Natural Computing (2006) 5: 15–42
DOI: 10.1007/s11047-004-5490-x

Finite memory loading in hairy neurons

CHENG-YUAN LIOU[1,*] and SHIAO-LIN LIN[2]
[1]*Department of Computer Science and Information Engineering, National Taiwan University, Taipei. Taiwan 106, ROC (*Author for correspondence, e-mail: cyliou@csie.ntu.edu.tw);* [2]*M.D., Health Center, National Taiwan Normal University, Taipei, Taiwan 106, ROC*

Abstract. This paper presents a method to expand the basins of stable patterns in associative memory. It examines fully-connected associative memory geometrically and translate the learning process into an algebraic optimization procedure. It finds that locating all the patterns at certain stable corners of the neurons' hypercube as far from the decision hyperplanes as possible can produce excellent error tolerance. It then devises a method based on this finding to develop the hyperplanes. This paper further shows that this method leads to the hairy model, or the deterministic analogue of the Gibb's free energy model. Through simulations, it shows that this method gives better error tolerance than does the Hopfield model and the error-correction rule in both synchronous and asynchronous modes.

Key words: associative memory, error-correction rule, Gibb's free energy, hairy model, Hopfield network, Little model, music perception, neural network, spin glass model

Abbreviations: AM – associative memory; EAM – expanded associative memory; ECR – error-correction rule; LM – Little model; RK – Runge–Kutta method

1. Introduction

Neural networks have facilitated recognition, classification, mapping, and many other tasks. One of these networks is auto-associative memory (AM). Auto-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 that pattern. The fully connected network is a common architecture for auto-AM. The interconnected weights between processing neurons provide feedback for recurrent evolution of the network.

There are many models and algorithms for training this network that improve its accuracy, efficiency and capacity. One of these is

the famous network proposed by Hopfield (Hopfield, 1982). It applies the Hebb's postulate of learning in (Hebb, 1949) to generate weights. Another method commonly used to train auto-AM is the error-correction learning rule in (Widrow and Hoff, 1960). It adjusts interconnected weights only when the output is not the expected result. Several advanced designs have achieved varying degrees of success (Gardner, 1987,1989; Kanter and Sompolinsky, 1987; Tao et al., 2001).

The learning algorithm for the Boltzmann machine in (Ackley, 1985) can drastically improve both the tolerance of noisy patterns and loading capacity in the network by introducing both hidden neurons and a thermal annealing process. This algorithm is obtained by tuning the weights so as to minimize an entropy measure, G. Due to use of the noisy clamping technique and the annealing process, the computation cost is very high. The reason for augmenting the noisy clamping technique is that the cross entropy measure, $G = \sum_\alpha p_\alpha \ln p_\alpha / \grave{p}_\alpha$, cannot regulate any noisy pattern which is not included in the training set. This is because the term inside the summation sign, $\sum_\alpha$, is zero for an absent training pattern, which has $p_\alpha = 0$. It is hard to exhaustively include all the noisy patterns in the training set when a pattern size is large. Note that p_α is the probability of the αth state of the visible units when their states are determined by the environment, and that $\grave{p}_\alpha$ is the corresponding probability when the network is running freely with no environmental input. A reversed measure, $G^{\mathrm{R}} = \sum_\alpha \grave{p}_\alpha \ln \grave{p}_\alpha / p_\alpha$, is used in (Liou and Lin, 1989) to derive the learning algorithm for the machine without using the noisy clamping technique. This is because the term inside the sign, $\sum_\alpha$, is infinitely large (infinite cost) for every noisy pattern not included in the training set. This reversed measure gives excellent tolerance for noisy patterns. The annealing cost is still heavy. There is a perfect design for the hidden neurons in (Liou and Sou, 2003). The design in (Liou and Yuan, 1999) provides a biologically plausible solution for the tolerance ability without any hidden neuron and annealing process. This work will present a method to further explore the idea behind this design and formulate it for finite memory loading. The goal is somewhat similar to that of Gardner for extensive memory loading in (Gardner, 1989). This work will use the same experimental simulations as those in (Liou and Yuan, 1999) to ease comparison.

One of the fully connected network's features is that it can be viewed as a geometrical hypercube (Li et al., 1989); therefore, the learning problem of auto-AM can be transformed into a geometric optimization

problem. This work will explore learning algorithms based on this point of view. First, this work will briefly introduce the notations and the geometrical interpretation (Cover, 1965). Then, this work will present the learning method in detail. Computer simulations and discussions will be given at the end of the paper.

Auto-AM is a fully connected network with N neurons. Each neuron i has N weights connecting it to all the neurons j, including itself, a threshold θ_i, and a state value v_i. The state value is updated according to the rule

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

or in matrix form,

$$V(t+1) = \operatorname{sgn}\left[W^{\mathrm{T}} V(t) - \theta\right], \tag{2}$$

where 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 state at evolution (or iteration) time t, and sgn( ) is the signum 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, $k = 1, \ldots, P$, using various learning algorithms. In the retrieving phase, the input is presented to the network as $V(0)$. If (2) is applied to all the neurons in each evolution, then this network is said to operate in synchronous mode; if (1) is applied to only one neuron, then this network is said to operate in asynchronous mode. This work will focus on the *synchronous mode* and discuss the asynchronous mode with respect to the experiments. Then, the network operates repeatedly according to (1) or (2) until evolution converges to a stable state or falls into a limitcycle (Bruck, 1990). A stable state meets the following requirement:

$$V(t) = \operatorname{sgn}\left[W^{\mathrm{T}} V(t) - \theta\right] \tag{3}$$

no matter whether the network is operating in synchronous mode or asynchronous mode.

Each neuron has a bipolar state value (a bit), and there are 2^N states in total. Therefore, one may view the whole network as an N-dimensional (N-D) hypercube with each state located at a corner. Any two neighboring corners differ in only one neuron state, or with a Hamming distance of two for a bipolar neuron. For example, Figure 1 shows a 3-D cube corresponding to a network with three neurons. The

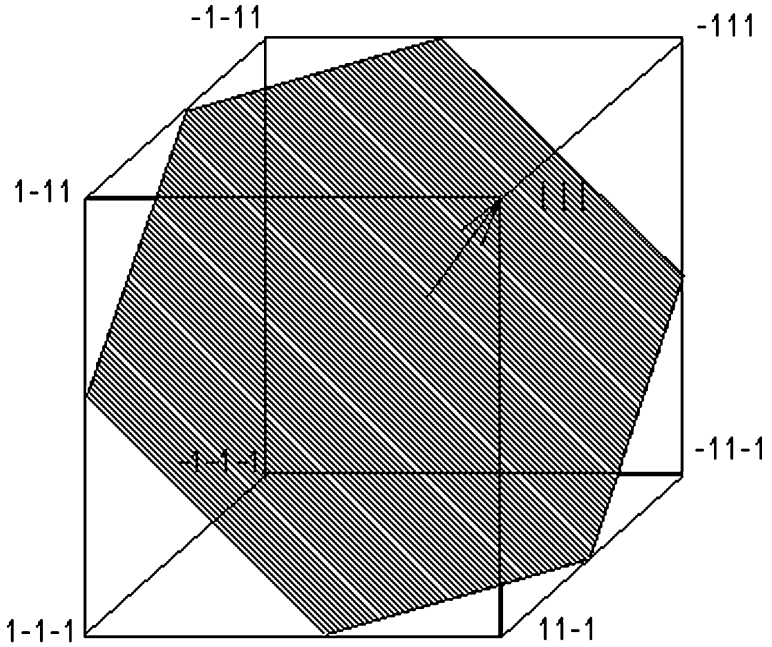

Figure 1. A neuron represents a plane (the shadow part) dividing the cube into positive and negative divisions (sides). In this figure, (1,1,1), (1,−1,1), (1,1,−1), and (−1,1,1) are in the positive division, while other corners are in the negative divisions.

current state is located at one corner and serves as the next input to the network. After updating according to (1) or (2), the current state either moves to another corner or stays at the original corner. Corners that always remain unchanged are stable states. The patterns one intends to save are located at certain stable corners. The goal of AM is to evolve an initial state to a nearby stable corner where a pattern is stored.

In the hypercube, the state of each neuron i is determined by an $(N-1)$-D decision hyperplane with the equation

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

The $N \times 1$ weight vector $W_i = (w_{i1}, w_{i2}, \ldots, w_{iN})^{\mathrm{T}}$ of neuron i is the normal vector of the corresponding hyperplane, and this hyperplane divides the hypercube into a positive part (division) to which the normal vector points and a negative division. We require that the length of the vector W_i, $|W_i|$, be normalized to one. In most cases, $|\theta_i|$ is less than $\sqrt{N}$, the half diagonal length of the hypercube. The learning process adjusts

the hyperplane to make all the patterns stable. That is, when the ith bit of a pattern is equal to 1, this stable pattern should be located in the positive division of the ith hyperplane; on the other hand, if its ith bit equal to -1, it should be located in the negative division. Such division is exactly the same as the binary coded division for multilayer networks in (Liou and Yu, 1995).

The basin of a stable pattern is defined as the collection of all patterns which will evolve to this stable pattern eventually using the dynamic equation (2). We subdivide a basin according to the number of iterations, λ. Therefore, the basin-λ of a stable pattern contains all the noisy patterns that will evolve to this stable pattern in λ iterations. The basin-0 contains the stable pattern only. All such basins, $\{\text{basin-}\lambda; \lambda = 0, 1, 2, \ldots\}$, constitute the entire basin of a stable pattern.

When the neighboring cube corners are the noisy patterns (in terms of the Hamming distance) of a stable pattern and are in the same division as that in which this stable pattern is located for every decision hyperplane, they can be restored to this stable pattern in a single evolution (iteration) by using the dynamic equation (2). Therefore, they are in basin-1 of this stable pattern, and they are the only patterns in basin-1. The division of a stable pattern contains both basin-1 and basin-0 of this stable pattern. Each division is enclosed by decision hyperplanes. All the basins except for the basin-0 are separated from each other by hyperplanes. Note that each state (the signs of the state's elements) in basin-1 indicates a unique division of basin-2 which contains noisy patterns that will evolve to the stable pattern in two iterations. All divisions indicated by the states in basin-1 constitute basin-2. Each state in basin-2 indicates a division of basin-3, and so on. When one increases the number of states in basin-1, one increases the number of divisions in basin-2 indirectly, and so on. This means that whenever one increases a state in the basin-i of a stable pattern, one increases a whole division in its basin-$(i+1)$. This is in some sense similar to a chain reaction or a positive feedback system. Such reaction would be a dominant evolution in any system. This kind delicate reaction mechanism, which is different from the random boolean network in (Kauffman, 1991), has evolutionary implications and can serve as the foundation for exploring the biological evolution. We expect that the increase of the number of states in basin-1 will increase the overall basin for a stable pattern indirectly and increase the number of noisy patterns with low iteration numbers. We hope that this chain-like reaction will exhaust all the states for stable patterns and leave no limit cycles. This is beneficial for restoration. The noisy patterns in basin-1 are close to their stable pattern and are in line

with the Hamming distance neighbors. Hence, this work formulates a method as attempts to tune the hyperplanes and enlarge the basin-1 divisions of stable patterns without damaging the stability of those patterns.

2. Optimal hyperplane

Since each decision hyperplane can be adjusted separately based on the above idea, this work will only discuss neuron i. For neuron i, one has P equations:

$$\begin{cases} w_{i1}X_1^k + w_{i2}X_2^k + \cdots \\ \quad +w_{iN}X_N^k - \theta_i = W_i^{\mathrm{T}}X^k - \theta_i > 0 \quad \text{if} \quad X_i^k = 1 \\ w_{i1}X_1^k + w_{i2}X_2^k + \cdots \\ \quad +w_{iN}X_N^k - \theta_i = W_i^{\mathrm{T}}X^k - \theta_i < 0 \quad \text{if} \quad X_i^k = -1 \end{cases} \qquad k = 1, \ldots, P. \tag{5}$$

We discard the case in (5), where $W_i^{\mathrm{T}}X^k - \theta_i = 0$ because it rarely happens in analog operation. Whenever we say that the ith hyperplane is stable, it means that all P patterns on the right (stable) side (division) of the hyperplane satisfy (5). This hyperplane is not stable whenever there is a pattern in the wrong division. This work will explore the flexible space in between the two pattern sets, $\{X^k; k = 1, \ldots, P, \text{ and } X_i^k = 1\}$ and $\{X^k; k = 1, \ldots, P, \text{ and } X_i^k = -1\}$, in order to locate the decision hyperplane under the stability conditions in (5). We will suppose that both sets are not empty. There exits at least one pattern in each set. For computational convenience, we multiply every element in the second equation by -1 and obtain new equations:

$$\begin{cases} s_i^k = w_{i1}X_1^k + w_{i2}X_2^k + \cdots \\ \quad +w_{iN}X_N^k - \theta_i = W_i^{\mathrm{T}}X^k - \theta_i > 0 \quad \text{if} \quad X_i^k = 1 \\ s_i^k = -w_{i1}X_1^k - w_{i2}X_2^k - \cdots \\ \quad -w_{iN}X_N^k + \theta_i = -W_i^{\mathrm{T}}X^k + \theta_i > 0 \quad \text{if} \quad X_i^k = -1 \end{cases} \qquad k = 1, \ldots, P. \tag{6}$$

Note that W_i is a normalized normal vector. s_i^k is the distance (Euclidean distance) between the kth pattern and the ith decision hyperplane, which is positive when the kth pattern is on the correct

(stable) side of the ith hyperplane. By (6), when a hyperplane has a large value of s_i^k, it can tolerate many more perturbations in the states X^k and keep the inequality. Since W_i is a normalized normal vector, the threshold θ_i will be the major parameter used in increasing the value of s_i^k.

We expect that increasing s_i^k will enlarge the division overall and improve the tolerance ability. This work devises techniques to tune weights to make these distances, $\{s_i^k\}$, as large as possible. To achieve this goal, we maximize the minimum of $\{s_i^k, k = 1, \ldots, P\}$ to improve the hyperplane's tolerant ability for those Hamming distance neighbors as possible. This work presents a method to do this in the following.

2.1. *Method*

The method uses the gradient ascent to increase the distance s_i^k. We consider $(s_i^k)^2$ in this method. $(s_i^k)^2$ is as follows:

$$\begin{aligned}(s_i^k)^2 &= (w_{i1}X_1^k + w_{i2}X_2^k + \cdots + w_{iN}X_N^k - \theta_i)^2 \\ &= \left(\sum_{l=1}^{N} w_{il}^2\right) + \theta_i^2 + \sum_{l=1}^{N}\sum_{l'=1,l'\neq l}^{N} w_{il}w_{il'}X_l^k X_{l'}^k - 2\theta_i\left(\sum_{l=1}^{N} w_{il}X_l^k\right).\end{aligned} \tag{7}$$

There are $N+1$ variables in (7). Each variable has degree two (quadratic form) and a positive leading coefficient. Since this work requires that the normal vector W_i be normalized, the term $\sum_{l=1}^{N} w_{il}^2$ is equal to 1. This work applies the partial derivative to $(s_i^k)^2$:

$$\begin{aligned}\frac{\partial (s_i^k)^2}{\partial w_{ij}} &= 2\sum_{l=1,l\neq j}^{N} w_{il}X_j^k X_l^k - 2\theta_i X_j^k \\ &= 2X_j^k\left(\sum_{l=1,l\neq j}^{N} w_{il}X_l^k - \theta_i\right) \\ &= \begin{cases} 2X_j^k(s_i^k - w_{ij}X_j^k) = 2(s_i^k X_j^k - w_{ij}) & \text{if} \quad X_i^k = 1, \\ 2X_j^k(-s_i^k - w_{ij}X_j^k) = 2(-s_i^k X_j^k - w_{ij}) & \text{if} \quad X_i^k = -1, \end{cases}\end{aligned} \tag{8}$$

$$\frac{\partial (s_i^k)^2}{\partial \theta_i} = -2 \sum_{l=1}^{N} w_{il} X_l^k + 2\theta_i$$
$$= -2 \left(\sum_{l=1}^{N} w_{il} X_l^k - \theta_i \right)$$
$$= \begin{cases} -2s_i^k & \text{if} \quad X_i^k = 1, \\ 2s_i^k & \text{if} \quad X_i^k = -1. \end{cases} \tag{9}$$

In (8, 9), we see that for the case $X_i^k = 1$, when $2(s_i^k X_j^k - w_{ij})$ (or $-2s_i^k$) is positive, increasing w_{ij} (or θ_i) will increase $(s_i^k)^2$, and when $2(s_i^k X_j^k - w_{ij})$ (or $-2s_i^k$) is negative, increasing w_{ij} (or θ_i) will decrease $(s_i^k)^2$. w_{ij} must be tuned (or trained) according to the following equations to increase $(s_i^k)^2$:

$$w_{ij}(t+1) = w_{ij}(t) + \varepsilon_1 \frac{\partial (s_i^k)^2}{\partial w_{ij}}$$
$$= \begin{cases} w_{ij}(t) + 2\varepsilon_1 (s_i^k(t) X_j^k - w_{ij}(t)) & \text{if} \quad X_i^k = 1, \\ w_{ij}(t) + 2\varepsilon_1 (-s_i^k(t) X_j^k - w_{ij}(t)) & \text{if} \quad X_i^k = -1. \end{cases} \tag{10}$$

$$\theta_i(t+1) = \theta_i(t) + \varepsilon_2 \frac{\partial (s_i^k)^2}{\partial \theta_i}$$
$$= \begin{cases} \theta_i(t) - 2\varepsilon_2 s_i^k(t) & \text{if} \quad X_i^k = 1, \\ \theta_i(t) + 2\varepsilon_2 s_i^k(t) & \text{if} \quad X_i^k = -1, \end{cases} \tag{11}$$

where the training rates ε_1 and ε_2 are small positive constants. Note that t denotes the training iteration. Substituting $w_{ij}(t+1)$ and $\theta_i(t+1)$ into (6), we obtain

$$s_i^k(t+1) = \begin{cases} W_i^{\mathrm{T}}(t+1) X^k - \theta_i(t+1) & \text{if} \quad X_i^k = 1, \\ -W_i^{\mathrm{T}}(t+1) X^k + \theta_i(t+1) & \text{if} \quad X_i^k = -1. \end{cases} \tag{12}$$

Since $(s_i^k)^2$ cannot indicate which division the kth pattern belongs to, (10, 11) always increase the distance between the kth pattern and the hyperplane but cannot correct the incorrect division of the hyperplane. $s_i^k(t)$ must be kept positive during the whole training process. This work monitors the value of $s_i^k(t)$ to ensure positivity. Its positivity can always be preserved by tuning down the training rates. The initial $w_{ij}(0)$ and

$\theta_i(0)$ must be set so as to make all the patterns stable (Li et al., 1989) to ensure positivity. They are set as follows:

$$w_{ij} = \begin{cases} 1 & \text{if} \quad i = j, \\ 0 & \text{if} \quad i \neq j, \end{cases} \qquad (13)$$
$$\theta_i = 0.$$

We set the initial weights according to (13) when the number of patterns is very large, $2^N \geq P \ggg N$. These weights can store stable patterns up to the network limit, 2^N. When the number of patterns is comparable to that of neurons, $2^N \ggg P \approx N$, we present the following hyperplane.

We select a pair of patterns $\{c^p, c^n\}$ from each of the two sets, $\{X^k; k = 1, \ldots, P, \text{ and } X_i^k = 1\}$ and $\{X^k; k = 1, \ldots, P, \text{ and } X_i^k = -1\}$, where c^p is a pattern in $\{X^k; k = 1, \ldots, P, \text{ and } X_i^k = 1\}$ and c^n is a pattern in the other set. This pair is the closest pair (in terms of the Euclidean distance) among all the pairs between the two sets. The weights of the hyperplane are set as

$$w_{ij} = c_j^p - c_j^n, \text{ normalize } W_i,$$
$$\theta_i = \sum_{j=1}^{N} w_{ij} \left(\frac{c_j^p + c_j^n}{2} \right). \qquad (14)$$

This hyperplane is right in the middle between the two patterns of the closest pair. It is perpendicular to the line section connecting the two patterns $\{c^p, c^n\}$ and passes the center of this section. $c^p - c^n$ is the direction of the normal vector of this hyperplane, see (Eqs. 8–11, Liou and Yuan, 1999). Further studies show that this hyperplane preserves all the merits of Hebb's postulate and the stability. We have sought such a simple crystal memory for the Hopfield network for 15 years. This novel memory can serve as the foundation for exploring the physiological implications. We will use this hyperplane and (13) in all auto-AM simulations to start the training iteration.

In each training iteration, this method calculates all the distances, $\{s_i^k, k = 1, \ldots, P\}$, using (6) or (12) and use the pattern with minimal s_i^k in (10) and (11) to improve the position of the decision hyperplane. This pattern is in some sense close to the support vector in (Boser, 1992). This method sets the hyperplane in the first iteration as in (14) or (13). $W_i(t)$ must be normalized after each iteration. The iteration stops whenever s_i^k shows no more improvement. This method is applied to adjust the ith hyperplane only when the ith bits of all P patterns are not equal. When

the ith bits of all P patterns are equal (backbone bit), we may locate the hyperplane directly without training. This method sets this hyperplane outside the hypercube through proper division. An easy way to do this is to set θ_i greater than $\sqrt{N}$. Figure 1 shows the training result for two patterns, (1,1,1) and $(-1,-1,-1)$. All three optimal hyperplanes are almost coincident. All N hyperplanes can be trained in sequence or in parallel. After training, this method finds the closest hyperplane to each stable pattern corner and use that hyperplane to determine the basin radius for that pattern. The basin-1's border can be traced along the corners near the hyperplanes which enclose this pattern corner.

3. Experiments I: Auto-associative memory

This work performed simulations to compare the performance of the Little model (LM) in (Little, 1974), the error-correction rule (ECR) in (Widrow and Hoff, 1960), the Runge–Kutta method (RK) in (Tao et al., 2001), and the proposed expanded associative memory (EAM) in synchronous mode. Several issues, such as the number of stable states, the number of limit cycles, and the achieved fault tolerance, will be discussed. The networks, LM, ECR, and EAM, are all applied with a fully-connected network structure in synchronous mode. The difference between these three networks lies at the learning phase. This work also implemented a pattern recognition application. In all the simulations, this method sets $\varepsilon_1 = \varepsilon_2 = 0.00055$ in EAM. This work uses fourth-order Runge–Kutta method in RK. The simulations set the same time steps, 0.01 and 0.1, as those used in (Tao et al., 2001). The stopping criterion for RK is according to the difference between the updated new value and its old value. Whenever the difference is less than 0.000005, the simulations stop RK. Below, this work briefly reviews these networks.

3.1. *Hopfield model (HM)*

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

$$w_{ij} = \begin{cases} \frac{1}{N} \sum_{k=1}^{P} X_i^k X_j^k & \text{if} \quad i \neq j, \\ 0 & \text{if} \quad i = j. \end{cases} \tag{15}$$

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 spurious stable states for the reason that overlarge self-feedback will tend to cause neurons to retain their previous states. This self-feedback has been discussed previously in (Kanter and Sompolinsky, 1987; Wilde, 1997). The zero-diagonal and symmetric weight matrix cause the HM to always converge to a stable state in asynchronous dynamics. However, the model we are interested in is called the Little model and is similar to HM. It differs from HM only in that it uses synchronous dynamics. This causes the network to always converge not only to a stable state, but also to a limit cycle of length two.

3.2. *Error-correction rule (ECR)*

ECR adjusts weights proportional to the error term $(X_i^k - v_i(t))$. At the beginning, the simulation randomly assigns initial values to all the weights and then adjust all the weights according to following equations:

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \eta(X_i^k - v_i(t))X_j^k, \\ \theta_i(t+1) &= \theta_i(t) - \eta(X_i^k - v_i(t)), \\ v_i(t) &= \operatorname{sgn}\left[\sum_{j=1}^{N} w_{ij}(t)X_j^k - \theta_i(t)\right], \end{aligned} \tag{16}$$

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 continues until there is no more error.

3.3. *Experimental results*

Table 1 lists experimental results for $(N = 5, P = 5)$, $(N = 5, P = 3)$, $(N = 10, P = 5)$ and $(N = 10, P = 3)$, respectively. In each experiment, the simulations presented 10 sets of randomly produced patterns to the four networks and then got the averaged results. We explain each row item below:

SP (#of Stored Patterns ($/P$)): given P patterns, the number of patterns successfully stored.
SS (#of Stable States ($/2^N$)): the number of stable states.
TS (#of States to Stable ($/2^N$)): the number of transient states converging to all stable states.

Table 1. Comparison among Little model (LM), error-correction rule (ECR), expanded associative memory (EAM) and Runge–Kutta method (RK)

	LM	ECR	EAM	RK
$N = 5$, $P = 5$				
SP (/5)	1.8	5	5	5
SS (/32)	4.2	15.2	17.6	5
TS (/32)	14.6	16.8	14.4	29
C	2.7	0	0	0
IC (/32)	5.4	0	0	0
TC (/32)	7.8	0	0	0
R (/25)	3.2	3.9	5.0	15.4
Time secs	0.03	0.02	0.11	5136
$N = 5$, $P = 3$				
SP (/3)	2.4	3	3	3
SS (/32)	5.2	7.1	5.0	2.8
TS (/32)	13.0	24.9	27.0	29.1
C	5.5	0	0	0
IC (/32)	11	0	0	0
TC (/32)	2.8	0	0	0
R (/15)	4.0	4.0	10.5	12
Time secs	0.03	0.02	0.07	1570
$N = 10$, $P = 5$				
SP (/5)	1.9	5	5	5
SS (/1024)	5.0	43.9	52.7	5
TS (/1024)	744.8	978.4	971.3	993.6
C	44.3	0.2	0	0
IC (/1024)	88.6	0.4	0	0
TC (/1024)	185.6	1.3	0	0
R (/50)	12.1	13.5	39.6	48.6
Time secs	0.12	0.11	0.39	2115629
$N = 10$, $P = 3$				
SP (/3)	2.6	3	3	3
SS (/1024)	6.8	20.2	6.2	3
TS (/1024)	553.4	1003.2	1017.8	903
C	84.9	0.2	0	0
IC (/1024)	169.8	0.4	0	0
TC (/1024)	294.0	0.2	0	0
R (/30)	21.3	8.6	29.2	29.6
Time secs	0.11	0.1	0.27	625077

C (#of Cycles): the number of limit cycles.
IC (#of States in Cycles ($/2^N$)): the number of states involved in all limit cycles. ($\geq$ 2C)
TC (#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.
Time in seconds: total cpu time on an IBM PC including training and restoration for the 10 sets of patterns.

The given patterns can be successfully memorized using ECR, EAM, and RK. RK gives better results with expensive cpu time. It has a non-Hebbian structure. These patterns are guaranteed to be saved by EAM because they are the conditions of (6). LM has rather limited capacity, so it cannot even store three patterns in a ten-neuron network in all cases, and neither can HM. Many researchers have pointed out that the maximum number of patterns stored in HM is $N/4 \ln N$ (McEliece et al., 1987). There are advanced designs for storing N different patterns with large basins in a straightforward way (Gardner, 1987; Kanter and Sompolinsky, 1987; Li et al., 1989). Many of them derive symmetric weight matrices. The EAM derives asymmetric matrices with nonzero diagonal elements and still keeps Hebb's postulate.

Although the number of stable states in LM is the smallest in many cases, it lacks the ability to store more patterns, so this advantage becomes useless. Comparing ECR and EAM, we find that EAM has fewer spurious stable states than ECR does when there are three patterns, and has more when there are five patterns. ECR stops training when all the patterns have been successfully saved, while EAM continues training until the minimal distance cannot be increased any more. The largest minimal distance is easy to obtain when there are fewer patterns but is difficult to obtain when there are more patterns. Therefore, the training is repeated, the self-feedback weights w_{ii} continue to increase, and more spurious stable states are generated. Overlarge self-feedback will cause a neuron to stay in its previous state and produce more stable states. The extreme situation occurs when all the weights are zero except w_{ii}. In this case, all the neurons remain unchanged, and all the states are stable.

LM has a much serious cycle problem, while ECR produces very few limit cycles. EAM produces no limit cycles as we expected. Again, this is because of the large self-feedback. When self-feedback is large, all the neurons tend to stay in their previous states; hence, the number of limit cycles can be effectively reduced.

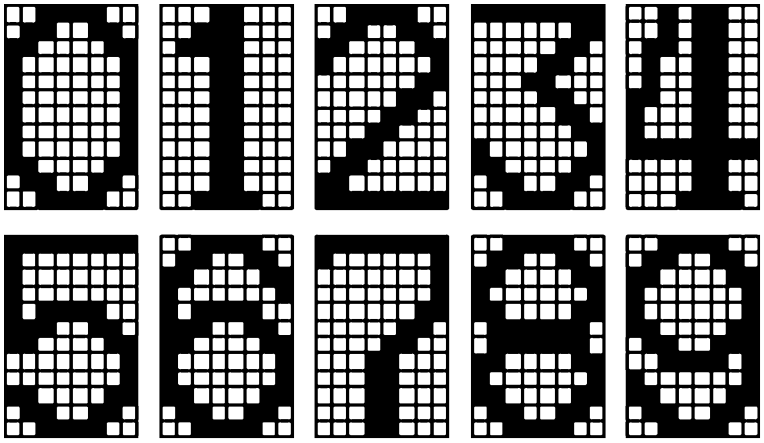

Figure 2. The bitmap for the 10 digital patterns.

When the number of patterns is small enough compared to that of neurons, LM has acceptable performance in recovery from distorted patterns as shown in Table 1 with $N = 10$ and $P = 3$. However, when there are more patterns, the error tolerance of LM is poor. This is also the case for ECR even when there are few patterns because the criterion for training ECR is accuracy, not error tolerance. The performance of EAM in recovery from noisy patterns is much better than that of the other two models. This is because EAM tunes the decision hyperplane so as to enlarge the decision division for each pattern and to include as many neighboring noisy patterns as possible in that division.

Table 2. Radius of each pattern for expanded associative memory (EAM) and error-correction rule (ECR)

	EAM	ECR
0	2	0
1	3	0
2	3	0
3	3	0
4	3	0
5	2	0
6	2	0
7	3	0
8	2	0
9	2	0

3.4. *Pattern recognition application*

Next, the simulation tested a more complex example of pattern recognition. In Figure 2, there are 10 patterns, from 0 to 9, constructed using 96 neurons (bits). After applying LM, ECR, and EAM to these patterns, the simulation found that LM could not memorize all the patterns, so we will not discuss LM with regard to further experiments. The basin radius produced by EAM is much better than those produced by ECR as shown in Table 2. The basin radius is defined as the maximal number of errors for which recovery is guaranteed, no matter which bits are wrong. This radius is the distance (in Hamming distance) between the stable pattern and a closest state along the basin-1's border. This is a much rigorous definition. The number in this table denotes the size of the radius (with a unit of one bit or a Hamming distance of 2 for each bipolar bit). The "0" numbers in ECR mean that there exists at least one bit or pixel that is vulnerable to noise contamination no matter how large the basin is.

The 10 expanded basins of attraction are huge. The simulations monitored the basin size of each pattern during the training of EAM. Because the pattern complexity of 96 neurons is 2^{96}, the simulations

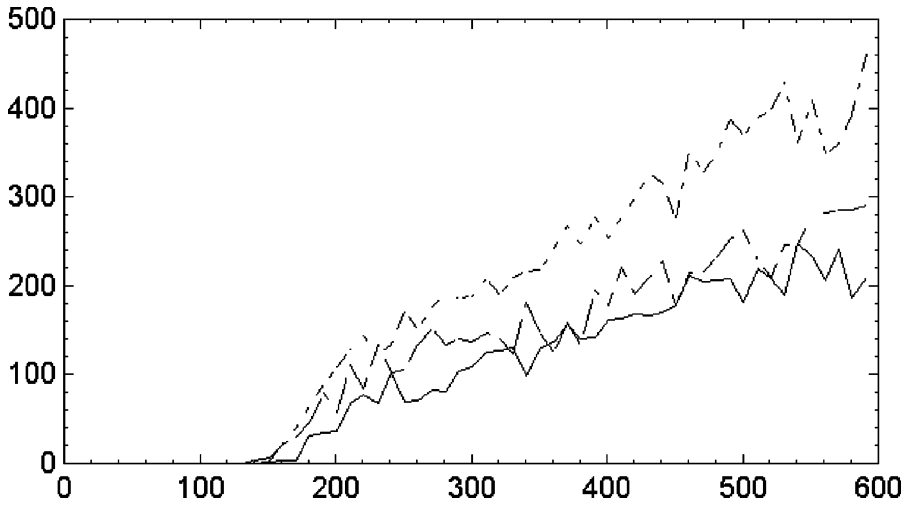

Figure 3. The growth of basins for patterns $\{'0', '1',$ and $'7'\}$ during training iterations using EAM. The number on the abscissa is the training iterations t. The number on vertical coordinate shows the basin size by counting the total number of patterns within the 10,000 random patterns which converge to a digital pattern. The solid line is for the digital patterns $'0'$, the dash-dot line is for $'1'$, and the dashed line is for $'7'$.

Table 3. Error Tolerance for expanded associative memory (EAM) and error-correction rule (ECR)

	0	1	2	3	4	5	6	7	8	9
EAM in synchronous mode										
10	1000	1000	1000	1000	1000	1000	998	999	1000	998
20	965	984	980	982	990	976	955	984	970	965
30	681	775	717	779	803	775	701	801	698	683
EAM in asynchronous mode										
10	1000	998	1000	1000	1000	1000	1000	1000	999	1000
20	970	987	984	985	986	981	935	988	964	958
30	710	786	810	811	809	717	617	863	750	712
ECR										
10	120	224	23	102	42	7	1	784	6	35
20	47	77	5	22	3	0	0	651	0	2
30	23	14	0	0	0	0	0	335	0	0

applied a statistical approach to roughly show the basin sizes. The experiments randomly produced 10,000 patterns and fed them into the trained network EAM. Then, the experiments counted how many random patterns converged to each pattern as shown in Figure 2. We plot the results for three patterns after every 10 iterations in Figure 3. One can see that although the basin sizes sometimes decreased, they expanded overall.

Table 3 shows the error tolerance of EAM. For each pattern of the 10 digits, we randomly generated 1000 noisy patterns with 10, 20, and 30 error bits, respectively, and used ECR and EAM to recover them. The numbers shown in this table are the total numbers of patterns in the 1000 noisy patterns that could be restored by the two methods in different operation modes. Results show that almost all the noisy patterns with 10 or 20 errors could be recovered by EAM. Even with 30 errors, more than 60% of the noisy patterns could be recovered. Also, we notice that for all 30,000 randomly generated patterns, none of them fell into a limit cycle in EAM. This also shows the excellent ability of EAM to avoid the production of limit cycles.

The experiments also tested the storage capability of the EAM. We randomly generated 500 patterns using these 96 neurons. The EAM could accomodate these patterns, and almost all of them had huge basins. The pseudo-inverse approach in (Kanter and Sompolinsky, 1987) will not

work for storing such a large number of patterns. This number greatly exceeds the storage capacity of many modern designs.

Although we designed the EAM for use in synchronous mode, the simulations show that they can operate satisfactorily in asynchronous mode with even better performance than they can in synchronous mode.

4. Experiments II: Temporal associative memory

The temporal AM is used to store one sequence or several sequences of patterns in the AM's dynamic state transitions (Amari, 1972). Given an initial input pattern, it will converge to the next pattern in a memorized sequence. All the successive patterns in this sequence will be recalled sequentially. Because of its dynamic property, it can be used to recognize or generate temporal patterns, such as speech, film, command, timbre, rhythm, bird song, or musical notations.

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

$$X_i^{k+1} = \operatorname{sgn}\left(\sum_{j=1}^{N} w_{ij} X_j^k - \theta_i\right). \tag{17}$$

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

This temporal AM can store various kinds of pattern sequences, such as a single chain (the dotted line in Figure 4), a cycle of patterns (the dashed line in Figure 4), or a tree (the upper two patterns in Figure 4). Generally, the temporal AM is able to save all one-to-one or many-to-one patterns, but not one-to-many patterns. This work will not discuss the one-to-many case.

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

$$w_{ij} = \sum_{k=1}^{P} X_i^{k+1} X_j^k. \tag{18}$$

This temporal AM has asymmetric weight and no threshold. It has the same drawback as HM: low capacity and incorrect recall. This implementation usually cannot memorize complete patterns, as we will show based on simulations described later.

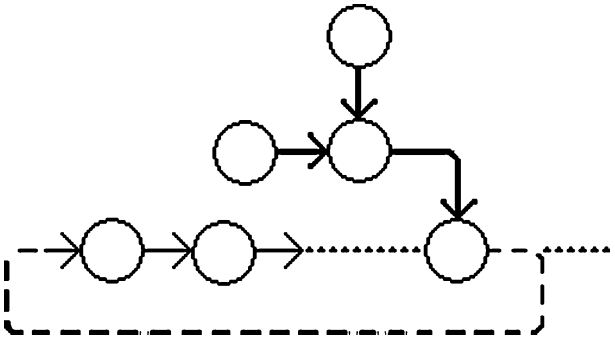

Figure 4. Various kinds of associative patterns are plotted in one graph, where each circle denotes a different pattern. The patterns linked by the dotted line constitute a single chain associative memory.The patterns linked by the dashed line constitute a cycle. The top three patterns constitute a tree.

The techniques in EAM 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-1 of a stable pattern will converge to this pattern, while in the temporal AM, all the states in the basin-1 of a pattern will evolve to its next pattern. To train weights is to allocate the hyperplane so as to separate patterns into two divisions according to the ith bits of their next patterns. That is when $X_i^{k+1} = 1$, X^k should be in the positive division of the ith hyperplane and in the negative division if $X_i^{k+1} = -1$. (6) should be modified as follows:

$$\begin{cases} s_i^k = w_{i1}X_1^k + w_{i2}X_2^k + \cdots \\ \quad + w_{iN}X_N^k - \theta_i = W_i^{\mathrm{T}}X^k - \theta_i > 0 & \text{if } X_i^{k+1} = 1, \quad \mathrm{k} = 1,...,\mathrm{P}. \\ s_i^k = -w_{i1}X_1^k - w_{i2}X_2^k - \cdots \\ \quad - w_{iN}X_N^k + \theta_i = -W_i^{\mathrm{T}}X^k + \theta_i > 0 & \text{if } X_i^{k+1} = -1 \end{cases} \tag{19}$$

The tuning equations (10–12), are also modified to fit the temporal case. We list them below:

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) \pm \varepsilon_1 \frac{\partial (s_i^k)^2}{\partial w_{ij}}, \begin{cases} + & \text{if } s_i^k \geq 0 \\ - & \text{if } s_i^k < 0 \end{cases} \\ &= \begin{cases} w_{ij}(t) \pm 2\varepsilon_1 (s_i^k(t) X_j^k - w_{ij}(t)) & \text{if } X_i^{k+1} = 1, \\ w_{ij}(t) \pm 2\varepsilon_1 (-s_i^k(t) X_j^k - w_{ij}(t)) & \text{if } X_i^{k+1} = -1, \end{cases} \end{aligned} \tag{20}$$

$$\theta_i(t+1) = \theta_i(t) \pm \varepsilon_2 \frac{\partial (s_i^k)^2}{\partial \theta_i} = \begin{cases} \theta_i(t) \mp 2\varepsilon_2 s_i^k(t) & \text{if} \quad X_i^{k+1} = 1, \\ \theta_i(t) \pm 2\varepsilon_2 s_i^k(t) & \text{if} \quad X_i^{k+1} = -1. \end{cases} \tag{21}$$

Substituting them in (19), we obtain

$$s_i^k(t+1) = \begin{cases} W_i^{\mathrm{T}}(t+1)X^k - \theta_i(t+1) & \text{if} \quad X_i^{k+1} = 1, \\ -W_i^{\mathrm{T}}(t+1)X^k + \theta_i(t+1) & \text{if} \quad X_i^{k+1} = -1. \end{cases} \tag{22}$$

We recommend an initial condition which places the initial hyperplane right in the middle between the center of patterns with $X_i^{k+1} = 1$ and the center of patterns with $X_i^{k+1} = -1$. That is,

$$\begin{aligned} C_j^p &= \frac{1}{N_1} \sum_{\{k | X_i^{k+1} = 1\}} X_j^k, \\ C_j^n &= \frac{1}{N_{-1}} \sum_{\{k | X_i^{k+1} = -1\}} X_j^k, \\ w_{ij}(0) &= C_j^p - C_j^n, \\ \theta_i(0) &= \sum_{j=1}^{N} w_{ij}(0) \left(\frac{C_j^p + C_j^n}{2} \right), \end{aligned} \tag{23}$$

where C^p and C^n are the centers of patterns with $X_i^k = 1$ and of patterns with $X_i^k = -1$, respectively.

The training iteration is similar to that for EAM. In each iteration we find the worst pattern corner and use that pattern in (20, 21) to improve the location of the hyperplane. Then, we normalize the weight vector W_i and calculate new distances $\{s_i^k, k = 1, \ldots, P\}$. The initial weights in (23) were used in the simulations. The distance, s_i^k, may have a negative value when the kth pattern is on the wrong side of the ith hyperplane, where the sign of X_i^{k+1} shows the correct side. We do not expect that the initialization, (18) or (23), will give a perfect dynamic at the beginning. Therefore, we modify EAM to include the cases where a pattern corner may be on the wrong side of the hyperplane as in (20, 21). This modification will decrease the distance between a pattern corner and a hyperplane when this corner is on the wrong side of the hyperplane. This modification will also increase the distance between a pattern corner and a hyperplane when this pattern is on the correct side.

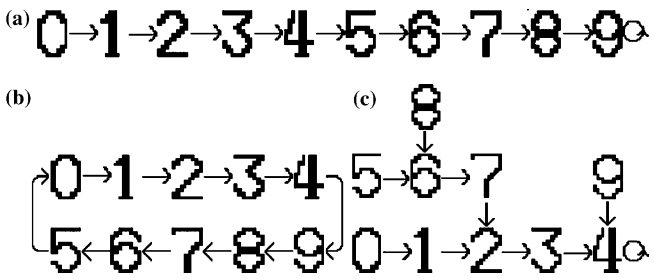

Figure 5. The 10 patterns in three different associative sequences. They from a single chain memory (a), a cycle memory (b), and a tree memory (c).

This work conducted simulations using the temporal AM. They require that the temporal AM store the 10 patterns in different orders. In Figure 5(a), they are saved as a chain; in Figure 5(b), they are saved as a cycle; and in Figure 5(c), they are saved as a tree. In general, the dynamic can be saved in a manner similar to a river system. The basins of these patterns are similar to river valleys. The water (system state) will flow (transition according to the dynamics) along the valleys (pattern basins) to the river mouth. Any contaminated temporal pattern, which falls into an upstream valley (basin), will be carried by the system through the valleys into the river mouth (tree root). A cumulated credit for a sequence of contaminated patterns can be collected at the mouth. One of the goals of the project is to devise a temporal AM with improved recognition of speech. This EAM has been used to learn musical notes. It can generate varying sequences of notes with similar melodies after learning (see http://red.csie.ntu.edu.tw/MG/index.htm).

In the three cases shown in Figure 5, we used both Amari's method as in (18) and the proposed EAM to train the networks. The simulations found that Amari's method could not store all the patterns. There existed serious limit cycles. With the proposed method, one could successfully store these patterns with enlarged basins to accomodate noisy patterns. A noisy pattern would always evolve into the basin-1 of the next pattern. Storing such sequential patterns is the focus of many advanced designs.

5. Discussion

First we will show that the idea behind the proposed method is in line with Hebb's postulate. (5) together with its *if* conditions, if one ignores

the fact that the weights W_i must be normalized, can be rewritten in the form

$$s_i^k = (w_{i1}X_1^k + w_{i2}X_2^k + \cdots + w_{iN}X_N^k - \theta_i) \cdot X_i^k = (W_i^{\mathrm{T}} X^k - \theta_i) \cdot X_i^k > 0. \tag{24}$$

We apply the partial derivative to the above equation and obtain

$$\frac{\partial s_i^k}{\partial w_{ij}} = X_j^k X_i^k \qquad \text{and} \qquad \frac{\partial s_i^k}{\partial \theta_i} = -X_i^k. \tag{25}$$

This result is *coherent* with the Hopfield model and Hebb's postulate. This means that Hebb's postulate also increases the distance, s_i^k, indirectly. The postulate says that w_{ij} will increase when X_i^k is equal to X_j^k. According to (25), the distance s_i^k will increase under this postulate. The present method extends the postulate to the threshold θ_i. They increase this distance directly and effectively in a global sense, and serve as a generalization of the postulate.

The energy function for the above two equations (24, 25) is

$$E_{\mathrm{Hebbian}}(W, \theta) = -\sum_{i=1}^{N}\sum_{k=1}^{P} s_i^k = -\mathrm{trace}\{X^{\mathrm{T}} W^{\mathrm{T}} X\} + \theta^{\mathrm{T}} X[I], \tag{26}$$

where the pattern matrix X contains each pattern X^k in its kth column, $X = [X^1, X^2, \ldots, X^P]$. The unit vector $[I]$ is a $P \times 1$ column vector that contains 1 in all its entries, $[I] = [1, 1, 1, \ldots, 1]^{\mathrm{T}}$. Note that we reverse the sign of (24) to obtain this energy function to be consistent with the popular definition. This work will follow this definition in all discussions. The energy function for the temporal AM, (18), is

$$E_{\mathrm{TAM}}(W, \theta) = -\mathrm{trace}\{X_{\mathrm{shift}}^{\mathrm{T}} W^{\mathrm{T}} X\} + \theta^{\mathrm{T}} X_{\mathrm{shift}}[I], \tag{27}$$

where X_{shift} is a shift column version of the matrix X, which contains the pattern vector X^{k+1} in its kth column. Note that the learning of the weights and θ derived from these two energies can sense a pattern that is in the wrong division of the hyperplane and improve this hyperplane. EAM and the method in (Liou and Yuan, 1999) possess such energy functions, E_{Hebbian} and E_{TAM}. The extreme value of E_{Hebbian} is less than or equal to $-NP$ for the weights in the proposed method. In most cases the value is in between $-NP$ and $-N\sqrt{N}P$.

Their trained weights, w_{ij} and θ_i, can be directly used in many other neural models.

Note that one can also enlarge the basin-1 by modifying the ECR. The energy function for the modified rule is

$$E_{\mathrm{ECR}}(W,\theta) = \sum_{k=1}^{P}\sum_{i=1}^{N}(X_i^k - v_i(W_i,\theta_i))^2$$
$$= \mathrm{trace}\{(X - V(W,\theta)[I]^{\mathrm{T}})^{\mathrm{T}}(X - V(W,\theta)[I]^{\mathrm{T}})\}, \quad (28)$$

where we use the sigmoid function instead of the "sgn" function in ECR. One may use the modified rule to decrease this energy to its extremes and enlarge the basin-1. We summarize in the following several important issues related to the proposed method.

5.1. *Energy function and hairy model*

The energy function for the evolution of weights, (7, 10, 11), is

$$E(W,\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{P}(s_i^k)^2 = -\sum_{k=1}^{P}[W^{\mathrm{T}}X^k - \theta]^{\mathrm{T}} \cdot [W^{\mathrm{T}}X^k - \theta]$$
$$= -\mathrm{trace}\{(W^{\mathrm{T}}X - \theta[I]^{\mathrm{T}})^{\mathrm{T}}(W^{\mathrm{T}}X - \theta[I]^{\mathrm{T}})\}. \quad (29)$$

This energy is different from E_{Hebbian}. Since the summation is accumulated over all the patterns in the above equations, $\sum_{k=1}^{P}$, one can develop a random sequential mode for EAM and the temporal AM with respect to each individual pattern. EAM searches the extreme on the landscape of $E(W,\theta)$ in a gradient manner while keeping all the patterns stable. The method contains skills needed to reach this extreme stably by maximizing the minimum of $\{s_i^k, k = 1,\ldots,P\}$. Note that there is no guarantee for any gradient based method to reach this extreme without the skills. The method in (Liou and Yuan, 1999) also uses all the patterns as a whole and tunes the weights according to the worst patterns. They all refine Hebb's postulate in order to obtain large basins. The extreme value of $E(W,\theta)$ is less than or equal to $-NP$. In most cases the value is in between $-NP$ and $-N^2P$ using EAM.

In almost all simulations, the evolution of states converged in a single iteration during recall after training. This is very different from the evolutionary recall in many other models. The energy function for recall evolution, like that of HM, is

$$E(V) = -[V^{\mathrm{T}}\mathtt{T}^{\mathrm{T}}V] + \theta^{\mathrm{T}}V, \tag{30}$$

where $\mathtt{T}$ is a network matrix and its element $\mathtt{T}_{ij}$ is related to $w_{ij}, w_{ij} = \sigma_1(\mathtt{T}_{ij})$. σ_1 is a sigmoid function, and

$$w_{ij} = \sigma_1(\mathtt{T}_{ij}), \qquad \text{and} \qquad \frac{\mathrm{d}w_{ij}}{\mathrm{d}\mathtt{T}_{ij}} \geq 0. \tag{31}$$

This sigmoid function, σ_1, can be replaced by a hard-limiting activation function to facilitate operation in discrete mode.

Combining the two kinds of energy functions, (29, 30), together, we obtain a global energy function, $E(V, W, \theta) = E(W, \theta) + E(V)$, where $E(W, \theta)$ can be replaced by any other energy function, i.e., $\{E_{\mathrm{Hebbian}}(W, \theta), E_{\mathrm{TAM}}(W, \theta),$ or $E_{\mathrm{ECR}}(W, \theta)\}$. Note that we set

$$w_{ij} = \sigma_1(\mathtt{T}_{ij}) = \mathtt{T}_{ij} \tag{32}$$

in all simulations. Let

$$v_i = \sigma_2\{\mathtt{T}V - \theta\} = \sigma_2\{u_i\}, \tag{33}$$

where σ_2 is also a sigmoid function (or a nondecreasing function) and $\frac{\mathrm{d}v_i}{\mathrm{d}u_i} \geq 0$. $E(V, W, \theta)$ constitutes a hairy model. See one example of this hairy model in (Liou and Wu, 1996). The hairy model possesses a global energy function for the evolution of both weights and states. This energy will give two dynamic equations, which are used to evolve both the weights and the states. They are

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{\partial E(V, W, \theta)}{\partial v_i} \qquad \text{and} \qquad \frac{\mathrm{d}\mathtt{T}_{ij}}{\mathrm{d}t} = -\frac{\partial E(V, W, \theta)}{\partial w_{ij}}. \tag{34}$$

The network will employ these two dynamic equations to reduce the global energy. These two dynamic equations reinforce each other to stablize the memories and tolerate contamination. It has the homeostatic quality as a self-regulating network to resist both minimal perturbations and structural perturbations (Kauffman, 1991). The hairy model plasticizes a neural network toward its goal. It possessses all of the merits of ECR and HM. It is the deterministic analogue of Gibb's free energy model (Szu, 1989, 1999). The proposed EAM is useful for plasticizing the discrete Turing's B-type unorganized machine in (Ince, 1992) as an associative memory.

5.2. *Convergence and stability*

The EAM operates in batch (or collective) mode. The EAM can be rewritten to operate in sequential mode. The EAM operates in a very conservative manner. When the network starts with a stable pattern, it remains stable with an enlarged basin-1 iteration after iteration. The EAM will stop when the basin-1 cannot be enlarged. One may stop the iteration at any time without damaging the stability of the pattern.

When the network starts with an incorrect pattern transition, the temporal AM improves the worst (most vulnerable) transition among all the transitions for each bit sequentially. There is no guarantee that a correct transition can be achieved if one sets an incorrect transition at the beginning.

5.3. *Capacity*

The EAM can store more than N patterns or transitions. This number has been used in many modern designs (Kanter and Sompolinsky, 1987). The EAM can accomodate stable patterns up to the network limit, 2^N.

5.4. *Computation cost*

The EAM operates in one shift. Each hyperplane is adjusted in turn. Each iteration improves the location of a hyperplane. The computational cost is linearly proportional to the network size, N, and the number of patterns, P.

5.5. *Learning parameters*

The learning parameters in EAM should be chosen carefully. When the number of neurons N is large, the distance from the corners to the origin, $\sqrt{N}$, is also large. A little adjustment in the direction W_i of a hyperplane will cause many corners to move from a positive division to a negative one or vice versa. Therefore, the learning rate should not too large in the case of a large N. The rate would be better if one sets it a value proportional to $1/\sqrt{N}$.

5.6. *Training with noisy patterns*

When there are P classes of noisy patterns, one may apply the EAM to adjust the hyperplanes such that an entire class will be isolated in the same division. All the patterns that belong to a class will be used in a batch during a training iteration. All the distances, s_i^k, that belong to a class will be calculated in each iteration, and the minimum distance (or the worst one) should be used as the distance between this class and the hyperplane. The center of a class will be the only stable corner in this class division enclosed in basin-1. This center may not be a training pattern. This kind of center is a cluster center in some sense close to that by the Hamming distance. This center is extremely useful for representing a class and restoring noisy images. The EAM can be used in place of the annealing techniques to achieve tolerance for stored patterns with less computation.

5.7. *Multilayer perceptrons, recurrent network, and Boltzmann machine*

The weights obtained using the EAM have been used in multilayer perceptions and recurrent networks between two connected layers (Liou and Yu, 1995) to accomplish various goals successfully, such as error tolerance, associative memory, and replicator. To directly use the trained weights, the two connected layers must have the same number of neurons. Since the EAM operates neuron by neuron, it can be slightly modified for two connected layers with different numbers of neurons. The modified method is similar to that for the temproal AM. The weights have also been used in Boltzmann machine and mean-field theory to accomplish similar goals.

5.8. *Summary*

This work has proposed a method to explore the flexible space in between two pattern sets, $\{X^k; k = 1, \ldots, P, \text{and } X_i^k = 1\}$; $\{X^k; k = 1, \ldots, P, \text{ and } X_i^k = -1\}$, so as to locate the decision hyperplane so that it is consistent with the dichotomy inequality, (5). This is done by enlarging the decision division basin-1 to include as many neighboring noisy patterns as possible for each stable pattern. This enlargement indirectly improves the entire basin in a chain-reaction-like manner and gives better tolerance. This work has shown its superior performance

based on the results of massive random simulations. We highly recommend to use the weights given in (14) in many networks.

The EAM derives asymmetric matrices and nonzero diagonal elements, and still keep the arguments of Hebb's postulate. Also, each hyperplane is independent of all others during training. This is very different from the correlation matrix designs, which are based on the pseudo-inverse approach.

The EAM can be used for sparsely-connected networks in a way similar to that for the fully-connected networks. In a sparsely-connected network one set $\{w_{ij}\ (t) = 0\}$ for all iterations for those unconnected neurons and set the initial weights as in (13). The method then operate the EAM to tune the connected weights only. Note that a hyperplane which has many zero weights, $\{w_{ij}\ (t) = 0\}$, preserves similar properties of stability as that with the weights in (13).

This work has devised the required energy function (29) and dynamics, built them as standard neural models, and fit them as the hairy model, which has a global energy function, $E(V, W, \theta)$, and two dynamic equations (34). To our knowledge, this is the only hairy model directly constructed so far.

Finally, we would like to emphasize that the arguments of the EAM can be equally applied to the spin glass model in (Hopfield, 1982) and its stochastic version. Also, when one applies the EAM to a subset of neurons of a network, these subset neurons act as a whole and form stable local memories. These local memories serve to the whole network as multiple and temporal references. This is contrary to the frozen island in random boolean network in (Kauffman, 1991) which serves as a fixed rigid reference (backbone) to the network. One can train these subset neurons as a communicating pathway embedding in the network. One may train the interconnected weights between two subsets to build a potential pathway between subset memories similar to the bidirectional model. One can easily build an attentional subsystem for the network based on these subset neurons to implement selective attention tasks.

Acknowledgements

This work was supported by the National Science Council under projects NSC 93-2213-E-002-081 and NSC 81-0404-E-002-21. Thanks also go to Mr. S.-K. Yuan, who carried out computer experiments and produced figures.

References

Ackley DH, Hinton GE and Sejnowski TJ (1985) A learning algorithm for Boltzmann machine. Cognitive Science 9: 147–169

Amari SI (1972) Learning patterns and pattern sequences by self-organising nets. IEEE Transactions on Computers 21: 1197–1206

Boser B, Guyon I and Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Fifth Annual Workshop on Computational Learning Theory, Morgan Kaufmann, pp. 144–152

Bruck J (1990) On the convergence properties of the Hopfield model. Proceeding of IEEE 78: 1579–1585

Cover TM (1965) Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Transactions on Electronic Computers 14: 326–334

Gardner E (1987) Maximum storage capacity in neural networks. Electrophysics Letters 4(4): 481–485

Gardner E (1989) Optimal basins of attraction in randomly sparse neural network models. Journal of Physics A 22(12): 1969–1974

Hebb DO (1949) The Organization of Behavior: A Neuropsychological Theory. Wiley, New York

Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational ability. Proceeding of the National Academy of Science 79: 2554–2558

Ince DC (1992) Intelligent machinery. In: Ince DC (ed) Collected Works of A. M. Turing: Mechanical Intelligence. Elsevier Science Publishers

Kanter I and Sompolinsky H (1987) Associative recall of memory without errors. Physics Review A 35(1): 380–392

Kauffman SA (1991) Antichaos and adaptation. Scientific American: 64–70

Li J, Michel AN and Porod W (1989) Analysis and synthesis of a class of neural networks: linear systems operating on a closed hypercube. IEEE Transactions on Circuits and Systems 36(11): 1405–1422

Liou CY and Lin SL (1989) The other variant Boltzmann machine. In: Proceedings of the IJCNN, Washington DC, pp. 449–454

Liou CY and Sou UC (2003) Loading temporal associative memory using the neuronic equation. In: Kaynak O, Alpaydin E, Oja E and Xu L (eds), LCS, vol. 2714, Springer, pp. 52–59

Liou CY and Wu JM (1996) Self-organization using Potts models. Neural Networks 9(4): 671–684

Liou CY and Yu WJ (1995) Ambiguous binary representation in multilayer neural network. In: Proceedings of the ICNN, Perth, Australia, vol. 1, pp. 379–384

Liou CY and Yuan SK (1999) Error tolerant associative memory. Biological Cybernetics 81: 331–342

Little WA (1974) The existence of persistent states in the brain. Mathematical Biosciences 19: 101–120

Mceliece RJ, Posner EC, Rodemich ER and Venkatesh SS (1987) The capacity of the Hopfield associative memory. IEEE Transaction on information Theory 33: 461–482

Szu H (1989) Reconfigurable neural nets by energy convergence learning principle based on extended McCulloch–Pitts neurons and synapses. In: Proc IJCNN, Washington, DC, vol. 1, pp. 485–496

Szu H (1999) Thermodynamics energy for both supervised and unsupervised learning neural nets at a constant temperature. International Journal of Neural System 9: 175–186

Tao Q, Fang T and Qiao H (2001) A novel continuous-time neural network for realizing associative memory. IEEE Transactions on Neural Networks 12(2): 418–423

Widrow B and Hoff ME Jr. (1960) Adaptive switching circuits. In: IRE WESCON Convention Record, pp. 96–104

Wilde PD (1997) The magnitude of the diagonal elements in neural networks. Neural Networks 10(3): 499–504