

On the Analysis of The Hopfield Network: A Geometric Approach.

by

Mohamad A. Akra

B.E. American University of Beirut  
(1986)

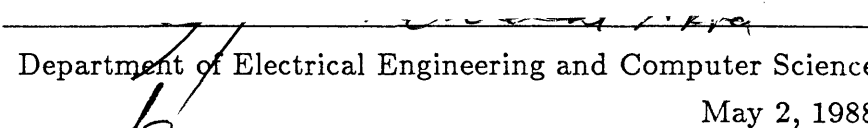
Submitted in Partial Fulfillment  
of the Requirements for the  
Degree of

Master of Science  
in Electrical Engineering and Computer Science


at the

Massachusetts Institute of Technology  
May 1988

© Massachusetts Institute of Technology 1988

Signature of Author   
Department of Electrical Engineering and Computer Science  
May 2, 1988

Certified by   
Sanjoy K. Mitter  
Thesis Supervisor

Accepted by   
Arthur C. Smith  
Chairman, Departmental Committee on Graduate Students

ARCHIVES  
MASSACHUSETTS INSTITUTE  
OF TECHNOLOGY

JUL 26 1988

LIBRARIES

**On The Analysis of The Hopfield Network:  
A Geometric Approach**

by

Mohamad A. Akra

Submitted to the  
Department of Electrical Engineering and Computer Science  
on May 20, 1988 in partial fulfillment of the requirements  
for the Degree of Master of Science

**Abstract**

Artificial neural networks have been studied for many years in the hope of achieving human-like performance in the fields of speech and image recognition. A great landmark in this field is the Hopfield Network. It is simple to implement, simple to analyze. However, it suffers from the existence of the so-called spurious states which tend to inundate the space and reduce the value of this network. In this thesis we provide a complete analysis of the spurious states. This analysis involves the number of such states, their distribution in the space of states, an experimental study of their basins of attractions, and finally, most importantly, their equations.

We also describe an algorithm that takes a Hopfield network and replaces it by an equivalent one but with less internal connections. The importance of this algorithm is obvious when it comes to the issues of implementation using VLSI.

Finally, we comment on this network and discuss its usefulness as a Content-Addressable-Memory.

Thesis Supervisor: Sanjoy K. Mitter

Title: Professor of Electrical Engineering, Department of Electrical Engineering  
and Computer Science

# Contents

<b>1</b>	<b>Neural Networks : A Survey</b>	<b>1</b>
1.1	History . . . . .	1
1.2	Models . . . . .	2
1.2.1	The Actual Nerve Cells . . . . .	2
1.2.2	The Artificial Nets . . . . .	4
1.3	Thesis Overview . . . . .	5
<b>2</b>	<b>The Hopfield Network</b>	<b>6</b>
2.1	The Basic Model . . . . .	6
2.2	Modes of Operation . . . . .	8
2.3	Properties of Hopfield Nets . . . . .	11
2.4	Limitations of the Network . . . . .	12
<b>3</b>	<b>The Main Problem</b>	<b>14</b>
3.1	Formulation . . . . .	14
3.1.1	Content-Addressable-Memory . . . . .	14
3.1.2	Hopfield's Approach . . . . .	15
3.2	Solution: Characterization of Spurious States . . . . .	16
3.2.1	Main Theorem: . . . . .	16
3.2.2	Example: $s=3$ . . . . .	17
3.2.3	Remarks . . . . .	18
3.2.4	Proof . . . . .	18
3.2.5	Experimental Results . . . . .	26
3.3	Discussion . . . . .	29
3.3.1	Relation to other's work . . . . .	29

3.3.2	Impact of our work . . . . .	29
3.3.3	Generality of Results . . . . .	30
3.4	An interesting Algorithm . . . . .	32
3.4.1	The Basic Idea . . . . .	33
3.4.2	Description of The Algorithm . . . . .	33
3.4.3	Example . . . . .	34
3.4.4	Comments . . . . .	35
3.5	Summary . . . . .	35
<b>4</b>	<b>Threshold Logic</b>	<b>37</b>
4.1	Definitions . . . . .	37
4.2	Number of Threshold Functions . . . . .	41
4.3	Characterization of Threshold Functions . . . . .	43
<b>A</b>	<b>List of Realizable functions</b>	<b>46</b>
A.1	$s = 3$ . . . . .	46
A.2	$s = 4$ . . . . .	46
A.3	$s = 5$ . . . . .	46
A.4	$s = 6$ . . . . .	47
A.5	$s = 7$ . . . . .	48
<b>B</b>	<b>Computer Programs</b>	<b>53</b>

## List of Figures

2.1	Two typical neurons $i$ and $j$ . . . . .	7
2.2	I/O Characteristic of a neuron: <b>DSDT</b> mode . . . . .	8
2.3	I/O characteristic of a neuron: <b>CSDT</b> mode . . . . .	8
2.4	Transition probabilities of a neuron: <b>DSST</b> mode . . . . .	9
3.1	Cube divided into 14 pieces by 4 planes passing through its center .	21
4.1	Space of Thresholds for one variable . . . . .	40
4.2	Unrealizability of the xor function . . . . .	44

# List of Tables

3.1	Number of attractors and spurious states . . . . .	17
3.2	$V_1, V_2$ and $V_3$ in canonical form . . . . .	20
4.1	Number of threshold functions . . . . .	44

## Acknowledgments

I would like to express my deep thanks to my Lord who guided me in this life and directed me towards the right path. Without His mercy, compassion and guidance this life would have been mere misery and desperation.

Also, my sincere appreciation goes to Professor Sanjoy Mitter who was for me much more than a research supervisor. His continuous support and cooperation, his understanding, his insights and ideas, his time and effort he has put into this thesis; all of this was undoubtedly of invaluable help to me during my stay at M.I.T. It has been a pleasure for me to work with the Director of the Center for Intelligent Control Systems.

I am also grateful to my fellow graduate student Tom Luo for his continuous willingness to discuss ideas and suggest questions and to check the validity of my proofs.

Credit should also go to Tom Richardson, my other fellow graduate student, for his cooperation and useful comments.

My parents deserve more than what I can express for being there when I needed them, for helping out in any way they could, both financially and spiritually, and for all their love, compassion, guidance and encouragement throughout my life.

Finally, I would like to express my deep gratitudes to my brothers Hasan, Waseem, Mazen, Anas, Isam, Khaled, Rifat, Nabil, Bassim, Jamal, Walid, Emre and Nasser for the environment they provided for me to have the peace of mind needed to survive at M.I.T.

This research was performed at the M.I.T. Center for Intelligent Control Systems and was supported in part by the U.S. Army Research Office under grant DAAAL-03-86-K0171 and by the Air Force of Sponsored Research under grant AFOSR 85-0227B.

# Chapter 1

## Neural Networks : A Survey

### 1.1 History

Artificial neural net models have been studied for many years with the hope of understanding certain essential features of the human central nervous system, and thereby incorporating these features in computational systems [16].

Some of these features are :

- Pattern recognition.
- Classification.
- Learning.
- Extraction of concepts or rules from instances.
- Adaptive computation.[2]

The definition of what some of these features really mean is a difficult task.

The most influential idea was the Perceptron, which ruled for about 12 years (1957–1969). First proposed by Rosenblatt, it was shown later by Minsky and Pappert[19] to have certain limitations.

In the sixties and early seventies, Caianiello and Little were independently attempting to model Mc.Culloch-Pitts neural networks within the physics community.[17, 18].

Grossberg, in 1968, started a large research effort that is still in progress. His modeling project has engaged in studying a network of elements which are intended to map more or less faithfully cortical neurons. A wealth of papers and books have



been published and an interesting collection of psychological results [4] — such as the development of the Adaptive Resonance Theory : ART — has been announced.

Finally in 1982, J. J. Hopfield began a research program into neural networks and rekindled interest in them by his extensive work [9,10,11] on different versions of the Hopfield net. This interest arose because of the simplicity of the network as well as some other interesting properties. His net can be used as an associative memory, for instance, or to solve optimization problems, as we will elaborate later [11,14].

## 1.2 Models

### 1.2.1 The Actual Nerve Cells

Neurons, or nerve cells, are the building blocks of the brain. Although they have the same genes, the same general organization and the same biochemical apparatus as other cells, they also have unique features that make the brain function in a different way from, say, the liver [24]. The important specializations of the neuron include a distinctive cell shape, an outer membrane capable of generating nerve impulses, and a unique structure, the synapse, for transferring information from one neuron to the next. The human brain is thought to consist of  $10^{11}$  neurons, about the same number of stars in our galaxy. No two neurons are identical in form. Nevertheless, their forms generally fall into only a few broad categories, and most neurons share certain structural features that make it possible to distinguish three regions of the cell: the cell body, the dendrites and the axon. The functioning of the brain depends on the flow of information through elaborate circuits consisting of networks of neurons. Information is transferred from one cell to another at specialized points of contact: the synapses. A typical neuron may have anywhere from 1,000 to 10,000 synapses and may receive information from something like 1,000 other neurons. Although neurons are the building blocks of the brain, they are not the only kind of cells in it. A major class of cells in the central nervous system is the glia cells or glia, which provide structural and metabolic support for the delicate meshwork of the neurons [12]. Although the human brain is the most complex of all known systems in the universe, some facts are known ( or thought to be ) about its operation. As a matter of fact, neuroscientists model the neuron

as follows [8] :

1. **The internal state** of a neuron is characterized by an electrical potential difference across the cell membrane at the *axon hillock*. This potential difference is called the *generating potential*. External inputs produce deviations in this potential from a baseline *resting potential* ( typically between 70 and 100 mv ). When the generating potential exceeds a certain *threshold potential*, an *action potential* is generated at the hillock and propagates away from the hillock along the axon.
2. **Axonal Signals:** The *action potential* is a large depolarizing signal ( with amplitude up to 110 mv ) of brief duration ( 1–10 ms ). In a given neuron, every action potential travels with the same constant velocity ( typically between 10 and 100 m/s ) and undiminished amplitude along all axon collaterals ( branches ) to their terminal *synaptic knobs*.

Axonal signals are emitted in bursts of evenly spaced action potentials with pulse frequencies typically in the range between 2 and 400 Hz for cortical pyramid cells, or 2 and 100 Hz for retinal ganglion cells. Single spike potentials are also spontaneously emitted. A single spike is not believed to carry information. It appears that all the information in an axonal signal reside in the pulse frequency of the burst. Thus, the signal can be represented by a positive real number in a limited interval.

3. **Synaptic Inputs and Outputs:** The flow of signals in and out of a neuron is unidirectional. A neuron receives signals from other neurons at points of contact on its dendrites or cell body known as *synapses*. A typical pyramid cell in the cerebral cortex receives input from about  $10^5$  different synapses. When an incoming axonal signal reaches the synaptic knob it induces the release of a substance called a *neurotransmitter* from small storage vesicles. The released transmitter diffuses across the small synaptic gap to the post synaptic cell where it alters the local *receptor potential* across the cell membrane. A synaptic input is either excitatory ( if it increases the receptor potential ) or inhibitory ( if it decreases it ), and inputs combine additively to drive a change in the generating potential.

4. **Summary:** Neurons are cells that are highly interconnected. Each cell sums the inputs coming to it (from the axons of other neurons) through its dendrites. If the sum exceeds a certain threshold level, a signal will be transmitted (through the axon) to the dendrites of the other neurons. Otherwise, nothing is sent out. Based on this observation, the reader will appreciate the artificial models presented in the next section.

### 1.2.2 The Artificial Nets

All the artificial neural net models that have been devised so far share the following common characteristics. They are composed of many nonlinear computational elements operating in parallel and arranged in patterns reminiscent of biological neural nets. Computational elements or nodes are connected via weights that are typically adapted during use to improve performance.

What caused the recent resurgence in this field ( after a long period of dormancy ) is the development of new net topologies and algorithms, new analog VLSI techniques, and some intriguing demonstrations, together with a growing fascination about the functioning of the human brain. Recent interest is also driven by the realization that human-like performance in the areas of speech and image recognition will require enormous amounts of processing. Neural nets provide one technique for obtaining the required processing capacity using large numbers of simple processing elements operating in parallel.

Although all the neural networks fall under the same category: Dynamical systems used for computational purposes, they differ in certain aspects. These aspects are:

- Dynamics: Synchronous or asynchronous update of the neurons.
- Connections: Weights, number of layers, etc...
- Input/Output: Continuous or binary.
- Nonlinearity: Hardlimiting, Sigmoid, etc...
- Weight adaptation: Different algorithms.

The Hopfield network will receive adequate discussion in the chapters to come. For a detailed study of other important landmarks in this field the reader is referred to [16], [4], [19], [13] and [1].

### **1.3 Thesis Overview**

The thesis is organized in the following way:

In chapter two we introduce the Hopfield network, describe its various modes of operation, discuss its properties and list some of its limitations.

In chapter three we formulate rigorously the problem of spurious states, provide a complete characterization of these states (e.g. their numbers, their equations, etc...), discuss the generality of our results and correlate with other available results in the field. In addition to this, we include an algorithm that replaces a Hopfield network by an equivalent one (same I/O behavior) but with less internal connections.

Chapter four is meant to provide the necessary background for this thesis and emphasize the ideas that are closely related to our work.

## Chapter 2

# The Hopfield Network

### 2.1 The Basic Model

Between 1982 and 1985, J. J. Hopfield came out with different versions of a new neural network. In his 1982 paper[9], Hopfield was looking for a network that would work as a Content-Addressable-Memory. The model he proposed constituted of N “Neurons”, the output of each was either  $V_i = 0$  (“not firing”) or  $V_i = 1$  (“firing at maximum rate”). Each neuron adjusted its state *asynchronously* setting

$$\begin{aligned} V_i &\rightarrow +1 && \text{if } \sum_j T_{ij}V_j > U_i \\ V_i &\rightarrow -1 && \text{if } \sum_j T_{ij}V_j < U_i \end{aligned} \quad (2.1)$$

with  $T_{ij}$  being the strength of the interconnection between neurons  $i$  and  $j$ , ( $T_{ij}=0$  if they are not connected ).

To illustrate further, for any two neurons  $i, j$  we have the representative diagram of figure (2.1).

He pointed out that the main features his model had over the perceptrons were: Back-coupling (i.e. output is fed back to input until convergence), exhibition of computational properties (whereas perceptrons were used only as classifiers his network could, in addition to that, solve some optimization problems), and finally, asynchronous operation.

In 1984, Hopfield modified his first version considerably by allowing continuous variations in the output rather than discrete ones. This was achieved by changing the I/O characteristics of each neuron from a simple step function to a sigmoid type of relation (see figures (2.2) and (2.3)). He tried to defend his new model

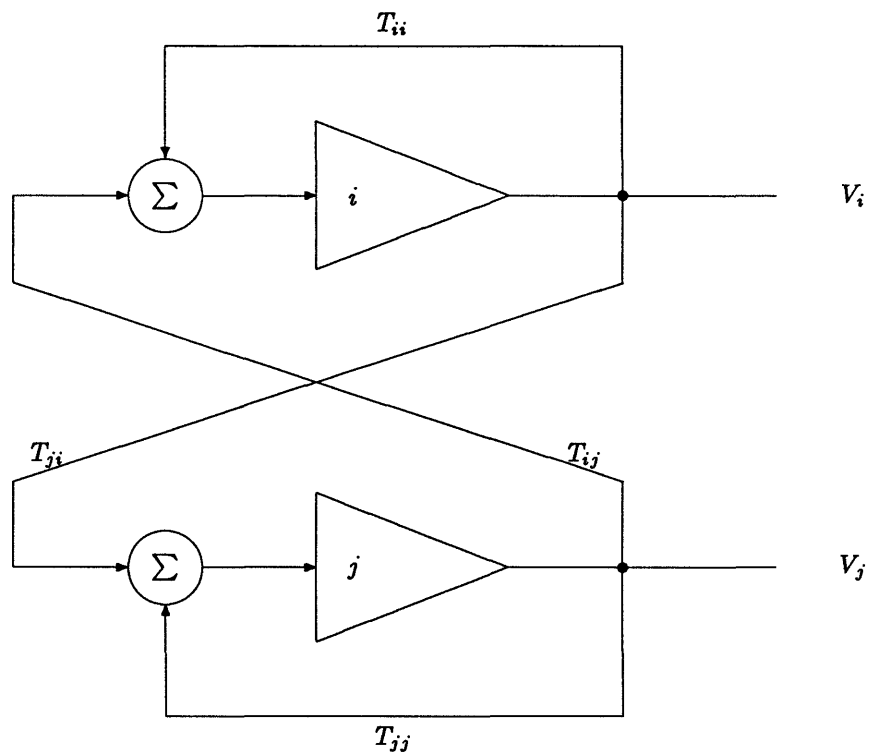


Figure 2.1: Two typical neurons  $i$  and  $j$

using some biological arguments, but also showed that if it were operating with a sigmoid relation close to a step function, then the stable points would be basically the same ones as the old model, thereby concluding that “the new continuous model supplements, rather than replaces, the old original stochastic description” [10]. A minor change to the continuous model was made in 1985, when Hopfield decided to make the output of a neuron vary continuously between  $-1$  and  $1$  instead of  $0$  and  $1$  thereby embedding some form of symmetry in the network space of states.

Later, several researchers (e.g. [2]) suggested a new version of the Hopfield net that had discrete state space ( $-1$  or  $1$ ) and where the transitions were not deterministic, but rather obey a certain probabilistic mechanism. In this new scheme, even if the sum of inputs exceeded the threshold value, the output would become  $1$  only with a certain probability (see figure (2.4)).

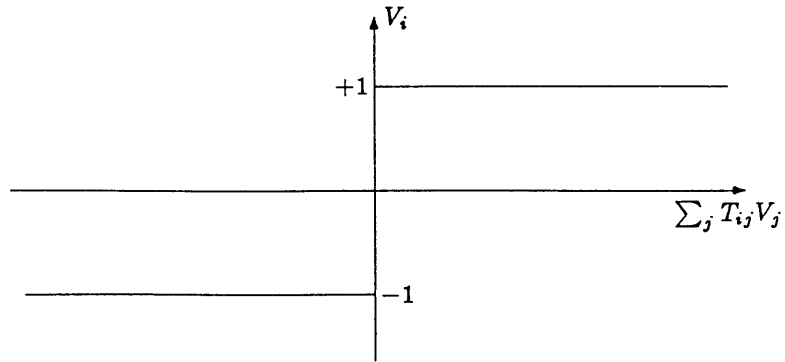


Figure 2.2: I/O Characteristic of a neuron: **DSDT** mode

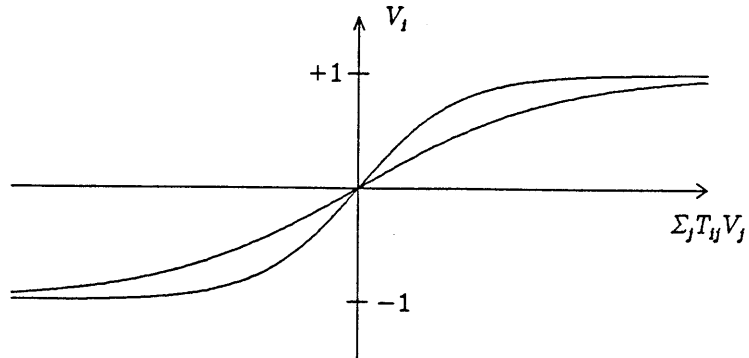


Figure 2.3: I/O characteristic of a neuron: **CSDT** mode

## 2.2 Modes of Operation

From our previous discussion of the different versions of the Hopfield net we can distinguish three types of dynamics.

1. **DSDT**: Discrete-Space, Deterministic-Transitions.
2. **CSDT**: Continuous-Space, Deterministic-Transitions.
3. **DSST**: Discrete-Space, Stochastic-Transitions.

In the following, we will describe in detail each of the various dynamics.

- **DSDT**: In this mode of operation, a neuron is chosen at random for state update. All neurons will have as output either  $-1$  or  $1$ . The chosen neuron

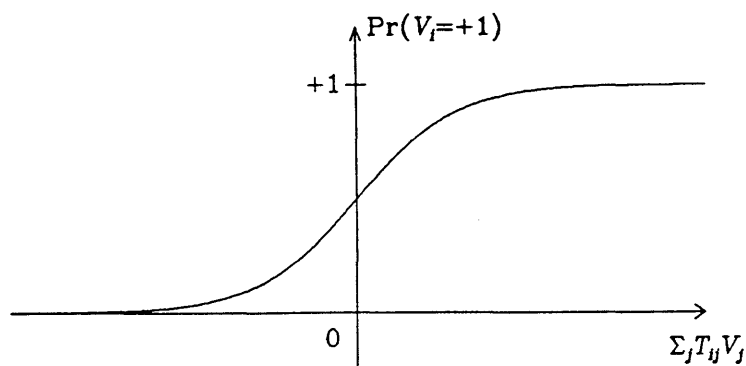


Figure 2.4: Transition probabilities of a neuron: **DSST** mode

will take the weighted sum of the output of all the neurons connected to it, with weights  $T_{ij}$  (connecting the  $j^{\text{th}}$  neuron to our  $i^{\text{th}}$  one) carefully predetermined. Now if the sum exceeds a given threshold  $U_i$  (which might vary from one neuron to another), the output takes the value  $\mathbf{1}$ , otherwise it goes to  $-\mathbf{1}$ . All of this can be written in equation form (see 2.1) repeated here for convenience.

$$\begin{aligned} V_i &\rightarrow +1 && \text{if } \sum_j T_{ij} V_j > U_i \\ V_i &\rightarrow -1 && \text{if } \sum_j T_{ij} V_j < U_i \end{aligned} \quad (2.2)$$

The I/O characteristics of each neuron can be modeled as a step function (see figure (2.2) for the case  $U_i = 0$ ).

To introduce some notational convenience, we will collect the output of all  $N$  neurons  $w_1, w_2, \dots, w_N$  in one vector  $\mathbf{w}$ . By doing this, we can now describe this mode of operation “at each clock cycle” as:

$$\mathbf{w}(k+1) = \text{sgn}(T\mathbf{w}(k) - \mathbf{U}) \quad (2.3)$$

with

$$\text{sgn}(x) = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}$$

where  $\mathbf{U}$  = vector of thresholds =  $(U_1, U_2, \dots, U_N)$ . However, for all practical purposes we can assume the thresholds to be equal since this yields to an easier VLSI implementation. In fact, most versions of the model have zero threshold.



Note also that, strictly speaking, only one entry of  $U$  is updated at a time. However, it will turn out that this distinction is immaterial for our work.

In the literature:  $T$ , the matrix of interconnection weights  $T_{ij}$ , is referred to as the matrix of “synaptic efficacies” by analogy with biology.

Depending upon the choice of  $T$ , the vector operation might have fixed points, as it might also exhibit some interesting features. Namely, by careful choice of  $T$ , one can let any starting initial vector  $w(0)$  converge after some time to one of a certain set of fixed points.

To illustrate with an example: Let  $T$  be

$$T = \begin{pmatrix} 1 & 3 & -1 \\ 2 & 1 & -2 \\ -1 & 0 & 3 \end{pmatrix}$$

chosen arbitrarily. Then  $T$  will have the vectors:

$$\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix}$$

as fixed points. More specifically,  $T$  will drive all other states to the attractors in the way described by the following schematic diagram:

$$\begin{array}{ccccccc} \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix} & \longrightarrow & \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix} & \longrightarrow & \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} & \longleftarrow & \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \longleftarrow \\ \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} & \longrightarrow & \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} & \longrightarrow & \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix} & \longleftarrow & \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \longleftarrow \end{array}$$

Note also that this  $T$  has a nice property: It takes each state to the closest attractor in Hamming distance measure. So, we can classify the network with the chosen  $T$  as a memory for 4 patterns, and starting with any unknown pattern containing partial information about one of the attractors it will converge to it, hence the name “*Content-Addressable-Memory*”.

- **CSDT:** Contrary to the previous mode, here all neurons will update their states simultaneously. Also, the output of each is not restricted to live in a discrete space, but rather is allowed to take continuous values between, say, 0 and 1. The major change being yielded in the I/O characteristics of a neuron which is now a sigmoid instead of the old step shape (see figures (2.2) and (2.3) for a comparison). Note, however, that even in this case, the final limiting states can be forced to lie on the corners of the hypercube by gradually shrinking the width of the linear region of the sigmoid function.
- **DSST:** This mode differs from the previous ones by the fact that equation (2.2) no longer holds, and transitions are random and follow a certain probability distribution. As illustrated in figure (2.4) the sum of inputs to each neuron will now only determine the probability that the output will go to 1 (or equivalently, to  $-1$ ).

## 2.3 Properties of Hopfield Nets

Hopfield studied extensively the model he proposed and concluded – after several computer experiments – that it could work as a content-addressable-memory [9]. As we have already illustrated in the previous section, a memory location is not addressed by an address but rather by incomplete information about its contents. For example, if the memory has the data 0111001 then a partial knowledge, such as 01\*1\*0\*, should be enough, for instance, to retrieve the full data.

In another part of his work[11], Hopfield used the network to provide “good” but “fast” solutions to such computationally hard problems like the *Travelling Salesman Problem*, thereby showing an important collective (i.e. parallel) computation feature of his network.

Finally, Hopfield used his network to simulate an A/D converter !

The probable reasons for the interest in the Hopfield Network are:

1. The internal connections remind us of the way neurons are connected in the brain.
2. It can be analyzed using statistical arguments.

3. It is a fail soft device since it will keep operating satisfactorily even when some of the wires are randomly disconnected.
4. It exhibits the feature of “learning”. For neurophysiologists, it is believed that the human being learns by modifying the brain synapses<sup>1</sup>. Since in the Hopfield network the synapses correspond to the interconnection weights  $T_{ij}$ , the ability to add memories by modifying these weights is an interesting property.
5. Finally, the computation is collective, and being able to obtain  $O(n^2)$  multiplication and  $O(n^2)$  addition instantaneously is a real saving in computation time. The type of computation that is being referred to is the multiplication by the matrix  $T$  (see equation (2.3)).

## 2.4 Limitations of the Network

So far we have not discussed any of the severe drawbacks associated with the network, some of which Hopfield himself mentioned in his paper[9]. The first and most restrictive drawback is the dramatically low number of memories it can handle. Using Hopfield’s algorithm to determine the interconnection weights  $T_{ij}$  for a network of  $N$  neurons, one can faithfully address by content no more than  $0.15N$  memories. Recall that the total possible number of states in this case is  $2^N$ . A technical question that arises is whether this drastic reduction in memory size has its origin in the algorithm Hopfield chose to find the  $T_{ij}$ ’s or is it really an inherent limitation of the network itself? The answer to this question will be of great importance and it is actually a fundamental one associated with neural networks.

Another limitation of the network is the appearance of little-understood spurious memories, those memories that represent patterns we did not intend to remember. Computer experiments have shown that they are huge in number. In my opinion, it is here where the real problem lies, and a better understanding of the occurrence of these states is essential to its use as an “associative memory”.<sup>2</sup>

Finally, the present model requires a relatively large chip area due to the numerous internal connections. It would be nice if one can find an algorithm which

---

<sup>1</sup>This is called “*Hebb’s rule*” for learning.

<sup>2</sup>another term for Content-Addressable-Memory.

will transform any matrix of weights  $[T_{ij}]$  into an “equivalent” one (i.e. one with the same input-output behavior) but having more zeroes in it. A zero in this case represents no connection.

## Chapter 3

### The Main Problem

#### 3.1 Formulation

##### 3.1.1 Content-Addressable-Memory

A Content-Addressable-Memory is, as we discussed in the previous chapters, a memory from which data can be retrieved by providing partial information about the content rather than supplying a certain address. This means that if the memory is able to remember  $s$  patterns (i.e. strings of  $N$ -bits of  $\pm 1$ 's) and we present to it a string of  $N$ -bits, it has to retrieve the best matching pattern. To describe such a memory more formally :

Let  $\Omega = \{-1, 1\}^N$ . Let  $G = \{V_1, V_2, \dots, V_s\} \subset \Omega$

Our purpose is to design a network that simulates a mapping  $\Phi$ , such that:

1.  $\Phi(\Omega) = G$ , where  $G$  is the set of patterns to be memorized.
2. Let  $U \in \Omega$ . We want<sup>1</sup>

$$\Phi(U) = \arg \min_{V \in G} d(U, V)$$

where  $d(U, V)$  is the Hamming distance between  $U$  and  $V$ .

$$d(V_1, V_2) = \sum_{i=1}^N \mu(V_1^i, V_2^i)$$

---

<sup>1</sup>in cases of ties,  $\Phi$  may be set-valued.

with<sup>2</sup>

$$\begin{aligned}\mu(x, y) &= \begin{cases} 1 & x \neq y \\ 0 & x = y \end{cases} \\ &= 1 - \delta_{xy}.\end{aligned}$$

In particular,  $\forall V \in G$  we require  $\Phi(V) = V$ .

### 3.1.2 Hopfield's Approach

In an attempt to build a Content-Addressable-Memory, J. J. Hopfield suggested a network with the following properties:

- Let  $V_{in} \in \Omega$  be the input vector and let  $T$  be an  $N \times N$  matrix.
- Let  $w(1), w(2), \dots$ , be a sequence of vectors defined by:<sup>3</sup>

$$\begin{aligned}w(1) &= V_{in} \\ w(k+1) &= \text{sgn}(Tw(k))\end{aligned}\tag{3.1}$$

where  $\text{sgn}(\cdot)$  is defined, component-wise, as:

$$\text{sgn}(x) = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}$$

- The output of the network is  $\lim_{k \rightarrow \infty} w(k)$  when it exists, and we denote it by  $V_{out}$ .

If the limit  $V_{out}$  exists  $\forall V_{in} \in \Omega$  then the mapping

$$\begin{aligned}H: \Omega &\rightarrow \Omega \\ V_{in} &\mapsto V_{out}\end{aligned}$$

---

<sup>2</sup>Note that  $d$  can also be written as  $d(V_1, V_2) = \frac{N - V_1^T V_2}{2}$ , and therefore

$$\Phi(U) = \arg \min_{V \in G} \frac{N - V^T U}{2} = \arg \max_{V \in G} V^T U.$$

<sup>3</sup>The actual way these vectors evolve differs from one version of the model to another. We will concentrate now on the synchronous, deterministic-transitions, discrete-space model and show later that the results apply equally well to all the other models.

is called a Hopfield operator.

Note that once the matrix  $T$  is chosen, the operator  $H$  is completely specified.

It is hoped that if we can synthesize  $T$  out of the patterns we want to memorize,  $H$  will have the desired properties of  $\Phi$  described in the previous section. However, as we will soon see, this is not a trivial issue. In fact, the most widely known algorithm to construct  $T$  – the so-called *outer product* algorithm<sup>4</sup>– results in the appearance of a much larger number of memories than required. These extra memories are for this reason called *spurious memories*.

**Definition:** Let  $H$  be a Hopfield operator. A vector  $V$  is *spurious* if

$$V \in \Omega \setminus G \text{ and } H(V) = V.$$

**Problem:** Characterize the spurious vectors of a given operator  $H$

This problem is of fundamental importance to the field of neural networks, and solving it (at least for the outer product case) will shed light on and provide considerable insight into the usefulness of the Hopfield network, specifically as a Content-Addressable-Memory.

In the next section, the main result regarding this question will be stated and followed by a theoretical proof together with some experimental tests.

## 3.2 Solution: Characterization of Spurious States

### 3.2.1 Main Theorem:

Let  $G = \{V_1, \dots, V_s\} \subset \Omega$ , where the  $V_i$ 's are mutually orthogonal, i.e.  $V_i^T V_j = N \delta_{ij}$ . Let  $T = \sum_{i=1}^s V_i V_i^T$ . Let  $A_s$  be the set of vectors the network ends up memorizing, i.e.  $A_s = \{V : H(V) = V\}$ .

Then the following is true :

---

<sup>4</sup>Under this scheme  $T$  is taken to be the sum of outer products of the vectors to be memorized.

$$T = \sum_{i=1}^s V_i V_i^T$$

- $G \subset A_s$ : This means that, indeed, the network does memorize the patterns we are interested in. Note, however, that  $G$  is a *proper* subset of  $A_s$ .
- **Elements of  $A_s$** : Let  $D_s =$  Set of all *realizable* Boolean functions of  $V_1, V_2, \dots, V_s$ .<sup>5</sup> Then, there is an injection between  $A_s$  and  $D_s$ .  
For the case  $s \leq 3$ , there is a bijection between  $A_3$  and  $D_3$ .
- **Cardinality of  $A_s$** : For  $s \leq 7$

$$\text{card}(A_1) = 2, \text{card}(A_2) = 4, \text{card}(A_3) = 14, \text{ etc. } \dots \text{ see table 3.1}$$

and for  $s > 7$

$$\frac{3^s}{2} < \text{card}(A_s) < \frac{2^{s^2}}{s!}$$

Note that the number of spurious memories is  $(\text{card}(A_s) - s)$  which is quite large.

$s$	1	2	3	4	5	6	7
$\text{card}(A_s)$	2	4	14	40	1402	21,228	3,548,358
$\text{card}(A_s) - s$	1	2	11	36	1397	21,222	3,548,351

Table 3.1: Number of attractors and spurious states

### 3.2.2 Example: $s=3$

Let  $V_1, V_2$  and  $V_3$  be 3 mutually orthogonal vectors of  $\Omega$ . Let  $T = V_1V_1^T + V_2V_2^T + V_3V_3^T$ . Let  $H$  be a Hopfield operator corresponding to  $T$ . Then the set of patterns the network ends up memorizing is:

$$A_3 = \left\{ \begin{array}{l} V_1 ; V_2 ; V_3 ; \bar{V}_1 ; \bar{V}_2 ; \bar{V}_3 \\ V_1V_2 + V_3(V_1 \oplus V_2) ; \bar{V}_1V_2 + V_3(\bar{V}_1 \oplus V_2) \\ V_1\bar{V}_2 + V_3(V_1 \oplus \bar{V}_2) ; V_1V_2 + \bar{V}_3(V_1 \oplus V_2) \\ \bar{V}_1\bar{V}_2 + V_3(\bar{V}_1 \oplus \bar{V}_2) ; \bar{V}_1V_2 + \bar{V}_3(\bar{V}_1 \oplus V_2) \\ V_1\bar{V}_2 + \bar{V}_3(V_1 \oplus \bar{V}_2) ; \bar{V}_1\bar{V}_2 + \bar{V}_3(\bar{V}_1 \oplus \bar{V}_2) \end{array} \right\}$$

<sup>5</sup>Realizable Boolean functions are a special class of Boolean functions. See next chapter for a complete definition.



where the logical operations are defined on an entry by entry basis. Note that there is a bijection between  $A_3$  and the set of *realizable* Boolean functions of 3 variables. This set is:

$$\{x_1, x_2, x_3, \bar{x}_1, \bar{x}_2, \bar{x}_3, x_1x_2 + x_3(x_1 \oplus x_2), \bar{x}_1x_2 + x_3(\bar{x}_1 \oplus x_2), x_1\bar{x}_2 + x_3(x_1 \oplus \bar{x}_2), x_1x_2 + \bar{x}_3(x_1 \oplus x_2), \bar{x}_1\bar{x}_2 + x_3(\bar{x}_1 \oplus \bar{x}_2), \bar{x}_1x_2 + \bar{x}_3(\bar{x}_1 \oplus x_2), x_1\bar{x}_2 + \bar{x}_3(x_1 \oplus \bar{x}_2), \bar{x}_1\bar{x}_2 + \bar{x}_3(\bar{x}_1 \oplus \bar{x}_2)\}.$$

### 3.2.3 Remarks

Taking a closer look at the above example, the following observations can be made:

1. For this special case, it turned out that  $\text{card}(A_3) = \text{card}(D_3)$ .
2. The number of attractors is always 14 no matter how large  $N$  is.
3. For each attractor, its negative is also an attractor.
4. The number of spurious states is  $14 - 3 = 11$ .
5. If we had the chosen  $s = 7$  case, we would have obtained 3,548,351 attractors, literally MILLIONS of spurious states!!!.

### 3.2.4 Proof

Let  $H$  be a Hopfield network as described in the previous section. Let  $V_1, V_2, \dots, V_s$  be  $s$  orthogonal vectors of dimension  $N$ . Let  $T = \frac{1}{N} \sum_{i=1}^s V_i V_i^T$ . Let  $\mathbf{v}$  be an attractor of  $H$ , i.e.  $H(\mathbf{v}) = \mathbf{v}$ , then <sup>6</sup>:

Lemma:  $\mathbf{v}$  is an attractor if and only if  $\text{sgn}(T\mathbf{v}) = \mathbf{v}$ .

Proof: Suppose  $\text{sgn}(T\mathbf{v}) = \mathbf{v}$ . Let  $\mathbf{w}(1) = \mathbf{v}$ . From equation 3.1 we conclude that  $\mathbf{w}(2) = \mathbf{w}(3) = \dots = \mathbf{v}$ . Therefore  $H(\mathbf{v}) = \mathbf{v}$  and hence  $\mathbf{v}$  is an attractor.

To prove the other direction, let  $\mathbf{v}$  be an attractor. Let  $\mathbf{w}(1) = \mathbf{v}$ . From equation 3.1 we conclude that  $\lim_{k \rightarrow \infty} \mathbf{w}(k) = \mathbf{v}$ . The key point to notice is that the space of states is finite, hence we will hit the limit, since we are given it exists, in  $2^N$  steps at most. Therefore,  $\mathbf{w}(2^N + 1) = \mathbf{w}(2^N + 2) = \dots = \mathbf{v}$ . Now since  $\mathbf{w}(2^N + 2) = \text{sgn}(T\mathbf{w}(2^N + 1))$ , we conclude that  $\text{sgn}(T\mathbf{v}) = \mathbf{v}$   $\square$

---

<sup>6</sup>This lemma applies to any real matrix  $T$

- Let  $\mathbf{v}$  be an attractor. Then,

$$\begin{aligned}\operatorname{sgn}(T\mathbf{v}) &= \mathbf{v}. \\ \operatorname{sgn}\left(\sum_{i=1}^s \frac{1}{N} V_i V_i^T \mathbf{v}\right) &= \mathbf{v}. \\ \operatorname{sgn}\left(\sum_{i=1}^s \alpha_i V_i\right) &= \mathbf{v}.\end{aligned}$$

Note that  $\alpha_i = V_i^T \mathbf{v}/N$  and hence  $\alpha_i V_i$  is simply the orthogonal projection of  $\mathbf{v}$  over  $V_i$ . Now it becomes apparent that the argument of the signum function is nothing but the orthogonal projection of  $\mathbf{v}$  over  $S$ , the space spanned by the eigenvectors of  $T$  that correspond to nonzero eigen-values. For this reason,  $T$  is called a *projective matrix*.

If we define an *orthant* of a vector  $\mathbf{v}$ ,  $\mathcal{O}(\mathbf{v})$ , to be the region of  $R^n$  that satisfies:  $\operatorname{sgn}(x_1) = \operatorname{sgn}(v_1)$ ,  $\operatorname{sgn}(x_2) = \operatorname{sgn}(v_2)$ , and so on  $\dots$ , (i.e. the orthant of  $\mathbf{v} = (1, 1, -1, 1)$  is the region of  $R^n$  specified by:  $x_1 > 0$ ,  $x_2 > 0$ ,  $x_3 < 0$ ,  $x_4 > 0$ ) then, we can characterize the  $\operatorname{sgn}$  function as a mapping that maps a vector into that vertex of the  $N$ -cube that shares the same orthant.

Making use of the above two observations, we can directly conclude that:

**Fact I:** *A necessary condition<sup>7</sup> for  $\mathbf{v}$  to be an attractor is that  $S$  intersects the orthant of  $\mathbf{v}$ .*

Stated in mathematical terms:

$$\operatorname{sgn}(T\mathbf{v}) = \mathbf{v} \implies \mathcal{O}(\mathbf{v}) \cap S \neq \emptyset.$$

Note also that, in general<sup>8</sup>

**Fact II:** *A necessary and sufficient condition for  $\mathbf{v}$  to be an attractor is that the orthogonal projection of  $\mathbf{v}$  over  $S$  lies also in the orthant of  $\mathbf{v}$ .*

In equation form:

$$\operatorname{sgn}(T\mathbf{v}) = \mathbf{v} \iff \operatorname{proj}_{\perp}(\mathbf{v})_S \in \mathcal{O}(\mathbf{v}).$$

- Let  $Q_s = \{\mathbf{v}: S \cap \mathcal{O}(\mathbf{v}) \neq \emptyset\}$ . Let  $A_s = \{\mathbf{v}: H(\mathbf{v}) = \mathbf{v}\}$ . Then, from **Fact I** we conclude:

$$A_s \subset Q_s. \tag{3.2}$$

<sup>7</sup>This condition is quite general and applies to any real matrix  $T$ .

<sup>8</sup>Although we will not use **FACT II** in the coming proof, we included it to make the picture clearer.

How to determine  $Q_s$ ?

For the sake of illustration, I will consider the case of 3 memories ( $s = 3$ ) and then generalize.

Let  $T = \sum_{i=1}^3 V_i V_i^T$ . Without loss of generality we can interchange any two rows of corresponding entries until we reach the canonical form of table (3.2).

$V_1$	$V_2$	$V_3$	
+1	+1	+1	} $a_{11}$ (rows)
$\vdots$	$\vdots$	$\vdots$	
+1	+1	-1	} $a_{21}$
$\vdots$	$\vdots$	$\vdots$	
+1	-1	+1	} $a_{31}$
$\vdots$	$\vdots$	$\vdots$	
+1	-1	-1	} $a_{41}$
$\vdots$	$\vdots$	$\vdots$	
-1	+1	+1	} $a_{42}$
$\vdots$	$\vdots$	$\vdots$	
-1	+1	-1	} $a_{32}$
$\vdots$	$\vdots$	$\vdots$	
-1	-1	+1	} $a_{22}$
$\vdots$	$\vdots$	$\vdots$	
-1	-1	-1	} $a_{12}$
$\vdots$	$\vdots$	$\vdots$	

Table 3.2:  $V_1, V_2$  and  $V_3$  in canonical form

Let  $S =$  Space spanned by  $V_1, V_2$  and  $V_3$ . To find the orthants that  $S$  intersects it is enough to study the sign of entries of the vector

$$\text{sgn}(\alpha V_1 + \beta V_2 + \gamma V_3) \tag{3.3}$$

which is, in turn, the same as exploring the vector of signs generated by:(see table (3.2))

$$\alpha + \beta + \gamma \tag{3.4}$$

$$\alpha + \beta - \gamma \tag{3.5}$$

$$\alpha - \beta + \gamma \tag{3.6}$$

$$\alpha - \beta - \gamma \tag{3.7}$$

The question now becomes : Using different choices of  $\alpha, \beta$  and  $\gamma$  how many distinct vectors of signs can we generate? Or, stated differently, how many regions of  $R^3$  do we obtain after drawing 4 planes? At this point, the need for Threshold logic results becomes obvious.

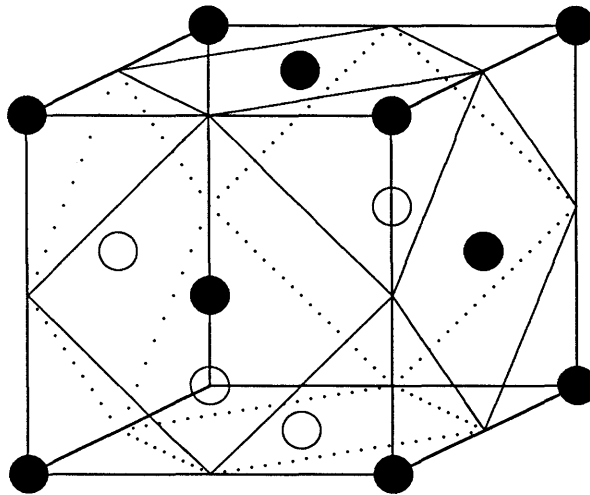


Figure 3.1: Cube divided into 14 pieces by 4 planes passing through its center

From the literature on Threshold logic<sup>9</sup>, we learn that 4 planes passing through the origin divide  $R^3$  into 14 regions (and not 16 !!, see figure (3.1)) such that all the triplets  $(\alpha, \beta, \gamma)$  lying in the same region generate the same vector of signs (in equations 3.4–3.7)

A set of 14 triplets,<sup>10</sup>one from each region, is

$$M = \left\{ \begin{array}{l} (1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1), \\ (1, 1, 1), (1, 1, -1), (1, -1, 1), (1, -1, -1), \\ (-1, 1, 1), (-1, 1, -1), (-1, -1, 1), (-1, -1, -1) \end{array} \right\}$$

<sup>9</sup>The next chapter is dedicated to provide the basic information about this field.  
<sup>10</sup>see Chapter 4 and Appendix A for more detail

From equation 3.3 and the discussion preceding it, it should be clear now that

$$Q_3 = \left\{ \mathbf{v}: \mathbf{v} = \text{sgn}(m_1 V_1 + m_2 V_2 + m_3 V_3) \quad \text{and} \quad (m_1, m_2, m_3) \in M \right\}$$

So, what is  $A_3$ ?

One way to determine  $A_3$  is, using equation (3.2), to check every vector of  $Q_3$  and see if it satisfies  $\text{sgn}(T\mathbf{v}) = \mathbf{v}$ . However, with some thought, we need only check 2 vectors of  $Q$ , namely those that correspond to

$$\mathbf{m} = (1, 0, 0) \quad \text{and} \quad \mathbf{m} = (1, 1, 1)$$

Taking  $N = 4, V_1 = (1, 1, 1, 1), V_2 = (1, 1, -1, -1)$  and  $V_3 = (1, -1, -1, 1)$  (the vectors are in their canonical form) we find out that, indeed,  $V_1$  and  $\text{sgn}(V_1 + V_2 + V_3)$  are attractors. Therefore, for  $s = 3$

$$\begin{aligned} A_3 &= Q_3 \\ &= \left\{ \mathbf{v}: \mathbf{v} = \text{sgn}(m_1 V_1 + m_2 V_2 + m_3 V_3) \quad \text{and} \quad (m_1, m_2, m_3) \in M \right\} \end{aligned}$$

What is even more interesting is the fact that the elements of  $Q_3$  can be written as Boolean functions<sup>11</sup> of  $V_1, V_2$  and  $V_3$ , where it turns out that

$$\begin{aligned} \text{sgn}(V_1 + V_2 + V_3) &= V_1 V_2 + V_3 (V_1 \oplus V_2) \\ \text{sgn}(V_1 + V_2 - V_3) &= V_1 V_2 + \bar{V}_3 (V_1 \oplus V_2) \end{aligned}$$

and so on ... with the operations on the right hand side expressions being Boolean operations defined on an entry-by-entry basis.

Finally, we can conclude the following important result that was stated before:

Let  $H$  be a Hopfield network made out of  $N$  neurons having zero threshold. Let  $T = V_1 V_1^T + V_2 V_2^T + V_3 V_3^T$ , where  $V_1, V_2$  and  $V_3$  are 3 mutually orthogonal vertices of the N-cube. Then, the attractors of  $H$  (or equivalently, the fixed points of the operation  $\text{sgn}(T\mathbf{v}) = \mathbf{v}$ ) are the following

$$\begin{aligned} &V_1 ; V_2 ; V_3 ; \bar{V}_1 ; \bar{V}_2 ; \bar{V}_3 \\ &V_1 V_2 + V_3 (V_1 \oplus V_2) ; \bar{V}_1 V_2 + V_3 (\bar{V}_1 \oplus V_2) \end{aligned}$$

---

<sup>11</sup>see next chapter for more details.

$$\begin{aligned}
& V_1\bar{V}_2 + V_3(V_1 \oplus \bar{V}_2) ; V_1V_2 + \bar{V}_3(V_1 \oplus V_2) \\
& \bar{V}_1\bar{V}_2 + V_3(\bar{V}_1 \oplus \bar{V}_2) ; \bar{V}_1V_2 + \bar{V}_3(\bar{V}_1 \oplus V_2) \\
& V_1\bar{V}_2 + \bar{V}_3(V_1 \oplus \bar{V}_2) ; \bar{V}_1\bar{V}_2 + \bar{V}_3(\bar{V}_1 \oplus \bar{V}_2)
\end{aligned}$$

where the logical operations between vectors are defined on an entry-by-entry basis.

#### A Micronote on $s$

So far we have not discussed anything about  $s$ , the number of mutually orthogonal vertices of the  $N$ -cube. Contrary to what one might originally think, the maximum number of  $s$  needs not be always  $N$  but might in fact be a much smaller number. If  $N$  is odd then  $s = 1$  since no two vertices can be mutually orthogonal (adding an odd number of  $-1$  and  $+1$  can never result in  $0$ ). However, it can be shown that if  $N = 2^p$  for some  $p$ , it is guaranteed that we can let  $s = N$ .

- Case  $s > 3$ : Going back to our previous discussion. To obtain analogous results for the case  $s > 3$ : Let  $V_1, \dots, V_s$  be  $s$  mutually orthogonal elements of the  $N$ -cube. We interchange the rows until we obtain the canonical form similar to table (3.2).

The question is: What is  $M$  for  $s > 3$ ?

Answer: Books on Threshold Logic (e.g. [5]) have listed the elements of  $M$  for  $s = 1, 2, \dots, 7$ . If  $s = 4$  for instance, then (omitting permutations and negations) we get

$$M = \{(1, 0, 0, 0), (1, 1, 1, 0), (2, 1, 1, 1)\}$$

(the full  $M$  has 104 elements).

Checking which of these correspond to attractors we find that only  $(1, 0, 0, 0)$  and  $(1, 1, 1, 0)$  do.

Counting all possible permutations and negations we obtain that the overall number of attractors in the case  $s = 4$  is 40, where each attractor is of the form

$$\text{sgn}(\pm m_1 V_1 \pm m_2 V_2 \pm m_3 V_3 \pm m_4 V_4)$$

with  $\mathbf{m} = (m_1, m_2, m_3, m_4) =$  a permutation of either  $(1, 0, 0, 0)$  or  $(1, 1, 1, 0)$ .

Doing the same for  $s = 5, 6$  and  $7$  we obtain the following fundamental result:

Consider a Hopfield network made out of  $N$  neurons having threshold zero. Let  $[T_{ij}]$  be the interconnection matrix of the network. let  $s$  be the rank of  $T$ . Let  $V_1, \dots, V_s$  be the eigen-vectors of  $T$  (with nonzero eigenvalues). Assume that these vectors are constrained to be corners of  $(-1, +1)^N$ . Assume further that the corresponding eigen-values are the same (and positive). Let  $A_s$  be the set of attractors of  $H$ , then

$$\text{card}(A_1) = 2 \quad \text{card}(A_2) = 4 \quad \text{card}(A_3) = 14 \quad \text{etc...}$$

and in general

$$\frac{3^s}{2} < \text{card}(A_s) < \frac{2^{s^2}}{s!} \ll 2^{2^s}$$

with the attractors being realizable Boolean functions of  $V_1, \dots, V_s$ .

Again the reader is referred to appendix A for a complete listing of the realizable Boolean functions with zero threshold.

A point here needs to be clarified. In certain cases (e.g.  $N = 8, s = 8$ ) some of the attractors may turn out to be the same. The overall number of spurious states will be less than what is listed in table (3.1). However, this will not occur if all the typical rows in the canonical form are present (i.e.  $\forall i, a_{i1} + a_{i2} \neq 0$  in table (3.2)). It is easy to see that a necessary condition for the presence of all typical rows is to have  $\log_2 N \geq s - 1$ .

We have not proved yet the last part of the theorem, namely

$$\frac{3^s}{2} < \text{card}(A_s) < \frac{2^{s^2}}{s!} \ll 2^{2^s} \tag{3.8}$$

To see why this is true, we refer to the fact that there is an injection between  $A_s$  and  $D_s$ , the set of realizable Boolean functions of  $V_1, \dots, V_s$ . Therefore,

$$\text{card}(A_s) \leq \text{card}(D_s).$$

But it was proved by Winder[26] that

$$\text{card}(D_s) < \frac{2^{s^2}}{s!} \ll 2^{2^s}$$

thereby establishing the upper bound.

For the lower bound it is enough to note that for any  $s$  all vectors  $\mathbf{m}$  having an odd number  $p$  of  $\pm 1$ 's and  $(s - p)$  zeroes will correspond to an attractor. The overall number of such attractors is

$$\sum_{p=1, \text{odd}}^s \binom{s}{p} 2^p = \frac{3^s - (-1)^s}{2}$$

the result of the summation can be obtained by expanding  $(1 + 2)^s$  and  $(1 - 2)^s$  and then taking the difference.

Experimentally we see that the lower bound is very loose but still it expresses the exponential growth of the spurious states.

So far, no attempt has been made to correlate between  $D_s$  and  $Q_s$ . In fact, a careful analysis (see next chapter) can show that there is a bijective mapping between  $D_s$  and  $Q_s$ . This means –in words– that a space spanned by vectors of the  $N$ -cube intersects an orthant if and only if the *representative* vector<sup>12</sup> of the orthant is a realizable Boolean function of these vectors.

---

<sup>12</sup>The representative vector of the orthant is the  $N$ -cube element that lies in it.



### 3.2.5 Experimental Results

Appendix B includes a program written in LISP. This program is used to search the whole  $N$ -cube –looking for the attractors of the Hopfield Network– for the cases:  $s=3, N=8$ ;  $s=4, N=8$  and  $s=5, N=16$ . The whole list is printed for the first two cases while for the last case, and due to lack of space, only the *number* of attractors is computed and printed.

#### 1. $s=3, N=8$

$$\text{The vectors to be memorized are } \begin{cases} V_1 = (+, +, +, +, -, -, -, -) \\ V_2 = (+, +, -, -, +, +, -, -) \\ V_3 = (+, -, +, -, +, -, +, -) \end{cases}$$

```
1 ==> (list-attractors (enumerate-vertices 8))
```

```
1 (+ + + + - - - -) <---- V1
2 (+ + + - + - - -)
3 (+ + - + - + - -)
4 (+ + - - + + - -) <---- V2
5 (+ - + + - - + -)
6 (+ - + - + - + -) <---- V3
7 (+ - - - + + + -)
8 (- + + + - - - +)
9 (- + - + - + - +)
10 (- + - - + + - +)
11 (- - + + - - + +)
12 (- - + - + - + +)
13 (- - - + - + + +)
14 (- - - - + + + +)
DONE
```

```
-----
```

2. s=4, N=8

Here we add to the vectors from last case:

$$V_4 = (+, +, +, +, +, +, +, +)$$

```
1 ==> (list-attractors (enumerate-vertices 8))
```

```

1 (+ + + + + + +) <---- V4
2 (+ + + + + - -)
3 (+ + + + - + -)
4 (+ + + - + - +)
5 (+ + + - - + +)
6 (+ + + - - - -) <---- V1
7 (+ + - + + + -)
8 (+ + - + - - -)
9 (+ + - + + - +)
10 (+ + - + - + -)
11 (+ + - - + + +)
12 (+ + - - + + -) <---- V2
13 (+ + - - - - -)
14 (+ - + + + - +)
15 (+ - + + - - +)
16 (+ - + - + + +)
17 (+ - + - + - -) <---- V3
18 (+ - + - - - -)
19 (+ - - - + + -)
20 (+ - - - + - -)
21 (- + + + - + +)
22 (- + + + - - +)
23 (- + - + + + +)
24 (- + - + - + +)
25 (- + - + - - -)
26 (- + - - + + -)
27 (- + - - - + -)
28 (- - + + + + +)
29 (- - + + - - +)
30 (- - + + - - -)
31 (- - + - + - +)
32 (- - + - - - -)
33 (- - - + - + +)
34 (- - - + - - +)
35 (- - - - + + +)
36 (- - - - + - -)
37 (- - - - + - -)
38 (- - - - - + +)
39 (- - - - - + +)
40 (- - - - - - -)

```

DONE

-----

### 3. s=5, N=16

In this case the vectors to be memorized are:

$$V_1 = (+, +, +, +, +, +, +, +, +, +, +, +, +, +, +, +)$$

$$V_2 = (+, +, +, +, +, +, +, +, -, -, -, -, -, -, -)$$

$$V_3 = (+, +, +, +, -, -, -, -, +, +, +, +, -, -, -, -)$$

$$V_4 = (+, +, -, -, +, +, -, -, +, +, -, -, +, +, -, -)$$

$$V_5 = (+, -, +, -, +, -, +, -, +, -, +, -, +, -, +, -)$$

```
1 ==> (count-attractors (enumerate-vertices 16))
1402
```

By inspecting the computer results, we find a perfect match with the theoretical expectations. The reader is referred to section 3.3.3 for a discussion about the basins of attractions.

### 3.3 Discussion

In the coming sections, we will examine the obtained results closely, trying to correlate them with present available results that have used the statistical approach. In addition, we will discuss the implications of such results. Finally, we will assess the Hopfield approach as a means for simulating a Content-Addressable-Memory.

#### 3.3.1 Relation to others' work

In a paper by Amit et al [3] as well as a paper by Newman [21], it is shown that in the case of  $N \rightarrow \infty$ , the spurious states correspond to mixtures of finitely many original patterns. Each such configuration, which they denote  $X(\nu)$  is given by:

$$X(\nu) = \text{sgn}(\nu_1 V_1 + \nu_2 V_2 + \dots + \nu_s V_s)$$

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

- (i)  $\nu$  has a finite number of non zero components.
- (ii)  $\pm\nu_1 \pm \nu_2 \pm \dots \neq 0$  for any choice of  $\pm$  signs.
- (iii) For each nonzero  $\nu_s$ , the sum  $\pm\nu_1 \pm \nu_2 \dots$  has the same sign as  $\pm\nu_s$  for exactly a fraction  $(1 + \nu_s)/2$  of all possible sign choices.

Our work was essentially to exhaust all such possible  $\nu$ 's; enumerate the spurious states;<sup>13</sup> give a different representation in terms of Boolean functions of the original patterns and show that the results hold exactly even for finite  $N$  as long as  $\log_2 N \geq s - 1$  and the vectors are orthogonal in the geometric sense.

#### 3.3.2 Impact of our work

The result we have already proven has serious implications. It states that no matter how many neurons we use to memorize  $s$  words (using Hopfield's

---

<sup>13</sup>Enumerating the spurious states by simply searching the N-cube is computationally very expensive. For the case  $s = 7$  for instance,  $N$  has to be at least  $2^{7-1} = 64$ . This means that the size of the N-cube will be  $2^{64} = 1.8 \times 10^{19}$ . If  $10^{-9}s$  is needed to check every vector, the whole cube will require 300 years !! Our approach provide for a very efficient way to find the attractors.

algorithm) we always end up memorizing  $A_s$ . Such a result falsifies a recent impression which states that increasing the number of neurons will reduce the number of spurious states. In fact, increasing  $N$  can only help pushing the attractors to stay apart from each other (i.e. to increase their mutual Hamming distance).

### 3.3.3 Generality of Results

The following questions may be asked by the reader:

- (a) What if the original vectors are not orthogonal?

Answer: If they are not exactly orthogonal then they are at least uncorrelated, i.e.  $E(V_i^T V_j) = 0$  for  $i \neq j$ . In this case our results will still hold for large enough  $N$  (see Amit et al [3] for a discussion on the case  $N \rightarrow \infty$ ).

- (b) What if  $T$  was not a sum of outer products of vectors?

Answer: If we assume that  $T$  is symmetric, then from linear algebra any  $N \times N$  symmetric matrix has  $N$  orthogonal eigen-vectors:  $V_i, i = 1, \dots, N$ . Therefore we can write it as

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

However, we imposed two constraints: First, the eigen-vectors have to be elements of  $\Omega$ ; second, the eigen-values have to be the same.

- (c) What if  $T$  is not symmetric?

Answer: From linear algebra we know that a non-symmetric  $T$  will have complex eigen-values. This will result in oscillatory output provided that the input is the vector with the complex eigen-value (as can be seen by direct computation).

- (d) Even if  $T$  is symmetric why should we let the eigen-values be the same as the original vectors ?

Answer: They need not be. In fact the results are given in terms of the eigen-vectors of  $T$  regardless of their interpretation. The only restriction is that they belong to the  $N$ -cube.

(e) Why should we restrict the eigen-vectors to be vertices of the  $N$ -cube?

Answer: Even when we relax this constraint **FACT I** remains true and therefore the upper bound in equation 3.8 is still applicable. To obtain a lower bound more work is needed, and in fact this is left as an open technical question.

(f) What if the eigen-values are not the same?

Answer: If we fix the eigen-values, the attractors can again be found by checking which elements of  $Q_s$  are fixed points of  $T$  (see discussion on page 22).

(g) What if we are not operating the network under the mode you have chosen?

Answer: Although we have considered in our analysis the synchronous, deterministic-transitions, discrete-space model, our results still apply equally well to all other types of dynamics. To see the validity of this assertion, note first that synchrony is not an issue because we are studying the fixed points of  $H$  and consequently the fixed points of  $\text{sgn}(T)$ . Therefore it does not matter whether we update one neuron at a time or all of them together.

Second, even when the chosen mode has a continuous space (which amounts to a sigmoid I/O relationship) instead of a discrete space (which amounts to a signum, see figure (2.3)), the linear region of the sigmoid will gradually shrink during the operation of the network to force the final output state to be on the  $N$ -cube (see Hopfield [10]).

Finally, in the case of stochastic transitions, where *the probability* of changing the neuron state is given by figure 2.4, the number of spurious states is significantly reduced at high temperature (i.e. a long transition region in figure 2.4). However, when we start “cooling” the network other spurious states start to appear and for  $T = 0$  we are faced with all of them. One might argue that by heating the network enough at the beginning we will produce a correct output with very high probability. However the number of neurons needed in this case is approximately  $N = s/0.15$ , and if we know that  $N$  neurons mean  $N^2$  interconnections we readily see that this

aspect is not very practical for implementation purposes.

- (h) What is harmful about having spurious states if they are quite away from the original memories?

Answer: To push the spurious states away from the original memories we need a large  $N$  and we are back to the previous discussion.

- (i) How about the basins of attractions of these spurious states?

Answer: It is proved rigorously by Newman[21] that these spurious patterns have energy barriers for sufficiently small  $\alpha(= \lim s/N)$ .

### 3.4 An interesting Algorithm

One of the problems associated with the Hopfield Network is that of the numerous internal connections. If we can localize the connections we can implement the network using smaller chip area. In what follows we will provide an optimum solution for this problem.

**Definition:** Let  $T$  and  $T'$  be  $2 N \times N$  matrices. Let  $H$  and  $H'$  be the Hopfield operators characterized by  $T$  and  $T'$  respectively. We say that  $T$  is isomorphic to  $T'$  if  $H = H'$ .

**Problem:** Given a matrix  $T$ , find the matrix  $T'$  which is isomorphic to  $T$  and has the largest number of 0's.i.e. find

$$\arg \max_{T'isoT} \sum_{i,j} \delta(T'_{ij}, 0)$$

where  $\delta$  is the Kronecker delta. Note that no restriction is imposed on  $T$  whatsoever. There are, however, some constraints on the mode of operation. It should be either *deterministic* with a signum type of I/O relationship, or *stochastic* with the probability of transition taking one of two values only (i.e.  $\Pr(V_i=1)=p$  if  $\sum T_{ij}V_j > 0$ , and  $(1 - p)$  if  $\sum T_{ij}V_j < 0$ ).

**Solution:** Let  $T$  and  $T'$  be two isomorphic matrices. Then,

$$\forall \mathbf{v} \in \Omega \text{sgn}(T\mathbf{v}) = \text{sgn}(T'\mathbf{v})$$

Let  $\mathbf{a}$  and  $\mathbf{a}'$  be the first rows of  $T$  and  $T'$  respectively. Then

$$\forall \mathbf{v} \in \Omega \operatorname{sgn}(\mathbf{a}\mathbf{v}) = \operatorname{sgn}(\mathbf{a}'\mathbf{v}) \quad (3.9)$$

Given  $\mathbf{a}$ , the above equation does not have a unique solution in terms of  $\mathbf{a}'$ . Our aim is to find a solution with the maximum possible number of zeroes in it.

If we draw all the binary<sup>14</sup> hyperplanes in  $R^N$ , then from equation (3.9), the two vectors  $\mathbf{a}$  and  $\mathbf{a}'$  should fall at the same side with respect to all the hyperplanes. That is to say, they should belong to the same convex cone (e.g. see figure (4.1) and the following discussion).

### 3.4.1 The Basic Idea

We want to determine the vector in the same cone which has the largest number of zeroes in it. Because of the inherent symmetry of the problem, this vector has to be equidistant from all the intersections of the hyperplanes with the cube surface on which it lies. One way to accomplish this is by assigning *positive charges* to both the vector in consideration and to the traces of the hyperplanes on the surface on which this vector lies (if the vector does not lie on the cube surface, i.e. the greatest entry in magnitude is not  $\pm 1$ , we can force it to be so by scaling it by the magnitude of the largest entry). Having done so, the vector will move and settle down in the unique minima (unique because of symmetry) of the potential of the already defined electric field.

### 3.4.2 Description of The Algorithm

We define a surface of the  $N$ -cube as :

$$\{(x_1, x_2, \dots, x_N) : x_i = \pm 1 \text{ for some } i, -1 \leq x_j \leq 1 \forall j \neq i\}$$

- Let  $\mathbf{a}$  be the first row of  $T$ .

---

<sup>14</sup>A binary hyperplane is one whose coefficients are either +1 or -1.



- Let  $P_1, P_2, \dots, P_m$  be the binary hyperplanes in  $R^N$ . Their number is  $2^{N-1}$ . Each is of the form  $c_i x = 0$ . Where  $c_i \in \Omega$  and  $x = (x_1, x_2, \dots, x_N)$  is the vector of the space variables.
- Scale  $\mathbf{a}$  by the magnitude of its largest entry (i.e. by its  $H_\infty$  norm).
- Let  $S_{\mathbf{a}} =$  The surface of the  $N$ -cube on which  $\mathbf{a}$  lies.
- Let  $L_1, L_2, \dots, L_m$  be hyperlines of dimension  $(N - 2)$  such that

$$\forall i L_i = P_i \cap S_{\mathbf{a}}$$

i.e.  $L_i$  is the trace of  $P_i$  on  $S_{\mathbf{a}}$ .

- Let  $d_i(\mathbf{r}) =$  distance between  $L_i$  and an arbitrary vector  $\mathbf{r}$ . For example, if  $P_i: x_1 + x_2 + \dots + x_N = 0$  and  $S_{\mathbf{a}}: x_1 = 1$ , then,

$$d_i(\mathbf{r}) = \frac{|1 + r_2 + r_3 + \dots + r_N|}{\sqrt{N}}$$

- Let  $f(\mathbf{r}) = \sum_{i=1}^{2^{N-1}} \frac{1}{d_i^2(\mathbf{r})}$
- Using steepest descent algorithm or otherwise find

$$\min_{\mathbf{r}} f(\mathbf{r})$$

with  $\mathbf{r} = \mathbf{a}$  as initial value.

- Replace the first row of  $T$  by the already found minimum.
- Repeat for all the other rows.

### 3.4.3 Example

Let  $T$  be the  $4 \times 4$  matrix

$$\begin{bmatrix} 1 & 3 & -1.2 & 4 \\ 2.2 & -1 & 0.1 & 5 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 0.9 & 0 \end{bmatrix}$$

Then the optimum isomorphic matrix to  $T$  is

$$\begin{bmatrix} 1/2 & 1/2 & -1/2 & 1 \\ 0 & 0 & 0 & 1 \\ 1/2 & 1 & 1/2 & 1/2 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Note that the new matrix has much less variations in the entries, which has a clear practical consequence.

#### 3.4.4 Comments

In the process of computing the minimum of the function  $f(\boldsymbol{r})$  we have to compute an exponential number ( $2^{N-1}$ ) of terms. This is not practical for large values of  $N$ . Note however that this is the best that can be achieved since the problem is NP-complete.

To see why it is so, let  $\boldsymbol{a}$  be a row of  $T$ . To transform  $a_i$ , the  $i^{\text{th}}$  entry of  $\boldsymbol{a}$ , to zero, we have to check if

$$\left| \sum_{j \neq i} \pm a_j \right| > |a_i|$$

but this is equivalent to the “Partition” problem which was proved in [6] to be NP-complete.

### 3.5 Summary

In this chapter we have formulated rigorously the problem of spurious states, provided an analysis based on a geometrical approach and supported with computer experiments. We also discussed the generality of our results, and clarified the relation between our work and other ongoing research in the same field.

In addition to this, we described an algorithm which transforms a Hopfield network into an equivalent one having the least possible number of internal connections as well as the least possible number of parameter values. Based

on these two characteristics, this algorithm represents a major step forward in terms of implementation efficiency.

## Chapter 4

### Threshold Logic

In this chapter we shall study Threshold Logic, the properties of threshold functions, and discuss the results in the literature that are in close relation with our research.

#### 4.1 Definitions

Threshold logic deals with the construction of switching functions using threshold devices as building blocks. Threshold devices form linear summations of their inputs, and discriminate on the magnitude of the sum.

We will start first with few definitions of terms as considered necessary for the clear understanding of threshold functions, the geometrical interpretation of switching functions in the  $n$ -dimensional space, and the clarification of certain notations.

A *binary variable*  $x_i$  is a variable which assumes the value of either element of the two-element set  $B$ , i.e.

$$x_i \in B = \{0, 1\}, \text{ or } \{-1, 1\}, \text{ or } \{F, T\}, \text{ or in general, } \{a, b\}.$$

Throughout our work we shall use  $-1$  and  $1$  for the two elements of  $B$ . This will turn out to be more convenient because of the inherent symmetry.

A *switching* or *Boolean function* is a function of  $n$  binary variables, such that it assumes a value of either  $-1$  or  $1$  when each of its  $n$  variables assumes a value of  $-1$  or  $1$ .

The Cartesian product of  $n$  copies of  $B$ ,  $B^n = B \times B \times \dots \times B$ , is called the *n-cube*.

The *n-cube*  $B^n$  is obviously a subset of the continuous Euclidean  $n$ -space  $R^n$ . There are  $2^n$  elements in the *n-cube*, called *vertices*. They are the  $2^n$  different *valuations* of the *ordered n-tuple*  $X = (x_1, x_2, \dots, x_n)$ , where  $x_i, i = 1, 2, \dots, n$ , is the *i*-th coordinate or component.

In the geometrical sense, a switching function of  $n$  variables,  $F(x_1, \dots, x_n)$ , is defined by assigning either  $-1$  or  $1$  to the  $2^n$  vertices of  $B^n$ , i.e. a mapping from the *n-cube*  $B^n$  into  $B$ , or

$$F : B^n \rightarrow B.$$

A Boolean function  $F(X)$  of  $n$  binary variables  $X$  is said to be a *threshold function* if the following conditions are satisfied:

$$F(X) = +1, \text{ if } f_A(X) = \sum_{i=1}^n a_i x_i > T \quad (4.1)$$

$$F(X) = -1, \text{ if } f_A(X) = \sum_{i=1}^n a_i x_i < T \quad (4.2)$$

where

$x_i$  = a binary variable assuming a value of either  $1$  or  $-1$ , for  $i = 1, 2, \dots, n$ .

$X = (x_1, x_2, \dots, x_n)$   
= an ordered  $n$ -tuple or vector on  $n$  binary variables.

$a_i$  = a real coefficient called the *weight* of  $x_i$ , for  $i = 1, 2, \dots, n$ .

$A = (a_1, a_2, \dots, a_n)$   
= an ordered  $n$ -tuple or vector of  $n$  coefficients for the  $n$  variables

in the algebraic function, or the *weight vector*.

$T$  = a constant called the *threshold value*.

$F(X)$  = a Boolean function of  $X$ .

$f_A(X)$  = an algebraic function of  $X$ .

A different way of writing equations 4.1 and 4.2 is:

$$\begin{aligned} F(X) &= \operatorname{sgn}\left(\sum_{i=1}^n a_i x_i - T\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{n+1} a_i x_i\right) \end{aligned}$$

where  $a_{n+1} = -T$ ,  $x_{n+1} = 1$  and  $\operatorname{sgn}$  is defined by

$$\operatorname{sgn}(x) = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}$$

In order to avoid having  $F(X) = 0$  which does not make sense, we choose the  $a_i$ 's such that  $\pm a_1 \pm a_2 \pm \dots \pm a_{n+1} \neq 0$ .

By moving to the  $(n + 1)$  dim space of the  $a_i$ 's we see that each equation of the form  $\sum x_i a_i = 0$  is an  $n$ -dim hyperplane for a specific choice of the boolean  $x_i$ 's. Each such hyperplane divides the space of weights into two regions one of which contains the weights we need to realize  $F(X)$  and the other contains the weights for  $\overline{F}(X)$ .

To clarify this argument consider the case  $n = 1$ . To implement  $F(X) = \overline{x}$  (the "not" function), for instance, we should find  $a_1$  and  $a_2$  such that

$$\operatorname{sgn}(a_1 x_1 + a_2) = \overline{x}_1$$

$$\text{or } a_1 x_1 + a_2 > 0 \text{ for } x_1 = -1$$

$$\text{and } a_1 x_1 + a_2 < 0 \text{ for } x_1 = +1$$

which is equivalent to solving the inequalities

$$-a_1 + a_2 > 0$$

$$a_1 + a_2 < 0$$

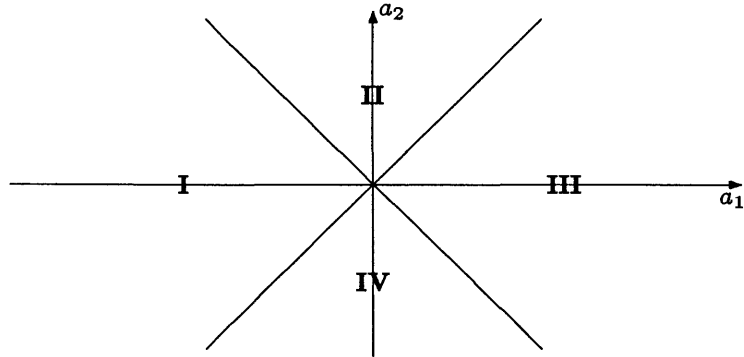


Figure 4.1: Space of Thresholds for one variable

this is best solved geometrically in the space of weights (figure 4.1).

It is clear that the pair  $(a_1, a_2)$  should lie in region **I** for it to realize the “not” function. Similarly, any pair in region **III** realizes the identity function  $F(x_1) = x_1$ , any pair in region **II** corresponds to the “true” function  $F(x_1) = +1$ , and finally any pair in region **IV** corresponds to the “false” function.

From this simple example we can gain a lot of understanding about Threshold logic. By going to the space of weights and drawing all the binary hyperplanes we divide this space into a number of convex cones such that each corresponds to a threshold function. Therefore, from the number of convex cones we can generate with  $2^n$  binary hyperplanes ( $2^n$  since each  $x_i$  in  $\sum x_i a_i = 0$  is either  $+1$  or  $-1$  except for  $x_{n+1}$ ), we can determine the number of possible realizable boolean functions.

In the next section we will see that if  $R(n)$  is the number of regions obtained by dividing  $R^n$  with  $m$  binary hyperplanes of dimension  $(n - 1)$ , then

$$R(n) < 2 \sum_{k=0}^{n-1} \binom{m-1}{k} < 2 \frac{m^n}{n!} \ll 2^{2^n}$$

Note that the space of weights is  $R^{n+1}$  only if we are allowed to vary the threshold; however, if we are constrained to a fixed threshold (e.g. zero), due to implementation issues, then the space of weights is really  $R^n$ .

Threshold function may also be referred to as a *linearly separable function*, a *1-realizable function*, a *linear-input function*, a *majority decision function*, etc.

The term “linearly separable function” means, geometrically, that in the  $n$ -dimensional space the set of vertices represented by  $F$  (i.e.  $\{X : F(X)=1\}$ ) can be separated from the set of vertices represented by the complementary function  $\bar{F}$  by a *hyperplane*. However this hyperplane is now in the space of the binary variables  $x_1, \dots, x_n$  and should not be confused with the binary hyperplanes of the space of weights.

The term “1-realizable function” means that the function can be realized by a *single* threshold logic element.

The term “linear-input function” means that the algebraic equation representing the separating surface contains only first-order terms and therefore is *linear*.

The term “majority decision function” is slightly misleading. Strictly it should mean the function which can be realized by a majority gate, a gate whose output is 1 if and only if a majority of the inputs are 1, for instance, a 2 out of 3 majority gate. However, in the literature, the term majority decision function has been used for a general threshold function.

A threshold logic element is a physical element whose inputs consist of  $n$  binary variables  $\{x_1, x_2, \dots, x_n\}$  and whose output is  $F(X)$  where  $F$  is a threshold function of the  $n$  variables.

In other words, a threshold logic element is a physical realization of a threshold function. Note in this context that a neuron is nothing but a threshold logic element.

## 4.2 Number of Threshold Functions

For up to 7 variables, the number of threshold functions has been exactly determined[27]. The general problem of how many threshold functions there are for  $n$  variables remains unsolved at present. Several upper and lower bounds, however, have been derived for  $n > 7$ .



The calculation of an upper bound for the number of threshold functions,  $R(n)$ , can be easily done by considering the problem of the maximum number of cones into which any number of hyperplanes passing through the origin may divide a space of any dimensions.

Let  $C_{m,n}$  be the maximum number of cones into which  $m$  hyperplanes (of dimension  $n-1$ ) passing through the origin may divide a space of  $n$  dimensions. It is obvious that  $C_{m,1} = 2$  for each  $m > 0$ ,  $C_{m,2} = 2m$ ,  $C_{1,n} = 2$  for any  $n > 0$ . The general formula can be derived by the following argument. Suppose that a formula has been established for  $m-1$  hyperplanes in the  $n$ -dimensional space. The  $m^{\text{th}}$  hyperplane will be divided by  $m-1$  hyperplanes (along at most  $m-1$  hyperplanes) into  $C_{m-1,n-1}$  pieces. Each of these hyperplanar pieces divides the region it belongs to into two new regions, i.e. out of the original  $C_{m-1,n}$  regions at most  $C_{m-1,n-1}$  regions are doubled. Therefore we have:

$$C_{m,n} = C_{m-1,n-1} + C_{m-1,n} \quad (4.3)$$

For  $m \leq n$ , Equation 4.3 can be expanded as follows:

$$\begin{aligned} C_{m,n} &= C_{m-1,n-1} + C_{m-1,n} \\ &= C_{m-2,n-2} + 2C_{m-2,n-1} + C_{m-2,n} \\ &= \dots \\ &= \binom{m-1}{0} C_{1,n-m+1} + \binom{m-1}{1} C_{1,n-m+2} + \dots + \binom{m-1}{m-1} C_{1,n} \end{aligned} \quad (4.4)$$

where

$$\binom{i}{j} = \frac{j!}{i!(j-i)!}$$

Since  $C_{1,n} = 2$  for any integer  $n > 0$ , Equation 4.4 becomes

$$\begin{aligned} C_{m,n} &= 2 \binom{m-1}{0} + 2 \binom{m-1}{1} + \dots + 2 \binom{m-1}{m-1} \\ &= 2 \times 2^{m-1} \\ &= 2^m \end{aligned} \quad (4.5)$$

For  $m > n$ , Equation 4.3 can be expanded as follows:

$$\begin{aligned} C_{m,n} &= \binom{m-1}{0} C_{1,n-m+1} + \dots + \binom{m-1}{m-n} C_{1,1} \\ &+ \binom{m-1}{m-n+1} C_{1,2} + \dots + \binom{m-1}{m-1} C_{1,n} \end{aligned} \quad (4.6)$$

Since  $C_{1,n} = 0$  for any integer  $n \leq 0$  and since  $\binom{j}{i} = \binom{j}{j-i}$ , Equation 4.6 becomes

$$\begin{aligned} C_{m,n} &= 2\binom{m-1}{n-1} + 2\binom{m-1}{m-n+1} + \dots + 2\binom{m-1}{m-1} \\ &= 2\binom{m-1}{n-1} + 2\binom{m-1}{n-2} + \dots + 2\binom{m-1}{0} \\ &= 2\sum_{k=0}^{n-1} \binom{m-1}{k} \end{aligned} \quad (4.7)$$

Thus the number of threshold functions realized by threshold logic elements of  $n$  inputs and  $n$  weights is

$$R_T(n) < 2\sum_{k=0}^{n-1} \binom{2^n - 1}{k} \quad (4.8)$$

which is the number when we are forced to use a fixed value of the threshold (e.g. zero). However, if this restriction is released, then the number of threshold function satisfies:

$$R(n) < 2\sum_{k=0}^n \binom{2^n - 1}{k} \quad (4.9)$$

For purposes of computation,  $2n^2/n!$  is a convenient approximation to  $2\sum_0^n \binom{2^n-1}{1}$ . For a complete proof of this fact see Appendix B in [15].

Finally, a lower bound was also derived by Winder [26] who showed that:

$$R(n) > 2^{0.33n^2}$$

Both bounds can be written down in a single equation:

$$2^{0.33n^2} < R(n) < \frac{2^{n^2}}{n!} \ll 2^{2^n} \quad (4.10)$$

The ratio of  $R(n)$  to the total number of functions of  $n$  variables decreases rapidly as  $n$  increases.

### 4.3 Characterization of Threshold Functions

It can be shown that for a threshold function of  $n$  variables, the complementation of variables, the permutation of variables, and/or the complementation of the function preserves the 1-realizability.

Number of variables, $n$	Number of switching functions, $2^{2^n}$	Number of threshold functions, $R(n)$	Functions of fixed threshold, $R_T(n)$
1	4	4	2
2	16	14	4
3	256	104	14
4	65,536	1882	104
5	$\sim 4.3 \times 10^9$	94,572	1882
6	$\sim 1.8 \times 10^{19}$	15,028,134	94,572
7	$\sim 3.4 \times 10^{38}$	8,378,070,864	15,028,134

Table 4.1: Number of threshold functions

For a complete listing of threshold functions for  $n$  variables, the reader is referred to Appendix C. However, two examples will be given now, where the results of the second one were already used in the previous chapter.

**Example 4.3.1:** For the case of two variables  $x_1$  and  $x_2$ , the threshold functions are all the boolean functions of 2 variables except  $x_1 \oplus x_2$  and  $\overline{x_1 \oplus x_2}$  since as we see in figure 4.2 the *true* vertices cannot be separated from the *false* ones by a hyperplane (which is a line in this case).

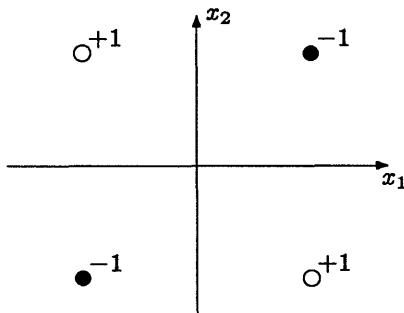


Figure 4.2: Unrealizability of the xor function

**Example 4.3.2:** For the case of three variables, with the constraint that  $T = 0$  only 14 boolean functions are realizable. These functions are[15]:

$$x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3, x_1x_2 + x_3(x_1 \oplus x_2), \bar{x}_1x_2 + x_3(\bar{x}_1 \oplus x_2), x_1\bar{x}_2 + x_3(x_1 \oplus \bar{x}_2), x_1x_2 + \bar{x}_3(x_1 \oplus x_2), \bar{x}_1\bar{x}_2 + x_3(\bar{x}_1 \oplus \bar{x}_2), \bar{x}_1x_2 + \bar{x}_3(\bar{x}_1 \oplus x_2), x_1\bar{x}_2 + \bar{x}_3(x_1 \oplus \bar{x}_2), \bar{x}_1\bar{x}_2 +$$

$$\bar{x}_3(\bar{x}_1 \oplus \bar{x}_2).$$

## Appendix A

### List of Realizable functions

The following lists<sup>1</sup> were obtained from Dertouzos[5]. Checked vectors correspond to attractors of the Hopfield Network.

#### A.1 $s = 3$

- |                                     |           |     |
|-------------------------------------|-----------|-----|
| <input checked="" type="checkbox"/> | (1, 0, 0) | (1) |
| <input type="checkbox"/>            | (1, 1, 1) | (2) |

#### A.2 $s = 4$

- |                                     |              |     |
|-------------------------------------|--------------|-----|
| <input checked="" type="checkbox"/> | (1, 0, 0, 0) | (1) |
| <input checked="" type="checkbox"/> | (1, 1, 1, 0) | (2) |
| <input type="checkbox"/>            | (2, 1, 1, 1) | (3) |

#### A.3 $s = 5$

- |                                     |                 |     |
|-------------------------------------|-----------------|-----|
| <input checked="" type="checkbox"/> | (1, 0, 0, 0, 0) | (1) |
|-------------------------------------|-----------------|-----|

---

<sup>1</sup>Since we are fixing the threshold to be zero, the value of  $s$  is 1 more than the value printed in Dertouzos.

- (1, 1, 1, 0, 0) (2)
- (2, 1, 1, 1, 0) (3)
- (1, 1, 1, 1, 1) (4)
- (2, 2, 1, 1, 1) (5)
- (3, 1, 1, 1, 1) (6)
- (3, 2, 2, 1, 1) (7)

#### A.4 $s = 6$

For the next two cases, only the vectors with nonzero elements will be listed. The unlisted vectors can be obtained from the ones for  $s=3,4$  and 5 by padding enough zeroes.

- (2, 1, 1, 1, 1, 1) (1)
- (4, 3, 3, 2, 2, 1) (2)
- (2, 2, 2, 1, 1, 1) (3)
- (4, 1, 1, 1, 1, 1) (4)
- (5, 2, 2, 2, 2, 1) (5)
- (4, 2, 2, 1, 1, 1) (6)
- (5, 3, 3, 2, 1, 1) (7)
- (3, 2, 1, 1, 1, 1) (8)
- (4, 3, 2, 2, 1, 1) (9)
- (4, 3, 3, 1, 1, 1) (10)
- (5, 4, 3, 2, 2, 1) (11)
- (3, 2, 2, 2, 1, 1) (12)
- (3, 3, 2, 1, 1, 1) (13)
- (3, 3, 2, 2, 2, 1) (14)

## A.5 $s = 7$

- ✓ (5, 3, 3, 1, 1, 1, 1) (1)
- ✓ (6, 4, 4, 2, 1, 1, 1) (2)
- ✓ (5, 4, 2, 2, 2, 1, 1) (3)
- ✓ (3, 2, 2, 1, 1, 1, 1) (4)
- ✓ (6, 5, 3, 3, 2, 1, 1) (5)
- ✓ (4, 3, 2, 2, 2, 1, 1) (6)
- ✓ (9, 8, 5, 4, 3, 2, 2) (7)
- ✓ (7, 6, 3, 3, 2, 2, 2) (8)
- ✓ (5, 4, 3, 3, 2, 1, 1) (9)
- ✓ (6, 5, 4, 3, 3, 2, 2) (10)
- ✓ (4, 3, 3, 2, 2, 2, 1) (11)
- ✓ (5, 4, 3, 3, 3, 2, 1) (12)
- ✓ (5, 5, 3, 2, 2, 1, 1) (13)
- ✓ (4, 4, 2, 2, 1, 1, 1) (14)
- ✓ (3, 3, 1, 1, 1, 1, 1) (15)
- ✓ (7, 6, 5, 4, 3, 2, 2) (16)
- ✓ (5, 4, 4, 3, 2, 2, 1) (17)
- ✓ (6, 5, 4, 4, 3, 2, 1) (18)
- ✓ (3, 2, 2, 2, 2, 1, 1) (19)
- ✓ (4, 4, 3, 2, 2, 1, 1) (20)
- ✓ (3, 3, 2, 2, 2, 1, 1) (21)
- ✓ (2, 2, 1, 1, 1, 1, 1) (22)
- ✓ (8, 7, 6, 5, 4, 3, 2) (23)
- ✓ (4, 3, 3, 3, 2, 1, 1) (24)
- ✓ (5, 5, 4, 3, 3, 2, 1) (25)
- ✓ (7, 6, 5, 5, 4, 3, 3) (26)

<input checked="" type="checkbox"/>	(3, 3, 3, 2, 2, 1, 1)	(27)
<input checked="" type="checkbox"/>	(5, 5, 4, 4, 3, 2, 2)	(28)
<input checked="" type="checkbox"/>	(4, 4, 3, 3, 3, 2, 2)	(29)
<input checked="" type="checkbox"/>	(4, 4, 3, 3, 3, 2, 2)	(30)
<input checked="" type="checkbox"/>	(4, 3, 3, 3, 2, 2, 2)	(31)
<input checked="" type="checkbox"/>	(2, 2, 2, 2, 1, 1, 1)	(32)
<input checked="" type="checkbox"/>	(1, 1, 1, 1, 1, 1, 1)	(33)
<input type="checkbox"/>	(5, 1, 1, 1, 1, 1, 1)	(34)
<input type="checkbox"/>	(7, 2, 2, 2, 2, 1, 1)	(35)
<input type="checkbox"/>	(6, 2, 2, 2, 1, 1, 1)	(36)
<input type="checkbox"/>	(8, 3, 3, 3, 2, 1, 1)	(37)
<input type="checkbox"/>	(5, 2, 2, 1, 1, 1, 1)	(38)
<input type="checkbox"/>	(7, 3, 3, 2, 2, 1, 1)	(39)
<input type="checkbox"/>	(4, 2, 1, 1, 1, 1, 1)	(40)
<input type="checkbox"/>	(7, 3, 3, 3, 1, 1, 1)	(41)
<input type="checkbox"/>	(9, 4, 4, 3, 2, 2, 1)	(42)
<input type="checkbox"/>	(6, 3, 2, 2, 2, 1, 1)	(43)
<input type="checkbox"/>	(3, 1, 1, 1, 1, 1, 1)	(44)
<input type="checkbox"/>	(6, 3, 3, 2, 1, 1, 1)	(45)
<input type="checkbox"/>	(8, 4, 3, 3, 2, 2, 1)	(46)
<input type="checkbox"/>	(5, 2, 2, 2, 2, 1, 1)	(47)
<input type="checkbox"/>	(7, 4, 4, 3, 1, 1, 1)	(48)
<input type="checkbox"/>	(9, 5, 5, 3, 2, 2, 1)	(49)
<input type="checkbox"/>	(5, 3, 2, 2, 1, 1, 1)	(50)
<input type="checkbox"/>	(7, 4, 3, 2, 2, 2, 1)	(51)
<input type="checkbox"/>	(7, 3, 3, 3, 2, 2, 1)	(52)
<input type="checkbox"/>	(7, 4, 4, 2, 2, 1, 1)	(53)
<input type="checkbox"/>	(6, 4, 3, 3, 1, 1, 1)	(54)



- (8, 5, 4, 3, 2, 2, 1) (55)
- (4, 2, 2, 2, 1, 1, 1) (56)
- (6, 3, 3, 2, 2, 2, 1) (57)
- (8, 5, 5, 3, 2, 1, 1) (58)
- (9, 6, 5, 4, 2, 2, 1) (59)
- (6, 4, 3, 2, 2, 1, 1) (60)
- (7, 5, 3, 3, 2, 2, 1) (61)
- (5, 3, 3, 3, 1, 1, 1) (62)
- (7, 4, 4, 3, 2, 2, 1) (63)
- (5, 3, 2, 2, 2, 2, 1) (64)
- (7, 5, 4, 3, 2, 1, 1) (65)
- (4, 3, 2, 1, 1, 1, 1) (66)
- (8, 6, 4, 3, 3, 2, 1) (67)
- (8, 5, 5, 4, 2, 2, 1) (68)
- (5, 3, 3, 2, 2, 1, 1) (69)
- (6, 4, 3, 3, 2, 2, 1) (70)
- (7, 5, 5, 2, 2, 1, 1) (71)
- (8, 6, 5, 3, 3, 1, 1) (72)
- (5, 4, 3, 2, 1, 1, 1) (73)
- (9, 7, 5, 4, 3, 2, 1) (74)
- (6, 5, 3, 2, 2, 2, 1) (75)
- (6, 4, 4, 3, 2, 1, 1) (76)
- (7, 5, 4, 4, 2, 2, 1) (77)
- (7, 5, 4, 3, 3, 2, 1) (78)
- (5, 3, 3, 3, 2, 2, 1) (79)
- (6, 5, 4, 2, 2, 1, 1) (80)
- (7, 6, 4, 3, 2, 2, 1) (81)
- (8, 7, 4, 3, 3, 2, 2) (82)

<input type="checkbox"/>	(7, 5, 5, 3, 3, 1, 1)	(83)
<input type="checkbox"/>	(4, 3, 3, 2, 1, 1, 1)	(84)
<input type="checkbox"/>	(8, 6, 5, 4, 3, 2, 1)	(85)
<input type="checkbox"/>	(5, 4, 3, 2, 2, 2, 1)	(86)
<input type="checkbox"/>	(6, 4, 4, 3, 3, 2, 1)	(87)
<input type="checkbox"/>	(5, 4, 4, 1, 1, 1, 1)	(88)
<input type="checkbox"/>	(7, 6, 5, 2, 2, 2, 1)	(89)
<input type="checkbox"/>	(8, 7, 5, 3, 3, 2, 1)	(90)
<input type="checkbox"/>	(7, 6, 4, 3, 3, 1, 1)	(91)
<input type="checkbox"/>	(5, 4, 4, 2, 2, 1, 1)	(92)
<input type="checkbox"/>	(9, 7, 6, 4, 4, 2, 1)	(93)
<input type="checkbox"/>	(6, 5, 4, 3, 2, 2, 1)	(94)
<input type="checkbox"/>	(7, 5, 5, 4, 3, 2, 1)	(95)
<input type="checkbox"/>	(4, 4, 3, 1, 1, 1, 1)	(96)
<input type="checkbox"/>	(6, 5, 5, 2, 2, 2, 1)	(97)
<input type="checkbox"/>	(7, 6, 5, 3, 3, 2, 1)	(98)
<input type="checkbox"/>	(6, 5, 4, 3, 3, 1, 1)	(99)
<input type="checkbox"/>	(5, 4, 3, 3, 2, 2, 2)	(100)
<input type="checkbox"/>	(7, 6, 5, 4, 4, 3, 2)	(101)
<input type="checkbox"/>	(3, 3, 3, 1, 1, 1, 1)	(102)
<input type="checkbox"/>	(5, 5, 4, 2, 2, 2, 1)	(103)
<input type="checkbox"/>	(5, 5, 3, 3, 3, 1, 1)	(104)
<input type="checkbox"/>	(6, 5, 5, 3, 3, 2, 1)	(105)
<input type="checkbox"/>	(7, 6, 5, 4, 4, 2, 1)	(106)
<input type="checkbox"/>	(6, 5, 4, 4, 3, 3, 2)	(107)
<input type="checkbox"/>	(5, 4, 4, 3, 3, 2, 2)	(108)
<input type="checkbox"/>	(4, 4, 3, 3, 2, 2, 1)	(109)
<input type="checkbox"/>	(3, 3, 2, 2, 2, 2, 1)	(110)

- (5, 4, 4, 3, 3, 1, 1) (111)
- (6, 5, 5, 4, 3, 2, 2) (112)
- (4, 4, 3, 3, 3, 1, 1) (113)
- (5, 5, 4, 4, 3, 3, 3) (114)
- (3, 3, 3, 2, 2, 2, 2) (115)

## Appendix B

### Computer Programs

The following program is written in SCHEME (a dialect of LISP). The input are the vectors  $V_1, \dots, V_s$ . Several procedures can be used to produce a list of the attractors, their numbers, the limit to which each state converges, etc. . .

```
<define (attractor? v)
  (equal? (operation v) v))
;;)-----
<define (operation v) ;; returns the next state of the Hop. Net.
  (signum (*mat T v)))
<define (check v) ;; checks if a given realization a is also
  (signum (*mat (transpose TM) v))) ;; an attractor
;;)-----
<define (signum v)
  (cond ((null? v) nil)
        ((> (top v) 0) (cons +1 (signum (rest v))))
        (<< (top v) 0) (cons -1 (signum (rest v))))
        (else (cons 0 (signum (rest v)))))
<define (spurious-counter list) ;; finds the number of non-degenerate
  (cond ((null? list) 0) ;; attractors.
        (else (+ (count-varieties (car list))
                  (spurious-counter (cdr list)))))
;;)-----
```

*Continued.*

```
(define (count-attractors stream)
  (newline)
  (let ((number-of-attractors 0)
        (present-vertex 0))
    (define (count-helper the-stream)
      (set! present-vertex (1+ present-vertex))
      (cond ((null? the-stream) (1+ number-of-attractors))
            ((attractor? (top-of the-stream))
             (set! number-of-attractors (1+ number-of-attractors))
             (count-helper (remaining-of the-stream)))
            (else (count-helper (remaining-of the-stream)))))
    (count-helper stream)))

(define (count-varieties vector) ;; finds all possible permutations of a
  (let ((k (length vector))      ;; list of elements.
        (define (num-of-rep list)
          (cond ((null? (cdr list)) 1)
                ((= (car list) (cadr list)) (1+ (num-of-rep (cdr list))))
                (else 1)))
        (define (remove-j-elements j list)
          (if (zero? j) list
              (remove-j-elements (- j 1) (cdr list))))
        (if (null? vector) 1
            (* (combination (num-of-rep vector) k)
               (count-varieties (remove-j-elements (num-of-rep vector)
                                                    vector))))))

(define (combination k n)
  (define (fact m)
    (if (= 0 m) 1 (* m (fact (- m 1)))))
  (/ (fact n) (* (fact k) (fact (- n k)))))
););-----
);); Abstraction definition.
););
(define top car)
(define rest cdr)
(define first-of car)
(define first-element car)
(define remaining cdr)
(define top-of head)
(define remaining-of tail)
););-----

);); definition of matrix operation +mat.
););
(define (+mat mat1 mat2)
  (if (null? mat1) '()
      (cons (accumulate-n + 0 (list (first-element mat1)
                                     (first-element mat2)))
            (+mat (rest mat1) (rest mat2)))))
););
);); definition of matrix operation *mat
););
(define (*mat mat vector)
  (mapcar (lambda (row) (dot-product row vector)) mat))
(define (dot-product v w)
  (accumulate + 0 (accumulate-n * 1 (list v w))))

(define (transpose matrix)
  (accumulate-n cons '() matrix))
););-----
);); definition of general list manipulation procedures
```

*Continued.*

```
)))
(define (accumulate combiner initial-value list)
  (if (null? list)
      initial-value
      (combiner (car list)
                (accumulate combiner initial-value (cdr list)))))
(define (accumulate-n op init lists)
  (if (null? (car lists))
      '()
      (cons (accumulate op init (mapcar car lists))
            (accumulate-n op init (mapcar cdr lists)))))
)
)
-----
(define (sum-of-diadic-products vectors)
  (define (diadic-product vector1 vector2)
    (define (scale-by a)
      (lambda (x)
        (* a x)))
    (if (null? vector1) '()
        (cons (mapcar (scale-by (first-element vector1)) vector2)
              (diadic-product (rest vector1) vector2))))
  (if (null? (remaining vectors)) (diadic-product (first-of vectors)
                                                  (first-of vectors))
      (+mat (diadic-product (first-of vectors) (first-of vectors))
            (sum-of-diadic-products (remaining vectors)))))
)
)
-----
(define (filter pred list)
  (cond ((null? list) '())
        ((pred (car list))
         (cons (car list)
               (filter pred (cdr list)))))
        (else (filter pred (cdr list)))))
)
)
-----
))) definition of several matrices, cases N=8, N=16
)))
(define v81 '(1 1 1 1 -1 -1 -1 -1))
(define v82 '(1 1 -1 -1 1 1 -1 -1))
(define v83 '(1 -1 1 -1 1 -1 1 -1))
(define T8M (list v81 v82 v83))
(define T8 (sum-of-diadic-products (list v81 v82 v83)))
)
(define v161 '(1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1))
(define v162 '(1 1 1 1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1))
(define v163 '(1 1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 -1))
(define v164 '(1 -1 1 -1 1 -1 1 -1 1 -1 -1 1 -1 1 -1 1))
(define v165 '(1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1))
(define v166 '(-1 -1 -1 -1))
(define T16M (list v161 v162 v163 v164 v165))
(define T16 (sum-of-diadic-products (list v161 v162 v163 v164 v165)))
)
(define v321 (append v161 v161))
(define v322 (append v162 v162))
(define v323 (append v163 v163))
(define v324 (append v164 v164))
(define v325 (append v165 v166 v166 v166 v166))
(define v326 (append v165 v165))
(define T32M (list v321 v322 v323 v324 v325 v326))
(define T32 (sum-of-diadic-products (list v321 v322 v323 v324 v325 v326)))
)
)
-----
```

## Bibliography

- [1] Ackley D. H., Hinton G. E. and Sejnowski T. J., “*A Learning Algorithm for Boltzmann Machines*”, *Cognitive Science* vol.9, pp 147-169 (1985).
- [2] Amit D. J., “*Neural Networks : Achievements, Prospects and Difficulties*”, *The Physics of Structure Formation, Int. Symposium, Tubingen, Oct. 1986.*
- [3] Amit D.J., Gutfreund, H. Sompolinsky, H. (1985) “*Spin-glass models of neural networks.*” *Phys.Rev.A.* 32: 1007-1018.
- [4] Carpenter G. A. and Grossberg S., “*Neural Dynamics of Category Learning and Recognition : Attention, Memory Consolidation, and Amnesia*”. In *Brain Structure, Learning, and Memory* . J. Davis, R. Newburgh, and E. Wegman (Eds.). AAAS Symposium Series, 1985.
- [5] Dertouzos M., “*Threshold Logic: A Synthesis Approach*”. (Cambridge, Mass: MIT Press, 1965).
- [6] Garey M. R. and Johnson D.S., “*Computers and Intractability: A Guide to the theory of NP-Completeness*”. Bell Telephone Laboratories, 1979.
- [7] Hajek B., “*A Tutorial Survey of Theory and Applications of Simulated Annealing*”. *IEEE Proceedings of 24th Conference on Decision and Control.* Ft. Lauderdale, FL. Dec. 1985.
- [8] Hestenes D., “*How The Brain Works*”. Presented at the Third Workshop on Maximum Entropy and Bayesian Methods in Applied Statistics. (Univ. of Wyoming, Aug. 1-4, 1983).

- [9] Hopfield J. J., “*Neural Networks and Physical Systems With Emergent Collective Computational Abilities*”, Proc. Natl. Acad. Sci. USA, vol 79, pp 2554-2558, April 1982, Biophysics.
- [10] Hopfield J. J., “*Neurons With Graded Response Have Collective Computational properties Like Those of Two-State Neurons*”, Proc. Natl. Acad. Sci. USA, vol 81, pp 3088-3092, May 1984, Biophysics.
- [11] Hopfield J. J. and Tank D. W., “*Neural Computation of Decisions in Optimization Problems*”, Biol. Cyber. vol 52, pp 141-152, 1985.
- [12] Kandel E. R., “*Nerve Cells and Behavior*”. Principles of Neural Science 2ed., Kandel and Schwartz, 1985, Elsevier Science Publishing Co.
- [13] Kohonen T., *Self-Organization and Associative Memory*, pp 127-161, Springer-Verlag, New York.
- [14] Levy B. C. and Adams M. B., “*Global Optimization With Stochastic Neural Networks*”. IEEE First International Conference on Neural Networks, San Diego, Ca, June 21-24, 1987.
- [15] Lewis P. M. and Coates C.L., *Threshold Logic*. John Wiley & sons Inc. New York 1967.
- [16] Lippmann R. P., “*An Introduction to Computing with Neural Nets*”, IEEE ASSP Magazine, pp 4-22, April 1987.
- [17] Little W. A. and Shaw G. L., “*Analytic Study of the Memory Storage Capacity of a Neural Network*”, Mathematical Biosciences vol 39, pp 281-290, 1978.
- [18] Little W. A., “*The Existence of Persistent States in the Brain*”, Mathematical Biosciences vol 19, pp 101-120, 1974.
- [19] Minsky M. and Papert S., *Perceptrons*. (Cambridge, Mass: MIT Press, 1969).
- [20] Mitter S. K., “*Estimation Theory and Statistical Physics*” in “*Stochastic Processes and their Applications*”. Lecture Notes in Mathematics, 1203, 157-176, Springer-Verlag.



- [21] Newman Charles M., "*Memory Capacity in Neural Network Models: Rigorous Lower Bounds*". Preprint available from author. Results were announced at the Rutgers University Statistical Mechanics Meeting, Dec. 19, 1986 and at a Los Alamos National Laboratory CNLS seminar, Dec. 31, 1986. Prof. Newman is affiliated with Dept. of Math., Univ. of Arizona.
- [22] Sejnowski T. J. and Rosenberg C. R., "*Parallel Networks That Learn to Pronounce English Text*". *Complex Systems* vol. 1, (1987), pp 145-168.
- [23] Sheng C. L., *Threshold Logic*, Academic Press, London and New York, 1969.
- [24] Stevens C. F., "*The Neuron*", *Scientific American*, September 1979.
- [25] Winder R. O., "*More About Threshold Logic*", *Proc. IRE*, pp 55-64, July 1960.
- [26] Winder R. O., "*Threshold Logic*", Doctoral Dissertation, Mathematics Department, Princeton University.
- [27] Winder R. O., "*Enumeration of Seven Argument Threshold Functions*," *I.E.E.E. Trans. Electron. Computers*, **EC-14**, (3), 315-325 (June 1965).