{b_y} of b_y as the former item, and output t_{former} as the time order of the former item.
    12. Find the index k in node b_y with $T_{b_y}[k] = t_{former}$, update parameters by $k_F = k$, $b = b_y$, go to Step 10 to recall the former items until all former items of the key vector are recalled.

In Algorithm 4.3, we first recall the item corresponding to the key vector; then we recall the latter items and former items 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 plenty of repeated or shared items exist in the temporal sequence, then Algorithm 4.3 can recall the full temporal sequence correctly. With the learning process in Algorithm 3.3 and recalling process in Algorithm 4.3, we can realize association of the temporal sequence well.

V. Experiment

In this section, we describe some experiments to test the proposed method. First, we adopt real-world data to test

TABLE I

COMPARISON: RECALLING RESULTS OF THE PROPOSED METHOD AND OTHER METHODS UNDER AN INCREMENTAL ENVIRONMENT

Method	Number of nodes	Recall rate
Proposed	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%

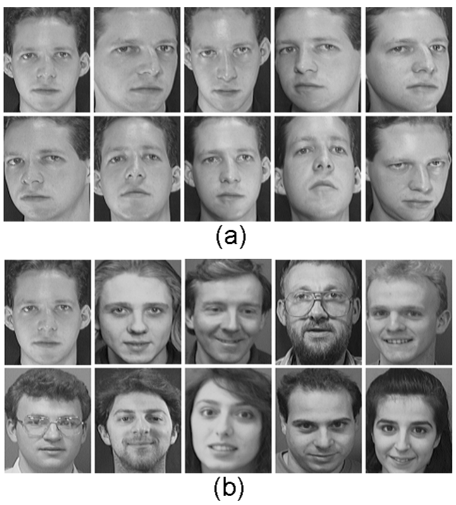

Fig. 4. Facial image (a) 10 images of one person, (b) 10 different person

the incremental learning efficiency of the proposed method. Then, we use some temporal sequential data to test the proposed method and compare it with some other methods.

A. Binary (bipolar) data

Many traditional associative memory systems can only process binary data. In this experiment, we use a binary text character dataset taken from the IBM PC CGA character font to test the proposed method. This dataset is adopted by some methods such as SOIAM [1] and the Kohonen Feature Map associative memory (KFMAM) dataset [9] to test their performance. Figure 2 portrays the training data, comprising 26 capital letters and 26 small letters; each letter is a 7×7 pixel image, and every pixel has only -1 (black) or 1 (white) value. During memorization, capital letters are used as the key vectors; small letters are used as the response vectors, i.e., $A \rightarrow a$, $B \rightarrow b$, ..., $Z \rightarrow z$.

In [1], with the dataset presented in Fig.2, A. Sudo et al. compare results of their proposed SOIAM with bidirectional associative memory with the Pseudo-Relaxation Learning Algorithm for BAM (PRLAB) [10], KFMAM [9], and KF-MAM with weights fixed and semi-fixed neurons (KFMAM-FW) [11]. Here, using the same dataset, we compare the proposed method with other methods. For SOINN used in the proposed method, $age_{max} = 50$, $\lambda = 50$. For other methods, we adopt the same parameters as those reported in Table I of [1].

For the proposed method, every letter is thought of as one class; there are 52 classes for this task. For every 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. Such original patterns are set as the typical prototype of the classes. The noise is generated using the following method: randomly choose some pixels (e.g., 10% of total pixels) and transform the value of such pixels from 1 to -1 or from -1 to 1. Figure 3 portrays one example. Only the proposed method is able to memorize patterns with class, thus newly generated patterns are used only for training of the proposed method. For other methods, original patterns are used as training set.

We first test the proposed method, SOIAM, BAM with PRLAB, KFMAM, and KFMAM-FW under a stationary environment. All pairs $A \rightarrow a$, $B \rightarrow b$, ..., $Z \rightarrow z$ are therefore used to train the systems without changing the data distribution. For the proposed method, 90 nodes are generated automatically to memorize the input patterns in the memory layer; 52 nodes exist in the associative layer to represent the 52 classes. An association relation is also built between capital letters and small letters. During the recall process, capital letters (without noise) serve as key vectors. With Algorithm 4.2, all associated letters are recalled, the correct recall rate is 100%. For SOIAM, it clustered the combine vector $A + a$, $B + b$, ..., $Z + z$, and generated 93 nodes to represent the 26 clusters. When a capital letter is served as a key, the letter is compared with the former part of every node and finds the nearest one; then the backward part is reported as the associated results. Actually, SOIAM also got a 100% recall ratio. For BAM with PRLAB, KFMAM, and KFMAM-FW, the training data are 26 pairs, $A \rightarrow a$, $B \rightarrow b$, ..., $Z \rightarrow z$. Under this stationary environment, BAM with PRLAB and KFMAM-FW can get perfect recall results (100%), but KFMAM worked poorly, with only a 63% recall ratio.

Secondly, we consider incremental learning. The patterns of $A \rightarrow a$, $B \rightarrow b$, ..., $Z \rightarrow z$ are input into the system sequentially. At the first stage, only $A \rightarrow a$ are memorized, then $B \rightarrow b$ are input into the system and memorized, and so on. This environment is non-stationary, new patterns and new classes will be input incrementally into the system. Table I shows comparison results between the proposed method and other methods. For the proposed method, 94 nodes in all are needed for memorization of all 52 classes in the memory layer. One node represents one class in the associative layer. Therefore, 52 nodes exist in the associative layer. The correct recall rate is 100% for the proposed method. It is difficult for BAM and KFMAM to realize incremental learning. Later input patterns will destroy the memorized patterns. For SOIAM, it needs 99 nodes to represent the association pairs; it recalls the associated patterns with a 100% correct recall

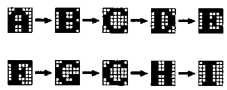

Fig. 5. Two open temporal sequences: C is shared by both sequences

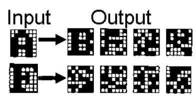

Fig. 6. Recall results of Temporal Associative Memory (TAM)

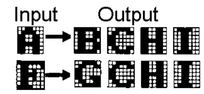

Fig. 7. Self-organizing Map (SOM) recall results

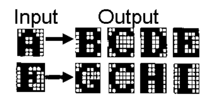

Fig. 8. SOM Associative Memory (SOM-AM) recall results

rate. For KFMAM-FW, if we adopt sufficient nodes (more than 36), then it can achieve perfect recalling results. We must mention that if the maximum number of patterns to be learned is not revealed in advance, then we do not know how to give the total number of nodes for KFMAM-FW [1].

B. Real-value data

For this experiment, we adopt 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 were taken at different times. There are variations in facial expression (open/closed eyes, smiling/nonsmiling) and facial details (glasses/no glasses). All images were taken against a dark homogeneous background with subjects in an upright frontal position, with tolerance for some tilting and rotation of up to about 20 deg. There is some variation in scale of up to about 10%.

The original images are grayscale, with a resolution of 112×92. Before presenting them to the proposed system, we normalize the value of each pixel to the range [-1, 1]. Figure 4(a) presents 10 images of the same person; Fig. 4(b) portrays the 10 different people to be memorized. For every person, five images are used to memorize the person, and the remaining five images of such person will be 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 training, SOIAM puts together a key vector and the response vector (here is the typical prototype) and sends it to SOIAM for memorization. There are 50 combination vectors, for which SOIAM generated 101 nodes to store those associative pairs. For the proposed method, the key vectors are memorized under auto-associative mode. The memory layer of the proposed method will memorize such patterns incrementally; the number of nodes of every class is learned automatically. With $age_{max} = 50$, $\lambda = 50$, 22 nodes in all are generated to store the input patterns. The associative layer has 10 nodes representing 10 classes. No association between classes is produced (auto-associative mode). During the recall process, the remaining test data of the same person are served as key vectors. The recall performance will be affected by the selection of training images. Therefore, the reported results are obtained by training 20 times and selecting the average recall rate overall results. For each training time, we adopt different training examples (random selection of five images from 10 per each subject). With different parameters, SOIAM yields different results. Its best recall rate is 95.3%. With Algorithm 4.1, we recall the memorized pattern according to the key vectors, and with $k = 1$, the recalling rate is 96.1%, which is slightly better than SOIAM.

For a stationary environment, nearly the same results as those for an incremental environment were obtained for the proposed method and SOIAM.

In this experiment, we only compared the proposed method with SOIAM under a non-stationary incremental environment, with no comparison to other methods. It is for the reason that there are no other methods suit non-stationary incremental learning with real-value data.

C. Temporal sequence

In this section, we describe some experiments that were undertaken to test the ability of the proposed method for storing and recalling temporal sequences. In [5], N. Sakurai et al. compared their proposed SOM-AM with Temporal Associative Memory (TAM) [2], and conventional SOM [11]. According to [5], two open temporal sequences (Fig. 5) are used. At first, sequence A→B→C→D→E was learned using each method; then F→G→C→H→I was learned incrementally as new information. The two temporal sequences have a shared item C. After training, pattern A and F are used as the key vectors to recall temporal sequences. In fact, TAM can not store one-to-many associations, the learning of TAM did not converge; it failed to recall the sequences (Fig. 6). For conventional SOM, because the contextual information of temporal sequences is not considered both in the learning and in the recall process, the correct sequence was not recalled (Fig. 7). Actually, SOM-AM resolved the ambiguity using recurrent difference vectors and recalled both temporal sequences correctly (Fig. 8).

For the proposed AM system, the sequence items are first memorized in the memory layer as different classes; then the association relation are built in the associative layer. With Algorithm 4.3, we can recall all sequences with any input key vector. For SOM-AM, only pattern A served as a key vector can recall the first sequence; only pattern F served as a key vector can recall the second sequence. For the proposed

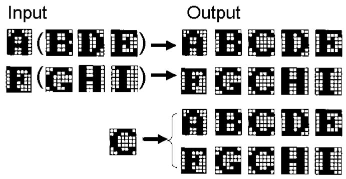

Fig. 9. Proposed associative memory recall results

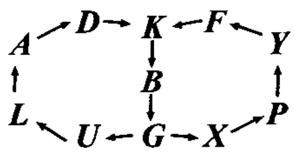

Fig. 10. Closed temporal sequence

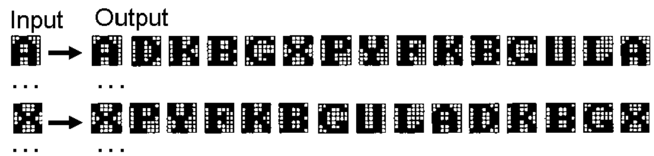

Fig. 11. Proposed associative memory for closed temporal sequence recall results

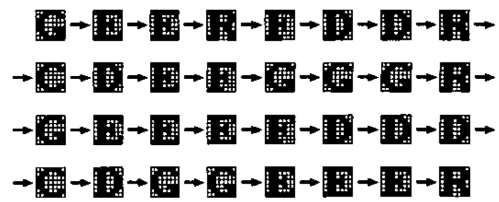

Fig. 12. Complicated temporal sequence: *chou chou* song

method, if A, B, D, or E is served as a key vector, then the first sequence is recalled. If F, G, H, or I is served as a key vector, then the second sequence is recalled. If C is served as a key vector, then both the first sequence and the second sequence are recalled. Figure 9 portrays the recall results.

Figure 10 portrays a closed temporal sequence; the items K, B, and G are consecutive and occur twice. Such a situation is difficult for some methods such as TAM, SOM, and Contextually guided AM system [4]. Actually, SOM-AM can store and recall this sequence with the first item A served as key vector, but if we adopt a key vector other than A, then SOM-AM is unable to recall the sequence. For the proposed method, we can recall the sequence with any input key vector. Figure 11 portrays the recall results.

Finally, we compare the proposed method with SOM-AM using a complicated sequence: half of a famous Japanese children's song, *chou chou*. The song has 32 notes including four-quarter rests. According to [5], to store the song, we can respectively encode do, re, mi, fa, sol, and a quarter rest with alphabetical patterns, C, D, E, F, G, and R. Figure 12 portrays the song sequence. This temporal sequence is complex because each item appears several times: C, D, E, F, G, and R repeat respectively 2, 7, 9, 3, 7, and 4 times. Then SOM-AM is examined for 50 trials: the first note G served as key vector for recalling. In some trials, SOM-AM failed to recall the song, the perfect recall rate was 86%. Using the proposed method, any note can be used as a key vector; the song can be recalled perfectly. We also do 50 trials for the proposed method; the perfect recall rate is 100%.

VI. Conclusion

As described in this paper, an associative memory is proposed to realize incremental learning and temporal sequence learning, based on three-layer network. Input vectors are memorized in memory layer. Association relations are built in associative layer. Patterns are memorized with classes. New information (new patterns or new classes) can be stored incrementally. Patterns with binary or non-binary data can be stored in the memory layer. With the associative layer, new associations can be added incrementally between stored items. The storage and recall of temporal sequences is also available. Experiments for binary data, real-value data, and temporal sequence show the efficiency of the proposed method.

Acknowledgments

This work was supported in part by China NSF grant (#60975047, #60723003, #60721002), Jiangsu NSF grant (#BK2009080), and 973 Program (2010CB327903). This work is also supported in part by NEDO of Japan.

References

[1] Akihito Sudo, Akihiro Sato, and Osamu Hasegawa. Associative memory for online learning in noisy environments using self-organizing incremental neural network. *IEEE Trans. on Neural Networks*, 20(6):964–972, 2009.

[2] B. Kosko. Bidirectional associative memories. *IEEE Trans. Systems, Man and Cybernetics*, 18(1):49–60, 1988.

[3] M. Hattori and M. Hagiwara. Episodic associative memory. *Neurocomputing*, 12:1–18, 1996.

[4] G. de A. Barreto and A. F. R. Ara ujo. Storage and recall of complex temporal sequences through a contextually guided self-organizing neural network. *Proc. of the IEEE-INNS-ENNS Int. Joint Conf. on Neural Networks*, 2000.

[5] Naoaki Sakurai, Motonobu Hattori, and Hiroshi Ito. Som associative memory for temporal sequences. *Proceedings of the 2002 International Joint Conference on Neural Networks*, pages 950–955, 2002.

[6] F. Shen and O. Hasegawa. An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, 19:90–106, 2006.

[7] F. Shen and O. Hasegawa. An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks*, 20:893–903, 2007.

[8] Daniel S. Rizzuto and Michael J. Kahana. An autoassociative neural networkmodel of paired-associate learning. *Neural Computation*, 13:2075–2092, 2001.

[9] T. Kohonen. Self-organization and associative memory. *Berlin: Springer-Verlag*, 1984.

[10] H. Oh and S.C. Kothari. Adaptation of the relaxation method for learning in bidirectional associative memory. *IEEE Trans. On Neural Networks*, 5(4):576–583, 1994.

[11] T. Yamada, M. Hattori, M. Morisawa, and H. Ito. Sequential learning for associative memory using kohonen feature map. *Proc. of the 1999 International Joint Conference on Neural Networks*, pages 1920–1923, 1999.