ON THE DYNAMICS OF ARTIFICIAL NEURAL NETWORKS

by

Shobana Venkatraman

Memorandum No. UCB/ERL M88/85

22 December 1988

ON THE DYNAMICS OF ARTIFICIAL NEURAL NETWORKS

by

Shobana Venkatraman

Memorandum No. UCB/ERL M88/85

22 December 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ON THE DYNAMICS OF ARTIFICIAL NEURAL NETWORKS

by

Shobana Venkatraman

Memorandum No. UCB/ERL M88/85

22 December 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Acknowledgements

I thank my research advisor Prof. Shankar Sastry for his encouragement and support through the course of my master's program. I also thank Prof. Jitendra Malik for the interest and time he has taken to read my report.

I have enjoyed interacting with everyone in the robotics group, particularly Saman Behtash, Jeff Mason and Pietro Perona. I also thank my friends Yogesh Gubbi, Sudeshna Sengupta, Elizabeth Finlayson, and Patrick Phelan, who have made my graduate school experience very special.

This reseach was supported in part by JSEP, under grant # F49620-87-C-0041.

Table of Contents

CHAPTER I

INTRODUCTION :

1.1. A Survey of Neural Networks

The mathematical modeling and idealization of biological memory has led to a variety of networks with interesting properties. Associative memory was one of the earliest such properties to be studied. Interest in the perceptron, originally conceived by Rosenblatt waned when Minsky and Pappert studied and criticized its limitations. McCulloch and Pitts proposed the first mathematical model of a single nerve cell. In recent years, Hopfield and Grossberg have greatly revived interest in models of the human brain built by connecting several such McCulloch Pitts neurons. Grossberg has attempted to build faithful models of the cortical nerve cells while Hopfield has proposed and studied a simplified model mainly applying them for associative memory and optimization.

There are about 10^{11} neurons in the human brain. The functioning of the brain depends on the flow of information through elaborate networks of neurons. Information is transferred from one neuron to another at specialized points of contact, called the synapses. A typical neuron has 1000 to 10,000 synapses. neuroscientists model the neuron using three characteristics : 1) their internal state, 2) the axonal potential, and 3) the synaptic inputs and outputs.

The Internal State: The internal state of a neuron is characterized by the potential difference across the cell membrane. The baseline is a resting potential between 70 and 100 mV and when external inputs cause it to go beyond a certain threshold, an action potential is generated which travels across the axon.

Axonal Signals : The action potential is a depolarizing signal with a peak amplitude of about 110 mV and lasting for 1-10 msecs. Axonal signals are bursts of evenly spaced action potentials with pulse frequencies of 2 to 400 Hz for the pyramid cells in the cerebral

cortex and 1 to 100 Hz for retinal ganglion cells. The information carried in the axonal signals is believed to be in the pulse frequency of the burst. Thus the signal can be represented by a positive real number over a limited short time duration.

Synaptic Inputs and Outputs : The flow of signals into and out of the neurons is unidirectional. Signals are received at points of contact on the dendrites called *synapses*. When an incoming signal reaches the synaptic knob it induces the release of a substance called a *neurotransmitter* which diffuses into the synaptic cell and causes a change in the *receptor potential* across the cell membrane. A synaptic input is excitatory if it increases the receptor potential or inhibitory if it decreases it.

Artificial Neural Networks

All the artificial neural networks proposed thus far share a common broad framework. They have a set of processing units, each of which has a well defined state of activation, analogous to the internal state of a nerve cell. Each unit has an output function which is typically a threshold function. The inter-connection between the processing units is described by a pattern of connectivity, which is modified by "learning". Patterns of activity are propagated through the network by a propagation rule. The inputs to each processing units are combined by an activation rule (which is typically a sigmoid function) to alter the state of activation of the unit. The construct of the networks proposed by Hopfield [1], Grossberg [2], and that by Michel, Li and Porod [4] which we will be studying in detail are briefly discussed here.

The Hopfield network

Hopfield has proposed several models of networks. In his paper [1], he proposed a model that would function as a Content Addressable Memory. The structure of the network can be described as follows :

Let x = $\{x_1, x_2, x_3, ..., x_m\}$ be an m-set of n-dimensional binary (their entries belong to {-1,1}) column vectors which we want the network to store as memories. For each x_i, a

connection matrix describing the pattern of connectivity among the various neurons is constructed thus:

$$T_i = x_i x_i^t - I_n \qquad (1.1)$$

The connection matrix, also called weight matrix, for the entire network is then constructed by summing the T_i s.

$$T = \sum_{i=1}^{i=m} T_i \qquad (1.2)$$

Once the vectors $x_1, x_2, \ldots, x_m$ are programmed into the network they are "forgotten". In order to retrieve information, we start with an n-dimensional probe vector x. We wish to find the stored memory x_i closest to x in Hamming distance. Hopfield's asynchronous algorithm for this consisted of replacing the kth component of x by the kth component of Tx. For a symmetric T, this process can be shown to be convergent. Starting with any x, one will find a y such that y = Ty.

Hopfield also proposed a continuous-variable neural system which is realizable by electric circuits. It is represented by the following system of equations :

$$C_i(u_i/dt) = \sum_{j=1}^{j=n} T_{ij} v_j - u_i/R_i + I_i \qquad (1.3)$$

where C_i, R_i and I_i are constants for all i, $C_i > 0$ and $R_i > 0$.

T_{ij} is a real symmetric n x n matrix and g_i : R -> (-1,1) is a monotone increasing function satisfying the following conditions:

$g_i(0) = 0$, $g_i(\rho) = -g_i(-\rho)$. Further, g_i^{-1} : (-1,1) -> R exists and g_i and g_i^{-1} are C^1 functions.

The Grossberg Model

Cohen and Grossberg[3] studied a class of competitive dynamical systems described by

$$\dot{x} = a_i(x_i)[b_i(x_i) - \sum_{j=1}^{j=n} C_{ij} d_j(x_j)], \quad \text{for i = 1,2,3,.....n.} \qquad (1.4)$$

where $C_{ij} > 0$.

x_i is the potential (short term memory activity)

$d_k\ (x_k)$ is the output of node k

C_{ik} is the strength of the connection between node i and k

$C_{ik} > 0$ for an inhibitory system and C_{ik} is < 0 for an excitatory system

$a_i(x_i)$ is the amplification factor and

$b_i(x_i)$ is the intrinsic rate of decay or activity

The Hopfield model is in fact a special case of the Grossberg model and this can be established by a simple transformation of variables. To illustrate, if in equation (1.3) we set

$\frac{1}{C_i} = a_i(u_i)$, $\frac{-1}{(R_i)}u_i + I_i = b_i(u_i)$, $T_{ij} = -c_{ij}$ and $g_j(\lambda u_j) = d_j(u_j)$,

then equations (1.3) and (1.4) are identical.

Grossberg's Convergence Theorem:

Grossberg proved a remarkable convergence theorem[2]. If the functions b_i are monotonic increasing functions, then the system has a finite number of equilibrium points and each initial point converges to the equilibrium point that is closest to it.

1.2. Generalised Model proposed by Michel, Li and Porod

This is the model that we study in detail. The model is described by the following differential equation:

$$dx/dt = -H(x)(-Tx + S(x) - I) \qquad (1.5)$$

$H : (-1,1)^n \rightarrow R^{n \times n}$, i.e. for each x, H(x) is an n x n matrix;

$T \in R^{n \times n}$, and is the "connection" or "weight" matrix;

and $S(x) = [S_1(x_1) | S_2(x_2) | .....S_n(x_n)]$;

$I = (I_1, I_2, ..., I_n)'$ is a constant real vector and represents the input to the system;

Under certain assumptions imposed upon this model which are described in the next chapter, the qualitative behavior of the model has been shown to be predictable. It is instructive to study this model because because of its convenient mathematical representation and more importantly, because it represents the class of Hopfield and Grossberg models described earlier. This is an interesting fact because these two models are being

studied widely.

To illustrate, let us consider the continuous variable network realizable by electrical circuits proposed by Hopfield in [1]. It is represented by the system of equations

$$C_i(du_i/dt) = \sum_{j=1}^{j=n} T_{ij} - v_i/R_i + I_i \qquad (1.6)$$

where C_i, R_i and I_i are constants for each i, $C_i > 0, R_i > 0$ and $T_{ij} = T_{ji}, T \in R^{n \times n}$;

$u_i \in R$, $v_i \in (-1,1)$ and $v_i = g_i(\lambda u_i)$ where $\lambda > 0$;

and $g_i : R \rightarrow (-1,1)$ satisfies the following conditions:

a) it is monotone increasing.

b) $g_i(0) = 0$, $g_i(\rho) = -g_i(-\rho)$

c) $g_i^{-1} : (-1, 1) \rightarrow R$ exists, and g_i and g_i^{-1} are C^1 functions. Now, representing equation (1.6) in terms of v_i,

$$dv_i/dt = (dv_i/du_i)(du_i/dt)$$

$$= \frac{1}{(du_i/dv_i)} (1/C_i) \left(\sum_{j=1}^{j=n} T_{ij} v_j - u_i/R_i + I_i\right)$$

$$= - \frac{\lambda}{dg^{-1}(v_i)/dv_i} (1/C_1) - \sum_{j=1}^{n} T_{ij} + \frac{g^{-1}(v_i)}{\lambda R_i} - I_i \qquad (1.7)$$

Now, substituting $(v_1, v_2, \cdots v_n)^t$ by x,

$\text{diag}\left[\frac{\lambda}{C_i(dg_i^{-1}(v_i),.....}\right]$ by H(x), $[T_{ij}]$ by T, $\left[\frac{g^{-1}(v_1)}{\lambda R_1},....,\frac{g^{-1}(v_n)}{\lambda R_n}\right]^t$ by S(x),

and $(I_1, I_2, I_3, \cdots, I_n)$ by I, equation (1.7) is identical to equation (1.5), the model proposed by Michel, Li and Porod.

The model proposed by Cohen and Grossberg which we described earlier can also be represented by equation (1.5). The equation describing the Grossberg model is:

$$dx/dt = a_i(x_i)[b_i(x_i) - \sum_{j=1}^{j=n} C_{ij} d_j(x_j)], \text{ for i = 1, 2, .. n.} \qquad (1.8)$$

if we let $1/C_i = a_i(u_i), (-1/R_i)u_i + I_i = b_i(u_i)$, and $g_j(\lambda u_j) = d_j(u_j)$, then it

translates to equation (1.4).

In both cases, it can be easily verified that the assumptions made to establish the properties of the network models also mapped under the transformation of variables.

1.3. An overview of the chapters that follow

In Chapter II, the model proposed by Michel, Li and Porod and their algorithm for updating the weight matrix T are studied. A modified algorithm which makes the update rule recursive is proposed. In Chapter III we study the discrete time Hopfield network and analyze the spurious states of the network using the principles of Threshold logic. In Chapter IV, we study back propagation and its difference in approach to the associative memory problem.

CHAPTER II

THE MICHEL LI POROD MODEL:

2.1. Analysis of the model

The model can be thought of as a class of nonlinear, autonomous, nonlinear differential equations of the form

$$\dot{x} = -H(x)(-Tx + S(x) - I) \tag{2.1}$$

where $x = (x_1, x_2, \ldots x_n)^t \in (-1,1)^n$

$H:(-1,1) \rightarrow R^{n \times n}$, $T \in R^{n \times n}$

$S(x) = (S_i(x_i), \ldots\ldots S_n(x_n))^T$ where $S_i : (-1,1) \rightarrow R$

Assumptions (A):

i) For every $x \in (-1,1)^n$, $H(x)$ is symmetric and positive definite.

ii) $T_{ij} = T_{ji}$ for all i, j.

iii) $S_i:(-1,1) \rightarrow R$ is monotone increasing for $1 \leq i \leq n$, $S_i(0) = 0$;

$S_i(x_i) = -S_i(-x_i)$; S_i, S_i^{-1} are C^1 functions.

Also, it is required that $S_i''(x_i) = d^2S_i/dx_i^2$ exists.

A few of the important results that describe the qualitative behavior of the system are presented here.

Definition 2.1: An energy function is associated to the system and it is :

$$E(x) = -(1/2)x^t Tx + \sum_{i=1}^{i=n} \int_0^{x_i} s_i(\rho)d\rho - x^t I \tag{2.2}$$

The gradient of E,

$$\nabla E(x) = (\partial E(x)/\partial x_1, \partial E(x)/\partial x_2, \ldots\ldots, \partial E(x)/\partial x_n)^t \tag{2.3}$$

$$= -Tx + S(x) - I$$

This immediately leads us to the following conclusion:

Lemma 2.1:

If the system satisfies assumptions (i) through (iv) then $x \in (-1,1)^n$ is an equilibrium point of the system if and only if $\nabla E(x) = 0$

Assumptions (B):

i) There is no $x \in (-1,1)^n$ satisfying the equations:

(a) $\nabla E(x) = 0$,

(b) $det(J_E(x)) = 0$

(c) $J_E(x) \geq 0$ simultaneously.

ii) The set of equilibrium points of the system is discrete.

Theorem 2.1: If the system satisfies assumptions (A) and (B) then

1) For any $x \in (-1,1)^n$ there is a unique solution for (2.1).

2) Along a non-equilibrium solution, E decreases monotonically, implying that no non-constant periodic solutions exist.

3) Each solution exists on $[0,\infty)$

4) Each non-equilibrium solution of (2.1) converges to an equilibrium solution of (2.1) as $t \to \infty$.

5) There are only a finite number of equilibrium points for (2.1).

Theorem 2.2: Suppose that the system satisfies the assumptions (A) and (B). If $\hat{x}$ is an equilibrium of (2.1) then the following statements are equivalent:

a) $\hat{x}$ is stable;

b) $\hat{x}$ is a local minimum of function E;

c) $J_E(\hat{x}) > 0$;

d) $\hat{x}$ is asymptotically stable.

Assumptions(C):

1) $S_i : (-1,1) \to R$ is a C^2 function and

2) $S_i''(\rho) > 0$ for $\rho \in (0,1)$

Conjecture:

If the system (2.1) satisfies assumptions (A), (B) and (C), then there is at most one asymptotically stable equilibrium point of S in each of the 2^n regions $\Lambda(\eta_1,\eta_2,.....\eta_n)$ defined by

$\Lambda(\eta_1,\eta_2,\eta_3,\cdots\eta_n) = \{x \in (-1,1)^n : \eta_i x_i > 0, -1 \leq i \leq n\}$.

Michel, Li and Porod have proposed the above as a theorem in their results [1]. However, the proof proposed by them is not complete in our opinion. We present their proof here and explain why we consider it to be incomplete.

Proof proposed for Conjecture:

To prove the theorem we define the following:

$D = \{x \in \Lambda : DF(x) > 0\}$ and $C = P^{-1}(D)$

Lemma 1: C is convex.

Lemma 2: There is at most one zero of $\nabla E(x)$ in region D.

Proof of Lemma 2: Assume there is more than one zero of $\nabla E(x)$ in region D. Then there exist $y, z \in C$, $y \neq z$ such that $Q(y) = Q(z) = 0$. By the mean value theorem, there exists $\lambda \in (0,1)$ such that $Q(y) = Q(z) + DQ(z + t(y - z))(y - z)$.

It is this proof which finally leads to the proof of the conjecture that we disagree with because of the fact that the mean value theorem is not applicable for functions from R^n to R.

2.2. Synthesizing the Network

Given a set of specified vectors, we wish to store them as equilibrium points of a neural network of the form (2.1), satisfying assumptions (A), (B) and (C). We assume that $S(x)$ and $H(x)$ are given and that they satisfy the assumptions we have made. The m information vectors, $\{a_1,a_2,.....,a_n\}$ are specified. We wish to find a symmetric real n x n matrix T and an external input vector $I \in R^n$ such that $a_1,a_2,..,a_n$ are equilibrium points of (2.1).

From Lemma 2.1 we know that in order for the vector a_i, $1 \leq i \leq m$ to be equilibrium

points of (2.1) they must satisfy the system of equations

$$Ta_i + I = S(a_i),\ i = 1, 2, ...m. \qquad (2.4)$$

Lemma 2.2 :

Given m vectors $\{a_1, a_2, ..., a_n\}$ C $(-1,1)^n$, then T and I comprise a solution of (2.4) if and only if they are a solution of the equations

$$TA = B \qquad (2.5) \text{ and}$$

$$I = Sa_m - T(a_m) \qquad (2.6)$$

where $A = [a_1 - a_m, ..., a_{m-1} - a_m]$ and $B = [S(a_1) - S(a_m), ..., S(a_m - 1) - S(a_m)]$

2.2.a Summary of Synthesis Procedure

i) Check if there are any $a_i \neq a_j$ which are located in the same region $\Lambda(\eta_1, \eta_2, ..., \eta_n)$ defined by equation (2.4). If so it may not be possible to store both a_i and a_j as equilibria.

ii) Compute $A = [a_1 - a_m, ..., a_{m-1} - a_m]$, $S(a_i)$,for i=1,2,...,m ,

$B = [S(a_1) - S(a_m), ..., S(a_{m-1}) - S(a_m)$ and $A^t B$.

iii) Check if $A^t B$ is symmetric. If this is not true, $(a_1, a_2, ..., a_n)$ can not be all equilibrium points of (2.1).

iv)Perform a singular value decomposition of A to obtain the matrices U, V and Σ such that $A = U\Sigma V^t$ where U, V are unitary matrices and Σ is a diagonal matrix with the singular values of A on its diagonal. In doing so we find k , the rank of A.

v) Compute $U^t BV$

vi) Check if the last (m-1-k) rows of $U^t BV$ are all zero. If not $a_1, a_2, ..., a_m$ can not all be equilibria of (2.1).

vii) Choose an (n-k) x (n-k) symmetric parameter matrix $\Psi = -\alpha I_{(n-k)}$ where α is a large positive number.

viii) Compute $Q = \begin{bmatrix} C_1\Sigma_1^{-1} & (C_2\Sigma_1^{-1})^t \\ C_2\Sigma_1^{-1} & \Psi \end{bmatrix}$

where C_1 is formed by the first k rows and first k columns of $U^t BV$ and U_2 is formed by

the last n-k rows and first k columns of $U^t BV$. Σ_1 is formed by the first k columns of Σ.

ix) Compute $T = UQU^t$ and $I = S(a_m) - Ta_m$

x) Check if all of the eigenvalues of $J_E(a_i) = \text{diag}[\, s_1'(a_i^{(1)}),.....,s_n'(a_i^{(n)})] - T$ are positive. $a_i^{(j)}$ is the jth coordinate of vector a_i, i = 1, 2, ...m. If this is true then $a_1, a_2,...,a_n$ are asymptotically stable equilibrium points of (2.1)

2.3. Properties of the model

1. The model is fail soft in the sense that it will operate satisfactorily even if some of the connections are severed (some of the T_{ij} 's set to zero).

2. The update algorithm is straight forward and easy to program.

3. The computation is collective and this leads to a reduced computation time.

2.4. Limitations of the model

1. Given a set of memory vectors, the network is fixed and there is no "learning" process going on which will make the time to converge from an initial vector to the equilibrium closest to it shorter by starting from that initial vector frequently.

2. Every time we wish to add a new memory vector the entire network has to be synthesized from step 1.

2.5. Modified update Rule :

The algorithm described above has to be repeated from step (i) if we wish to program the network to store an additional vector as an equilibrium point. In order to make the computation a little easier we propose an update rule which is based on the orthogonalized projection algorithm [5]. To begin with we assume that the vectors a_i are an orthonormal set of vectors. This is a reasonable assumption if m << n. Then we build T in the following manner:

We let $Ta_1 = y_1$

We construct a first approximation for T as $\hat{T}_1 = y_1 \frac{a_1^t}{|a_1|^2}$

We define $e_2 = a_2 - a_1 \frac{a_1^t a_2}{|a_1|^2}$

Now, $Ta_2 = y_2 - y_1\frac{(a_1{}^t a_2)}{|a_1|^2} = \hat{y}_2$

So we build the next approximation to T as

$\hat{T}_2 = \hat{T}_1 + \hat{y}_2\frac{e_2^t}{|e_2|^2}$,

and so on to find the weight matrix. The advantage of this construction is that it is recursive. We illustrate both the methods with the example borrowed from Michel, Li and Porod.

Example 2.1 : Let n = 3 and let functions s : (-1,1) -> R be given by the sigmoid function

$s(\rho) = (1/\lambda)(2/\pi)tan((\pi/2)\rho)$ where $\lambda = 1.4$. Then

$s'(\rho) = (1/\lambda)[1/\cos^2((\pi/2)\rho)]$, and

$s''(\rho) = (\pi/\lambda)\sin((\pi/2)\rho)/\cos^3((\pi/2)\rho)$

$a_1 = (-0.6,-0.5,-0.4)^t$ and $a_2 = (-0.4,0.8,-0.7)^t$

We seek to find a 3 x 3 matrix T and an input vector I, for a neural network of the form of equation (2.1).

As explained in the procedure above we compute

$$A = \begin{bmatrix} -1.0 \\ -0.3 \\ 0.3 \end{bmatrix}$$

$$S(a_1) = \begin{bmatrix} -0.626 \\ -0.455 \\ 0.330 \end{bmatrix}, S(a_2) = \begin{bmatrix} -0.330 \\ 1.3995 \\ -0.892 \end{bmatrix}.$$

$$B = \begin{bmatrix} -0.956 \\ -0.944 \\ 0.560 \end{bmatrix}$$

$A^tB = [1.4083]$

The symmetry of A^tB is trivially verified. And this means that the chosen vectors can be programmed to be the equilibria of a network of the form (2.1).

Now, performing a singular value decomposition of A, such that $A = U\Sigma V^t$ yields

$$\begin{bmatrix} -0.9206 & 0.2761 & 0.2761 \\ -0.2761 & 0.9603 & 0.0397 \\ 0.2761 & 0.0397 & 0.9603 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1.086 \\ 0.000 \\ 0.000 \end{bmatrix}$$

and $V = [1.0]$ and the rank of A = k = 1.

We compute the matrix $U'\Sigma V$

$$U'\Sigma V = \begin{bmatrix} 1.2965 \\ -0.6208 \\ 0.2381 \end{bmatrix}$$

Since m-1-k = 0, we don't have to check for null columns in $U'BV$. We now choose the parameter matrix Ψ to be $\Psi = -\alpha\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ where $\alpha = 10$

$$\text{Matrix } Q = \begin{bmatrix} 1.194 & -0.571 & 0.219 \\ -0.571 & -1.0 & 0 \\ 0.2192 & 0.0 & -10.0 \end{bmatrix}$$

From Q we can compute the matrix T and the input vector I and they come out to be

$$T = \begin{bmatrix} -9.1608 & 3.2827 & -2.9584 \\ 3.2827 & -8.8479 & -1.0548 \\ -2.9284 & -1.0548 & -9.0425 \end{bmatrix}$$

$$\text{and } I = \begin{bmatrix} -4.0 \\ 6.43 \\ -5.20 \end{bmatrix}$$

By computing $S(a_1) - Ta_1 - I$ and $S(a_2) - Ta_2 - I$ it is verified that a_1 and a_2 are equilibrium points of system (2.1). Further, since the eigenvalues of $J_E(a_1)$ and $J_E(a_2)$ are both positive, a_1 and a_2 are asymptotically stable equilibria.

CHAPTER III

DISCRETE NEURAL NETWORKS

3.1. The Hopfield Network

In [1], Hopfield proposed a model which consisted of n "neurons", the output of each being either 0 or 1 . Each neuron adjusted its state asynchronously by setting

$$v_i \rightarrow +1 \text{ if } \sum_{j=1}^{j=n} T_{ij} v_j > U_i \qquad (3.1)$$

$$\text{and } v_i \rightarrow -1 \text{ if } \sum_{j=1}^{j=n} T_{ij} v_j < U_i$$

where T_{ij} is the strength of the interconnection between neurons i and j. The I/O characteristic of each neuron is modeled as a step function. The output of all the neurons, $v_1, v_2, \cdots, v_n$ can be represented by one vector $\vec{v}$. The mode of operation of the network can now be described by the equation

$$\vec{v}(k+1) = sgn(T\vec{v}(k)) - U \qquad (3.2)$$

where sgn(x) = +1 for x > 0 and -1 for x < 0, and U is the vector of thresholds $U_1, U_2, \cdots U_n$.

In general, the thresholds are set to be equal since this is easier to implement and they are very often set to be zero. Also, only one entry of U is updated at a time. The output of the network is $\lim_{k \rightarrow \infty} v(k)$ when it exists and we denote it by V_{out}. If the limit V_{out} exists, then H: {-1,1} -> {-1, 1} is called a Hopfield operator. H is completely specified by the matrix T. We seek to study the number of spurious equilibria of the system.

3.2. Properties of the network

1. It is fail soft.

2. It "learns". Neurophysiologists believe that human beings learn by modifying the

synapses in the brain. The ability of the network to modify the strength of the connection between two units to add memories can be considered analogous to that.

3. The collective nature of the computation makes it faster .

3.3. Limitations of the network

1. Perhaps the biggest limitation of the network is its memory capacity which is very low. The number of memories that can be faithfully programmed to be content addressable is as low as 1.5N [7], N being the number of units in the network.

2. Another limitation is the appearance of spurious equilibria, or memories that represent patterns we do not intend to remember.

3. The large number of interconnections between the various units makes it difficult to synthesize such a network in a VLSI chip.

Theorem 3.1 :

Let $G = \{a_1, a_2, ..., a_m\}$ belong to $(-1,1)^n$ where the a_i s are orthonormal, i.e. $a_i{}^t a_j = 1$ for i = j and 0 otherwise.

Let $T = \sum_{i=1}^{i=s} a_i a_i{}^t$. Let M_m be the set of memories that the network stores. The set of memories that we wish to store, G, is a proper subset of M_m.

Let R_m = the set of all realizable Boolean functions of the vectors $a_1, a_2, ..., a_m$. Then there is an injection between M_m and R_m. Further, the cardinality of M_m is given by

$$\frac{3^m}{2} < card(M_m) < \frac{M^{m^2}}{m!} \ll 2^{2^m}.$$

Proof :

Let x be an attractor. Then ,

$$sgn(Tx) = x; \tag{3.3}$$

$$sgn(\sum_{i=1}^{i=s} \frac{1}{N} a_i a^t{}_i x) = x; \tag{3.4}$$

$$sgn(\sum_{i=1}^{i=s} \alpha_i a_i) = x; \tag{3.5}$$

Hence the argument of the signum function is the orthogonal projection of x over the space spanned by the eigenvectors of T that correspond to non-zero eigenvalues. T is also called a *projective matrix* for this reason.

If we define an orthant of x , O(x) to be the region of R^n that satisfies sgn(a_1) = sgn (x_1), sgn(a_2) = sgn (x_2), and so on, then ths gn function can be characterized as the mapping of a vector into the vertex of the N-cube in that shares the same orthant as the vector.

Observation 1: A necessary and sufficient condition for x to be an attractor is that the orthogonal projection of x over the space spanned by the eigenvectors of T corresponding to the non-zero eigenvalues also lies in the orthant of x.

Observation 2: $sgn(Tx) = x$ if and only if projection of x on S $\varepsilon O(x)$;

where S is the space spanned by the eigenvectors of T corresponding to the non-zero eigenvalues.

Let $Q_m = \{ x: S \cap O(x) \neq 0\}$. Let $M_m = \{x: H(x) = x\}$, where H represents the Hopfield operator. Then, from observation 1, we conclude that $M_m \ C \ Q_m$. The following exercise illustrates how Q_m is determined.

Illustration :

Let us consider the case of 3 memories. (This example is borrowed from the work of Akra[5]).

$$T = \sum_{i=1}^{i=3} a_i a^t{}_i \qquad (3.6)$$

Let S = space spanned by the three vectors. To find the orthants that intersect S, it is sufficient to study the sign of the entries of the vector

$$sgn(\alpha a_1 + \beta a_2 + \gamma a_2)$$

which is the same as exploring the signs generated by

$$\alpha + \beta + \gamma$$

$$\alpha - \beta + \gamma$$

$$\alpha - \beta - \gamma$$

So the number of memories is the number of distinct signs of vectors we can generate using distinct choices of α, β and γ. From the literature on Threshold Logic[6], we learn that 4 planes passing through the origin divide R^3 into 14 regions such that all triplets (α,β,γ) lying in the same region generate the same vector of signs. Hence, we derive that

$Q_3 = \{ x : x = sgn(m_1a_1 + m_2a_2 + m_3a_3) \}$ where $(m_1, m_2, m_3) \varepsilon M$, M being defined by

M = { (1,0,0), (−1,0,0), (0,1,0), (0,−1,0), (0,0,1), (0,0,−1), (1,1,1)

(1,1,−1), (1,−1,1), (1,−1,−1), (−1,1,1), (−1,1,−1), (−1,−1,1), (−1,−1,−1)}.

M_m can be determined by using the fact that $M_m \ C \ Q_m$ to check if every vector of Q_m and see if it satisfies $T(x) = x$. However, we really need to check only two of the vectors, those corresponding to m = (1, 0, 0) and m = (1, 1, 1). Taking N = 4, a_1 = (1, 1, 1, 1),a_2 = (1, 1, −1, −1) and a_3 = (1,−1,−1,1), the attractors are a_1 and $sgn(a_1 + a_2 + a_3)$. therefore, for m = 3,

$M_m = Q_m = \{ x : x = sgn(m_1a_1 + m_2a_2 + m_3a_3)\}$. Thus we reach the important results that the attractors of H are the following:

$a_1, a_2, a_3, \overline{a_1}, \overline{a_2}, \overline{a_3},$

$a_1a_2 + a_3\ (a_1\ xor\ a_2), \overline{a_1}a_2 + a_3\ (\overline{a_1}\ xor\ a_2)$

$a_1\overline{a_2} + a_3\ (a_1\ xor\ \overline{a_2}), a_1a_2 + \overline{a_3}\ (a_1\ xor\ a_2)$

$\overline{a_1a_2} + a_3\ (\overline{a_1}\ xor\ \overline{a_2}), \overline{a_1}a_2 + \overline{a_3}\ (\overline{a_1}\ xor\ a_2)$

$a_1\overline{a_2} + \overline{a_3}\ (a_1\ xor\ a_2), \overline{a_1a_2} + \overline{a_3}\ (\overline{a_1}\ xor\ \overline{a_2})$,

the logical operations between the vectors being defined on an entry by entry basis and the *xor* operator refers to the logical exclusive OR operation. Thus the number of spurious equilibria is very large. The above example indicates that the number of equilibria is 14, no matter how large N is. Since the number of desired equilibria is 3, the remaining 11 equilibria are spurious equilibria. For a larger value of m, the number of attractors is extremely large. For m = 7, the number of attractors is 3, 548, 351. As m grows larger, the number of equilibria grows very rapidly.

3.4. Characterization of Spurious States

Consider a Hopfield network consisting of N neurons having threshold zero. Let $[T_{ij}]$ be the interconnection matrix of the network. Let m be the rank of T. Let $V_1, V_2, V_3, .. V_m$ be the eigenvectors of T corresponding to its non-zero eigenvalues. Let us assume that these vectors are constrained to be the corners of $(-1, 1)^N$. Further, let us assume that the corresponding eigenvalues are equal and positive. Let M_m be the set of attractors of the network. Then,

$card(M_1) = 2$, $card(M_2) = 4$, $Card(M_3) = 14$, and so on. (refer [5])

$$\text{In general, } \frac{3^m}{2} < card(M_m) < \frac{M^{m^2}}{m!} \ll 2^{2^m}. \qquad (3.7)$$

Let R_m be the set of all realizable Boolean functions of the m vectors. Since there is an injective mapping between M_m and R_m,

$$card(M_m) \leq card(R_m) \qquad (3.8)$$

It is known form threshold logic[6] that

$card(R_m) < \frac{2^{m^2}}{s!} \ll 2^{2^m}$, thus establishing the upper bound of M_m.

For the lower bound, note that for any m all vectors having an odd number of +1's or -1's and (m - p) zeros correspond to an attractor. The number of such attractors can be computed as follows :

$$\sum_{p=1}^{p=s} \binom{s}{p} 2^p = \frac{3^m - (-1)^m}{2} \text{ for odd p.} \qquad (3.9)$$

To summarize, the results mean that a space spanned by vectors of the n-cube intersect an orthant if and only if the N-cube element which lies in the orthant is a realizable Boolean function of these vectors.

3.5. Discussion

Amit et al[12] and Newman[13] have shown that in the case where $N \rightarrow \infty$, the spurious states correspond to mixtures of finite number of original patterns. Each such configuration, denoted $X(\nu)$ is given by:

$$X(\nu) = sgn(\nu_1 V_1 + \nu_2 V_2 + ... + \nu_m V_m) \qquad (3.10)$$

where $\nu=(\nu_1,\nu_2,..)$ is a fixed real vector independent of m with the following properties:

(i) ν has a fixed number of non-zero components.

(ii) $\pm\nu_1\pm\nu_2\pm\nu_3 \cdots \neq 0$ for any choice of $\pm$ signs.

(iii) For each non-zero ν_m the sum $\pm\nu_1\pm\nu_2\pm\nu_3 \cdots$ has the same sign as $\pm\nu_m$ for exactly a fraction $(1 + \nu_m)/2$ of all possible sign choices.

The threshold logic approach of enumerating the spurious states is an attempt to exhaust all the possible ν's . The above results hold for finite N as long as $\log_2 N \geq m - 1$ and the vectors are orthogonal to one another in the geometric sense.

3.6. Generality of the Results

1) The results hold even if the vectors a_i are not orthonormal. If the vectors are uncorrelated, i.e. $E(a_i^T a_j) = 0$ for $i \neq j$, the results hold if N is large enough.

2) It is not fundamental to the results that T be a sum of outer products matrix. It is only required that T be symmetric. Since any N x N symmetric matrix has N orthonormal eigenvectors, a_i, $i = 1,2,..N$, T can be written as

$$T = \sum_{i=1}^{i=N} \lambda_i a_i \frac{a_i^T}{N}$$

3) If T is asymmetric then the complex eigenvalues may cause the output to exhibit oscillatory behavior.

4) It is not necessary that the eigenvalues of T be equal. The only restriction placed on the eigenvectors of T is that they belong to the N-cube. This constraint helps establish a lower bound for the number of equilibria, as we saw earlier.

5) Since the fixed points of sgn(T) remain the same whether we update one neuron at a time or all of them, the synchronous and asynchronous models behave similarly. If the network is operated in continuous space mode, the sigmoid function that would be used will gradually shrink during the operation of the network to force the output of the state

to be on the N-cube. This is discussed by Hopfield[ ]. If the network is operated with stochastic transitions where the probability of changing the neuron state is given by figure 2., the number of spurious states is reduced significantly at high temperature (the probability of the transition in state of each unit is given by $\left[\frac{1}{1+e^{-\Delta E_k/T}}\right]$ and T is a parameter analogous to temperature and this corresponds to the long transition region in figure 2). However, as the network starts to cool, other spurious states start to appear and for T = 0, all of them are present. In order to heat the network at start and thus produce correct outputs with very high probability, the number of neurons required is very large, approximately m/0.15 [5]. The number of interconnections involved make this impractical from an implementation perspective.

6) In order to push the spurious states away from the desired memories, the number of neurons required is very large, which again leads to difficulty in implementation.

7) Newman[13] has proved that the spurious equilibria have energy barriers for sufficiently small $\alpha = \lim m/N$.

Applying Simulated Annealing to Hopfield nets :

Rumelhart and McClelland[10] have proposed a simple modification to the Hopfield update rule to implement simulated annealing. If the energy gap between the 1 and 0 states of the kth unit is ΔE_k then, regardless of the previous state set, a_k is 1 with probability

$p_k = \left[\frac{1}{1+e^{-\Delta E_k/T}}\right]$ where T is the parameter that is analogous to the temperature of a physical system. This corresponds to our discussion above on the stochastic models and its limitations.

Hopfield has also applied his networks to solve optimization problems[8]. Our discussion clearly indicate why these networks do not succeed in their goal of finding a global minimum or optimum solution. Not only do they get stuck in local mini-

ma, but also there is a possibility of encountering a large number of spurious equilibria. Applying simulated annealing is not an altogether satisfactory solution to the problem of spurious equilibria, primarily because of the difficulties in implementing such a network.

CHAPTER IV

BACK PROPAGATION:

4.1. The Problem

The networks models we have seen so far are two-layered association networks in which a set of input vectors arriving at the '"input" layer are mapped to a set of output vectors at an "output" layer. These networks do not have any *hidden* units. This implies that there is no internal representation for the things we wish to remember. The essential characteristic of these networks is that they map similar input patterns to similar output patterns. This could lead to an inability to learn certain mappings from input to output. Whenever the similarity structures of the input and output patterns are very different a network which does not have an internal representation will be unable to accomplish the necessary mapping. This is exemplified by the classic case of not being able to build the XOR function using a single threshold element. Minsky and Papert[11] have analysed the conditions under which two layer networks are capable of performing the required input-output mappings. They also pointed out that with a single layer of simple perceptron-like units, as in figure 3. with which the input layer can be augmented, any mapping between the input and output units can be supported.

If we have the right connections between the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from the input to the output units. To the output unit, the hidden unit is equivalent to another input unit. The problem, is that whereas a very simple guaranteed learning rule that applies to all problems that can be solved without hidden units, there is no equally powerful learning rule for networks with hidden units. There have been three approaches, as it were, to this problem. One of the learning procedures is competitive learning in which unsupervised learning rules are employed in order to make useful hidden units ap-

pear. The obvious drawback of this method is that there is no external force to ensure that hidden units appropriate for the desired mapping are developed. The second response is to assume an internal representation, based on *a priori* reasoning. The third is attempting to develop a learning procedure capable of learning an internal representation that is adequate for accomplishing a specified task. One such procedure involves the use of stochastic units, requires the network to reach equilibrium in two different phases and is limited to symmetric networks. There is also an alternative approach that uses deterministic units and involves only local units. This is called the generalized delta rule.

4.2. Description of a multi-layered network

The Hopfiled network can be viewed as a two-layer neural network with each network having n threshold units. The connection between the two layers is given by the symmetric matrix T. For the standard autoassociative Hopfield model, which we have discussed earlier, the output of the second layer is fed back into the first layer and the elements of T are given by the Hebbian(outer product) rule. The change in weights ΔT_{ij} in a very general form can be thought of as a function of

1. The input to the ith unit,
2. A teaching input to the ith unit,
3. The output of the jth unit and
4. The strength of the connection between the ith and jth units, w_{ij}.

In its simplest version, the Hebbian learning rule is given using only 1 and 3 above :

$$\Delta w_{ij} = \eta a_i o_j$$

where η is a constant of proportionality representing the rate of learning and a_i and o_j are the input to the ith unit and output of the jth unit respectively.

A multi-layered network, on the contrary, has several layers of units, each layer being a set of units whose relationship with the units belonging to another layer being different from its relationship to units in the same layer. Thus a layer is a logical concept (and not necessarily a physical one). We consider three-layered networks here.

Most of the results can be easily extended to networks with more than three layers. In the analysis that follows the input vectors are represented by i_i, and the output units by o_i. The weight between any two units, i and j is given by w_{ij}. We present several patterns to the network, in order to make it "learn". The target or desired output of the jth output unit on the presentation of the kth pattern is represented by t_{kj}. The output of the jth unit on the presentation of the kth pattern is represented by o_{kj} and the jth component of the kth input vector is represented as i_{kj}. Each of the units is a threshold unit like the one described in Chapter III.

4.3. The Generalized Delta Rule

To begin with we apply the procedure to a two-layer network. The learning procedure involves presenting the system with pairs of input and output patterns. The system uses the first input pattern to to produce an output vector, and compares this with the desired target vector. If there is no difference, no "learning" takes place. Otherwise, the weights are altered so as to reduce the difference. The rule for changing the weights after the kth input, output vector pair , input vector i_k and output vector o_k is given by

$$\Delta_k w_{ij} = \eta(t_{kj} - o_{kj})i_{ki} = \eta\delta_{kj} i_{ki} \qquad (4.1)$$

where t_{kj} = target input for jth component of the output pattern k.

o_{kj} = jth element of actual output pattern produced by input pattern k.

i_{ki} = value of the ith element of the input pattern.

$\delta_{kj} = t_{kj} - o_{kj}$

$\Delta_k w_{ij}$ = change to be made to the strength of the connection between the ith and jth units following the presentation of the pattern k.

There are many ways of deriving the the generalized delta rule. It is useful to observe that the method minimizes the squares of the differences between the actual and desired output vectors summed over the output vectors and all pairs of input/output vectors. One way of establishing this is to show that the derivative of the error measure with respect to each weight is proportional to the weight change dictated by the delta rule, the

constant of proportionality being negative. This corresponds to performing the steepest descent on a surface in weight space whose height at any point in the weight space is a measure of the error [10]. The delta rule essentially implements gradient descent in sum-squared error for linear(as opposed to sigmoid) activation functions. Without any hidden units, since the error surface is concave upwards, gradient descent is guaranteed to find the best set of weights. However, with hidden units, there is a possibility of getting stuck in local minima. The important observation to be made is that the problem of local minima is not relevant to a wide range of learning tasks. Also, since hidden units with linear activation functions provide no advantage, it is important to generalize the analysis to the case of nonlinear activation functions. The nonlinear activation function must meet the requirements that it be a nondecreasing and differentiable function of the net total output $net_{kj} = \sum_{i=1}^{i=n} w_{ji} p_{ki}$ where $o_i = i_i$ if unit i is an input unit. Thus the nonlinear activation function results in the following dynamics:

$$o_{kj} = f_j(net_{kj}) \tag{4.2}$$

where f is differentiable and nondecreasing. In order to generalize the delta rule, set

$$\Delta_k w_{ji} \ \alpha \ -\frac{\partial E_k}{\partial w_{ji}} \tag{4.3}$$

where E, as defined earlier, is the sum squared error function described earlier.

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial net_{kj}} \frac{\partial net_{kj}}{\partial w_{ji}} \tag{4.4}$$

By the definition of net_{kj} ,

$$\frac{\partial net_{kj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_{k=1}^{k=n} w_{kj} o_{ki} = o_{ki} \ .$$

Defining $\frac{-\partial E_p}{\partial net_{kj}} = \delta_{kj} o_{ki}$ This implies that to implement gradient descent in E, the weight changes should be made according to

$$\Delta_k w_{ji} = \eta \delta_{kj} o_{ki} \tag{4.5}$$

The only problem remaining is what δ_{kj} should be for *each unit* u_j of the network. It is

easy to establish that a simple recursive computation of the δ's can be implemented by propagating error signals backward through the network [10]. The results can be summarized as follows: the weight on each line should be changed by an amount proportional to the product of an error signal, δ , available to the unit receiving input along that line and output of the unit sending activation along that line. In symbols,

$$\Delta_k w_{ji} = \eta \delta_{kj} o_{ki} \qquad (4.6)$$

where $\delta_{kj} = (t_{kj} - o_{kj}) f'_j(net_{kj})$

$f'_j(net_{kj})$ being the derivative of the activation function which maps the total input of the unit to an output value. The error signal for the hidden units for which there is no specified target is determined recursively in terms of the error units to which it is directly connected. That is,

$$\delta_{kj} = f'_j(net_{kj}) \sum_{m=1}^{m=n} \delta_{km} w_{mj} \qquad (4.7)$$

whenever the unit is not an output unit.

The application of the generalized delta rule, thus involves two phases. In the first phase the input is presented and propagated forward through the network to compute the output value for each unit. The output is compared with the target to obtain an error signal for each output unit. In the second phase the error is passed back to each unit in the network and the appropriate weight changes are made. The second phase allows a recursive computation of δ. The δ's are first computed for all the units that are in the output layer. The weights of all the connections that feed into the output layer are changed accordingly. Then the δ 's for the units in the penultimate layer are computed. The same process is repeated for every layer of the network. Note that it is not required that we modify all the weights. It is possible to keep any number of weights fixed. In this case, the error is still propagated backwards, but the fixed weights are not modified. Also, output units can receive inputs from other output units in earlier layers. Such units receive two different kinds of error, one from direct comparison with the target vector and the other passed through the activation units they are connected to. The two weight

changes are summed to obtain the weight change corresponding to the two error signals.

4.4. The Activation Function for back propagation

The derivation of the learning rule requires that the derivative of the activation function exists. Also, it is necessary to have a nonlinear differentiable activation function since a linear function does not achieve any advantage from the hidden units. Let us consider an activation function of the form

$$a_{ki} = \frac{1}{1+e^{-net_{ki}}} \tag{4.8}$$

In order to apply the learning rule it is necessary to establish that the derivative of a_{ki} with respect to net_{ki} exists.

$$\frac{da_{ki}}{dnet_{ki}} = a_{ki}(1-a_{ki}) \tag{4.9}$$

Thus the error signal, δ_{ki}, is given by

$$\delta_{ki} = (t_{ki} - a_{ki})a_{ki}(1 - a_{ki}).$$

Note that the derivative reaches its maximum at $a_{ki} = 0.5$ and since $0 \leq a_{ki} \leq 1$, it reaches its maximum value as a_k approaches 0 or 1. Since the amount of change in the weights is proportional to the derivative, weights are changed most for those units that are close to their midrange, and in a sense , not committed to be either on or off.

4.5. Discussion

The learning procedure requires only that that the change in weight be proportional to the weight error derivative. True gradient descent requires that infinitesimal steps be taken. The constant of proportionality, ε, is the learning rate in the procedure and the larger it is, the greater the change in weights. For practical purposes, it is important to choose as large a learning rate as possible without leading to oscillation in order to achieve the most rapid learning. One way to increase the learning rate without leading to oscillations is to modify the back propagation learning rule to include an extra term that is analogous to momentum. This can be accomplished as follows:

$$\Delta w_{ij}(n+1) = \varepsilon(\delta_{ki} a_{kj}) + \alpha\Delta w_{ij}(n) \tag{4.10}$$

where n indexes the presentation number of the pattern and α is a constant that determines the effect of past weight changes on the current weight change. Thus it provides a momentum in weight space that filters out high frequency variations of the error surface in weight space.

The learning procedure has one more problem. If all the weights start out with equal values and if the solution requires that unequal weights be developed then the system will never learn, This is because the error is propagated back through the weights in proportion to the values of the weights. All the hidden units that are connected directly to output units will thus get identical weight changes and the weights from these units to the output units will thus remain the same. The system thus starts out in an unstable equilibrium point and keeps the weights equal. In order to counteract this problem the system must be started with small random weights.

The derivation of the back propagation rule supposes that we are taking the derivative of the error function summed over all the patterns. Hence it is possible that we may present all the patterns and then sum the derivatives before changing the weights. Instead, we can change the weights after we present each pattern. For a small learning rate, it is easy to establish by simulations that the difference in the two approaches is negligible. However for a very large number of patterns the approach of changing weights after the presentation of each pattern is more useful.

4.6. Extensions of Back Propagation

We have analysed back propagation in one-pass, feedforward networks. Here we discuss some extensions of the method in an attempt to make the scheme more recurrent.

4.6.a Cascaded Feedforward Networks

These networks preserve the desirable computational characteristics of the back propagation scheme but combines with them a gradual build-up of the activation. Here, the net input to each unit is given by

$$net_{ir}(t) = k_r \sum_{j=1}^{j=n} w_{ij} a_{js}(t) + (1 - k_r) net_{ir}(t-1) \qquad (4.11)$$

and the equation for its activation is

$$a_{ir}(t) = \frac{i}{1 + e^{-net_{ir}(t)}} \qquad (4.12)$$

In the implementation of the scheme there is a single parameter, called the cascade rate, that is used for all units instead of a separate rate parameter for each level. One advantage of this scheme is that if an input pattern comes on at time t=0 and stays on, the asymptotic activation each unit reaches is identical to the activation it reaches in a single step in the one-pass feedforward computation. Thus the one-pass network in fact computes the asymptotic activations that would result from a process that may in real systems be gradual and continuous. The network can then be trained to achieve a particular asymptotic activation value for each of several patterns. Another feature of such networks is that their dynamical properties depend on the initial state. It may be assumed that the initial state is the pattern of activation resulting from an input pattern consisting of all non-zero activations for all the input units. In order for this to work the network must be trained to produce an appropriate output state for this initial input state.

4.6.b Recurrent Networks

It is easy to establish that for every recurrent network, there is a feedforward network with identical behavior (over a finite period of time). However, the behavior of a recurrent network can be achieved in a feedforward network at the cost of duplicating the hardware many times over for the feedforward version of the network. Hence the advantages of building a recurrent network. We have distinct units and distinct weights for each point in time. As long as we constrain the weights at each level of the feedforward network to be the same, we have a recurrent network which performs identically with the feedforward network(refer to figures 4 & 5). The appropriate method for maintaining the constraint that all weights be equal is to simply keep track of the changes dictated for each weight at each level and then alter each of the weights by the sum of the required

changes. The general rule for determining the change prescribed for a weight in the system foe a particular time is to take the product of an appropriate error measure δ and the input along along the relevant line. Thus the problem reduces to computing the correct δ for each time. In a feedforward network, δ is determined by multiplying the derivative of the activation function by the sum of the δ's for those units it feed into weighted by the connection strengths. The same procedure works for recurrent networks, except that in this case, the value of δ associated with a particular unit changes in time as a unit passes error back, sometimes to itself. After each iteration, as error is being passed back through the network, the change in weight for that iteration must be added to the weight changes specified by the preceding iterations and the sum stored. This process of passing error through the network should continue for a number of iterations equal to the number of iterations through which the activation was originally passed. And at this point, the changes to all the weights are made.

Summary and Conclusions

We have examined two layer networks and found that their main limitations are that they can learn only certain classes of input/output patterns and that they end up learning a large number of "spurious" patterns. In order to increase the reliability of such networks in recalling the stored patterns it is necessary to have a network with large number of units in proportion to the size of the pattern they learn. They also have the problem that some initial states may get stuck in local minima.

The back propagation approach overcomes some of the limitations that are inherent to two-layer networks. By learning internal representations for patterns, they are able to learn a considerably wider range of patterns than two-layer networks. However, they also have the problem of learning extra or spurious patterns and have local minima. However, for a large variety of problems the local minima are not really a problem. Further, it is possible to obtain a recurrent implementation of the back propagation model and this is extremely useful for implementation.

Most artificial neural network models that have been developed have a common characteristic which is that they perform very well for a particular class of problems. And even so, it cannot be guaranteed that they will produce the correct solution in a new instance even after a long learning or training period. In this sense, they are a far cry from biological neural networks. And because it is not possible to guarantee that they will produce the correct solution every time, they are not entirely suitable as a tool for computation. However, many such networks have been implemented as VLSI chips and they are a very useful tool to augment the conventional tools for such problems as pattern association.

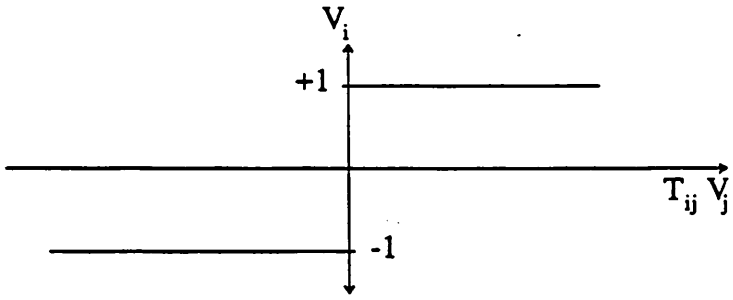

Figure 1: The I/O Characteristics of a neuron in the discrete space mode

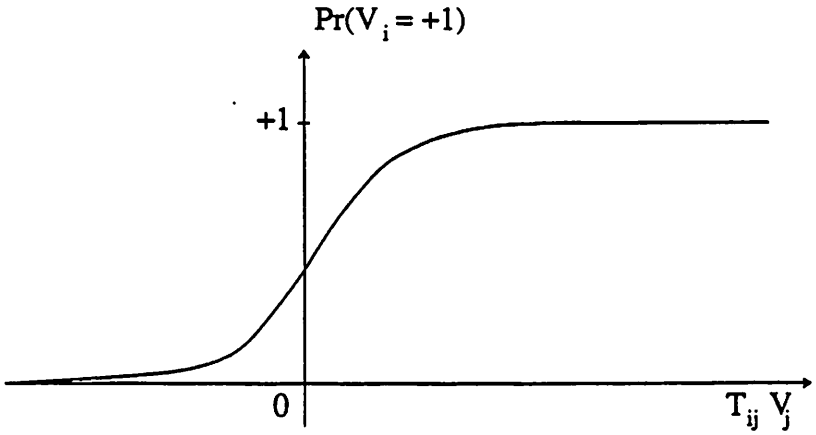

Figure 2: Transition probabilities of a neuron in the stochastic mode

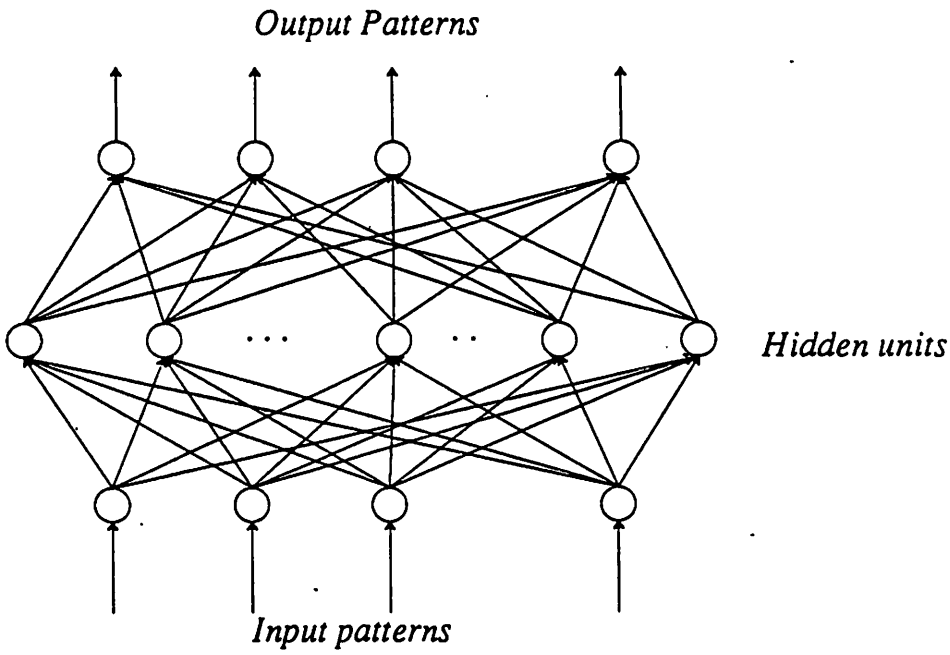

Figure 3: A multilayer network with internal representatiion units

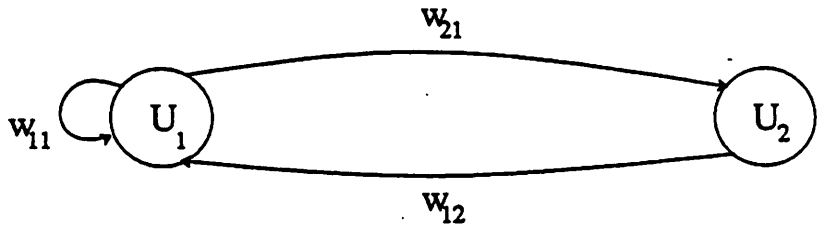

Figure 4: A completely connected two-unit recurrent network

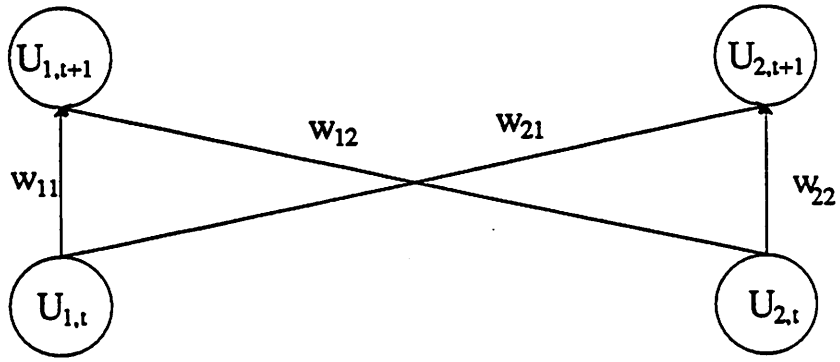

Figure 5: An equivalent feedforward network

REFERENCES

[1] Hopfield, J.J.(1982). *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences, USA, 79, *pp 2554-8.*

[2] Hopfield, J.J.(1984). *Neurons with graded response have collective properties like those of two state neurons*, Proceedings of the National Academy of Sciences, **81**, *pp. 3088-92.*

[3] Cohen, M. and Grossberg,S.(1983). *Absolute stability of global pattern formation and and parallel memory storage by competitive neural networks*, IEEE Transactions on Systems, Man and Cybernetics, **SMC-13**, *pp 815-26.*

[4] Michel, A.N., Li,J. and Porod, W.(1988). *Qualitative analysis and synthesis of a class of neural networks*, IEEE Transactions on Circuits and Systems, **35, no.8**, *pp 976-86.*

[5] Akra, M.A.(1988). *On the analysis of the Hopfield network: A geometric approach*, M.S.Thesis, MIT, Spring 1988.

[6] Lewis, P.M. and Coates, C.L.(1967). *Threshold Logic*, John Wiley and Sons Inc., New York .

[7] McEliece, R.J. et al(1987). *The capacity of the Hopfield associative memory*, IEEE Transactions on Information Theory, **IT-33, no.4**, *pp 461-82.*

[8] Hopfield, J.J. and Tank, D.W.(1985). *Neural computation of decisions in optimization problems*, Biological Cybernetics, **52**, *pp 141-52.*

[9] Goodwin, C.G. and Sin, K.S.(1984). *Adaptive Control and Filtering*, Prentice-Hall Inc., New Jersey.

[10] Rumelhart, J.E. and McClelland, J.L.(1986) *Parallel Distributed Processing: Explo rations in the microstructure of Cognition*, MIT Press, Masachussets.

[11] Minsky, M. and Papert, S.(1969). *Perceptrons*, MIT Press, Cambridge, Masachusetts.

[12] Amit, D.J. et al(1986). *Spin-glass models of neural networks*, Physics Review, A32, *pp 1007-18.*

[13] Newman, C.M., *Memory capacity in neural network models: Rigorous lower bounds*, Preprint available from author. Prof. Newman is affiliated with the Dept. of Math, University of Arizona.

[14] Carpenter, G.A., Cohen, M. and Grossberg, S.(1987). *Computing with neural networks*, Science, **235**, *pp 1226-7.*

[15] Kohonen, T.(1984). *Self-organization and associative memory*, Springer-Verlag, Berlin.

[16] Lippman, R.P.(1987). *An Introduction to computing with neural nets*, IEEE ASSP magazine, *pp 4-22, April 1987.*

[17] Rudin, W.(1960). *Principles of Mathematical Analysis, McGraw-Hill, New York.*