Chapter 6

Synthesis of Associative Memory

Abstract - We present a new approach to enlarging the basin of attraction of associative memory, including auto-associative memory and temporal associative memory. The memory trained by means of this method can tolerate and recover from seriously noisy patterns. Simulations show that this approach will greatly reduce the number of limit cycles.

1 Introduction

Associative memory (AM) is a mechanism used to store patterns: when a reasonable subset of a certain pattern is received with the other part corrupted, it has the ability to recover the original pattern. The fully connected network is a common architecture for the AM. The interconnections between processing neurons provide feedback, which enables the whole network to recurrently evolve into equilibrium. Many well-developed models (Gardner 1987, 1988; Kanter and Sompolinsky 1987) and algorithms have been devised to train the fully connected AM to improve its accuracy, efficiency and capacity. The Hopfield model (HM) (Hopfield, 1982) applies the Hebb's (1949) postulate of learning to generate interconnected weights. It has the advantage of easy training and guarantees convergence when operation is in asynchronous mode, but it also has the vital drawback that it has numerous limit cycles in synchronous mode due to its zero-diagonal, symmetric weight matrix, and zero threshold. The unexpected limit cycle restrains its ability to operate in synchronous mode. There is no reason to exclude synchronous mode evolution among neurons.

According to the relation between input patterns and output patterns, AM can be divided into two classes: the auto-AM is able to recall a pattern which is the same as the input pattern while the hetero-AM is able to present an output pattern which is different from the input pattern. HM is an effective method for implementing the auto-AM. This model is also extensively utilized to implement the hetero-AM, such as bidirectional AM (Kosko 1987, 1988), the multidirectional AM (Hagiwara, 1990), and the temporal AM (Amari, 1972).

One of the features of the fully connected structure is that it can be viewed as a hypercube; therefore, the learning problem of the AM can be transformed into a geometric problem. In this work, we will present a new learning algorithm called error tolerant associative memory (ETAM), which enlarges the basins of attraction, centered at the stored patterns, to the greatest extent possible to improve error tolerance. Simulations show that this algorithm also reduces the number of limit cycles. We will focus on the auto-AM and the temporal-AM in this work. First we will briefly illustrate the model used in this paper, where the geometric interpretations and the ETAM algorithm will be presented. Computer experiments, comparisons, and discussions will be given.

2 Geometric Interpretation of the Hopfield Network

AM is a fully connected network with N neurons. Each neuron i has N weights $\{w_{ij}, j=1,\ldots,N\}$ connecting all neurons j, a threshold θ_i, and a state value $v_i \in \{1,-1\}$. The state value is updated according to the rule

$$v_i(t+1) = \mathrm{sgn}\left[\sum_{j=1}^{N} w_{ij} v_j(t) - \theta_i\right], \qquad (1)$$

or in matrix form,

$$V(t+1) = \mathrm{sgn}[WV(t) - \theta\ ]\ , \qquad (2)$$

where $\boldsymbol{W}$ is an $N \times N$ weight matrix, θ is an $N \times 1$ threshold vector, V(t) is an $N \times 1$ vector representing the global state at time t, and sgn() is the sigmoid function returning 1 with input greater than or equal to zero and -1 with negative input. In the learning phase, the network is trained by P patterns X^k, k=1, …,P, according to various learning algorithms (Widrow and Hoff, 1960; Hopfield, 1982). In the retrieving phase, the input is presented to the network as $V(0)$. Equation (2) is applied to all the neurons in each iteration, so this network is said to operate in synchronous mode; Eq. (1) is applied to only one neuron, and this network operates in asynchronous mode. Then, the network operates repeatedly according to Eq. (1) or Eq. (2) until it converges to a stable state or enters a limit cycle. A stable state meets the following requirement:

$$V(t) = \mathrm{sgn}[WV(t) - \theta\ ]\ , \qquad (3)$$

regardless of whether it operates in synchronous or asynchronous mode. Each neuron has a bipolar state value (a bit), and there are 2^N global states in total. Therefore we can view the whole network as an N-dimensional (N-D) hypercube with each global

state located at a corner. Any two neighboring corners differ in only one neuron state or one Hamming distance. For example, in Fig. 1, there is a 3-D cube corresponding to a network with three neurons. The current global state is located at one corner and serves as the next input to the network. After updating according to Eq. (1) or Eq. (2), the current global state either moves to another corner or stays at the original corner. Corners that always remain unchanged are stable states. The patterns we intend to save are located at certain stable corners. The goal of AM is to move the initial global state to a nearby stable corner where a pattern is stored.

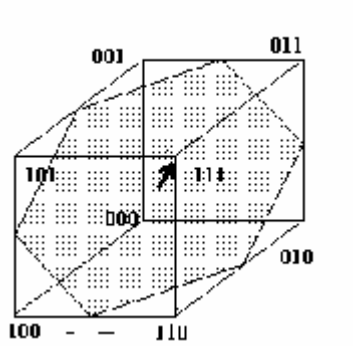


Fig. 1. A fully connected associative memory (AM) with three neurons can be viewed as a 3-D cube. There are eight corners representing the eight global states. A neuron represents a plane (*shaded area*) dividing the cube into positive and negative divisions. In this figure (1,1,1), (1,-1,1), (1,1,-1), and (-1,1,1) are in the positive division while the other corners are in the negative division

In this hypercube, each neuron i describes an (N-1)-D hyperplane through the equation

$$w_{i1}v_1 + w_{i2}v_2 + w_{i3}v_3 + ... + w_{iN}v_N - \theta_i = 0\,,$$
$$i = 1,...N\,. \qquad (4)$$

The $N \times 1$ weight vector $\boldsymbol{W}_i = (w_{i1}, w_{i2}, \ldots, w_{iN})^T$ of the ith neuron is the normal vector of the corresponding hyperplane, and the hyperplane divides the hypercube into positive division to which the normal vector points, and a negative division. We then follow the geometrical point of view developed by Cover (1965). We will require that the length of the vector $\boldsymbol{W}_i$, $|\boldsymbol{W}_i|$, be normalized to one, and that θ_i be divided by this length accordingly. The learning process is used to adjust the hyperplane to make all the patterns stable. That is, when a pattern has an ith bit which is equal to 1, it should be located in the positive division of the ith hyperplane; if it has an ith bit equal to -1, it should be located in the negative division. Furthermore, if we wish to achieve good error tolerance, which means that a reasonably noisy pattern can

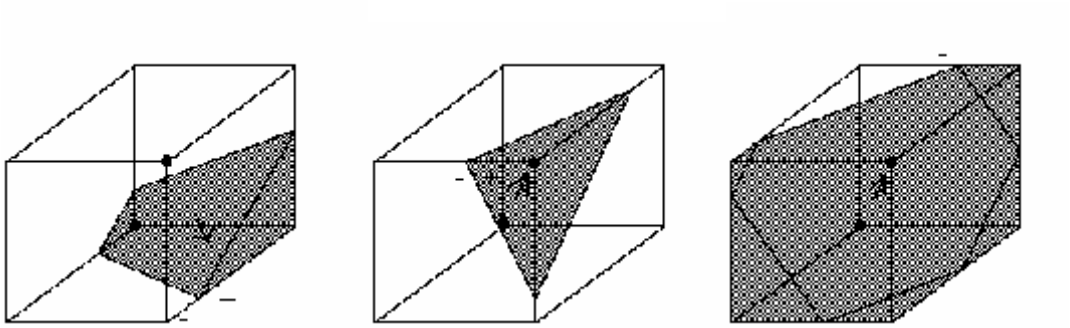
converge to its original pattern, the neighbors of one pattern should be in the same division in which the pattern is located. This can be achieved by rotating and shifting the hyperplane so that it will face the pattern corner and include as many neighboring corners in the same division as possible. These rotations and shifts can be performed by adjusting the weights and the thresholds.

Each hyperplane can be adjusted separately to stabilize the *i*th bits of patterns. In the following, unless otherwise stated, we will only discuss neuron *i*. First, we will illustrate a 3-D cube to observe the mechanism by which the AM saves patterns, then we will expand this mechanism into a higher dimensional hypercube; that is, we will present a general idea for training an AM with more neurons.

An example is shown in Fig. 2. It is an AM with three neurons, in which two patterns, (1,1,1) and (-1,-1,-1), are stored. For the first neuron, we require that the weights satisfy the following conditions:

$$\begin{cases} w_{11}\cdot 1 + w_{12}\cdot 1 + w_{13}\cdot 1 - \theta_1 > 0 \\ w_{11}\cdot(-1) + w_{12}\cdot(-1) + w_{13}\cdot(-1) - \theta_1 < 0 \end{cases} \tag{5}$$

The other two neurons also have to satisfy similar conditions, so we do not show their corresponding planes. The black dots are patterns to be stored. The dark parts are the positive division of the plane.



(a) (b) (c)

Fig. 2a-c. An associative memory (AM) saving (1, 1, 1) and (-1, -1, -1) by means of the error correction rule (ECR). **c** The ideal hyperplane

Figure 2a shows a randomly chosen initial weights and the corresponding hyperplane. We apply the error-correction rule (ECR) (Widrow and Hoff, 1960) to train these weights, and in only one step we get the updated weights shown in Fig. 2b. The weights satisfy the conditions in Eq. (5). However, in order to increase the error tolerance, we wish all the one-bit neighbors of (1,1,1), {(1,1,-1), (1,-1,1), and(-1,1,1)} to be in the positive division, and all one-bit neighbors of (-1,-1,-1), {(-1,-1,1), (-1,1,-1), and (1,-1,-1)}to be in the negative division, just like the ideal hyperplane

shown in Fig. 2c. The normal vector of the plane, (w_{11}, w_{12}, w_{13}), is adjusted to point at (1,1,1) with a right angle to the corner, and so that the plane lies in the middle between (1,1,1) and (-1,-1,-1). In this case, all three planes will coincide.

In Fig. 3, we see only one pattern (1,-1,1) to be stored, and the first neuron hyperplane is shown. In Fig. 3a, there is no error tolerance; in Fig. 3b, the one-bit error can be recovered; in Fig. 3c, even a two-bit error can be recovered. In all three cases, the normal vector of the plane, (w_{11}, w_{12}, w_{13}), points at the pattern corner to be stored with a right angle. The only difference is the threshold. When we modify the threshold to move the pattern farther from the plane, we get better error tolerance. If there is more than one pattern, such as in the example shown in Fig. 4a, b, the normal vector points at the midpoint of the stored patterns; that is, the normal vector is the sum of all the vectors which point from the origin to the patterns. In Fig. 4c, for the second neuron, the two patterns have to lie separately in the two divisions, and the best way to locate the plane is to make the distances from the two patterns to the plane equal. Therefore the two patterns will have the same error tolerance.

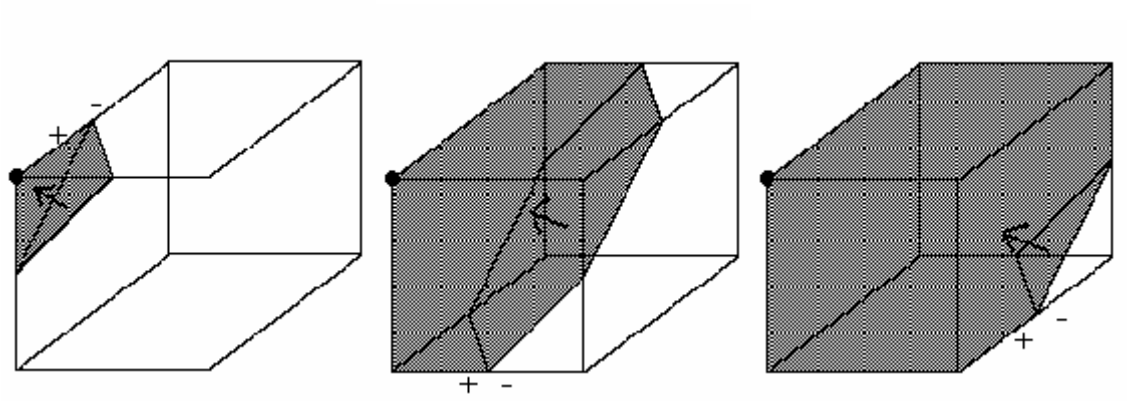
As a result, we obtain the general idea that the normal vector of a certain hyperplane representing neuron i is the sum of all patterns in the positive division and all the patterns in the negative division multiplied by -1. In terms of a neural network, the weights (w_{i1}, w_{i2}, $w_{i3, \ldots,}$ w_{iN}) are the sum of all patterns in which the ith bit is equal to 1 and all the patterns multiplied by -1 in which the ith bit is equal to -1. This results in the equation:

$$w_{ij} = \sum_{k=1}^{P} X_j^k X_i^k , \quad i, j = 1, \ldots, N. \tag{6}$$

We find that Eq. (6) coincides with the Hopfield model except for all the diagonal weights w_{ii}. In the HM, all the self-feedback w_{ii} is made zero to achieve convergence. This self-feedback has been discussed in Kanter and Sompolinsky (1987) Araki and Saito (1995) and DeWilde (1997). In asynchronous mode, zero self-feedback and symmetry ensure that the AM will finally converge to a stable state, but in synchronous mode, there will be some limit cycles with length two (Bruck, 1990; Goles and Martinez 1990). Following Xu et al. (1996) we know that the adequate conditions for the AM to converge in synchronous mode are that the weight matrix $\boldsymbol{W}$ is symmetric and nonnegative definite, or that $\boldsymbol{W}$ with certain values subtracted from its diagonal is nonnegative definite. Equation (6) will make all w_{ii} equal to P, the largest among all weights, and large self-feedback will benefit nonnegative definiteness and convergence.

Besides the diagonal elements, the HM has null thresholds, which means that all hyperplanes pass the origin. Even though the patterns may be stored without a threshold, error tolerance can be greatly improved by tuning thresholds, such as in the

example in Fig. 3. The problem of threshold tuning strategies has been addressed in Schwenker et al. (1996) in the 0,1 model. By tuning thresholds, we can move the planes as far away (distance) from the patterns as possible to increase error tolerance. If there is more than one pattern, we can adjust threshold to maximize the minimal distance among all distances from the patterns to the plane, as shown in Figs. 2c and 4c.



(a) (b) (c)

Fig. 3a-c. An associative memory (AM) saving (1, -1, 1) with different degrees of error tolerance

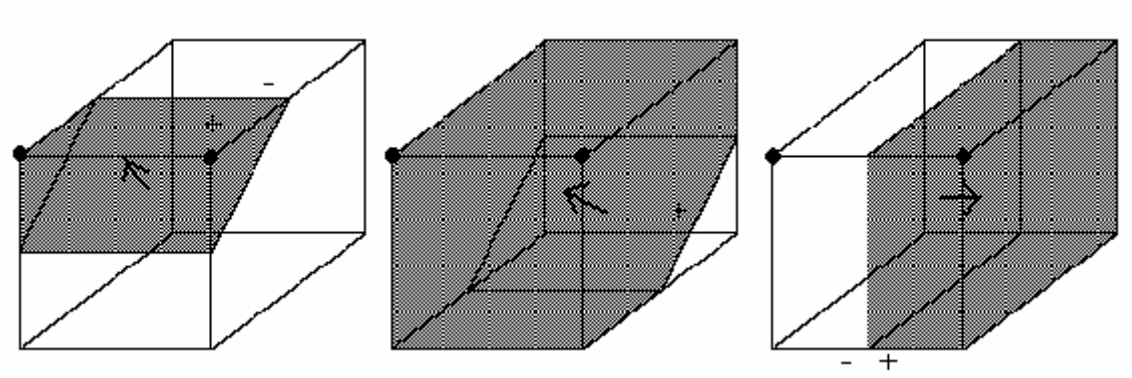


Fig. 4a-c. An associative memory (AM) saving (1, -1, 1) and (1, 1, 1) with different degrees of error tolerance

3 Error Tolerant Associative Memory

With the above general idea in mind, we have developed a method called the ETAM, which trains AM to achieve the best possible error tolerance. According to the ideas presented in the previous section, we can adjust the weights so as to rotate the hyperplane facing the pattern corner with a right angle as much as possible, and adjust

the threshold to maximize the minimal distance from the patterns to the hyperplane as possible. We propose the following retraining algorithm:

1. Initialize the weights according to the following equations:

$$w_{ij}(0) = \sum_{k=1}^{P} X_j^k X_i^k \,, \qquad i, j = 1, \ldots, N, \tag{7}$$

$$\theta_i(0) = 0 \,, \qquad i = 1, \ldots, N.$$

   Then normalize $\boldsymbol{W}_i$, $i = 1, \ldots, N$.
2. Set i to 1.
3. For the ith neuron, calculate the distances from all patterns to the hyperplane, that is,

$$d_i^k = w_{i1} X_1^k + w_{i2} X_2^k + \ldots + w_{iN} X_N^N - \theta_i \,, \qquad k = 1, \ldots P. \tag{8}$$

   Find the positive minimal distance d_p and negative minimal distance d_n :

$$\begin{cases} d_i^p = \min\{d_i^k \mid X_i^k = 1, \quad k = 1, \ldots P\} \\ d_i^n = \max\{d_i^k \mid X_i^k = -1, \quad k = 1, \ldots P\} \end{cases}. \tag{9}$$

4. If all the patterns have $X_i^k = 1$, set θ_i to a large negative value less than $-\sqrt{N}$, increase i by one, and go to step 3.

   If all patterns have $X_i^k = -1$, set θ_i to a large positive value greater than $\sqrt{N}$, increase i by one, and go to step 3.

   Note that we move the hyperplane outside the range of the hypercube, where $\sqrt{N}$ is the distance between each corner and the origin.
5. We shift the hyperplane to the middle of pattern p and pattern n to maximize the minimal distance as follows:

$$\theta_i(t+1) = \theta_i(t) + (d_i^p + d_i^n)/2 \,. \tag{10}$$

   Now the minimal distance d_i^m is in the middle of d_i^p and d_i^n :

$$d_i^m = (d_i^p - d_i^n)/2 \,. \tag{11}$$

6. We rotate the hyperplane to increase the distances from both pattern p and pattern n to the hyperplane:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \, (X_i^p X_j^p + X_i^n X_j^n) \,,$$

$$j = 1, \ldots, N. \quad (12)$$

where α is the learning rate. Normalize $\boldsymbol{W}_i$.

7. Repeat Eq. (8) and Eq. (9) and compute the new d_i^p and d_i^n. If the new $(d_i^p - d_i^n)/2$ is larger than the previous d_i^m, go to step 3 and continue. If not, undo Eq. (12) and go to step 8.
8. Increase *i* by one. If *i* is less than or equal to *N*, go to step 3. If not, stop.

The normalization procedure used in learning step 6 after Eq. (12) is necessary to limit the range of the normal vector $\boldsymbol{W}_i$. We may obtain an asymmetric matrix $\boldsymbol{W}$ using this normalization. This asymmetric matrix is not excluded by any existing physiological evidence (e.g. Hertz et al. 1986, 1991, Porat 1989). The diagonal elements of this matrix are neither equal to any others nor equal to zero. These elements will have large positive values after learning. To our knowledge, this kind of matrix has not been derived using any existing method.

The training time is case-dependent, and the learning process can be accelerated by increase the learning rate. Since the weights are normalized in the ETAM, they cannot increase without any limit and neither can the distance. Therefore, the learning process guarantees termination.

4 Error Tolerant Associative Memory with Temporal Patterns

The temporal AM is used to store one sequence or several sequences of patterns in the AM's dynamic state transitions. Given an initial input pattern, it will converge to the next pattern in a memorized sequence. All the patterns in this sequence will be recalled sequentially. Due to its dynamic property, it can be used to recognize or generate temporal patterns, such as speeches or images, or musical notations.

The temporal AM is trained to remember all patterns in the following dynamics:

$$X_i^{k+1} = \mathrm{sgn}\left(\sum_{j=1}^{N} w_{ij} X_j^k - \theta_i\right) . \quad (13)$$

The superscript of the pattern X^k can be computed using modulo $P + 1$. With an initial input state $V(0)$ is close to a stored pattern X^k, the pattern X^{k+1} will be the first pattern recalled, and the remainder of the patterns will be recalled sequentially.

This temporal AM can store various kinds of pattern sequences, such as a single

chain (dotted line in Fig. 5), a cycle of patterns (dashed line in Fig. 5), or a tree (the upper two patterns in Fig. 5). Generally, the temporal AM is able to save all one-to-one or many-to-one patterns, but not one-to-many patterns like that shown in Fig. 6. We will not discuss the patterns in Fig. 6 in this paper.

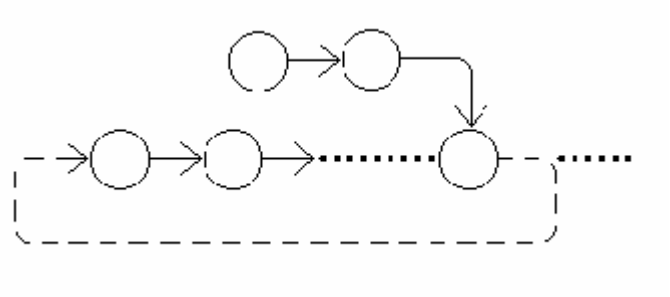


Fig. 5. Various kinds of sequential patterns

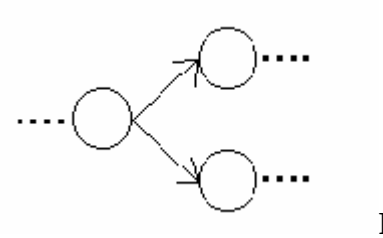


Fig. 6. One-to-many patterns

The original temporal AM proposed by Amari (1972) is implemented according to the Hebb's postulate of learning, which is similar to the HM, as follows:

$$w_{ij} = \sum_{k=1}^{P} X_i^{k+1} X_j^k \quad . \tag{14}$$

This temporal associative memory has asymmetric weight and no threshold. However, it has the same drawback as HM, low capacity. Amari's method usually cannot memorize complete patterns, as we will see in simulations described later.

The idea presented in Sect. 2 is also applicable to the temporal AM or any other hetero-AM. The difference is that in the auto-AM, all the states in the basin of a pattern will converge to this pattern while in the temporal AM, all the states in the basin of a pattern will evolve to its next pattern. To train weights is to shift and rotate the hyperplanes to separate patterns into two divisions according to the ith bits of their next patterns. If $X_i^{k+1} = 1$, X^k should be in the positive division of the hyperplane and in the negative division if $X_i^{k+1} = -1$.

Therefore, the whole algorithm in the previous section can be used to train the

temporal AM with slight modification. The modified algorithm is listed below.

1. Initialize the weights according to the following equations:

$$w_{ij}(0) = \sum_{k=1}^{P} X_i^{k+1} X_j^k , \qquad i, j = 1, \ldots, N, \tag{15}$$

$$\theta_i(0) = 0 , \qquad i = 1, \ldots, N,$$

   Then normalize $\boldsymbol{W}_i$, $i = 1, \ldots, N$.

2. Set i to 1.
3. For neuron i, calculate the distances from all patterns to the hyperplane, that is,

$$d_i^k = w_{i1} X_1^k + w_{i2} X_2^k + \ldots + w_{iN} X_N^k - \theta_i , k = 1, \ldots P. \tag{16}$$

   Find the positive minimal distance d_p and negative minimal distance d_n :

$$\begin{cases} d_i^p = \min\{d_i^k \mid X_i^{k+1} = 1, \quad k = 1,\ldots P\} \\ d_i^n = \max\{d_i^k \mid X_i^{k+1} = -1, \quad k = 1,\ldots P\} \end{cases} . \tag{17}$$

4. If all patterns have $X_i^{k+1} = 1$, set θ_i to a large negative value less than $-\sqrt{N}$, increase i by one, and go to step 3.

   If all patterns have $X_i^{k+1} = -1$, set θ_i to a large positive value greater than $\sqrt{N}$, increase i by one, and go to step 3.

   Note that we move the hyperplane outside the range of the hypercube, where $\sqrt{N}$ is the distance between each corner and the origin.
5. We shift the hyperplane to the midpoint between pattern p and pattern n to maximize the minimal distance as follows:

$$\theta_i(t+1) = \theta_i(t) + (d_i^p + d_i^n)/2 . \tag{18}$$

   Now the minimal distance d_i^m is in the middle of d_i^p and d_i^n :

$$d_i^m = (d_i^p - d_i^n)/2 . \tag{19}$$

6. We rotate the hyperplane to increase the distances from both pattern p and pattern n to the hyperplane :

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \ (X_i^{p+1} X_j^p + X_i^{n+1} X_j^n) ,$$

$$j = 1, \ldots, N. \tag{20}$$

   where α is the learning rate. Normalize $\boldsymbol{W}_i$.

7. Repeat Eq. (16) and Eq. (17) and compute the new d_i^p and d_i^n. If the new $(d_i^p - d_i^n)/2$ is larger than the previous d_i^m, go to step 3 and continue. If not, undo Eq. (20) and go to step 8.
8. Increase *i* by one. If *i* is less than or equal to *N*, go to step 3. If not, stop.

The algorithm above is the same as that used to train auto-AM in the previous section except for Eqs. (15), (17), and (20).

We initialize the weights using Amari's method in Eq. (15) and update weights according to Hebb's postulate in Eq. (20). Then, we calculate all the distances from all the patterns to the hyperplane. Among the patterns which are located at the positive division, the one with minimal distance is chosen in step 3, and so is the pattern in the negative division. Then, we move the hyperplane to the midpoint between these two patterns just chosen according to step 5 and rotate this hyperplane to make it face these two patterns more right in step 6. These steps are repeated until the minimal distances cease to increase. If all the patterns are located in a single division, we simply move the hyperplane outside the hypercube, as described in step 4.

5 Experiments

Next we will give some simulations. We will first give experimental results to compare the Little model (LM) (Little 1974; Little and Shaw 1975), the ECR model, and the ETAM algorithm. Several issues, such as the number of stable states, the number of limit cycles, and fault tolerance, will be discussed. These three approaches were applied to a fully connected network. The difference among these three approaches lies in the learning phase. We will give examples of utilizing the ETAM in a pattern recognition problem. Below, we will first briefly review these algorithms.

5.1 Little Model

The HM is constructed by using the outer product rule to compute the weights as follows:

$$w_{ij} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{k=1}^{P} X_i^k X_j^k, & if\ i \neq j; \\ 0, & if\ i = j. \end{cases} \tag{21}$$

Several characteristics are worth noting. Elements on the diagonal of the weight

matrix, w_{ii}, are zero. This means that all the neurons have no self-feedback, and this has the effect of reducing the number of spurious stable states for the reason that overlarge self-feedback will cause neurons to tend to retain their previous states. The zero-diagonal and symmetric weight matrix forces the HM to always converge to a stable state in asynchronous dynamics. However, the model we are interested in here is called the Little model (LM) and is similar to the HM. It differs from the HM only in that it uses synchronous dynamics. This forces the network to always converge to not only a stable state, but also limit cycles of length two.

5.2 Error-Correction Rule

The ECR is used to adjust the weights in proportion to the error term $[X_i^k - v_i(t)]$. At the beginning, we randomly assign initial values to all the weights, then we adjust all the weights according to following equations:

$$w_{ij}(t+1) = w_{ij}(t) + \eta [X_i^k - v_i(t)] X_j^k \tag{22}$$

$$\theta_i(t+1) = \theta_i(t) - \eta [X_i^k - v_i(t)] \tag{23}$$

$$v_i(t) = \operatorname{sgn}\left[\sum_{j=1}^{N} w_{ij}(t) X_j^k - \theta_i(t) \right], \tag{24}$$

where η is a positive constant which determines the rate of learning. The pattern X^k used to train the network is chosen randomly from among all the patterns. Adjustment will continue until there are no more errors.

5.3 Comparisons for Auto-Associative Memory

Table 1 lists experimental results for (N = 5, P = 5), (N = 5, P = 3), (N = 10, P = 5), (N = 10, P = 3). In each experiment, we present ten sets of randomly produced patterns to the three methods, obtained the information about the number of stable states, limit cycles, etc., then calculated the averaged results. For each row item, the following explanation is offered:

SP [No. of stored patterns (/P)]:

given P patterns, the number of patterns successfully stored.

SS [No. of stable states (/2^N)]:

the number of stable states.

TS [No. of states to stable (/2^N)]:

the number of states converging to all stable states.(≥SS)

C [No. of cycles]:

the number of limit cycles.

IC [No. of states in cycles (/2^N)]:

the number of states involved in all limit cycles.

TC [No. of states to cycles (/2^N)]:

the number of transient states falling into limit cycles.

R [recovery (/NP)]:

given *NP* 1-bit error patterns, the number of patterns converging to the original stored patterns.

Table 1. Comparisons among the Little model (LM), error correction rule (ECR), and error tolerant associative memory (ETAM)

	LM	ECR	ETAM		LM	ECR	ETAM
N=5, P=5				N=10, P=5			
SP(/5)	1.8	5	5	SP(/5)	1.9	5	5
SS(/32)	4.2	15.2	17.6	SS(/1024)	5.0	43.9	60.0
TS(/32)	14.6	16.8	14.4	TS(/1024)	744.8	978.4	964.0
C	2.7	0	0	C	44.3	0.2	0
IC(/32)	5.4	0	0	IC(/1024)	88.6	0.4	0
TC(/32)	7.8	0	0	TC(/1024)	185.6	1.3	0
R(/25)	3.2	3.9	5.5	R(/50)	12.1	13.5	38.9
	LM	ECR	ETAM		LM	ECR	ETAM
N=5, P=3				N=10, P=3			
SP(/3)	2.4	3	3	SP(/3)	2.6	3	3
SS(/32)	5.2	7.1	4.8	SS(/1024)	6.8	20.2	6.2
TS(/32)	13.0	24.9	27.2	TS(/1024)	553.4	1003.2	1017.8
C	5.5	0	0	C	84.9	0.2	0
IC(/32)	11	0	0	IC(/1024)	169.8	0.4	0
TC(/32)	2.8	0	0	TC(/1024)	294.0	0.2	0
R(/15)	4.0	4.0	10.9	R(/30)	21.3	8.6	29.0

The LM has a rather low capacity, so it cannot store all the patterns in all the situations. The maximum number of patterns stored in the LM is similar to that of the HM, which is $N/4 \ln N$ (Weisbuch and Fogelman-Soulié 1985; McEliece et al. 1987).

This number has been improved by Mazza (1997). Also, the LM produces many limit cycles.

For these two reasons, even though it produces the fewest stable states, this advantage becomes redundant. When the number of patterns is small enough compared to that of neurons, recovery from distorted patterns using the LM has acceptable performance, as Table 1 shows with ten neurons and three patterns. However, when there are more patterns, the LM performs poorly in terms of error tolerance, and so does the ECR. Note that in this experiment, the patterns were randomly generated, and some differed by only one or two bits. In this situation, it is reasonable that not all 1-bit errors can be corrected.

The patterns given could be successfully memorized using the ECR. Regarding other issues, the ECR produced very few limit cycles. However it produced spurious stable states, and the error tolerance of the ECR was not very good either. This is because the criterion based on which training to terminates is accuracy, not error tolerance.

For the ETAM, these patterns are guaranteed to be saved because if errors occur, Eqs. (9) and (10) will correct them immediately. Comparing the ECR and the ETAM, the ECR produces more spurious stable states than the ETAM does when there are few patterns and produces fewer spurious stable states when there are more patterns. This is because the ETAM continues training until the minimal distance cannot be increased further. A good minimal distance is easy to achieve when there are fewer patterns but difficult to obtain when there are more patterns. Therefore, Eq. (12) is repeated, the self-feedback weights w_{ii} continue increasing, and more spurious stable states are generated. Overlarge self-feedback will cause a neuron to tend to stay in its previous state and will produce more stable states. The extreme situation is that in which all the weights and thresholds are zero except w_{ii}, which are all positive. In this case, all the neurons remain unchanged, and all the global states are stable. We expect the converged matrix to have weights of this kind for a large set of difficult patterns.

The ETAM produces no limit cycles. Again, this is because of the larger self-feedback. When self-feedback is large, all neurons tend to stay in their previous states; hence, the number of limit cycles can be effectively reduced. With the ETAM, the performance in terms of recovery from noisy patterns is much better than that of the other two methods for the reason we enlarge the neighborhood of the stored pattern until no more improvement is possible. The larger the distance is, the more error the AM can tolerate.

Although the number of spurious stable states is increased by the ETAM because of the nonzero diagonal elements in the weight matrix, this has the effect of reducing the number of states in limit cycles and improving proformance. This is a trade-off.

Since we do not want a slightly noisy pattern to converge to an incorrect stable state, better error tolerance has priority. Also, detecting whether a pattern falls into a limit cycle is much harder than detecting whether a pattern converges to an incorrect stable state. Therefore, we would rather avoid limit cycles. Note the computation cost of ETAM is about five to ten times that of ECR in all of our simulations.

5.4 Example I for Auto-Associative Memory

This example is taken from the work of Tank and Hopfield (1987). The original example is like a memo with six attributes for each person. The goal is to store three patterns (1, 1, 1, -1, -1, -1), (1, -1, 1, 1, -1, 1), and (1,1, -1, 1, -1, -1) in an AM with six neurons representing six attributes. Using the LM, we obtain six stable states, which is a pretty good result. However, 15 2-cycles and 24 more transient states which converge to these limit cycles in synchronous mode exist. This means that the AM will seldom converge to the expected stable states and usually will not stop. Using the ECR no cycle exists, but eight more spurious stable states appear. Also, its error tolerance is poor because almost none of the one-bit neighbors of these three patterns can converge to the expected patterns. Using the ETAM, the performance is better. The number of stable states slightly increases to seven, but there is no limit cycles appear. About two-thirds of the one-bit neighbors can converge to the patterns to which they belong.

5.5 Example II for Auto-Associative Memory

Here, we will present another example of pattern recognition. There are ten template patterns from characters A to J as shown in Fig. 7. Each pattern is an 8×2 character, and 96 neurons are required. In Table 2, we list the Hamming distances between any two patterns. Ideally, the minimal Hamming distance for a pattern divided by two should be the ideal error tolerance for that pattern. However, the result is not as perfect as imagined due to the linear separability. Consider Fig. 8. The ideal radius is illustrated by the circle around each dot. To linearly separate these circles is hopeless. The actually achievable radius is the distance from the pattern to the straight line.

Fig. 7. The ten patterns

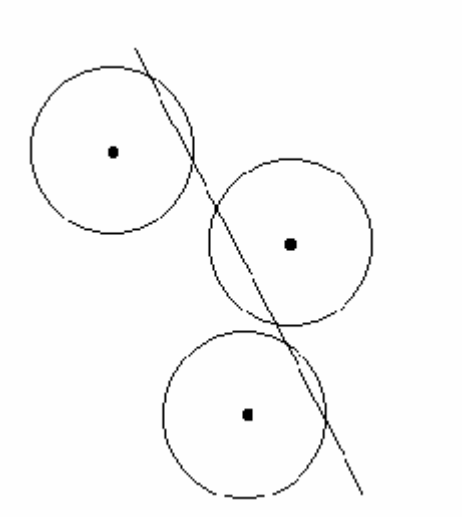


Fig. 8. The ideal radius is illustrated by the circle around each point. The actually achievable radius is the distance from the point to the straight line

To determine the actually achievable radius of each pattern, we calculate d_i^k for all the neurons according to Eq. (8) for each template pattern after the learning converges. If a bit *j* of pattern *k* is reverted, the d_i^k will increase or decrease by $2w_{ij}$, depending on whether the signs of X_j^k and w_{ij} are identical or not, respectively. If $|d_i^k|$ is larger than the sum of the r_i^k largest weights multiplied by two, then r_i^k is the actual radius of neuron *i* for pattern *k*. That is,

$$2\sum_{j=1}^{r_i^k+1} w'_{ij} X_j^k X_i^k > \left|d_i^k\right| > 2\sum_{j=1}^{r_i^k} w'_{ij} X_j^k X_i^k \qquad (25)$$

where w'_{ij} are weights sorted in descending order according to their corresponding $w_{ij} X_j^k X_i^k$. Note that there is at least one term (the largest one being $w_{ii} X_i^k X_i^k$), which has a positive value. Equation (25) will be satisfied, at least when $r_i^k = 0$. The radius of pattern *k* is the minimal one among the radii of all the neurons. Table 3 presents the result. First, we find that the HM memorizes none of these patterns, and neither does the LM. There are always several incorrect bits in the recalled pattern.

Therefore, we did not include the HM or the LM in the following simulation. By applying the ECR, all patterns are successfully saved. However, the radii are all zero because training stops as soon as no further errors occur, and error tolerance is not considered. With the ETAM, all the patterns are stored and have two or three radii. Apparently, the actual radius is much smaller than the ideal radius, but the worse situation rarely occurs; we can see this in Table 4. For each pattern, we randomly generated 1000 noisy patterns with 10, 20, 30, and 40 error bits, and fed these noisy patterns into the trained AM. The numbers of successfully recovered patterns are listed. Almost all noisy patterns with 10 errors are recovered, and more than 80% of the patterns with 20 errors were recovered. The number of recovered patterns is roughly proportional to the ideal radius of each pattern. One thing worth noticing is that there is no pattern fell into limit cycles in any of 40,000 experiments. The ETAM performs excellently in reducing the number of limit cycles and improving error tolerance.

There is no efficient way to measure the exact sizes of basins of attraction in such a large network, which has 2^{96} totally states. Actually, we have tested the basin sizes on a modern personal computer using brute force. It took us three weeks and we still failed to reach the boundaries of the basins of the three patterns, A, B, and C. According to the records, their basins must be huge. Instead, we use an eclectic way, testing the error tolerance, to approximately reveal the huge basins of the stored patterns. Better error tolerance must result from large basins of attractions.

Also, as shown in Table 4, we tried to retrieve patterns in asynchronous mode. Although the ETAM is designed for the synchronous mode, it is suitable in asynchronous mode, neurons are processed sequentially. An earlier recovered error may aid the correction of later errors. Restoring a noisy pattern will result in much benefit from local corrections. This kind of correction is different from that in synchronous mode since all neurons are processed in a global sense. However, we still cannot predict which the synchronous mode or the asynchronous mode will be better in every situation.

Table 2. Hamming distances between all the patterns

	A	B	C	D	E	F	G	H	I	J

A		44	36	36	56	57	24	46	68	56
B	44		34	10*	20	27	32	24*	58	50
C	36	34		28	42	45	16*	54	48	46
D	36	10*	28		26	33	30	34	60	56
E	56	20	42	26		13*	46	24*	50	48
F	57	27	45	33	13*		53	27	49	43
G	24*	32	16*	30	46	53		48	56	50
H	46	24	54	34	24	27	48		66	52
I	68	58	48	60	50	49	56	66		28*
J	56	50	46	56	48	43	50	52	28*	

Table 3. Actual radius of each pattern for the error tolerant associative memory (ETAM) and error correction rule (ECR)

	A	B	C	D	E	F	G	H	I	J
ETAM	3	2	2	22	2	2	2	3	3	3
ECR	0	0	0	0	0	0	0	0	0	0

Table 4. Error tolerance for the error tolerant associative memory (ETAM) and error correction rule (ECR)

	A	B	C	D	E	F	G	H	I	J
ETAM in synchronous mode										
10	1000	980	1000	989	993	989	998	1000	1000	1000
20	996	840	975	868	908	874	966	995	979	999
30	935	381	697	456	602	612	610	866	746	951
40	450	37	150	47	117	163	57	255	97	461
ETAM in asynchronous mode										
10	1000	998	1000	1000	999	999	1000	1000	1000	1000
20	997	950	983	969	957	965	996	988	997	1000
30	938	673	833	811	671	707	820	701	854	945
40	474	190	400	289	115	104	312	78	126	432
ECR in synchronous mode										
10	350	289	50	73	123	25	219	31	92	32
20	189	163	11	19	44	11	75	4	32	8
30	54	40	3	1	4	5	12	3	5	0
40	4	1	0	8	0	0	0	0	0	0

5.6 Example I for Temporal Associative Memory

In this section, we present simulations for the temporal AM. There are ten patterns, from characters A to J. We require the temporal-AM to store these patterns in different orders. In Fig. 9a, they are saved as a chain, in Fig. 9b, they are saved as a cycle, and in Fig. 9c, they are saved as a tree. In general, the patterns can be saved similar to a water system. The basins of the patterns from the river valley in the system. All the patterns are designed to transferred to a certain next pattern, and finally converge to a final stable state or a limit cycle.

In these three simulations, we use both Amari's method as Eq. (14) and the algorithm presented in the previous section to train the memory. We found that using Amari's method in our simulation, cannot store all the patterns. Only character J can be correctly recalled in the chain of patterns, only A and B in the cycle of patterns, and only B in the tree of patterns.

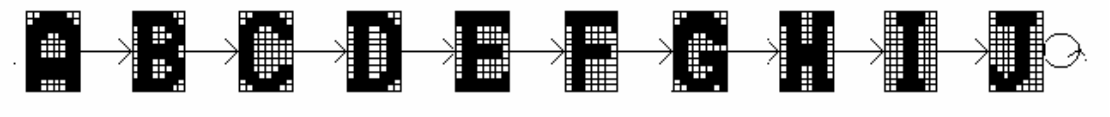
The tree AM is extremely useful in speech recognition. Space is needed to elaborate such an application. We will briefly introduce the techniques developed in our laboratory for recognizing Chinese speeches. We build a tree for each Chinese character pronunciation. There are 1309 character pronunciations in total. Fifty sequences of binary coded speech templates in melscale are collected for each character and used to train its tree AM. Each tree has a different form, which is obtained by means of training with these 50 sequences. This tree replaces the linear neural array (Liou et al. 1996). A sequence is composed of time-warped temporal patterns for a character pronunciation. These 50 sequences are used to train the ETAM repeatedly using the provided algorithm. The partial matches among the training sequences determine the tree form. The time warping will not affect the form of the tree. It will introduce noise into the templates. The ETAM can restore such noisy patterns. During the retrieving phase, a sequence of unknown speech templates is applied to the trained 1309 trees in parallel. The credit is accumulated at each tree root. This is similar to a river system, where we can collect all the water at the river mouth. The water flows into any branch river or its valley will follow the river to the mouth. Any unknown pattern that falls in the tree valley will evolve to the tree root. Recognition is achieved by finding the best credit among the 1309 trees. This technique is particularly useful in recognizing Chinese voice commands. We have obtained much better performance using these trees than the arrays given by Liou et al. (1996). The ETAM has been designed to learn rhymes from musical notes. After training, it can generate varying sequences of notes with similar melodies.

With the algorithm presented in the last section, we can successfully store these

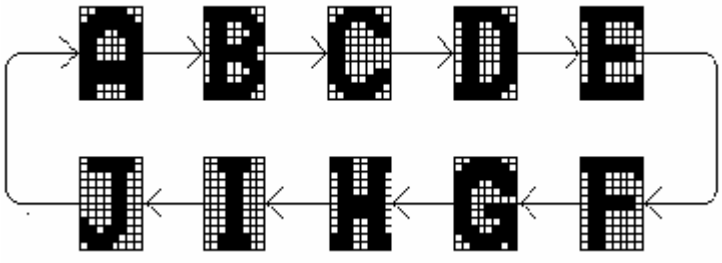
patterns in varying order. To test its error tolerance, we randomly generate noisy patterns, then check how many can converge to the template patterns. Due to its dynamic property, the temporal-AM will shift to a different pattern after every pass and will not converge to a stable pattern in the long run like the auto-associative memory. Therefore, we have two parts of each simulation. The first one is used to operate the network in one single pass to see whether the correct next pattern will be recalled. We call this part the recall rate. The second part is to operate the network until it converges to a stable or a limit cycle to see whether the final state or cycle belongs to the sequence trained. We call this part the converge rate. The results of these two parts are shown in Fig. 10.

In Fig. 10a, the recall rate is more than 90% when five errors occur, and it decreases rapidly with the increase in the number of errors. However, as shown in Fig. 10b, the convergence rate is much higher than the recall rate and remains at 90%, even when there are 30 errors. Sometimes, a noisy pattern cannot shift to the next correct next pattern but will fall into the basin of attraction of the next pattern. Therefore, one more pass will make the pattern converge to the stored sequence. This figure shows that the basins of all the patterns combine into a huge valley system for the tree.

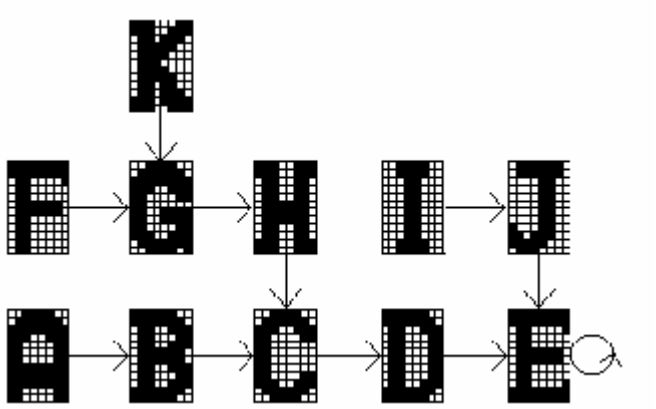
The data shown in Fig. 10 were generated in the same way as those shown in Table 4. The data were averaged over patterns to plot the figure.



(a)

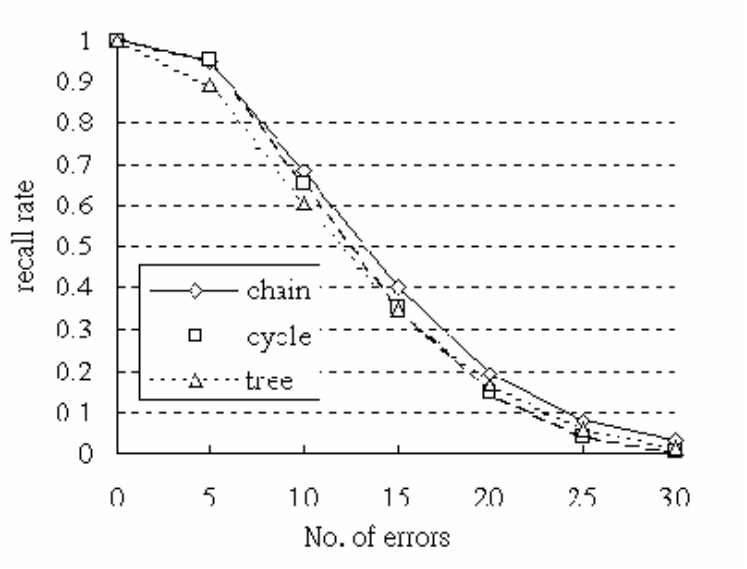


(b)

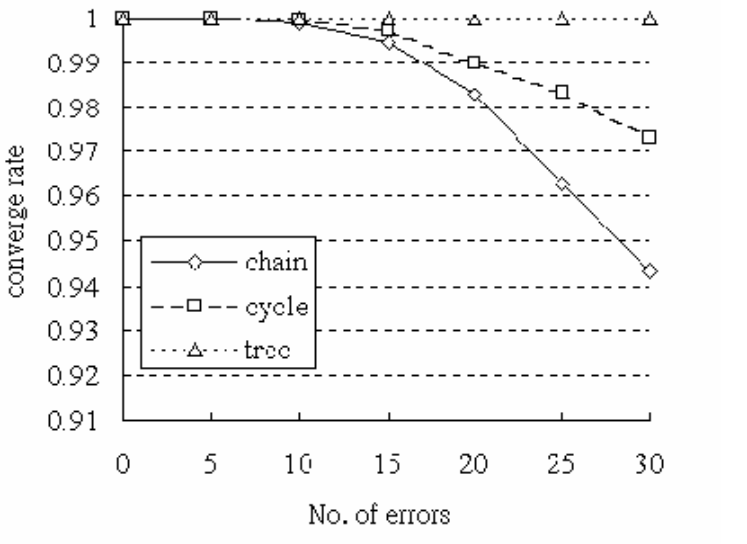


(c)

Fig. 9a-c. The ten patterns in varying order. **a** A chain; **b** a cycle; **c** a tree



(a)



(b)

Fig. 10a-b. The recall rates and convergence rates

5.7 Example II for Temporal Associative Memory

In this section, we will present one more application. We can use our method to implement a deterministic finite state automata (DFA) in a temporal-AM. In Fig. 11, there is a DFA which accepts all strings containing 01011 as a substring. There are six states and two input alphabet {a, b}, and $6 \times 2 = 12$ transition rules. We encode each transition rule into a pattern, as shown in Table 5. Each of the first six neurons represents a state. The neuron representing the current state is on, and the others are off. The other two neurons are input neurons. When input is a (or b) the seventh (or eighth) neuron is on, and the other one is off. First, we set the state to initial state S_0. When an input is received, we clamp it into the two input neurons and let the six state neurons operate in synchronous mode. After a single pass, we implement the next input, and so on. When the inputs have all been implementd, we check if the current state is the final state S_5. After training the AM using our method, we find that this DFA can be encoded into this temporal-AM successfully. When we increase the number of state neurons, this DFA can tolerate noise in state neurons.

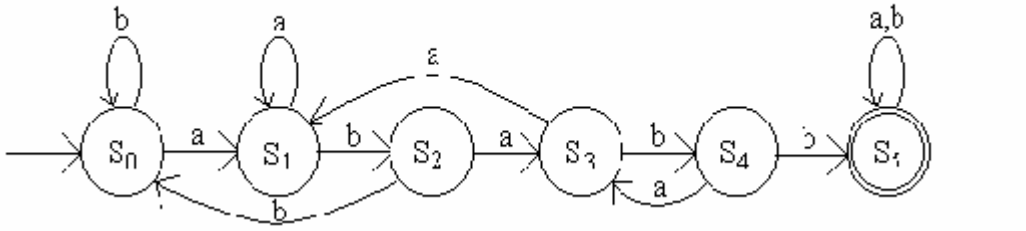


Fig. 11. The deterministic finite state automata (DFA) recognizing 01011

Table 5. The transition rules and corresponding patterns

State	Input	Next	Pattern
S_0	a	S_1	100000 10 → 010000
S_0	b	S_0	100000 01 → 100000
S_1	a	S_1	010000 10 → 010000
S_1	b	S_2	010000 01 → 001000
S_2	a	S_3	001000 10 → 000100
S_2	b	S_0	001000 01 → 100000
S_3	a	S_1	000100 10 → 010000
S_3	b	S_4	000100 01 → 000010
S_4	a	S_3	000010 10 → 000100
S_4	b	S_5	000010 01 → 000001
S_5	a	S_5	000001 10 → 000001
S_5	b	S_5	000001 01 → 000001

6 Discussions

6.1 About the Error-Correction Rule

In Eq. (22), the ECR trains AM according to the error term, and the iteration stops when there are no more errors. This terminating criterion can definitely achieve correctness, but an AM without error tolerance is useless since the patterns we present will usually be distorted. Therefore, we may modify the terminating criterion of the ECR to postpone the end of training and enlarge the summation result in Eq. (24), which is the distance in the ETAM.

Then, Eq. (24) becomes

$$v_i(t) = \text{sgn}\left[\sum_{j=1}^{N} w_{ij}(t)X_j^k - \theta_i(t) - \gamma X_i^k\right], \qquad (26)$$

where γ, the terminating criterion, is a positive constant. This equation was given by Gardner (1987, 1988).With γ equal to zero, we obtain the original error correction rule. Since γ is positive, it is rather difficult for the learning to achieve the correct result. If X_i^k are also positive, we need a larger $\sum_{j=1}^{N} w_{ij}(t)X_j^k - \theta_i(t)$, which is equal to the distance in Eq. (8), to make v_i(t) positive. Therefore, even though a few bits are wrong, the summation can still retain the same sign. Doing this has the same effect that the ETAM has in pushing the plane as far as possible away from the patterns. However, Eq. (22) together with Eq. (26) will not rotate the planes the same way as Eq. (12).

How to select γ is a problem. If γ is too small, good error tolerance cannot be achieved; while γ is too large, the iteration time will be too long and may never stop. We can gradually increase the value of γ whenever a lower value is reached by the learning in Eq. (26).

6.2 About Error Tolerant Associative Memory

Pushing planes forces each pattern to occupy more state space and, hence, enlarges the basin of attraction of the corresponding pattern to which all nearby global states will finally converge. Hence, these patterns are harder to forget and easier to recover.

In the ETAM algorithm, the terminating criterion is that the minimal distance no longer increases. In order to avoid the states being trapped, we can improve the terminating criterion (in learning step 7 in Sect. 2) by monitoring the decrease in the minimal distance two or more times successively. We find that doing this effectively improves the result. However, this may cause the iteration time to be long while the original ETAM is guaranteed to stop.

When it is found that all the patterns' *i*-th bit are the same, say 1, we simply stop the training procedure for this neuron and set the threshold θ_i to a large negative number. This ensures that this neuron will eventually converge to 1, no matter what the initial state is. Also, this will save much time and reserve more space for newly arriving memories.

Finally, in the ETAM, the weights are updated according to Eq. (12) or Eq. (20). Although the ETAM is designed based on the geometric viewpoint, it meets Hebb's postulate of learning. When two neurons fire at the same time, the weight between them is increased; otherwise, the weight is decreased. This is quite similar to many other learning methods. The difference is in the way the pattern used to adjust the weights is chosen. Neurons' weights will be tuned only for those most critical patterns selected by Eq. (9) or Eq. (17). These critical patterns have the attention of the neurons. This is similar to support vectors (Boser 1992).

As for biological modeling, the self-feedback synapse and the threshold of each neuron are active where the neuron screens out vast amounts of noncritical patterns to reduce the adaptation activities on its synapses. Its threshold will adapt to stand for the critical patterns. Its synapses will adapt to keep the critical patterns as discriminable as possible. A simple thresholding unit and Hebbian synapses are necessary for a neuron to carry out this modeling.

Note when one uses {0, 1} as the state values instead of {1, -1}, one can use coordinate transformation as for the HN to transform these two kinds of state value representations. This transformation scales and shifts the hypercube $\{-1, 1\}^N$ to match the hypercube $\{0, 1\}^N$. Substituting $\{Xi = 2Yi - 1, i = 1, ...N\}$ in the retraining algorithm will give the formulas in $\{0, 1\}^N$ space. The diagonal length of the hypercube should be scaled accordingly.

6.3 Nonnegative Definiteness

We noted earlier that large self-feedback (w_{ii}) is of benefit to nonnegative definiteness and, hence, to convergence. This is obvious in the following equation:

$$x^T Wx = \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} x_i x_j \geq 0 . \qquad (27)$$

Nonnegative definiteness requires that $x^{\mathrm{T}}Wx$ be greater than or equal to zero for all vectors x. On the right-hand side of Eq. (27), a positive w_{ii} can increase $x^{\mathrm{T}}Wx$. Therefore, large self-feedback has the effect to improving convergence and reducing limit cycles. Also, it increase the number of spurious stable states.

In the geometric sense, large w_{ii} means that the ith hyperplane is nearly perpendicular to its corresponding axis. If the thresholds are small or zero, then most of the corners in which the ith bit is equal to 1 will lie in the positive division while most of the corners in which the ith bit is equal to -1 will lie in the negative division. Again, this shows that large self-feedback is good to stability.