Pattern Retrieval and Learning in Nets of Asynchronous Binary Threshold Elements

MOSHE KAM, MEMBER, IEEE, ROGER CHENG, AND ALLON GUEZ

Abstract—We study the state space of a popular network of asynchronous multi-connected linear threshold elements. The properties of the state space are analyzed during a learning process: the network learns a set of patterns which appear in its environment in a random sequence. The patterns influence the network's weights and thresholds through an adaptive algorithm, which is based on the Hebbian hypothesis. The algorithm tries to install the patterns as fixed points in the network's state space, and to guarantee that a large region of attraction surrounds each fixed point. We obtain the stabilization probabilities of each pattern in the learned set, as well as the stabilization rate, as a function of the training time. In addition, we obtain a lower bound on the probability of convergence to any stored pattern, from an initial state at a given Hamming distance from it. A specific case of our training algorithm is the widely used, nonadaptive sum-of-outer-products parameter assignment. Properties of networks with this assignment can, therefore, be evaluated and compared to properties that are obtained under the adaptive training, which is more suited to the pattern environment. Our derivation allows the evaluation of the quality of information storage for a given set of patterns, and the comparison of different information-coding schemes for items that need to be stored and retrieved from the network. Also evaluated are the steady-state values of the network's parameters, following a long training with a stationary set of patterns. Finally, we study the differences between networks that were trained by "hard" and "soft" limiter learning curves.

I. Introduction

NETWORKS OF threshold elements have been the subject of much interest due to the remarkable properties of learning, content-addressable memory and pattern recognition that they exhibit, e.g., [2], [13], [8]–[10], [3], [4]. The recent resurgence of interest in neural networks is also tied to the development of new realizations and fabrication techniques for these large-scale systems (see a collection of papers in [18]). A considerable effort has been invested towards the study of the networks' dynamics [17], stable states [14], the memory storage capacity [1], [14], and conditions on the input patterns which guarantee successful storage [6]. If the net is to be useful for signal processing in an arbitrary, changing environment, there are two pre-requisites: the network must be *adaptive*, to reflect changes in the environment that it learns; and its properties should be predictable for ***any*** arbitrary set of patterns whose storage is attempted. In this paper, we study the dynamical and steady-state behavior of an adaptive network of binary threshold elements (="binary neurons".) The neurons are random in the way that they learn their environment: each neuron updates its weights and threshold at random times, independently of all other neurons. The environment is random: the set of learned patterns appears randomly in the network's environment. Finally, the operation (or "production") is random: each neuron reassesses its (binary) activity level at random times, asynchronously and independently of all other neurons. The probability density functions of the network (during learning and production) are assumed to be known. The probability distribution of the environment is unknown to the network.

A binary network of N neurons operates in a 2^N point N-cube state space, describing all the possible combinations of the activity levels. Some of the 2^N points are transient and some are fixed; some may belong to ordered paths, such as stable limit cycles, while others belong to chaotic paths. For pedagogical reasons, we describe the operation of the network in two distinct phases: the *learning phase*, and the *production phase*. During the learning phase the network probes the environment, attempting to change the neurons' parameters (weights and thresholds) to favor the installment of the observed patterns as stable states. The stable states are to be recovered later, in the production phase, from partial or erroneous versions of these patterns. During the production phase, the network is presented with a "probing" binary tuple pertaining to one of the stored patterns. Using this tuple as an initial state, the network changes its state until it settles in one of the fixed points. This stable state is to be "close" in some sense to the presented tuple; assuming successful operation, the network recovers the "correct" pattern, whose distorted version was initially presented. Throughout this paper we assume that the two phases are interlaced in time, but are non-concurrent. In an actual network these

Manuscript received September 22, 1987; revised May 9, 1988. This work was supported by the National Science Foundation under Grant IRI-88101186 and under Grant DMCE-8505235. This paper was recommended by Associate Editor M. Ilic.

M. Kam and A. Guez are with the Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104.

R. Cheng is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544.

IEEE Log Number 8825760.

0098-4094/89/0300-0353$01.00 ©1989 IEEE

phases can be intertwined, and our model is indeed capable of learning as it "produces."

The training algorithm, used during the learning phase, is based on the Hebbian hypothesis [7, p. 62] and is designed to allow flexible adjustment of the network to environments with changing patterns. A network designer would like to know what are the ultimate pattern retrieval probabilities that the network may achieve under this training procedure; in addition—how long should the learning period last, in order to guarantee a pre-specified desirable performance. We find the *probability of stabilization* for each input pattern versus training time, and develope a lower bound on the *probabilities of convergence* to the stabilized patterns from initial states at various Hamming distances from them. The derived probabilities allow an assessment of the network's learning rate, and indeed determine the feasibility of achieving the pre-specified performance, and the duration of training. In addition, they determine the necessity and effectiveness of applying pre-processing to the input pattern set: different codes for the information to be stored yield different stabilization and convergence properties. An interesting specific case of our training procedure is the sum-of-outer-products assignment. The design with this method is compared to the more general, adaptive Hebbian procedure.

We further find the steady-state statistics of the neurons' parameters, assuming that there are bounds on the largest absolute values of their magnitudes. The resulting design gives the average performance of the network in a static pattern environment. Finally, we examine the network's learning curve, and compare the steady state properties of networks trained with a sigmoidal, "soft" limiter learning curve, to networks trained with a "hard" limiter learning curve. To develop the results we use the theory of Markov chains, in particular, birth-and-death processes (e.g., [11]). We show that the synaptic weights, the thresholds and the stability margins of stored patterns can be interpreted as Markov processes over finite sets.

After a presentation of the notation, Section III defines the model and the "hard" and "soft" limiter Hebian learning rules which are used to train the network. In Section IV we analyze the dynamics of the network, assuming that the parameters are not bounded during the learning phase. We quantify stabilizability and attraction properties for the stored patterns versus training time, and determine the necessary training period. Section V presents analysis of the network's parameters in steady state, with bounds on the absolute values of the network's parameters (obtained from training-period considerations.) We compare the adaptive-training scheme to the fixed-parameter sum-of-outer-products algorithm, and examine the consequences of replacing the "hard" limiter learning curve by the "soft", sigmoidal limiter.

II. Notation

A symbol key is provided in alphabetical order. Some quantities are defined in the text when they appear for the first time, to preserve contextual continuity.

$(\ )^{(n_L)}$	Value after the n_Lth *learning* (= training) step.
$(\ )^{(n_p)}$	Value after the n_pth production-phase *state-reassessment* step
$\boldsymbol{B}$	$\{1, -1\}$.
$b_{kj} \in \boldsymbol{B}$	The jth element of the pattern B_k.
B_k	The kth pattern $B_k = [b_{k1}, b_{k2}, \cdots, b_{kN}]$.
$C(n, \lambda)$	The binomial coefficient, $n!/[\lambda!(n-\lambda)!]$.
I_i	A real-valued input of the ith neuron; except for forcing an initial value on u_i, $I_i = 0$.
Γ	A $1 \times (N+1)$ probability vector of a birth-and-death $N+1$-state Markov process.
Γ_i	ith component of Γ.
$\Delta(B_M, 0)$	Rate of stabilization of pattern B_M.
$\Delta(B_M, k)$	Rate of growth for the *lower bound on the contraction probability* towards the pattern B_M, from an initial state at Hamming distance k from B_M.
$\Delta S(B_M, B_Q, i)^{(\lambda_L)}$	The change in the stability margin S_{Mi} from the λ_Lth training step. to the (λ_L+1)th if pattern B_Q is taught at the λ_Lth training step.
$\boldsymbol{E}(x)$	Expected value of x.
$\mathrm{erf}(x)$	The error function, $1/\sqrt{2\pi}\ \int_0^x \exp(-(t^2/2))\, dt$.
HD	Hamming distance.
L	An upper bound on the absolute value of the synaptic weights, w_{ij}.
$\vartheta_k(n_L, B_M)$	A probability related to the contraction probability to a stable state (B_M), from an initial state at a given Hamming distance (k) from it.
$\mathrm{pdf}(x)$	Probability density function of x.
$\pi(d, B_{M,i})$	Probability that the stability margin S_{Mi} is increased by d.
Π_i	Probability of occurrence of the ith pattern in the environment.
$\boldsymbol{P}_i^+$	Probability that the threshold τ_i is drifted downwards to $\tau_i - 1$, when neuron i updates its parameters.
$\boldsymbol{P}_i^-$	Probability that the threshold τ_i is drifted upwards to $\tau_i + 1$, when neuron i updates its parameters.
$\boldsymbol{P}_{ij}^+$	Probability that the weight w_{ij} is drifted upwards to $w_{ij} + 1$, when neuron i updates its parameters.
$\boldsymbol{P}_{ij}^-$	Probability that the weight w_{ij} is drifted downwards to $w_{ij} - 1$, when neuron i updates its parameters.
$\boldsymbol{P}_r(x)$	Probability of x.
$\boldsymbol{Q}_{ij}$	The weight drifting rate, $\boldsymbol{P}_{ij}^+/\boldsymbol{P}_{ij}^-$.
$\boldsymbol{Q}_i$	The threshold drifting rate, $\boldsymbol{P}_i^-/\boldsymbol{P}_i^+$.
ρ_i	Probability that the ith neuron reassesses its activity level during the

	production phase = probability that the ith neuron updates its weights and thresholds during the learning phase.
$\mathrm{sgn}(x)$	The algebraic sign of x.
$\boldsymbol{\Psi_{bd}}$	The state-transition matrix of a birth-and-death Markov process.
$\boldsymbol{S_{Mi}^{(\lambda_L)}}$	Stability margin of the ith neuron with respect to pattern B_M, at the λ_Lth training step of neuron i.
T	An upper bound on the absolute value of the thresholds, τ_i.
$\tau_i \in \boldsymbol{R}$	Threshold of the ith neuron.
$\boldsymbol{U}$	An $N \times 1$ state-vector of the network $\boldsymbol{U} \in \boldsymbol{B}^N$.
$u_i \in \boldsymbol{B}$	Activity level (or "state") of the ith neuron.
$w_{ij} \in \boldsymbol{R}$	Strength of the synaptic weight from the jth neuron to the ith.

III. Nets of Random Threshold Elements

We consider a multi-connected network of N linear-threshold elements (also called binary neurons.) Using terms which are borrowed from simplistic models of the brain, the ith neuron is characterized by $N-1$ real numbers representing the synaptic *weights* (w_{ij}, $j=1,2,\cdots, i-1, i+1,\cdots, N$; w_{ii} is assumed to be zero for all i), a real *threshold* (τ_i) and a binary *activity level* ($u_i \in \boldsymbol{B}$,) which we shall also refer to as the neuron's *state*. At random times during the production phase, asynchronously and independently of all other neurons, the ith neuron reassesses its state, according to the rule

$$u_i = \mathrm{sgn}\left(\sum_{j=1,\ j\neq i}^{N} w_{ij}u_j - \tau_i + I_i \right). \tag{1}$$

The term I_i in (1) represents an external real-valued input, which in the present paper will assume nonzero values only when used to force an initial value on u_i ($i=1,2,\cdots,n$). This initial value is the "key" or "initial probe vector" [14] which is necessary in order to retrieve information from the network.

In order to allow a more convenient representation of the asynchronous operation, we shall count the reassessment steps of the network, using the symbol n_p. The count will increase by 1 every time any neuron reassesses its state. Let $u_i^{(n_p)}$ indicate the state of the ith neuron after n_p productions steps. Then (1) can be modified to

$$u_i^{(n_p+1)} = \begin{cases} u_i^{(n_p)}, & \text{with probability } \Gamma_i^{(n_p)} \\ -u_i^{(n_p)}, & \text{with probability } 1-\Gamma_i^{(n_p)} \end{cases} \tag{2}$$

where

$$\Gamma_i^{(n_p)} = 1 - \rho_i^{(n_p)} \boldsymbol{P_r}\left[\mathrm{sgn}\left(\sum_{j=1,\ j\neq i}^{N} w_{ij}u_j^{(n_p)} - \tau_i \right) = -u_i^{(n_p)} \right]$$

and $\rho_i^{(n_p)}$ is the probability that the ith neuron reassesses its state at the n_pth production step. The probabilities $\rho_i^{(n_p)}$, $i=1,\cdots,N$ are assumed stationary ($\rho_i^{(n_p)} = \rho_i$.)

The weights and thresholds used in (2) have been determined during a prior learning period when w_{ij} and τ_i were trained to encourage the storage of a set of patterns $B_1,\cdots,B_R$ as *stable states* of the network. A pattern $B_k = [b_{k1}, b_{k2},\cdots, b_{kN}]$ is assumed to appear in the environment that the network studies with the stationary probability Π_k, ($\sum_{i=1}^{R}\Pi_i = 1$). It is a *stable state* of the network, if $u_i^{(m_p)} = b_{ki} \Rightarrow u_i^{(n_p)} = b_{ki}$ for all $n_p > m_p$, $i=1,\cdots,N$. In order to study the stabilizability of a set of patterns, we introduce *the stability margin of the* ith *neuron with respect to pattern* B_M

$$\boldsymbol{S_{Mi}} = b_{Mi}\left(\sum_{j=1,\ j\neq i}^{N} w_{ij}b_{Mj} - \tau_i \right). \tag{3}$$

The pattern is *stable* if

$$\boldsymbol{S_{Mi}} > 0, \qquad i=1,\cdots,N \tag{4a}$$

and it is *stable with probability* q if

$$\sum_{i=1}^{N} \rho_i \boldsymbol{P_r}(\boldsymbol{S_{Mi}} > 0) = q. \tag{4b}$$

Learning Rules:

a) Hard limiter: To stabilize a set of patterns $B_1, B_2,\cdots, B_R$ in the network's state space, we adopt an adaptive training procedure for updating the weights and the thresholds during the learning phase. The procedure is inspired by the biological arguments of Hebb ([7]) and their adaptation, with a limiter-type learning curve, by Rosenblatt [19] in his training procedure for the C' perceptron (see also [20].)

The basic principle is that when b_{kj}, the jth binary digit of the learned pattern B_k is the same as the intended output of the ith neuron, namely b_{ki}, ($b_{ki} = b_{kj}$,) then w_{ij} will be increased by a positive increment (say 1,) introducing positive reinforcement. If $b_{ki} \neq b_{kj}$, w_{ij} is decreased, to supply negative reinforcement. Similarly, the threshold will be lowered by 1 if we wish to achieve $u_i = b_{ki} = 1$, and increased by 1 if we need $u_i = b_{ki} = -1$. An exception to the rule occurs if the training rule resulted in a weight or a threshold exceeding, in absolute value, some pre-specified limits (denoted L for the weights, T for the thresholds.) In a similar manner to that of the production phase operation, we shall perform weight and threshold updating in a random manner during the learning phase: the patterns $B_1,\cdots,B_R$ appear in the environment with probabilities $\Pi_1,\cdots,\Pi_R$, respectively. Asynchronously, at random times, independently of all other neurons and the input patterns, each neuron updates its N real parameters. We shall count the learning steps, using the symbol n_L. n_L is increased by 1 every time any neuron updates its parameters. If neuron i updates its parameters after n_L steps, and

if at that time pattern k appears in the learned environment, then the updating will follow the rule:

$$w_{ij}^{(n_L+1)} = \begin{cases} w_{ij}^{(n_L)}+1, & \text{if } b_{ki}b_{kj}=1 \text{ and } w_{ij}^{(n_L)} \neq L \\ w_{ij}^{(n_L)}-1, & \text{if } b_{ki}b_{kj}=-1 \text{ and } w_{ij}^{(n_L)} \neq -L \\ w_{ij}^{(n_L)}, & \text{if } b_{ki}b_{kj}=1 \text{ and } w_{ij}^{(n_L)}=L \\ w_{ij}^{(n_L)}, & \text{if } b_{ki}b_{kj}=-1 \text{ and } w_{ij}^{(n_L)}=-L \end{cases}$$

$$\tau_i^{(n_L+1)} = \begin{cases} \tau_i^{(n_L)}+1, & \text{if } b_{ki}=-1 \text{ and } \tau_i^{(n_L)} \neq T \\ \tau_i^{(n_L)}-1, & \text{if } b_{ki}=1 \text{ and } \tau_i^{(n_L)} \neq -T \\ \tau_i^{(n_L)}, & \text{if } b_{ki}=-1 \text{ and } \tau_i^{(n_L)}=T \\ \tau_i^{(n_L)}, & \text{if } b_{ki}=1 \text{ and } \tau_i^{(n_L)}=-T. \end{cases} \quad (5)$$

b) Soft limiter: The tuning of the weights and the thresholds according to (5) depends only on the properties of the taught patterns (as long as the network's parameters are not close to their limits.) In certain circumstances, there are advantages to introducing the network's experience in a more gradual manner during the learning process; this is accomplished by using the "present" values, $w_{ij}^{(n_L)}$ and $\tau_i^{(n_L)}$, to decide upon the "future" values, $w_{ij}^{(n_L+1)}$ and $\tau_i^{(n_L+1)}$. If the network learns in a noisy environment, particularly when short bursts of noise occasionally corrupt the learned patterns, the network's experience, as represented by $w_{ij}^{(n_L)}$ and $\tau_i^{(n_L)}$ could be used to reduce the effects of erroneous updating. Once a weight has been trained to move substantially from the origin ($w_{ij}^{(n_L)} \gg 0$,) it should 'resist' moving back, unless there is ample evidence that the pattern environment has changed to warrant such a move. When a genuine change in the environment is detected, the movement towards a new steady state is to be accelerated. This recipe is one way to cope with the tradeoff between the contradicting objectives of plasticity and noise-immunity in the network design (see discussion in [5].) To facilitate this kind of network behavior we suggest using a nonlinear sigmoidal transformation on *support variables*, which are trained by the hard-limiter Hebbian rule (5). Let $r_{ij}^{(n_L)}$ and $q_i^{(n_L)}$ be the support variables for the weights and the thresholds, respectively. As a sigmoidal nonlinearity we shall use the hyperbolic tangent. The training will follow the rules:

For weights:

$$r_{ij}^{(n_L)} = r_0 \tanh^{-1}\left[\frac{w_{ij}^{(n_L)}}{L}\right] \qquad r_0\text{: a normalization factor} \quad (6a)$$

$$r_{ij}^{(n_L+1)} = \begin{cases} r_{ij}^{(n_L)}+1, & \text{if } b_{ki}b_{kj}=1 \text{ and } r_{ij}^{(n_L)} \neq L \\ r_{ij}^{(n_L)}-1, & \text{if } b_{ki}b_{kj}=-1 \text{ and } r_{ij}^{(n_L)} \neq -L \\ r_{ij}^{(n_L)}, & \text{if } b_{ki}b_{kj}=1 \text{ and } r_{ij}^{(n_L)}=L \\ r_{ij}^{(n_L)}, & \text{if } b_{ki}b_{kj}=-1 \text{ and } r_{ij}^{(n_L)}=-L \end{cases} \quad (6b)$$

$$w_{ij}^{(n_L+1)} = L \tanh\left[\frac{r_{ij}^{(n_L+1)}}{r_0}\right] \quad (6c)$$

For thresholds:

$$q_i^{(n_L)} = q_0 \tanh^{-1}\left[\frac{\tau_i^{(n_L)}}{T}\right] \qquad q_0\text{: a normalization factor} \quad (6d)$$

$$q_i^{(n_L+1)} = \begin{cases} q_i^{(n_L)}+1, & \text{if } b_{ki}=-1 \text{ and } q_i^{(n_L)} \neq T \\ q_i^{(n_L)}-1, & \text{if } b_{ki}=1 \text{ and } q_i^{(n_L)} \neq -T \\ q_i^{(n_L)}, & \text{if } b_{ki}=-1 \text{ and } q_i^{(n_L)}=T \\ q_i^{(n_L)}, & \text{if } b_{ki}=1 \text{ and } q_i^{(n_L)}=-T \end{cases} \quad (6e)$$

$$\tau_i^{(n_L+1)} = T \tanh\left[\frac{q_i^{(n_L+1)}}{q_0}\right]. \quad (6f)$$

The normalization factors r_0 and q_0 control the tradeoff between noise immunity (small normalization factors,) and response speed (large normalization factors.) In the sequel we shall present the analysis for the simpler rule (5). When the application of the more complicated rule (6) is necessary, the weight and threshold variables (w and τ) are replaced by the support variables (r and q) and the analysis of the network's performance under the rule (6) proceeds in the same way that the analysis of (5) is performed.

Transition probabilities: We introduce $\boldsymbol{P}_{ij}^{+}$, the probability that the patterns shown to the network in the learning phase (using (5)) have the same signs in their ith and jth positions and $\boldsymbol{P}_{ij}^{-}$, the probability that these signs are different:

$$\boldsymbol{P}_{ij}^{+} = \sum_{r|b_{ri}b_{rj}=1} \Pi_r \quad (7a)$$

$$\boldsymbol{P}_{ij}^{-} = \sum_{r|b_{ri}b_{rj}=-1} \Pi_r. \quad (7b)$$

Let $\boldsymbol{P}_i^{+}$ represent the probability that the patterns shown to the network in the learning phase have a "1" in the ith position; $\boldsymbol{P}_i^{-}$ will represent the probability of a "-1" there:

$$\boldsymbol{P}_i^{+} = \sum_{r|b_{ri}=1} \Pi_r \quad (7c)$$

$$\boldsymbol{P}_i^{-} = \sum_{r|b_{ri}=-1} \Pi_r. \quad (7d)$$

To get the general transition probabilities for the network's parameters, we take into account the pattern statistics to

obtain equation (8):

Weight Transitions

A) For $|w_{ij}^{(n_L)}| < L$

$$w_{ij}^{(n_L+1)} = \begin{cases} w_{ij}^{(n_L)} + 1 & \text{with probability } \rho_i P_{ij}^{+} \\ w_{ij}^{(n_L)} - 1 & \text{with probability } \rho_i P_{ij}^{-} \\ w_{ij}^{(n_L)} & \text{with probability } 1 - \rho_i \end{cases}$$

B) For $w_{ij}^{(n_L)} = -L$

$$w_{ij}^{(n_L+1)} = \begin{cases} w_{ij}^{(n_L)} & \text{with probability } 1 - \rho_i P_{ij}^{+} \\ w_{ij}^{(n_L)} + 1 & \text{with probability } \rho_i P_{ij}^{+} \end{cases}$$

C) For $w_{ij}^{(n_L)} = L$

$$w_{ij}^{(n_L+1)} = \begin{cases} w_{ij}^{(n_L)} & \text{with probability } 1 - \rho_i P_{ij}^{-} \\ w_{ij}^{(n_L)} - 1 & \text{with probability } \rho_i P_{ij}^{-} \end{cases}$$

Threshold Transitions

D) For $|\tau_i^{(n_L)}| < T$

$$\tau_i^{(n_L+1)} = \begin{cases} \tau_i^{(n_L)} - 1 & \text{with probability } \rho_i P_i^{+} \\ \tau_i^{(n_L)} + 1 & \text{with probability } \rho_i P_i^{-} \\ \tau_i^{(n_L)} & \text{with probability } 1 - \rho_i \end{cases}$$

E) For $\tau_i^{(n_L)} = -T$

$$\tau_i^{(n_L+1)} = \begin{cases} \tau_i^{(n_L)} & \text{with probability } 1 - \rho_i P_i^{-} \\ \tau_i^{(n_L)} + 1 & \text{with probability } \rho_i P_i^{-} \end{cases}$$

F) For $\tau_i^{(n_L)} = T$

$$\tau_i^{(n_L+1)} = \begin{cases} \tau_i^{(n_L)} & \text{with probability } 1 - \rho_i P_i^{+} \\ \tau_i^{(n_L)} - 1 & \text{with probability } \rho_i P_i^{+} \end{cases} \tag{8}$$

As (8) demonstrates, the learning curves for the hard-limiter rule are linear for $|w_{ij}^{(n_L)}| < L$ and $|\tau_i^{(n_L)}| < T$, and exhibit saturation otherwise. If the soft-limiter learning rule (6) is used, a more gradual saturation is achieved. We examine the difference in steady-state performance under the two different rules in Section V.

The assignment of parameters for the network using our learning algorithms is considerably more complicated than the commonly used fixed assignment via the sum-of-outer-products algorithm (e.g., [8].) The payoff for the complexity is the adaptation to a changing environment that our scheme manifests. The regions of attraction around stored patterns expand and contract according to the frequency of those patterns in the learned environment, and their time history [12]. In addition, we allow for patterns with non-stationary probability distribution in the environment, and for additional degrees of freedom in the design: the state reassessment probabilities are not equal for all neurons, and they may change with the pattern environment.

IV. Performance of the Network Under Adaptive Hebbian Training

We shall study the properties that the network attains as a result of continued training, involving a single set of learned patterns. The patterns appear in the learned environment with stationary known probability density functions. Several assumptions are made: we assume that the network's parameters do not saturate ($w_{ij} < L, \tau_{ij} < T$); once the length of the training period which is necessary in order to guarantee desirable performance has been determined, the appropriate limits L and T would be set. We also assume that before learning has commenced, $w_{ij}^{(0)} = 0$, $\tau_i^{(0)} = 0$, $i = 1, \cdots, N$; $j = 1, \cdots, N$; $i \neq j$.[1] This assumption will be removed later, when the learned pattern set is allowed to change during the training period.

[1] Technically we choose small random values with zero mean to guarantee that

$$P_r\left[\sum_{j=1,\, j\neq i}^{N} w_{ij}u_j - \tau_i = 0\right] = 0.$$

The network is taught to store R patterns. Referring to the stability margins S_{ki} $i = 1, \cdots, N$, $k = 1, \cdots, R$ (equation (3),) let the stability margin of neuron i with respect to the pattern B_M after n_L training steps be $S_{Mi}^{(n_L)} = S_0$. If, at the training step $n_L + 1$, the ith neuron learns the pattern B_Q, then $S_{Mi}^{(n_L+1)}$ will take on the value

$$S_{Mi}^{(n_L+1)} = S_0 + \Delta S(B_M, B_Q, i)^{(n_L)} \tag{9a}$$

where

$$\Delta S(B_M, B_Q, i)^{(n_L)} = b_{Mi}b_{Qi}\left[\sum_{j=1,\, j\neq i}^{N} b_{Mj}b_{Qj} + 1\right]. \tag{9b}$$

Whenever the ith neuron updates its weights, the addendum to the stability margin, $\Delta S(B_M, B_Q, i)^{(n_L)}$, lies between $-N$ and N. Due to the non-deterministic nature of the appearance of patterns in the learned environment, a set of stationary transition probabilities is obtained.

Let neuron i update its parameters at the $\lambda_L + 1$ training step, and let

$$P_r\left[S_{Mi}^{(\lambda_L+1)} = S_0 + d \,|\, S_{Mi}^{(\lambda_L)} = S_0\right] = \pi(d, B_M, i) \tag{10a}$$

where $d = -N, -N+1, \cdots, N-1, N$, and

$$\pi(d, B_M, i) = \sum_{k|\Delta S(B_M, B_k, i)^{(\lambda_L)} = d} \Pi_k. \tag{10b}$$

Equation (10) describes a Markov process on the integers. For successive λ_L parameter updates of neuron i, the probability density function of $S_{Mi}^{(\lambda_L)}$ is related to the probability density function of $S_{Mi}^{(1)}$ through the relation:

$$\mathbf{pdf}\left[S_{Mi}^{(\lambda_L)}\right] = \left\{\mathbf{pdf}\left[S_{Mi}^{(1)}\right]\right\}^{*\lambda_L} \tag{11a}$$

where

$$\begin{aligned}\mathbf{pdf}\left[S_{Mi}^{(1)}\right] = \{ &\cdots 0,0,0, \pi(-N, B_M, i), \\ &\pi(-N+1, B_M, i), \cdots, \pi(0, B_M, i), \cdots, \\ &\pi(N-1, B_M, i), \pi(N, B_M, i), 0, 0, \cdots\}\end{aligned} \tag{11b}$$

and the superscript $*^{\lambda_L}$ indicates the λ_L-fold autoconvolution of the sequence inside the braces. This autoconvolution can be calculated efficiently using the discrete Fourier transform, as applied to linear convolution [15, Sect. 3.8].

Equation (11) now allows the calculation of the probability that pattern B_M has been stabilized at the training step n_L:

$$\vartheta_0(n_L, B_M) = \boldsymbol{P}_r^{(n_L)}\{\boldsymbol{U}^{(n_p+1)} = B_M | \boldsymbol{U}^{(n_p)} = B_M\} = \sum_{i=1}^{N} \rho_i \sum_{\lambda_L=1}^{n_L} C(n_L, \lambda_L)\rho_i^{\lambda_L}(1-\rho_i)^{n_L-\lambda_L} \cdot \boldsymbol{P}_r[S_{Mi}^{(\lambda_L)} > 0]. \quad (12)$$

The term in braces reads: the state of the network in the production step n_p+1 is the pattern B_M, provided that the network was at B_M at the previous step. Equation (12) therefore represents the probability that B_M has been stabilized after the n_Lth training step. The term $C(n_L, \lambda_L)$ is the binomial coefficient. The term $C(n_L, \lambda_L)\rho_i^{\lambda_L} \cdot (1-\rho_i)^{n_L-\lambda_L}$ is the probability that the ith neuron has updated its parameters λ_L times during n_L training steps of the learning phase. The summation over i weighs the inner sum by the probabilities that each neuron reassesses its activity level during the production phase. We assume that the probability of parameter-updating during the learning phase (ρ_i) is equal to the probability of activity-level reassessment during the production phase.

From the probability density function of the stability margins, we can calculate a lower bound on the probability that an initial state, at a pre-specified Hamming distance from a state $\boldsymbol{U} = B_M$, will be attracted to B_M. At the λ_Lth training step of neuron i, the probability density function of its stability margin with respect to the pattern B_M is $[\mathbf{pdf}(S_{Mi}^{(1)})]^{*\lambda_L}$. No weight or threshold of i has exceeded λ_L in absolute value. Therefore, if all stability margins are greater than $2\lambda_L$, then no single change in the sign of a weight will "destabilize" the state: a single change means that the network assumes a state at a Hamming distance 1 from the stable state. The Hamming distance after the next (production-phase) step will be 1 or less. This observation means that any state, at Hamming distance of 1 from B_M, will eventually stabilize at B_M. If all margins are greater than $2k\lambda_L$, where k is an integer, then any state at HD k from B_M will eventually stabilize at B_M. We shall first present an expression similar to (12), which is related to the probability of attraction to B_M from HD k:

$$\vartheta_k(n_L, B_M) = \sum_{i=1}^{N} \rho_i \sum_{\lambda_L=1}^{n_L} C(n_L, \lambda_L)\rho_i^{\lambda_L}(1-\rho_i)^{n_L-\lambda_L} \cdot \boldsymbol{P}_r[S_{Mi}^{(\lambda_L)} > 2\lambda_L k], \qquad k = 1, 2, \cdots, N. \quad (13)$$

Using (13) we express four important bounds on probabilities after n_L training steps (for derivation, see the Appendix):

1) A lower bound on the *contraction probability*: the probability that, when a state is at HD k from a state B_M, it will move to HD $k-1$ from it, at the next production-phase state reassessment. This lower bound is

$$P_k^{\text{contr}}(n_L, B_M) = \frac{k}{N}\vartheta_{k-1}(n_L, B_M). \quad (14a)$$

ii) A lower bound on the *stalemate probability*: the probability that, when a state is at HD k from a state B_M, it will stay at HD k from it after the next production-phase state reassessment. This lower bound is

$$P_k^{\text{stal}}(n_L, B_M) = \frac{N-k}{N}\vartheta_k(n_L, B_M) + \frac{k}{N}[1-\vartheta_{k-1}(n_L, B_M)]. \quad (14b)$$

iii) An upper bound on the *divergence probability*: the probability that when a state is at HD k from a state B_M it will move to HD $k+1$ from it after the next production-phase state reassessment. This upper bound, achieved after n_L training steps, is

$$P_k^{\text{div}}(n_L, B_M) = 1 - \frac{N-k}{N}\vartheta_k(n_L, B_M) - \frac{k}{N}. \quad (14c)$$

iv) Occasionally, we use a lower bound on the non-divergence probability

$$P_k^{\text{non-div}}(n_L, B_M) = P_k^{\text{contr}}(n_L, B_M) + P_k^{\text{stal}}(n_L, B_M) = \frac{k}{N} + \frac{N-k}{N}\vartheta_k(n_L, B_M). \quad (14d)$$

Example: We study the behavior of a 20-neuron network, designed to store 5 patterns. Three different sets of patterns are examined:

Set 1 (mutually orthogonal patterns):

Pattern 1:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Pattern 2:	1	1	1	1	1	−1	−1	−1	−1	−1	1	1	1	1	1	−1	−1	−1	−1	−1
Pattern 3:	1	1	1	1	1	1	1	1	1	1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Pattern 4:	1	1	−1	−1	−1	−1	−1	1	1	1	1	1	1	−1	−1	−1	−1	−1	1	1
Pattern 5:	−1	1	−1	1	−1	1	−1	1	−1	1	−1	1	−1	−1	−1	1	−1	1		

(15a)

Set 2 (no intended order):

Pattern 1:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Pattern 2:	1	1	1	1	1	1	1	1	1	−1	1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Pattern 3:	1	1	1	1	1	1	1	1	1	1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
Pattern 4:	1	1	−1	−1	−1	−1	−1	1	1	1	1	1	1	−1	−1	−1	−1	−1	1	1
Pattern 5:	1	1	−1	−1	−1	−1	−1	1	1	1	1	1	1	−1	−1	−1	−1	−1	1	−1	1.

(15b)

Set 3 (patterns are pairwise at HD4 from each other):

Pattern 1:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Pattern 2:	−1	−1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	−1	−1
Pattern 3:	1	1	−1	−1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	−1	−1
Pattern 4:	1	1	1	1	−1	−1	1	1	1	1	1	1	1	1	1	1	1	1	−1	−1
Pattern 5:	1	1	1	1	1	1	−1	−1	1	1	1	1	1	1	1	1	1	1	−1	−1.

(15c)

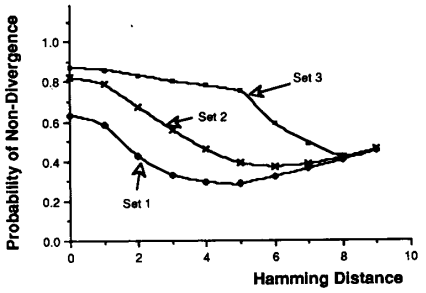


Fig. 1. Probability of non-divergence versus Hamming distance after a short training period (~ 5 steps per neuron).

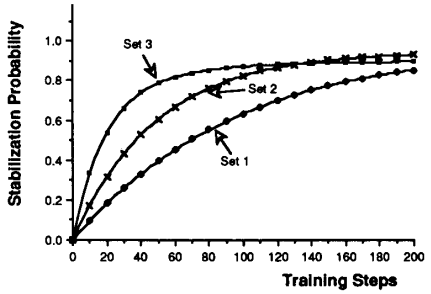


Fig. 2. Stabilization probability with Hebbian-training versus number of training steps.

The three sets represent three different coding schemes for five items of information that we want to store, and later retrieve, from the network. We shall use our stabilization results to compare the efficiency of these three codes. In the following, we shall compare efficiency with respect to a representative pattern, pattern 2 in each set.

In order to get a meaningful comparison with the sum-of-outer-products algorithm later, we shall assign equal probabilities of occurrence to the patterns ($\Pi_i = 0.2$), and equal state-reassessment and parameter-update probabilities to the neurons ($\rho_i = 0.05$.)

We compare the storage properties for pattern 2 in each one of the studied sets (see (15a)–(15c)). We examine the probability of stabilization, and the lower bound on the non-divergence probabilities. Fig. 1 shows these probabilities versus the Hamming distance between the network's states and the stored patterns. The training period represented in the figure is very short: 100 training steps are considered, meaning that, on the average, each neuron has been trained only 5 times. In Fig. 2 we examine the stabilization probability versus the number of training steps. The figures show that after 100 training steps the guaranteed performance of the pattern in set 3, as well as its learning rate, are superior to those of the two other patterns. In particular, we note that the learning of the pattern from the orthogonal set is the slowest, and that, unlike the set-3 pattern, whose learning rate seems to have saturated, it may benefit from continued training. In Fig. 1 we also observe the onset of a threshold: the guaranteed performance bound for the set-3-pattern falls sharply after a Hamming distance of 5. These tendencies will be examined again after a long training, and shown to persist in the steady state.

Birth-and-death-process analysis of attraction by stable states: The four bounds from (14) can now be used to define a Markov birth-and-death process (e.g., [11],) with birth probability = (upper bound on) divergence probability, and death probability = (lower bound on) contraction probability. This process will underestimate the probabilities of convergence to the stable states, and will therefore yield lower bounds on the regions of attraction. We shall consider the general birth-and-death process depicted in Fig. 3, with $N+1$ states. We denote the birth probabilities p_{bi}, and the death probabilities p_{dj}, where $i = 0, \cdots, N-1$, and $j = 1, \cdots, N$. The process has the general Markov transition matrix

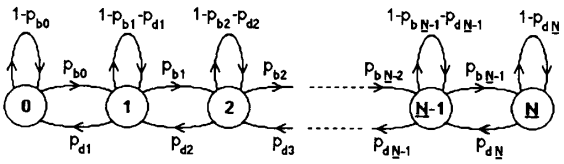


Fig. 3. Markov birth-and-death process.

$$\Psi_{bd}(p_{bi}, p_{di}) = \begin{bmatrix} 1-p_{b0} & p_{b0} & 0 & 0 & 0 & & & & 0 & 0 \\ p_{d1} & 1-p_{b1}-p_{d1} & p_{b1} & 0 & 0 & & & & & 0 \\ 0 & p_{d2} & 1-p_{b2}-p_{d2} & p_{b2} & 0 & & & & & 0 \\ 0 & 0 & & & & & & & & 0 \\ \vdots & \vdots & & 0 & p_{dk} & 1-p_{bk}-p_{dk} & p_{bk} & 0 & & \\ 0 & & & & & & 0 & & & \\ 0 & & & & & & 0 & p_{dN-1} & 1-p_{bN-1}-p_{dN-1} & p_{bN-1} \\ 0 & & & & & & 0 & 0 & p_{dN} & 1-p_{dN} \end{bmatrix} \quad (16)$$

In the evaluation of attraction probabilities for installed patterns we shall use $N = N$, as the Hamming distance between any two states of the network may never exceed N. The birth-and-death probabilities are

$$p_{bi} = 1 - \frac{N-i}{N}\vartheta_i(n_L, B_M) - \frac{i}{N}, \qquad i = 0,1,\cdots,N-1 \quad (17a)$$

$$p_{dj} = \frac{j}{N}\vartheta_{j-1}(n_L, B_M), \qquad j = 1,2,\cdots,N \quad (17b)$$

Equations (16) and (17) can now be used to determine the performance of the network with respect to the stored patterns as a function of time. We get a lower bound on the probability that a randomly chosen initial state, at a given HD from B_M, will converge to B_M. This probability is a function of the number of training steps used to teach

the pattern B_M to the network, and the number of production-phase steps, which we are willing to wait for convergence.

Let $\Gamma^{(n_p)} = [\Gamma_0^{(n_p)}, \Gamma_1^{(n_p)}, \cdots, \Gamma_N^{(n_p)}]$ be a vector, representing the probabilities that a studied parameter resides at positions 0 to N after a given production step n_p. The initial (production-phase) probability distribution is

$$\Gamma^{(0)} = [0,0,\cdots,0,1,0,\cdots] \tag{18}$$

where the "1" represents the initial state's HD from B_M (for an initial state at HD k, $\Gamma_k^{(0)} = 1$.) We assume that the network has been trained, using n_L steps, and ask: what is the (worst-case) probability that, having started with a state at HD k from B_M, it will be *at most* at an (arbitrary) HD m from B_M after n_p production-phase steps? In particular, what is the (worst-case) probability that it will have converged to B_M after n_p steps? Using the birth and death probabilities in (17), the (worst-case) probability distribution is:

$$\Gamma^{(n_p)} = \Gamma^{(0)}[\Psi_{bd}(p_{bi}, p_{di})]^{n_p} \tag{19}$$

and the worst-case convergence probability is just $\Gamma_0^{(n_p)}$. The probability to get to a HD of m or less from B_M in n_p steps is larger than $\sum_{i=0}^{m}\Gamma_i^{(n_p)}$ (for an example, see Section V.)

Rate of learning and numerical approximations: Using (12) and (13) we derive the *rate of learning* that the network possesses with respect to the stabilization objective. For the exact stabilization probability (12) this rate is

$$\begin{aligned}\Delta(M,0)^{(n_L)} &= \vartheta_0(n_L, B_M) - \vartheta_0(n_L - 1, B_M) \\ &= \sum_{i=1}^{N} \rho_i^{n_L+1} \boldsymbol{P}_r\left[S_{Mi}^{(n_L)} > 0\right] + \sum_{\lambda_L=1}^{n_L-1} C(n_L - 1, \lambda_L) \\ &\quad \cdot \sum_{i=1}^{N} \rho_i^{\lambda_L+1}(1-\rho_i)^{n_L-\lambda_L-1} \\ &\quad \cdot \left[\frac{n_L}{n_L - \lambda_L}(1-\rho_i) - 1\right] \boldsymbol{P}_r\left[S_{Mi}^{(\lambda_L)} > 0\right] \end{aligned} \tag{20}$$

while from (14a) we get the growth rate for the lower bound on the contraction probability

$$\begin{aligned}&\Delta(M,k)^{(n_L)} \\ &= \frac{k}{N}\left[\vartheta_k(n_L, B_M) - \vartheta_k(n_L - 1, B_M)\right] \\ &= \frac{k}{N}\left\{ \sum_{i=1}^{N} \rho_i^{n_L+1} \boldsymbol{P}_r\left[S_{Mi}^{(n_L)} > 2n_L k\right] \right. \\ &\quad + \sum_{\lambda_L=1}^{n_L-1} C(n_L - 1, \lambda_L) \sum_{i=1}^{N} \rho_i^{\lambda_L+1}(1-\rho_i)^{n_L-\lambda_L-1} \\ &\quad \left. \cdot \left[\frac{n_L}{n_L - \lambda_L}(1-\rho_i) - 1\right] \boldsymbol{P}_r\left[S_{Mi}^{(\lambda_L)} > 2\lambda_L k\right] \right\}. \end{aligned} \tag{21}$$

Equations (20) and (21) represent:

- Equation (20): the rate of stabilization, which provides the designer with the marginal addition to the stabilization probability, which is gained by an extra step of training, and is therefore useful to assess the efficiency of continued training.
- Equation (21): the rate of growth on the lower bound (14a), which is on the probability of contraction towards B_M from a pre-specified HD. The utility of this rate for design depends on the tightness of the lower bound.

We note that all the expressions used to assess dynamic learning involve the probability density function of the stability margin $S_{Mi}(\lambda_L)$. This function is calculated through auto-convolutions of $\mathbf{pdf}[S_{Mi}^{(1)}]$. A good numerical approximation for $\mathbf{pdf}[S_{Mi}^{(\lambda_L)}]$ can be obtained via the central limit theorem [16].

For a large λ_L (practically, ≥ 10):

$$\mathbf{pdf}\left[S_{Mi}^{(\lambda_L)}\right] \approx \frac{1}{\sqrt{2\pi\lambda_L}\,\delta_{Mi}} \exp\left[-\frac{\left(S_{Mi}^{(\lambda_L)} - \lambda_L \mu_{Mi}\right)^2}{2\lambda_L \delta_{Mi}^2}\right] \tag{22}$$

where

$$\mu_{Mi} = \sum_{d=-N}^{N} d\pi(d, B_M, i) \tag{23a}$$

$$\delta_{Mi}^2 = \sum_{d=-N}^{N} (d - \mu_{Mi})^2 \pi(d, B_M, i). \tag{23b}$$

We can obtain an approximation for $\boldsymbol{P}_r[S_{Mi}^{(\lambda_L)} > 0]$:

$$\boldsymbol{P}_r\left[S_{Mi}^{(\lambda_L)} > 0\right] \approx \frac{1}{2} + \operatorname{erf}\left(\frac{\sqrt{\lambda_L}\,\mu_{Mi}}{\delta_{Mi}}\right) \tag{24}$$

and an approximation for $\boldsymbol{P}_r[S_{Mi}^{(\lambda_L)} > 2\lambda_L k]$:

$$\boldsymbol{P}_r\left[S_{Mi}^{(\lambda_L)} > 2\lambda_L k\right] \approx \frac{1}{2} + \operatorname{erf}\left(\frac{\sqrt{\lambda_L}\,(\mu_{Mi} - 2k)}{\delta_{Mi}}\right). \tag{25}$$

Equations (24) and (25) make the calculations involved with (12), (13), (20), and (21) easier to perform, since high-order autoconvolutions are no longer needed.

Relations with the sum-of-outer-products assignment: The sum-of-outer-products algorithm determines the values of weights using the assignment:

$$w_{ij} = \sum_{k=1}^{R} b_{ki} b_{kj} \tag{26a}$$

and thresholds are chosen by

$$\tau_i = -\sum_{k=1}^{R} b_{ki}. \tag{26b}$$

If all parameter-updating probabilities are equal ($\rho_i = 1/N$,) and all patterns are equally probable ($\Pi_i = 1/R$,) then these weights and thresholds will be obtained as the *average* parameters in adaptive Hebbian training lasting $n_L = \mathrm{NR}$ steps. These restrictions make the sum-of-outer-products assignment (26) inferior to the dynamic assignment of the adaptive algorithm that we have employed (we demonstrate this fact in Section V which investigates steady-state performance.) Steady-state stabilization of pattern B_M with the sum-of-outer-products assignment will be achieved with probability

$$\vartheta_0(B_M) = \sum_{i=1}^{N} \rho_i \boldsymbol{P}_r[S_{Mi} > 0] \tag{27}$$

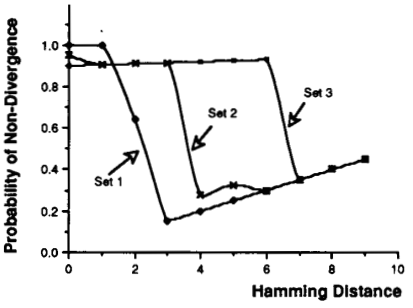


Fig. 4. Probability of non-divergence with sum-of-outer-products assignment versus Hamming distance.

where $P_r[S_{Mi} > 0]$ is either one or zero, depending on whether the deterministic S_{Mi} (equation (3)) is positive or negative. A comparison of lower bounds on the probability of contraction from HD k can also be achieved; let $\eta_{i1} = \max_j |w_{ij}|$, η_{i2} = second largest $|w_{ij}|$, etc. To get the lower bound we use, instead of (13),

$$\vartheta_k(B_M) = \sum_{i=1}^{N} \rho_i P_r\left[S_{Mi} > 2\sum_{r=1}^{k} \eta_{ir}\right] \tag{28}$$

where the probability is either one or zero, depending on the deterministic relation between S_{Mi} (equation (3)) and $2\sum_{r=1}^{k}\eta_{ir}$.

Example (continued): We calculate the weights and thresholds pertaining to the sets in (15), using the sum-of-outer-products assignment. Fig. 4 shows the probability of stabilization (For Hamming distance 0,) and the lower bounds on the non-divergence probabilities (for HD > 0,) versus the Hamming distance between the network's states and the stored patterns. The graphs can be used to assess the efficiency of using the codes in a network trained by the sum-of-outer-products assignment. Pattern 2 of the orthogonal set (set 1) has the largest stabilization probability, but the pattern from set 3 has a higher guaranteed non-divergence probability at the vicinity of the stored pattern (up to Hamming distance 6).

Adaptation to changing pattern sets: We have assumed so far that a single pattern set with a stationary probability distribution is presented to the network for training. If the distribution is not stationary, or if patterns are being added and omitted from the set, the distribution $\pi(d, B_M, i)$ changes with time. The repeated convolutions (as in (11)) are then performed with time-varying probability distributions: if the pattern set has changed, the distribution of S_{Mi}, which was obtained from auto-convolutions before the change, is now convolved with the new $\pi(d, B_M, i)$. Our analysis remains valid, except that the evaluation of **pdf**$[S_{Mi}^{(\lambda_L)}]$ gets more involved.

V. Steady-State Learning Behavior of the Network

We now turn to the steady state distribution of the weights and the thresholds in a network trained by the hard-limiter Hebbian rule. The weights and the thresholds are now bounded by limits (L and T,) and consequently the learning curve is no longer linear. The limits are determined from the analysis in the previous section, and are chosen to be larger than n_L, the number of steps required to guarantee desirable performance. Each weight w_{ij} and threshold τ_i can be described as a Markovian birth-and-death process with reflecting barriers. Using the general $(N+1)\times(N+1)$ transition matrix (16), there are $2L+1$ possible states for each weight and $2T+1$ possible states for each threshold. Hence $N = 2L$ for the processes representing weights and $N = 2T$ for processes representing thresholds. When used to study the weight distribution, state m of the general model corresponds to weight value $m - L$. When used to study the threshold distribution, state m of the general model corresponds to threshold value $m - T$.

For the weights:

$$b_0 = b_1 = \cdots = b_{N-1} = \rho_i P_{ij}^{+} \tag{29a}$$

$$d_1 = d_2 = \cdots = d_N = \rho_i P_{ij}^{-}. \tag{29b}$$

For the thresholds:

$$b_0 = b_1 = \cdots = b_{N-1} = \rho_i P_i^{-} \tag{30a}$$

$$d_1 = d_2 = \cdots = d_N = \rho_i P_i^{+} \tag{30b}$$

Let $\Gamma^{(n_L)} = [\Gamma_0^{(n_L)}, \Gamma_1^{(n_L)}, \cdots, \Gamma_N^{(n_L)}]$ be a vector, representing the probabilities that the parameter resides in the positions 0 to N after the n_Lth updating step. For the weights, the limiting state probabilities are

$$\Gamma_k^{(\infty)} = \Gamma_0^{(\infty)}(Q_{ij})^k, \qquad k = 0,1,\cdots,N \tag{31}$$

where $Q_{ij} = P_{ij}^{+}/P_{ij}^{-}$ is the *weight drifting rate*, and

$$\Gamma_0^{(\infty)} = \begin{cases} \dfrac{1-Q_{ij}}{1-(Q_{ij})^{2L+1}}, & \text{for } Q_{ij} \neq 1 \\ \dfrac{1}{2L+1} & \text{for } Q_{ij} = 1. \end{cases} \tag{32}$$

The expected value of w_{ij} at steady state is

$$E\left(w_{ij}^{(\infty)}\right) = \begin{cases} \dfrac{Q_{ij}\left[1-(2L+1)(Q_{ij})^{2L}+2L(Q_{ij})^{2L+1}\right]}{(1-Q_{ij})\left[1-(Q_{ij})^{2L+1}\right]} - L, \\ \qquad\qquad \text{for } Q_{ij} \neq 1 \\ 0, \qquad\qquad \text{for } Q_{ij} = 1. \end{cases} \tag{33}$$

Similarly for the thresholds,

$$\Gamma_k^{(\infty)} = \Gamma_0^{(\infty)}(Q_i)^k, \qquad k = 0,1,\cdots,N \tag{34}$$

where $Q_i = P_i^{-}/P_i^{+}$ is the *threshold drifting rate*, and

$$\Gamma_0^{(\infty)} = \begin{cases} \dfrac{1-Q_i}{1-(Q_i)^{2T+1}}, & \text{for } Q_i \neq 1 \\ \dfrac{1}{2T+1} & \text{for } Q_i = 1. \end{cases} \tag{35}$$

The expected value of τ_i at steady state is

$$E\left(\tau_i^{(\infty)}\right) = \begin{cases} \dfrac{Q_i\left[1-(2T+1)(Q_i)^{2T}+2T(Q_i)^{2T+1}\right]}{(1-Q_i)\left[1-(Q_i)^{2T+1}\right]} - T, & \text{for } Q_i \neq 1 \\ 0, & \text{for } Q_i = 1 \end{cases} \tag{36}$$

The steady-state probability distributions, as well as the expected values of the network parameters, are independent of the initial states.

The expressions for steady-state parameters are of importance for the case of a stationary pattern set. If the storage of such a set is the goal of the design, then the performance of the network, using adaptive Hebbian training, can be analyzed to assure that enough training steps (n_L) have been allocated to the network. The parameter limits are set accordingly, and the network parameters assigned to be $w_{ij}^{(\infty)}$ and $\tau_i^{(\infty)}$ (provided that the pattern probability distribution is known or can be estimated.) The network's properties can be assessed using (27) and (28) with these parameters.

Example (continued):

Production-phase behavior with steady state training: We use the birth-and-death process analysis of the production phase (Section IV) for steady-state training, employing (27) and (28). We obtain the worst-case transition matrices (of the Hamming distance between the network's state and the stored patterns.) Using initial states at HD2 from the stored patterns, we get (lower bounds on) the probabilities that the network's state is the stored pattern—versus the number of production steps (Fig. 5(a)). Similarly we obtain the lower bounds when the initial states are at HD4 from the stored patterns (Fig. 5(b)). The bounds of the latter case are, of course, lower than those of the former, due to the difference in initial distances. We also observe that the pattern from set 1 is a better attractor for HD2-states than the set-3-pattern, while it is worse than the set-3-pattern when HD4-states are concerned.

Another demonstration of the information which our results convey is given in Fig. 6. Here we compare the bounds on probabilities of non-divergence for pattern 2 in set 1. The 100- and 200-training-step graphs are obtained by Hebbian training with no bounds on parameters. The steady-state bound uses the (relatively low) bounds $L = T = 5$. We observe again the growing discrimination between attracted states and non-attracted states, separated by a threshold.

Comparison with sum-of-outer-products assignment: We assign values to the weights and the thresholds, according to the Hebbian-training steady-state expected values (equations (33) and (36).) We compare the efficiency of storage that employs the steady state parameters, to the efficiency obtained with sum-of-outer-products assignment. In Fig. 7 we compare the two assignments for pattern 2 in set 1. The limits L and T were set to 5 (the largest parameter value which can be assigned by the

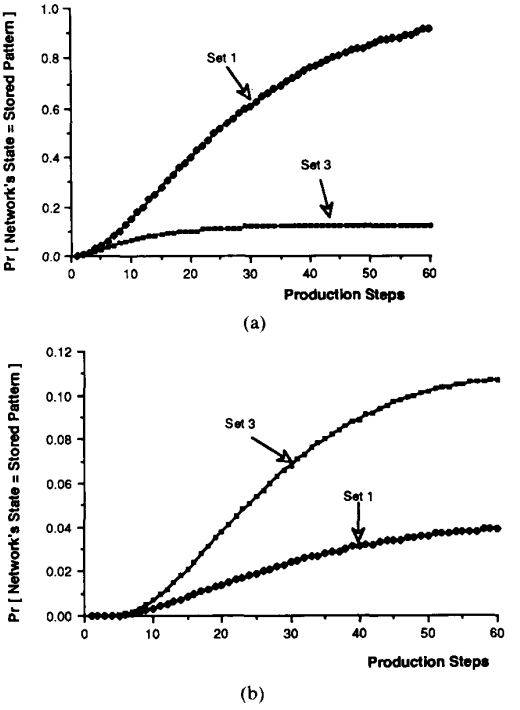


Fig. 5. Probability that the network's state is the stored pattern. (a) Initial state is at Hamming distance 2 from the stored pattern. (b) Initial state is at Hamming distance 4 from the stored pattern.

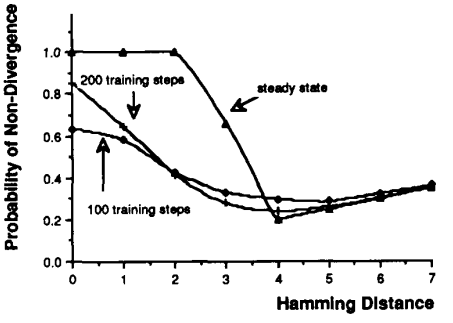


Fig. 6. Probability of non-divergence with different Hebbian-training periods versus Hamming distance.

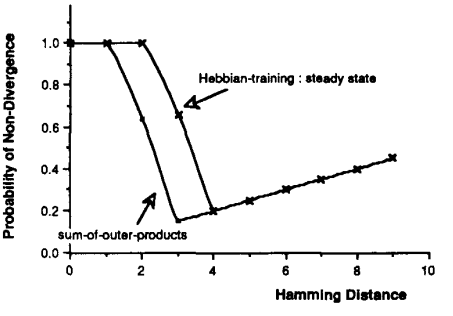


Fig. 7. Comparison of sum-of-outer-products assignment and steady-state Hebbian-training (pattern 2 of set 1).

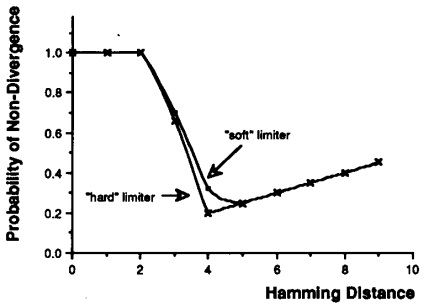


Fig. 8. Comparison of non-divergence probabilities: "soft"- and "hard"-limiter training.

sum-of-outer-products method.) The Hebbian-training steady-state assignment gives better bounds. We found that this is the situation for the other two sets also. Clearly, having better bounds does not guarantee actual better performance. However, in the absence of performance evaluation techniques that are not time-exponential, the guarantee of a better worst-case performance with steady-state assignment is valuable for network design.

The effect of the soft-limiter learning rule on steady-state behavior: We have compared the non-divergence probabilities at steady state for the hard-limiter rule (equation (5)) and the soft-limiter rule (equation (6).) The values used for the network parameters were $r_{ij}^{(\infty)}$, $q_i^{(\infty)}$, $w_{ij}^{(\infty)}$, and $\tau_i^{(\infty)}$. Equations (27) and (28) were used in order to calculate the steady-state stabilization probabilities and the lower bounds on convergence probabilities from a pre-specified HD. The results can serve to shape the sigmoidal transformations (choose r_0 and q_0).

Example (continued): We study pattern 2 from set 1 (mutually orthogonal patterns.) In Fig. 8 we show the relevant bounds, using steady-state values obtained with "hard" and "soft" (sigmoidal) learning curves ($r_0 = q_0 = 2$.) For this pattern, the guaranteed non-divergence probabilities of the sigmoidal training are somewhat higher than those obtained with "hard" limiter training. We conclude that the advantages of using the sigmoidal learning curve (in terms of noise immunity,) were achieved with no reduction in the non-divergence bounds.

VI. Conclusion

We have presented and analyzed a training algorithm for a network of asynchronous, learning threshold elements. The algorithm is based on the Hebbian hypothesis, and it allows adaptation of the learning-network parameters to changing pattern environments. In particular, the network's properties can be quantified in environments where pattern occurrence is random, with non-equal, non-stationary probability distributions. The state reassessment probabilities of neurons during information retrieval can also be nonstationary, and not equal for all neurons.

The trained network is a content-addressable memory. We evaluate its stabilization properties with respect to a given set of patterns, using the theory of Markov processes. The results are applicable for the determination of efficient coding for information that has to be stored, and for prediction of actual pattern-retrieval capabilities of the trained network. We include the popular sum-of-outer-products assignment as an analyzable specific case of our training procedure, and allow the steady state analysis of a large group of sigmoidal learning curves. While the determination of the coding and of the neuron-update probability distribution for optimal pattern retrieval remains an open question, the present work provides the network designer with an analytical tool for evaluation and comparison of various schemes.

Appendix
Derivation of Contraction and Divergence Probabilities

We consider the probability of non-divergence. If a state at HD k from a stable state B_M is "non-diverging," it will "contract" when one of the k neurons, whose activity level is different from the value in the corresponding position of B_M ($u_i \neq B_{Mi}$), updates its activity level. It will not change state if a neuron for whom $u_i = b_{Mi}$ updates its activity level. There are $C(n,k)$ states at HD k from B_M, and as no *a priori* knowledge is available about the mechanism that creates errors, we shall assume that they are equally likely. For each state at HD k from B_M, the (lower bound on the) contraction probability is, therefore,

$$\sum_{j|u_j \neq B_{Mj}} \rho_j \sum_{\lambda_L=1}^{n_L} C(n_L,\lambda_L)\rho_j^{\lambda_L}(1-\rho_j)^{n_L-\lambda_L} \cdot \boldsymbol{P_r}\left[S_{Mj}^{(\lambda_L)} > 2\lambda_L(k-1)\right]. \quad \text{(I-1)}$$

where we have taken into account the fact that when neuron j updates its activity level, it is influenced by at most another $(k-1)$ neurons whose states are different from those of B_M. The overall bound on the contraction probability from HD k is

$$P_k^{\text{contr}}(n_L, B_M) = \frac{1}{C(N,k)} \sum_{R=1}^{C(N,k)} \sum_{j|u_{Rj} \neq B_{Mj}} \rho_j \cdot \sum_{\lambda_L=1}^{n_L} C(n_L,\lambda_L)\rho_j^{\lambda_L}(1-\rho_j)^{n_L-\lambda_L} \cdot \boldsymbol{P_r}\left[S_{Mj}^{(\lambda_L)} > 2\lambda_L(k-1)\right] \quad \text{(I-2)}$$

where the first summation is over all relevant $C(N,k)$ states. The summation over j includes k elements, and therefore the double summation over R and j includes $kC(N,k)$ terms. Due to symmetry, each neuron is considered exactly $kC(N,k)/N$ times, and, therefore, (I-2) can be simplified to

$$P_k^{\text{contr}}(n_L, B_M) = \frac{k}{N}\sum_{i=1}^{N}\rho_i \sum_{\lambda_L=1}^{n_L}\rho_i^{\lambda_L}(1-\rho_i)^{n_L-\lambda_L} \cdot \boldsymbol{P_r}\left[S_{Mi}^{(\lambda_L)} > 2\lambda_L(k-1)\right] = \frac{k}{N}\vartheta_{k-1}(n_L, B_M). \quad \text{(I-3)}$$

The lower bound on the probability of a "stalemate", no contraction and no divergence, can be found using similar arguments. The neurons that need to be considered are the ones whose activity levels differ from those of B_M, and the ones that have the same levels, and will not diverge. The upper bound on the probability of divergence is the complement of the sum of the convergence and stalemate bound to one.

REFERENCES

[1] Y. S. Abu Mostafa and J-M. St. Jacques, "Information capacity of the Hopfield model," *IEEE Trans. Inform. Theory*, vol. IT-31, pp. 461–464, 1985.

[2] S-I. Amari, "Learning patterns and pattern sequences by self-organizing nets of threshold elements," *IEEE Trans. Computers*, vol. C-21, pp. 1197–1206, 1972.

[3] D. J. Amit, H. Gutfreund, and H. Sompolinsky, "Spin-glass model of neural networks," *Phys. Rev. A.*, vol. 32, no. 2, pp. 1007–1018, 1985.

[4] ____, "Storing infinite numbers of patterns in a spin-glass model of neural networks," *Phys. Rev. Lett.*, vol. 55, no. 14, pp. 1530–1533, 1985.

[5] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, Image Process.*, vol. 37, 1987.

[6] S. Dasgupta, A. Cosh, and R. Cuykendall, "Convergence in neural memories," in *Proc. 1987 Int. Symp. Circuits and Systems*, Philadelphia, PA, pp. 366–369, 1987.

[7] D. O. Hebb, *The Organization of Behavior*. New York: Wiley, 1949.

[8] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. National Academy of Science (USA)*, vol. 79, pp. 2554–2558, 1982.

[9] ____, "Neurons with graded response have collective computational properties like those of two-state neurons," in *Proc. National Academy of Science (USA)*, vol. 81, pp. 3088–3092, 1984.

[10] J. J. Hopfield, and D. W. Tank, "'Neural' computation of decision in optimization problems," *Biol. Cybern.*, vol. 52, pp. 1–12, 1985.

[11] R. A. Howard, *Dynamic Probabilistic Systems, Vol. I: Markov Models*. New York: Wiley, 1971.

[12] M. Kam, R. Cheng, and A. Guez, "On the design of a content-addressable memory via binary neural networks," in *Proc. First Int. Conf. on Neural Networks*, vol. 2, pp. 513–522, San Diego, CA, 1987.

[13] A. W. Little and G. L. Shaw, "Analytic study of the memory storage capacity of a neural network," *Math. BioSci.*, vol. 39, pp. 281–290, 1978.

[14] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The capacity of the Hopfield associative memory," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 461–482, 1987.

[15] A. V. Oppenheim, and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[16] A. Papoulis, *The Fourier Integral and Its Applications*. New York: McGraw-Hill, 1962.

[17] P. Perreto and J-J Niez, "Stochastic dynamics of neural networks," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, 1986.

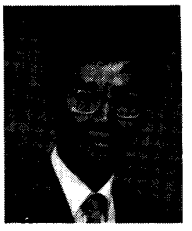
[18] W. T. Rhodes, (Editor): *Neural Networks. Appl. Opt.*, vol. 26, no. 23, 1987.

[19] F. Rosenblatt, *Principles of Neurodynamics*. New York: Spartan, 1962.

[20] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Sci.*, vol. 9, pp. 75–112, 1985.

Moshe Kam (S'75–M'86) received the B.Sc. degree from Tel Aviv University in 1977 and the M.S. and Ph.D. degrees from Drexel University in 1985 and 1987, respectively.

From 1977 to 1983, he was with the Israeli Defense Forces. Since September 1987, he has been an Assistant Professor in the Department of Electrical and Computer Engineering at Drexel University. His research interests are in the areas of large-scale systems, control theory, and information theory. He organized the session on "Relations between Information Theory and Control" in the 1987 American Control Conference, the session on "Neural Networks in Control" in the 1988 American Control Conference, and the 1987 IEEE Philadelphia seminar on "Neural Networks and their Applications." Dr. Kam serves as the Chairman of the joint Societies of the IEEE Circuits and Systems and IEEE Control Systems Chapter of the Philadelphia IEEE Section.



Roger Cheng received the B.S. degree in electrical and computer engineering from Drexel University, Philadelphia, PA, in 1987.

Since then, he has been a Research Assistant at Princeton University, NJ, where he is currently working towards the Ph.D. degree in Electrical Engineering. During 1986–1987 he was a Research Assistant at the Control System Laboratory at Drexel University. His research interests include neural networks, multi-user communications, and information theory.

Allon Guez received the B.S.E.E. degree from the Technion, Israel Institute of Technology, in 1978, and the M.Sc. and Ph.D. degrees from the University of Florida, Gainesville, in 1980 and 1982, respectively.

He has been with the Israeli Defense Forces from 1970 to 1975, with System Dynamics, Inc., from 1980–1983, and with Alpha Dynamics, Inc., from 1983 to 1984. In 1984, he joined the Faculty of the Electrical and Computer Engineering Department at Drexel University. He has done research and development and design in control systems, neurocomputing, and optimization.