CLAM: A New Model of Associative Memory

Antonio B. Bailón,* Miguel Delgado, Waldo Fajardo
Departamento Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, E.T.S.I. Informática, 18071 Granada, Spain

We present a new associative memory model that stores arbitrary bipolar patterns without the problems we can find in other models like BAM or LAM. After identifying those problems we show the new memory topology and we explain its learning and recall stages. Mathematical demonstrations are provided to prove that the new memory model guarantees the perfect retrieval of every stored pattern and also to prove that whatever the input of the memory is, it operates as a nearest neighbor classifier. © 2000 John Wiley & Sons, Inc.

1. INTRODUCTION

There are three types of memories attending to the way of returning previously stored data.

Kind of memory	Maps		To
Random access memory (RAM)	Address	⇒	Data
Content addressable memory (CAM)	Data	⇒	Address
Associative memory (AM)	Data	⇒	Data

Random access memory is made up by cells and in each one of them is stored an information unit. That information can be retrieved supplying to the memory the address of the cell that contains the data we want to recover.

Content addressable memory, like random access memory, is made up by cells that store the information but, in the retrieval process, what we supply to the memory is the data that we are searching and the memory answers returning the address of the cell that contains the data. Finally we feed an associative memory with the data we want to recover (possibly incomplete and/or incorrect) and it answers returning that information completed and corrected.

*Author to whom all correspondence should be addressed; e-mail: bailon@decsai.ugr.es.

Associative memories have several characteristics that make them useful in our needs of data storing:

- The concepts of cell and address disappear in associative memories. Those concepts are related to internal characteristics of the memory and are not related to the information we want to store. Then we will only use the information we need.
- The memory can complete and correct information when the input information of the memory is incomplete and wrong.
- This kind of memory is more robust than the others because of its behavior with noisy data.

The utility of associative memories is clear and there are several models that implement them like bi-directional associative memory (BAM)[1] or linear associative memory (LAM).[2]

Since associative memories are designed to store information, we demand enough capacity to hold the data we need to store or, at least, a good part of it. The existing associative memory models have two important problems that limit their capacity: spurious states and data dependency.

Spurious states appear in associative memories when in the data recovery process we reach a state that is not corresponded with any of the stored patterns, that is, we do not recover the wanted pattern but rather a nonexistent one. This is because each stored pattern is kept using all the connections of the network in such a way that when we store a new pattern we "mix it" with the rest of the stored patterns. This storage mode causes that many patterns interfere with the rest causing that the recovered pattern will be none of them but their mixture.

The problem of data dependency appears in some associative memory models that enhance the capacity imposing some conditions on the data that will be stored, so that it will not store the patterns we need to keep but rather those that the associative memory is capable of storing.

Classifier associative memory emerges from the study of the BAM. Various researchers have attempted to attenuate the capacity problems of such associative memory, modifying the learning process and obtaining some results that, though good, they can be enhanced.[3–8] This can make us think that capacity problems are intimately bound to the basic architecture of the BAM.

With the objective of finding the weak points of its architecture, a detailed study has driven us to accomplish various modifications on the BAM that guide to the creation of a new associative memory model named classifier associative memory (CLAM). This model is characterized by not imposing restrictions to the data to store and to obtain a very high storage capacity.

In our work we follow these steps.

- First we present the new associative memory model CLAM, indicating its objective and showing its architecture and the features of the elements that make it up.
- Next we will explain in detail the memory operation in the data learning and retrieval stages.

- After the presentation of the classifier associative memory we show its virtues demonstrating the perfect retrieval of all the stored patterns and the nearest neighbor classification.
- We evaluate the efficiency of the memory and compare it with bi-directional associative memory.
- Finally we expose the conclusions that we have obtained after the analysis of the features and operation of classifier associative memory.

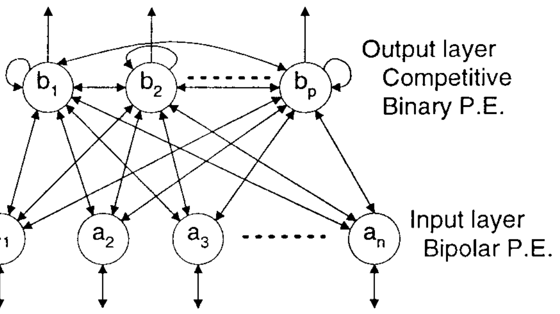
2. CLASSIFIER ASSOCIATIVE MEMORY

Classifier associative memory (CLAM) is an associative memory constituted by two layers of processing elements that classifies bipolar patterns by the nearest neighbor criterion according to the Hamming distance.

The topology of the associative memory is dynamic since the number of output processing elements and the number of connections varies in time according to the number of stored patterns.

The input/output (I/O) layer is constituted by bipolar processing elements in an equal number to that of components of the patterns that we want to memorize. This layer introduces the input pattern and, after the retrieval process, it shows the stored pattern that is the nearest neighbor of the input one.

The output layer is competitive (winner-takes-all competition), with binary processing elements that represent the classes by which the input patterns will be classified. For each stored pattern a PE will be created in the output layer representing that pattern and so the number of output processing elements p depends on the number of stored patterns. A classifier associative memory that has not stored any pattern will not have processing elements in the output layer.



CLAM with p stored patterns of n components.

This associative memory can store bipolar patterns $A_k = (a_{k_1}, \dots, a_{k_n})$, so that, given an input pattern, the system answers, activating the output processing element b_s that identifies the stored pattern A_s that is the nearest neighbor of the input pattern and is shown in the I/O layer.

All those patterns that produce the same output when they are presented in the input layer can be considered as belonging to a common class. So, storing k patterns we define a partition of the set of possible patterns in k classes.

This associative memory operates as a classifier indicating the class that an input pattern belongs to.

The weights of the existing connections among the processing elements of a layer with those of the other can be represented by a weights matrix $M_{p\times n}$, being n the number of components and p the number of stored patterns in that moment.

The activation state of the output layer processing elements can be represented by a vector in which each component represents the particular activation of each PE. As it is assigned an output PE to every stored pattern and every output PE belongs to a stored pattern, an output vector will have only one active component while the rest will stay off.

Furthermore, a component cannot have value 0 always since it will have value 1 at least when we present in the input layer the pattern identified by that component. To assure that this will always be true it is necessary to avoid storing the same pattern more than once in the learning stage.

As we can see, it is easily verifiable that the output vectors B_k are orthonormals.

Since B_i, $i = 1, \dots, p$, is the output patterns of a CLAM that has stored p patterns, we find

orthogonality

$$B_k B_j^T = 0 \quad \forall j \neq k \tag{1}$$

normality

$$\|B_k\| = B_k B_k^T = 1 \tag{2}$$

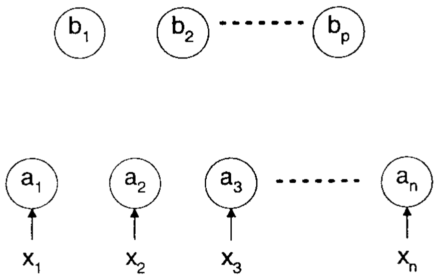
Although the classifier associative memory stores bipolar patterns, internally it stores pairs of patterns. One pattern is the bipolar one that we want to store; the other is a binary pattern that is assigned by the memory. This binary pattern is constituted by the activation signals of the output layer associated with the bipolar pattern. The binary patterns B_i, $i = 1, \dots, p$, where p is the number of stored patterns, are very important in the classifier associative memory.

To improve the capacity of the bi-directional associative memory, some researchers impose conditions on the patterns that are stored but this has the problem of data dependency. In classifier associative memory we improve the capacity imposing hard restrictions on the binary pattern while we give total freedom to the bipolar pattern that we want to memorize.

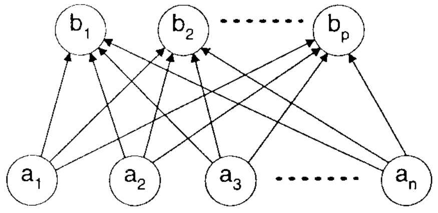
3. RECALL

We suppose that we have a classifier associative memory that has stored p patterns of n components. The classification of an input pattern follows the next process.

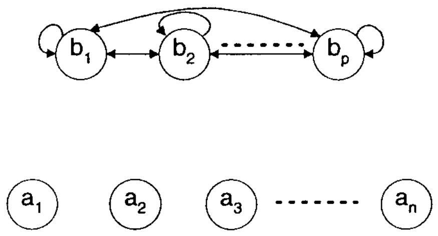
(1) The pattern $X = (x_1, \ldots, x_n)$ is presented in the input layer.



(2) The input signal is propagated toward the output layer through the weights matrix M. The output layer receives the signal $Y = XM^T$.



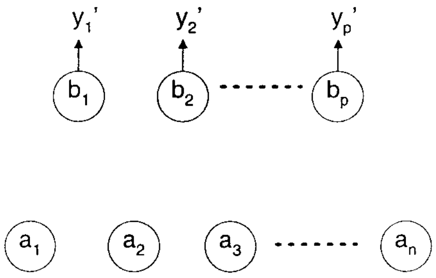
(3) The output layer processing elements compete mutually until only the one that received the greatest signal remains active.



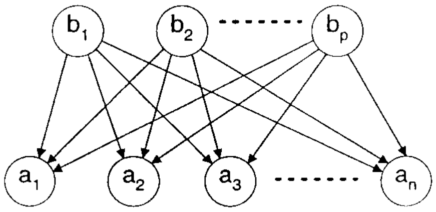
$$y_i' = \begin{cases} 1 & \text{if } y_i = \max\{y_k \ \forall\, k = 1, \ldots, p\} \\ 0 & \text{in the other case} \end{cases} \tag{3}$$

If more than one output processing element remains active it means that their hamming distance from the input pattern is the same in all cases. As there is no way of determining which pattern is the best, one of them is selected using some criterion and the other are made zero.

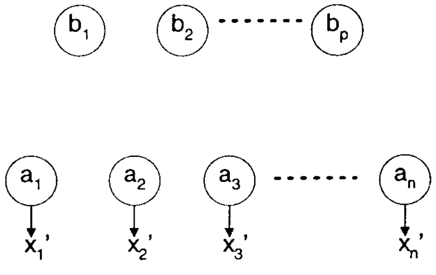
(4) In the output layer we obtain the classification of the input pattern. The only one processing element that remains active in the output layer represents the class by which the input pattern has been classified.



(5) The output activation signal is propagated backward to the input layer with $X' = Y'M$.



(6) In the input layer we obtain the stored pattern that is the closest one to the input pattern.



4. LEARNING

The classifier associative memory stores patterns by means of a process that has the following steps.

(1) The classifier associative memory modifies its topology dynamically in such a way that it has as many output processing elements as stored patterns. In its initial state it will have n input processing elements and no output processing elements, due to the fact that in this state it has not stored any pattern yet. The initial weights matrix M is constituted by n columns and 0 rows.

(2) Each time a pattern is memorized a new output PE is created and it is associated with it. To avoid having two PE associated to the same pattern, for each pattern to memorize we will verify if it has been previously stored.
We will begin presenting it in the input layer to see the recalled pattern. If, after the information retrieval process, the input pattern and the recalled pattern are the same, this means that the pattern is already stored and it is not necessary to store it again. If the patterns are not the same it means that the pattern is not stored and we continue by step (3).

(3) To store the new pattern $X = (x_1, \ldots, x_n)$, it has added a new processing element in the output layer representing the pattern that is being memorized. The weight of the connection made from each input PE to the new b_p is given by $m_{p_i} = x_i$.
Thus, the weights matrix M will be increased with a new row that consists of the weights of the connections from the input layer to the new processing element. The output vector associated with the new pattern is the only one that has the p component with value 1 (it represents the processing element b_p and the others remain with value 0. Likewise, the vector associated with the other stored patterns is increased with a new component (the pth) with value 0 indicating that the new class will not classify them.

(4) For each new pattern to be memorized we repeat steps (2) and (3).

In step (3) of the process we can see that, after storing in the memory the p patterns $A_i = (a_{i_1}, \ldots, a_{i_n})$, $i = 1, \ldots, p$, the matrix M that we obtain is

$$M = \sum_{i=1}^{p} B_i^T A_i = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \cdots \\ a_{p1} & \cdots & a_{pn} \end{pmatrix} \tag{4}$$

$$B_i = (b_1, \ldots, b_p) \qquad b_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{5}$$

This means that the learning process followed to obtain the weights of the connections made between the input and the output layer is Hebbian,[9] that is, the weight of the connection depends on the values of the processing elements it connects.

5. AN EXAMPLE

We will now look at the learning and recall stages with an example. We will store in CLAM three patterns of five components and we will try to recall one of them with and without noise.

The patterns that we will store are

$$A_1 = (+1, -1, -1, +1, +1)$$
$$A_2 = (-1, -1, +1, +1, -1)$$
$$A_3 = (+1, +1, -1, -1, +1)$$

- Before having stored any pattern there are no processing elements in the output layer and the weights matrix has five columns and zero rows.

$$M = (\ )$$

- Learning of A_1.

We present the pattern A_1 in the input layer to see if it was previously stored. The recall process gives us the pattern $(0, 0, 0, 0, 0) \neq A_1$.

A new processing element is created in the output layer and the corresponding connections are made. The new weights matrix is

$$M = (+1 \quad -1 \quad -1 \quad +1 \quad +1)$$

- Learning of A_2

We present A_2 in the input layer. The recall process gives

$$(+1, -1, -1, +1, +1) \neq A_2$$

A new PE is created associated with the new stored pattern. The new weights matrix is

$$M = \begin{pmatrix} +1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & +1 & -1 \end{pmatrix}$$

- Learning of A_3.

We present A_3 in the input layer and the recall process gives

$$(+1, -1, -1, +1, +1) \neq A_3$$

A new PE is created and the new weights matrix is

$$M = \begin{pmatrix} +1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & +1 & -1 \\ +1 & +1 & -1 & -1 & +1 \end{pmatrix}$$

Now we will see the recall process with the stored pattern $A_3 = (+1, +1, -1, -1, +1)$ and a noisy pattern $\tilde{A}_2 = (+1, -1, +1, +1, -1)$.

- Recall of $A_3 = (+1, +1, -1, -1, +1)$.

The input signal is propagated to the output layer:

$$Y = A_3 M^T = (+1, -5, +5)$$

The output processing elements compete:

$$Y' = (0, 0, 1)$$

In the output layer the only active processing element shows the class that represents the input pattern.

The signal is propagated backwards to the input layer:

$$X' = Y'M = (+1, +1, -1, -1, +1) = A_3$$

The stored pattern A_3 is perfectly recalled.

- Recall of $\tilde{A}_2 = (+1, -1, +1, +1, -1)$.

The input signal is propagated to the output layer

$$Y = \tilde{A}_2 M^T = (+1, +3, -3)$$

The output processing elements compete:

$$Y' = (0, 1, 0)$$

In the output layer the only active processing element shows the class that represents the input pattern.

The signal is propagated to the input layer

$$X' = Y'M = (-1, -1, +1, +1, -1) = A_2$$

The recalled pattern is the nearest neighbor of the input pattern.

6. RECALL OF EVERY STORED PATTERN

We will demonstrate below that the memory can recall every pattern stored in the learning stage, that is to say, if a stored pattern is given to the memory in the input layer, the memory will answer classificating and recalling it perfectly.

We suppose that the learning stage of the classifier associative memory has been done with the following set of p patterns:

$$A_i = (a_{i_1}, \ldots, a_{i_n}) \qquad i \in [1, p] \tag{6}$$

Each pattern has been classified by the class represented by the vector

$$B_i = \left(b_{i_1}, \ldots, b_{i_p}\right) \qquad b_{i_j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{7}$$

In this way, the resulting matrix M that represents the weights of the connections made between the two layers is

$$M = \sum_{i=1}^{p} B_i^T A_i \tag{8}$$

If we feed the memory with the pattern A_e, $e \in [1, p]$, we hope that it will be classified by the class e and that we will obtain the pattern A_e in the input layer. First the input activation signal is fed to the output layer through the weights matrix M:

$$Y = A_e M^T = (y_1, \ldots, y_p) \tag{9}$$

As seen in Eq. 8, the propagated signal, Eq. 9 can be expressed

$$Y = A_e \left(\sum_{i=1}^{p} B_i^T A_i \right)^T = \sum_{i=1}^{p} A_e A_i^T B_i \tag{10}$$

Let's see the value of y_k:

$$y_k = \sum_{i=1}^{p} \left(b_{i_k} \sum_{j=1}^{n} a_{e_j} a_{i_j} \right) \tag{11}$$

As seen in Eq. 7 in each B_i, there can only be one b_{i_j} distinct from zero and so (11) can be simplified in the following way:

$$y_k = \underbrace{b_{k_k}}_{1} \sum_{j=1}^{n} a_{e_j} a_{k_j} + \underbrace{\sum_{\substack{i=1 \\ i \neq j}} \left(\underbrace{b_{i_k}}_{0} \sum_{j=1}^{n} a_{e_j} a_{i_j} \right)}_{0} = \sum_{j=1}^{n} a_{e_j} a_{k_j} \tag{12}$$

Because the a_{i_j} have been coded in the bipolar mode, the product of two of them will be 1 if their values are the same (both are 1 or -1) or -1 if the values are different. Suppose that there are d_k pairs $a_{e_j} a_{k_j}$, therefore $n - d_k$ pairs are the same. Then, expression 12 can be rewritten as

$$y_k = 1(n - d_k) + (-1) d_k = n - 2d_k \tag{13}$$

The Hamming distance from the vector V to W can be defined as the number of different components between those vectors. The Hamming distance function is given by the expression

$$H(A_e, A_k) = \sum_{j=1}^{n} g(a_{e_j}, a_{k_j}) \tag{14}$$

where the function g is defined as

$$g(a,b) = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases} \tag{15}$$

If there are d pairs $a_{e_j} a_{k_j}$, the expression 13 can be rewritten as

$$y_k = n - 2H(A_e, A_k) \tag{16}$$

Because of the learning process [step (2) of the process] a pattern cannot be stored twice. This means that

$$H(A_v, A_w) = \begin{cases} 0 & \text{if } v = w \\ > 0 & \text{if } v \neq w \end{cases} \tag{17}$$

From Eqs. 16 and 17 we conclude

$$\left. \begin{aligned} y_e &= n \\ y_{k \neq e} &< n \end{aligned} \right\} \Rightarrow y_{k \neq e} < y_e \tag{18}$$

After this propagation of the input signal from the input layer to the output layer, the processing elements of the output layer begin the competition. As shown in Eqs. 3 and 18,

$$\begin{aligned} y_e &= 1 \\ y_{k \neq e} &= 0 \end{aligned} \qquad \text{u.e.d.} \tag{19}$$

The pattern has been classified correctly. Let's see the back-propagation process of the vector $Y(= B_e)$ to obtain the new vector X which we will name X':

$$X' = B_e M = B_e \sum_{i=1}^{p} B_i^T A_i \tag{20}$$

From the summation of Eq. 20 we are going to take out the addend for $i = e$. As the B vectors are orthonormal, Eqs. 1 and 2, we have

$$X' = \underbrace{B_e B_e^T}_{1} A_e + \sum_{\substack{i=1 \\ i \neq e}} \underbrace{B_e B_i^T}_{0} A_i = A_e \qquad \text{u.e.d.} \tag{21}$$

It has been demonstrated that every stored pattern is recalled perfectly.

7. NEAREST NEIGHBOR CLASSIFICATION

Let's check that, whatever the system input is, the system will answer classifying the pattern by that output class that represents the nearest stored pattern by means of the Hamming distance.

Let's suppose that the memory has stored the patterns $A_i = (a_{i_1}, \ldots, a_{i_n})$, associating to each one the output class i represented by the vector $B_i = (b_{i_1}, \ldots, b_{i_p})$, where $b_{i_i} = 1$ and $b_{i_j} = 0 \ \forall j \neq i$. After the learning process, the weights matrix is

$$M = \sum_{i=1}^{p} B_i^T A_i \tag{22}$$

Let $X = (x_1, \ldots, x_n)$ be the input pattern. The signal propagated toward the output layer is

$$Y = XM^T = X\left(\sum_{i=1}^{p} B_i^T A_i \right)^T = \sum_{i=1}^{p} XA_i^T B_i \tag{23}$$

The kth component of Y is

$$y_k = \sum_{i=1}^{p} XA_i^T b_{i_k} = XA_k^T \tag{24}$$

In the input layer we have bipolar codification. Therefore in the product XA_k^T, the components that are the same add 1 and the rest subtract 1. In this way, as we show in Eqs. 13 and 16, we can rewrite Eq. 24 as

$$y_k = n - 2H(X, A_k) \tag{25}$$

In the competitive stage of the output processing elements, the one with maximum value is selected to represent the input pattern.

We have the functions

$$\max_k \{m_k\} = r \quad m_r \geq m_j \ \forall j \tag{26}$$

$$\min_k \{m_k\} = r \quad m_r \leq m_j \ \forall j \tag{27}$$

The class that represents the input pattern is given by the only processing element that remains active (i.e., with value 1) after the competition. Then the selected class s is

$$s = \max_k \{y_k\} = \max_k \{n - 2H(X, A_k)\} = \min_k \{H(X, A_k)\} \tag{28}$$

In other words, the class by which the input patterns are classified is the one that is associated with the stored pattern whose Hamming distance from the input pattern is the lowest.

8. EFFICIENCY

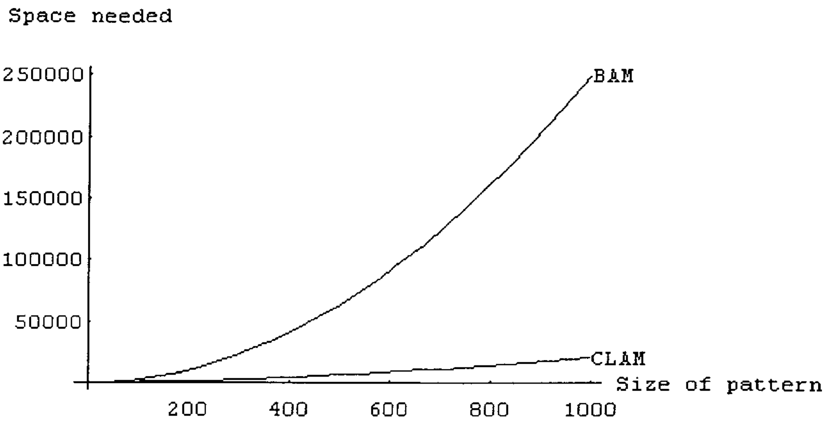
The space efficiency is the relation between the number of stored patterns and the total number of processing elements and connections needed to hold these patterns.

When we store n patterns of p components, the classifier associative memory has $p + n$ processing elements and the number of connections is $pn + [(p^2 + p)/2]$ [pn is the number of connections made between the two layers; $(p^2 + p)/2$ is the number of connections of the competitive layer]. The connections of the competitive layer are not weighted and we do not need to store them. Then the number of connections that we really need to keep is pn.

Working with patterns of p components, to store n patterns, we need $p + n$ processing elements and pn connections, that is, the space efficiency of classifier associative memory is $O(n)$.

We will compare this result with the space efficiency of bi-directional associative memory. There are some capacity estimations of BAM[5,10] and the most optimistic of them[10] shows a capacity of $n = q/4 \log q$ where $q = \min(r, s)$ and r and s are the number of processing elements of each layer. The capacity of BAM reaches its maximum value when $r = s$ and we will split the patterns of p components in pairs of $p/2$ components. The total number of processing elements of BAM is p and the number of connections is $p^2/4$. Then the storage capacity of BAM is $n = [p/8 \log(p/2)]$.

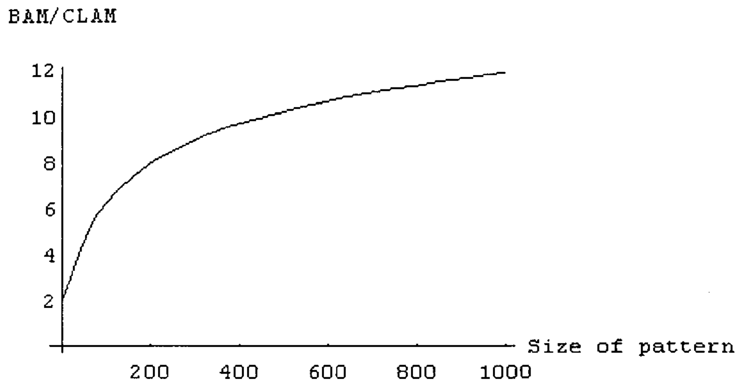
If we want to store $[p/8 \log(p/2)]$ patterns of p components, the space needed by BAM is $p + (p^2/4)$ and the space needed by CLAM is $p + [(p + p^2)/8 \log(p/2)]$. Let's see these results graphically.



We can see that the space needed by BAM is bigger than the space needed by CLAM and this difference increases as the size of the stored patterns grows. The relation between the space needed by BAM and CLAM is given in the expression

$$\frac{p + \dfrac{p^2}{4}}{p + \dfrac{p + p^2}{8 \log\left(\dfrac{p}{2}\right)}} \tag{29}$$

The next graphic shows that relation:



The time needed to store or recall a pattern is little enough to make the classifier associative memory a fast memory.

First of all, in the recall process the memory does not need to cycle until it reaches an energy minimum, that is, there are a limited number of steps in the process and we never jump back to a previous one.

The mathematical operations used to store or recall a pattern are very simple and the time spent is very little.

9. CONCLUSIONS

- The dynamic topology of the classifier associative memory makes it suit the number of patterns stored, so that every pattern can be perfectly memorized.
- The classifier associative memory stores *arbitrary* bipolar patterns.
- Every stored pattern can be perfectly recalled.
- When the input pattern is incomplete and/or wrong it is classified by that stored pattern that is the closest to it, i.e., the pattern that is the most similar to it.
- There are no spurious patterns and so we will never recall a nonstored pattern.
- The relation between the size of the memory and the number of stored patterns is the best of those analyzed and so its implementation is less difficult.
- Only a little time is spent in the learning process of the patterns because we do not need hard mathematical operations or slow learning algorithms.
- The recall stage does not have cycles, that is, it is not necessary to wait until the memory stabilizes in an energy minimum.

References

1. Kosko B. Bidirectional associative memories. IEEE Trans Syst Man and Cybernetics, SMC-18. 1988;42–60.
2. Anderson J. A memory storage model utilizing spatial correlation functions. Kybernetik 1968;5:113–119.
3. Leung C. Encoding method for bidirectional associative memory using projection on convex sets. IEEE Trans Neural Networks 1993;4:879–881.

4. Wang Y, Cruz J, Mulligan JH. Two coding strategies for bidirectional associative memory. IEEE Trans Neural Networks 1990;1:81–91.
5. Wang Y, Cruz J, Mulligan JH. On multiple training for bidirectional associative memory. IEEE Trans Neural Networks 1990;1:275–276.
6. Wang Y, Cruz J, Mulligan JH. Guaranteed recall of all training pairs for bidirectional associative memory. IEEE Trans Neural Networks 1991; 2:559–567.
7. Wang T, Zhuang X, Xing X. Weighted learning of bidirectional associative memories by global minimization. IEEE Trans Neural Networks 1992; 3:1010–1018.
8. Zhang B, Xu B, Kwong C. Performance analysis of the bidirectional associative memory and an improved model from the matched-filtering viewpoint. IEEE Trans Neural Networks 1993; 4:864–872.
9. Hebb DO. The organization of behavior. New York: John Wiley & Sons; 1949.
10. Haines K, Hecht-Nielsen R. A BAM with increased information storage capacity. Proceedings of the 1987 IEEE International Conference on Neural Networks. 1988 Vol. 1, pp. 181–190.