A Multidirectional Associative Memory Based on Self-organizing Incremental Neural Network

Hui Yu[1,2], Furao Shen[1,2], and Osamu Hasegawa[3]

[1] National Key Laboratory for Novel Software Technology, Nanjing University, China
frshen@nju.edu.cn
http://cs.nju.edu.cn/rinc/
[2] Jiangyin Information Technology Research Institute, Nanjing University, China
[3] Imaging Science and Engineering Lab., Tokyo Institute of Technology, Japan
hasegawa.o.aa@m.titech.ac.jp

Abstract. A multidirectional associative memory (AM) is proposed. It is constructed with three layer networks: an input layer, a memory layer, and an associate layer. The proposed method is able to realize many-to-many associations with no predefined conditions, and the association can be incrementally added to the network without destruction of old associations. Experiments show that the proposed AM works well for real tasks.

Keywords: Incremental learning; Self-organizing incremental neural network; Many-to-many association.

1 Introduction

An associative memory (AM) is a memory that stores data in a distributed fashion and which is addressed through its contents. Traditional methods such as the Hopfield network [1], the bidirectional associative memory (BAM) [2] and their variants realize one-to-one association, i.e., according to one key vector, only one stored vector is recalled. However, with a stimuli, we human beings usually remember much things rather than one. We hope AM can simulate the memory of human beings by realizing many-to-many association.

Another challenge is incremental learning of associations. We human beings are capable of learning new knowledge without destruction of learned knowledge. Therefore, AM should incrementally memorize new key-response information without destroying stored key-response information.

Some neural models have been proposed for multidirectional AM or incremental learning. Recently published self-organizing incremental associative memory (SOIAM) [3] incrementally stores new key-response pairs without destruction of memorized information, however, to realize incremental learning, SOIAM spends plenty of storage and computation cost. M. Hagiwara proposed a multidirectional AM [4] for many-to-many association, but the association was not flexible. It was necessary to preset the number of layers with the predetermined number of associations.

K.W. Wong et al. (Eds.): ICONIP 2010, Part II, LNCS 6444, pp. 344–351, 2010.

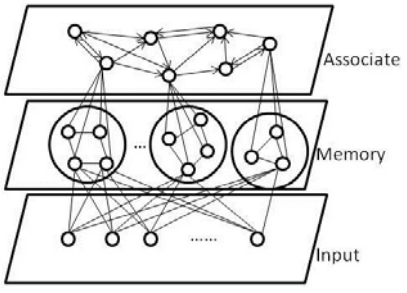

Fig. 1. Network structure of the proposed multidirectional AM

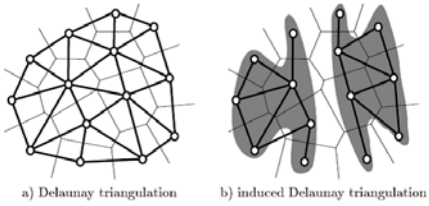

Fig. 2. a) *Delaunay triangulation* (thick lines) connects points having neighboring *Voronoi polygons* (thin lines). b) *Induced Delaunay triangulation* (thick lines): masking Delaunay triangulation with a data distribution (shaded)

In this paper, we propose a multidirectional AM to realize many-to-many association and incremental learning. A three-layer network is adopted for our targets (Fig. 1). The input layer inputs key vector, response vector, and association into the AM system. The memory layer stores information coming from the input layer. The associate layer builds the associative relation between the key vector and the response vector. In associate layer, we incrementally construct many-to-many associations.

2 Learning Algorithms

2.1 Memory Layer

Herein, we adopt a self-organizing incremental neural network (SOINN) [5] to build the memory layer. SOINN 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. The memory layer comprises some sub-networks, and each sub-network is used to represent one class. For each class we adopt one SOINN to represent the distribution of that class. Algorithm 1 shows the learning algorithm of memory layer.

Algorithm 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: $A = \emptyset$, $S = \emptyset$, $C = \emptyset$.
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**
4: Add a sub-network c_x to memory layer. This sub-network has a node c_x^1, the weight of c_x^1 is set as x.
5: Add node c_x^1 to the node set A, i.e., $S = S \cup \{c_x\}$, $A = A \cup \{c_x^1\}$.
6: **else**
7: Update the sub-network c_x with SOINN (Algorithm 2.1 in [5]).
8: **end if**

According to [6], to build connections among neural nodes, SOINN adopts the competitive Hebbian rule [7]: for each input signal, connect the two closest nodes with an edge. This rule forms a network whose edges are in the area suggested by input data distributions. The network represents a subgraph of the Delaunay triangulation (Fig. 2-a). Using the competitive Hebbian rule, the resultant graph approximates the shape of the input data distributions (Fig. 2-b). Hence, Algorithm 1 is capable of representing the input data distribution well. The nodes of sub-networks are the centers of Voronoi regions. Such nodes serve as the attractors for future recalling phase, the Voronoi regions form the basins of attraction.

In Algorithm 1, each class is allocated a sub-network. It shows that, for different classes, the dimension of vectors might be different. On other words, the memory layer is capable of memorizing data of different types (patterns with any different dimensions).

Algorithm 2. Learning of the associate layer

Initialize the associate layer network: node set B, arrow edge set $D \subset B \times B$ to the empty set: $B = \emptyset$, $D = \emptyset$
Input a key vector $x \in R^n$, the class name of x is c_x.
Use Algorithm 1 to memorize key vector x in the memory layer.
if No node b exists in the associate layer representing class c_x **then**
    Insert a new node b representing class c_x into the associate layer: $B = B \cup \{b\}$, $c_b = c_x$, $m_b = 0$, $W_b = x$.
else
    Increment the associative index of b: $m_b \leftarrow m_b + 1$;
    Find node i that is most frequently being winner in sub-network c_x.
    Update the weight of node b in associate layer: $W_b = W_{c_x^i}$.
end if
Input the response vector $y \in R^m$, the class name of y is c_y.
Use Algorithm 1 to memorize the response vector y in the memory layer.
if No node d representing class c_y in the associate layer **then**
    Insert a new node d representing class c_y into the associate layer: $B = B \cup \{d\}$, $c_d = c_y$, $m_d = 0$, $W_d = y$.
else
    Find node i which is most frequently being winner in sub-network c_y.
    Update the weight of node d in associate layer: $W_d = W_{c_y^i}$.
end if
if There is no arrow between node b and d **then**
    Connect node b and d with an arrow edge.
    Add arrow (b, d) to connection set D: $D = D \cup \{(b, d)\}$,
    Set the m_bth response class of b as c_d: $RC_b[m_b] = c_d$,
    Set the weight of arrow (b, d) as 1: $W_{(b,d)} = 1$.
else
    Set the m_bth response class of b as c_d: $RC_b[m_b] = c_d$,
    Increment the weight of arrow (b, d) with 1: $W_{(b,d)} \leftarrow W_{(b,d)} + 1$.
end if

2.2 Associate Layer

Associate layer is 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 associate layer, 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.

For training of the associate layer, firstly, Algorithm 1 is used to memorize information of both the key vector and the response vector. Then, the class name of the key class and response class are sent to the associate layer. In the associate layer, if there already exist nodes representing the key class and response class, 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 associate layer, we add a node to the associate layer and use that node to express the new class and then we build an arrow edge between the key class and response class.

Algorithm 2 gives details of learning associate layer. The building of the associate layer with Algorithm 2 discloses that it can realize many-to-many associations. The third layer of Fig. 1 presents an example of a many-to-many association network.

In the associate layer, weight vector of every node is selected from the corresponding sub-network of memory layer. Step 8, 9, 16, and 17 in Algorithm 2 show that the node that is most frequently being winner is chosen as the typical node of the sub-network in memory layer, and the weight vector of the typical node is set as the weight of that class node in associate layer.

Algorithm 2 is able to realize incremental learning. For example, we presume that Algorithm 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}. This new arrow edge has no influence to the arrow 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 1 memorizes the pattern x_2 in sub-network c_{x_1} incrementally, and Algorithm 2 updates the weight and associative index of node c_{x_1} in the associate layer. Then Algorithm 2 finds or generates a node c_{y_2} in the associate layer and build an arrow edge from c_{x_1} to c_{y_2}, which differs from arrow 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.

3 Recall and Associate

3.1 Recall in Auto-associative Mode

Figure 3 shows the basic idea for auto-associative task. There exists some attractor, and every attractor has an attraction basin. If the key vector is located in an attraction basin, the corresponding attractor will be the associated result.

According to Fig. 2, the memory layer separates input patterns to different Voronoi regions, every Voronoi region acts as attraction basin for associative

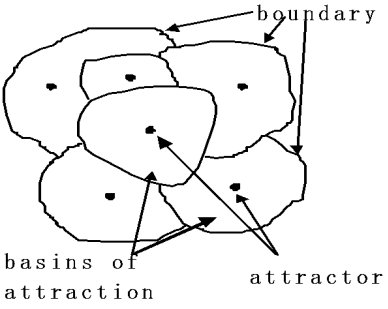

Fig. 3. Every attractor has an attraction basin. If key vector is located in an attraction basin, the corresponding attractor is the associated result.

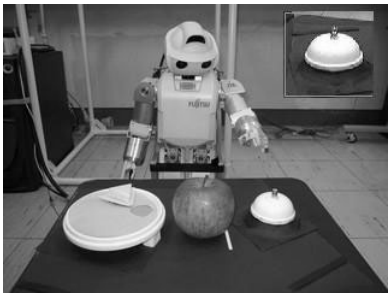

Fig. 4. Humanoid robot HOAP-3. After hearing the bell sound, HOAP-3 turns its head to the bell and watches the bell; then it points to the bell with its finger.

process, and the node in the Voronoi region acts as attractor. For the associative process, if an input key vector lies in one Voronoi region V_i, we give the weight vector W_i of the corresponding node i as the associative result. Algorithm 3 gives the detail for auto-associative recalling process.

Algorithm 3. 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** $i = 1, 2, ..., n$ **do**
3: Calculate the weight sum of input vector, and $\frac{1}{2}||W_i||^2$ is a bias.

$$g_i(x) = W_i^T x - \frac{1}{2}||W_i||^2 \tag{1}$$

4: **end for**
5: Find the maximum $g_k(x) = \max_{i=1,2,...,n} g_i(x)$
6: Output W_k as the recalling pattern.
7: Output the class of node k as the class of x.

By step 2-5, Algorithm 3 judges to which Voronoi region the input x most likely belongs. It is because

$$||x - W_i||^2 = ||x||^2 - 2W_i^T x + ||W_i||^2 \tag{2}$$

$||x||$ is the common item for all nodes, thus minimize $||x - W_i||^2$ is equivalent to maximize $W_i^T x - \frac{1}{2}||W_i||^2$, which is calculated in step 3 of Algorithm 3.

3.2 Associate in Hetero-associative Mode

With Algorithm 2, the proposed AM memorizes the $x \rightarrow y$ pair. To associate y from x, firstly we use Algorithm 3 to recall the stored key class c_x of key vector x, the corresponding node for class c_x in the associate layer is b_x; then, we use

Algorithm 4. Hetero-associative: associate stored patterns with a key vector

1: Input a key vector x.
2: Using Algorithm 3 to classify x to class c_x.
3: In associate layer, find node b_x corresponding to sub-network c_x.
4: **for** $k = 1, 2, ..., m_{b_x}$ **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** $k = 1, 2, ..., m_{b_x}$ **do**
9: Find node $b_y[k]$ in the associate layer corresponding to sub-network $c_y[k]$.
10: Output weight $W_{b_y[k]}$ as the associated result of key vector x.
11: **end for**

$RC_{b_x}[k], k = 1, ..., m_{b_x}$ to obtain the response class c_y and corresponding node b_y. Finally, we output all W_{b_y} as the hetero-associative results for key vector x. Algorithm 4 shows details of associating y from key vector x.

4 Experiment

4.1 Binary (Bipolar) Data

Here we use a binary text character dataset taken from the IBM PC CGA character font. This dataset is adopted by some methods such as SOIAM [3], BAM with PRLAB [8], Kohonen feature map associative memory (KFMAM) [9], and KFMAM-FW [10] to test their performance. There are 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, and small letters are used as the response vectors, i.e., $A \rightarrow a$, $B \rightarrow b$, ..., $Z \rightarrow z$.

Firstly, 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 are incrementally input to the system. Table 1 shows comparison results between the proposed AM and other methods. For the proposed AM, 94 nodes in all are needed for memorization. The correct recall rate is 100%. 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 rate. For KFMAM-FW, if we adopt sufficient nodes (more than 36), it can achieve perfect recalling results. We must mention that if the maximum number of patterns to be learned is not revealed in advance, we do not know how to give the total number of nodes for KFMAM-FW [3].

Then, we consider the many-to-many association. The BAM based and KFMAM based methods are unsuitable for this task. In fact, SOIAM can realize many-to-many association. However, for SOIAM, if it incrementally learns a new

Table 1. Comparison: recalling results of the proposed AM and other methods under an incremental environment

Method	Number of nodes	Recall rate
Proposed AM	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%

association pair, SOIAM will put together the key vector and response vector of new pair to one combination vector and send it to SOIAM for clustering. In [3], pairs such as (A, a), (A, b), (C, c), (C, d) (C, e), (F, f), (F, g), (F, h), and (F, i) are used to test the one-to-many association. To realize this target, SOIAM puts together A and b to produce vector $A+b$, C and d to produce vector $C+d$, and so on, 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 relation between $A \rightarrow b$, $C \rightarrow d$, etc. With the proposed AM, we need only add new associative relation (arrow edge) between nodes in the associate layer without adding new nodes in both the memory layer and associate layer. For example, to realize $A \rightarrow b$ association, we need only add an arrow edge from node A to node b in the associate layer: no new nodes are generated. Both the proposed AM and SOIAM can recall old associated patterns and new added response vectors well (100% correct recall rate). However, SOIAM spends new storage and computation time to cluster new association pairs and adds 81 new nodes. The proposed AM requires no new storage and nearly no additional computation for building new associations, and it saves much more storage and computation time than SOIAM.

4.2 Real Task for Robot with GAM

This experiment uses a humanoid robot with an image sensor and sound sensor to test whether the proposed AM is applicable to real tasks. A humanoid robot HOAP-3 (as depicted in Fig. 4) (Fujitsu Ltd.) is adopted for this experiment.

We show a bell to HOAP-3 and then push the button of the bell to present the bell sound to HOAP-3. Sounds of the bell are served as key vectors and images of the bell are served as response vectors. The associative action is set as the instruction for HOAP-3 to point to the bell with its finger. The association pairs are presented to the proposed AM, which is built in the brain of HOAP-3, to build association between key-response pair vectors. For features of images collected by the image sensor of HOAP-3, we do grayscale transformation and adopt 36-dimension low-frequency DCT coefficients as the feature vector. For

features of the sounds collected by sound sensor of HOAP-3, we extract the 15-dimensional spectrum feature on the 20 kHz – 50 ms sampling rate.

After the AM is trained, we show HOAP-3 the sound of the bell, and the sound of the bell serves as the key vector. HOAP-3 recalls the image of the bell, turns its head to the bell and watches the bell, then it points to the bell with its finger, as depicted in Fig. 4.

This experiment demonstrates that the proposed AM is able to realize real tasks with good performance. It is also noteworthy that, in this experiment, the dimensions of image and sound are different and the proposed AM builds an association between vectors with different dimensions quite well.

Acknowledgement

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

References

1. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Nat. Acad. Sci. USA 79, 2554–2588 (1982)
2. Kosko, B.: Bidirectional associative memories. IEEE Trans. Systems, Man and Cybernetics 18(1), 49–60 (1988)
3. Sudo, A., Sato, A., Hasegawa, O.: Associative memory for online learning in noisy environments using self-organizing incremental neural network. IEEE Trans. on Neural Networks 20(6), 964–972 (2009)
4. Hagiwara, M.: Multidirectional associative memory. In: Proc. of the 1990 International Joint Conference on Neural Networks, pp. 3–6 (1990)
5. Shen, F., Hasegawa, O.: An incremental network for on-line unsupervised classification and topology learning. Neural Networks 19, 90–106 (2006)
6. Shen, F., Hasegawa, O.: An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Networks 20, 893–903 (2007)
7. Martinetz, T.M., Berkovich, S.G., Schulten, K.J.: "neural-gas" network for vector quantization and its application to time-series prediction. IEEE Trans. On Neural Networks 4(4), 558–569 (1996)
8. Oh, H., Kothari, S.C.: Adaptation of the relaxation method for learning in bidirectional associative memory. IEEE Trans. On Neural Networks 5(4), 576–583 (1994)
9. Kohonon, T.: Self-organization and associative memory. Springer, Berlin (1984)
10. Yamada, T., Hattori, M., Morisawa, M., Ito, H.: Sequential learning for associative memory using kohonen feature map. In: Proc. of the 1999 International Joint Conference on Neural Networks, pp. 1920–1923 (1999)