

Louisiana State University

## LSU Scholarly Repository

---

LSU Historical Dissertations and Theses

Graduate School

---

1989

### The Capacity of Artificial Neural Networks Using the Delta Rule.

Donald Louis Prados

*Louisiana State University and Agricultural & Mechanical College*

Follow this and additional works at: [https://repository.lsu.edu/gradschool\\_disstheses](https://repository.lsu.edu/gradschool_disstheses)

---

#### Recommended Citation

Prados, Donald Louis, "The Capacity of Artificial Neural Networks Using the Delta Rule." (1989). *LSU Historical Dissertations and Theses*. 4869.

[https://repository.lsu.edu/gradschool\\_disstheses/4869](https://repository.lsu.edu/gradschool_disstheses/4869)

This Dissertation is brought to you for free and open access by the Graduate School at LSU Scholarly Repository. It has been accepted for inclusion in LSU Historical Dissertations and Theses by an authorized administrator of LSU Scholarly Repository. For more information, please contact [gradetd@lsu.edu](mailto:gradetd@lsu.edu).

## INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

# U·M·I

University Microfilms International  
A Bell & Howell Information Company  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
313.761-4700 800/521-0600



Order Number 9025331

**The capacity of artificial neural networks using the delta rule**

Prados, Donald Louis, Ph.D.

The Louisiana State University and Agricultural and Mechanical Col., 1989

**U·M·I**  
300 N. Zeeb Rd.  
Ann Arbor, MI 48106



THE CAPACITY OF ARTIFICIAL  
NEURAL NETWORKS  
USING THE DELTA RULE

A Dissertation

Submitted to the Graduate Faculty of the  
Louisiana State University and  
Agricultural and Mechanical College  
in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy

in

Department of Electrical and Computer Engineering

by  
Donald L. Prados  
B.S., Tulane University, 1981  
M.S.(BME), Tulane University, 1982  
M.S.(EE), Louisiana State University, 1986  
December, 1989

## **ACKNOWLEDGEMENT**

I would like to thank Dr. Subhash C. Kak for his help, comments, advice, support, criticism, patience, understanding, and friendship.

## Table of Contents

<b>CHAPTER 1. Introduction</b> .....	1
1.1 Physiology .....	3
1.2 Layout of Dissertation .....	5
<b>CHAPTER 2. Hopfield Neural Networks</b> .....	6
2.1 Update Equations of the Hopfield Model .....	7
2.2 Energy .....	8
2.3 The Hebbian Rule .....	10
2.4 Capacity Using the Hebbian Rule .....	12
<b>CHAPTER 3. Neural Network Learning</b> .....	14
3.1 The Delta Rule .....	16
3.2 Capacity Using the Delta Rule .....	18
3.3 Hetero-Associative Neural Networks .....	40
3.4 Higher-Order Terms .....	45
3.5 Unlearning .....	49
3.6 Negative Feedback in Learning .....	50
3.7 Networks of Limited Connectivity .....	54
<b>CHAPTER 4. Non-Binary Neural Networks</b> .....	59
4.1 Update Equations of the Non-Binary Model .....	59
4.2 Delta Rule for the Non-Binary Model .....	59
4.3 Capacity of the Non-Binary Model .....	62
<b>CHAPTER 5. Shift-Invariant Neural Networks</b> .....	65
5.1 Update Equations for the Shift-Invariant Model .....	68
5.2 Delta Rule for the Shift-Invariant Model .....	69
5.3 Capacity of the Shift-Invariant Model .....	70
<b>CHAPTER 6. Conclusions</b> .....	73
<b>REFERENCES</b> .....	78
<b>VITA</b> .....	81



## ABSTRACT

Over the past several years, several papers have been published on the capacity of Hopfield neural networks. It has been shown that, using the Hebbian rule, the capacity of the Hopfield model is approximately  $N/4\log N$ . The number of patterns one can store in a neural network, however, can be greatly increased by using learning algorithms other than the Hebbian rule such as the delta rule. The motivation behind this dissertation is to study, both analytically and experimentally, the information capacity of various neural network models using a modified version of the delta-rule algorithm.

This modified version of the delta-rule algorithm allows one to store significantly more patterns than previously thought possible. Both analytical and experimental results are presented on the number of patterns one can expect to be able to store using this algorithm. The analytical results suggest that the probability of *separating*  $m$  patterns of  $N$  bits each is about 50% for  $m = 2N$ ; experimental results show that the probability of *storing*  $m$  patterns of  $N$  bits each is about 50% for  $m = 1.5N$ .

Modifications of the Hopfield model including a non-binary model, a shift-invariant model, and models that use higher-order terms are also discussed. Learning rules for these models are presented along with discussion of their capacity.

Also, the trade-off between capacity and performance of neural networks is discussed along with a further modification of the delta rule that leads to significant improvement in performance.

## CHAPTER 1. Introduction

Artificial neural networks have become very popular lately among computer scientists, cognitive scientists, engineers, psychologists, solid state physicists, and others. This popularity is due, in part, to the fascinating similarities between artificial neural network memory and human memory. Like the brain, an artificial neural network is an associative memory. As such, it can be used as a content-addressable memory, as a model of a set of related objects (i.e., a semantic net), as a tool in pattern recognition, and as a tool in solving optimization problems, among others. While the brain contains about a trillion neurons, however, state-of-the-art neural networks contain no more than a few thousand neurons. Like the first generation of computer users working with only a few kilobytes of memory, the first generation of neural network users are severely limited in the amount of memory available. Unlike the first generation of computer users, however, neural network users are often unfamiliar with, or even misinformed about, the information capacity of their systems. The motivation behind this dissertation is to study, both analytically and experimentally, the information capacity of various neural network models.

Over the past several years, several papers have been published on the capacity of neural networks. Hopfield predicted from experimental results<sup>1</sup> that the number of patterns that one could store in a neural network of  $N$  neurons using the Hebbian weight matrix is approximately  $0.15N$ . McEliece *et al.* have shown<sup>2</sup> that the capacity of the Hopfield model is approximately  $N/4\log N$ . They also assume that the Hebbian rule is used to determine the weight matrix  $T$ .

The number of patterns one can store in a neural network can be greatly increased by using learning algorithms other than the Hebbian rule such as the delta rule. Abu-Mostafa and St. Jacques<sup>3</sup> claimed that the number of arbitrary patterns that one can store in a Hopfield network of  $N$  neurons is bounded above by  $N$  regardless of the method of obtaining the weight matrix. They define the capacity of a neural network as the largest number  $m$  such that *every* possible set of  $m$  patterns that one wishes to store can be stored successfully in the neural network. Several serious problems with this definition will be discussed.

Venkatesh<sup>4</sup> also calculated theoretical bounds on the capacity that are unrelated to the method of obtaining the connection matrix. He has calculated what he calls the "epsilon capacity" of neural networks, which he defines as "the largest rate of growth of the number of associations that can be stored such that, with high probability, the retrieved memory after one synchronous step differs from the desired associated memory in no more than (essentially) a fraction  $\epsilon$  of components." He shows that for large  $N$ , and with  $0 \leq \epsilon < 1/2$ , the epsilon capacity  $C_\epsilon(N)$  is at most  $2N/(1 - 2\epsilon)$ . Whereas Abu-Mostafa and St. Jacques and Venkatesh present theoretical limits on the number of patterns one can store in a neural network, this dissertation discusses practical limits on the number of patterns one can store. Both experimental and analytical results are presented on the capacity of neural networks using a modification of the delta rule.

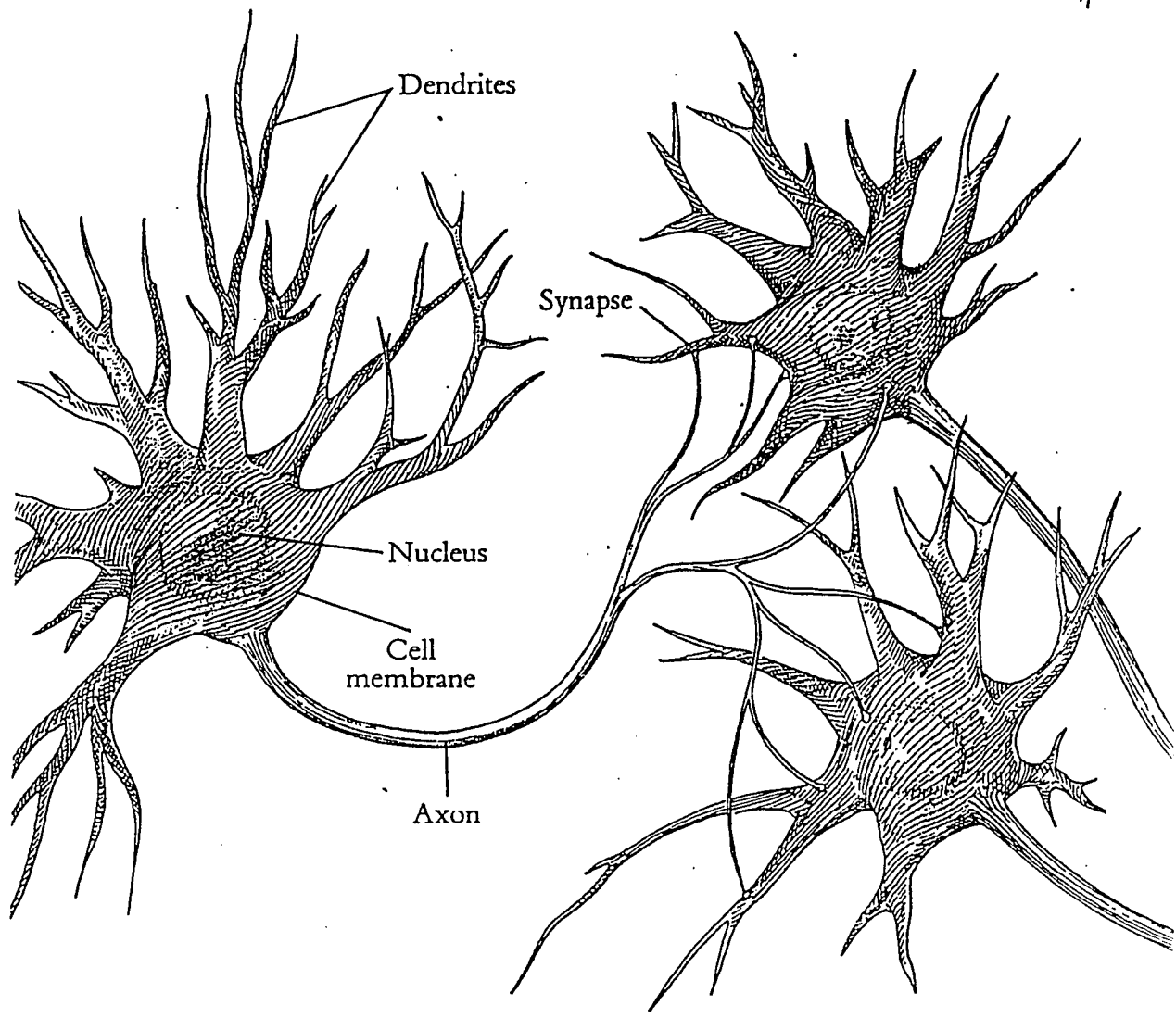
Since the probability of *storing* sets of  $m$  patterns of  $N$  bits each depends on the ability of a single neuron to *separate* sets of  $m$  patterns of  $N$  bits each, an upper bound

on the number of patterns that one can store can be determined by finding a bound on the number of patterns one neuron can separate. Analytical results show that the probability of one neuron successfully separating  $m$  patterns of  $N$  neurons each is greater than 50% if  $m$  is less than  $2N$  and less than 50% if  $m$  is greater than  $2N$ . If one defines capacity in terms of the probability of success at storing a set of patterns, one could say that the 50% capacity, or  $C_{50\%}$ , has an upper bound of  $2N$ . Experimental results show that  $C_{50\%}$  is about  $1.5N$  using the modified delta-rule algorithm presented in this dissertation, provided  $N$  is not too small (not less than about 50).

### 1.1 Physiology

The neurons of artificial neural networks are, of course, simplistic models of the neurons (or *nerve cells*) of our nervous system. Our brains contain about  $10^{12}$  nerve cells. A typical nerve cell has inputs from hundreds or thousands of other cells, and sends its output, in turn, to hundreds or thousands of cells.<sup>5</sup> Thus, the total number of interconnections, or *synapses*, is about  $10^{14}$  to  $10^{15}$ . The nucleus of the cell, along with the mitochondria and other organelles, is contained in the *cell body*, Figure 1. The *dendrites* are the branching fibers coming off of the cell body that serve to pick up signals sent from other neurons. The *axon* is the cylinder-shaped nerve fiber coming off the cell body, generally a few millimeters in length. The *cell membrane* encloses the entire cell.

Synapses can be either excitatory or inhibitory. At any given time the membrane potential is the result of all the excitatory and inhibitory influences added together. At



rest the membrane potential is about 70 millivolts, positive outside. When the membrane potential is *depolarized* from 70 millivolts to about 40 millivolts, the process becomes regenerative, and the depolarization continues all the way to 20 millivolts, negative outside. This pulse starts on the axon close to where it joins the cell body and travels along the axon away from the cell body toward the terminal branches at a rate of 0.1 to 10 or so meters per second. The magnitude of the pulse is determined by the neuron itself and not by the intensity of the depolarization that sets it going. The cells feeding into the neuron only effect the rate of fire of the cell.

## 1.2 Layout of Dissertation

In Chapter 2, the Hopfield neural network model in its simplest form is presented. The chapters that follow will deal with various modifications of this model. Chapter 3 addresses the capacity of the Hopfield model using the delta rule. In Chapter 4 a non-binary model is presented. In particular, models that allow each neuron one of four possible states are discussed. Chapter 5 discusses a neural network model that allows for shift-invariant pattern recognition. The final chapter, Chapter 6, concludes the dissertation.

---

## CHAPTER 2. Hopfield neural networks

Hopfield made a major contribution to the field of neural networks in 1982<sup>1</sup> when he pointed out that certain collective computational properties emerge when many very simple *neurons* were connected together. He pointed out that the outputs of the  $N$  binary neurons could be considered a point in  $N$ -dimensional state space and that updating a neuron's output using his update rule would always result in decreasing the energy of the network, leading, eventually, to an *energy minimum*. The energy minima in state space can be thought of as *stored memories*. Since applying the update rule will not change the state of such a stored memory, it is a *stable state* of the network. If a pattern similar to a particular stored memory is supplied to the network by setting the outputs of the neurons to that pattern, continuously updating the network until an energy minimum is reached can "retrieve" that stored memory. Failure to retrieve the stored memory will occur if the energy minimum reached is a stable point other than the desired stored memory. The network can function as a content-addressable memory (CAM) by retrieving a pattern given only part of it, and it can function as an error detection and correction device by retrieving a pattern given a noisy version of it. It can also serve as a pattern classifier by storing prototypes of a pattern class and retrieving the prototype when other members of the class are presented.

The stable points are sometimes referred to as *attractors*. The *attraction basin* of an attractor is the set of states which have the attractor as the next stable state reached after continuously updating the network. Since the next stable state reached depends on the order in which the neurons are updated, the attraction basin to which a pattern

belongs depends on the manner in which the neural network is updated.

## 2.1 Update Equations of the Hopfield model

Neural network models generally assign a weight to each synapse, positive for excitatory synapses and negative for inhibitory synapses. Artificial neurons are usually given a value of +1 if they are firing (or ON) and either 0 or -1 if they are not firing (or OFF). Whether a neuron is ON or OFF depends on the summation or integration of each input neuron times its synaptic weight. Only if this sum is greater than a particular threshold does the neuron fire.

The most common neural network model is the Hopfield model. The Hopfield model, in its simplest form, uses the following equations to update the output of neuron  $i$ :

$$x_i = \sum_{j=1}^N T_{ij} V_j \quad [1]$$

$$V_i = \begin{cases} 1 & x_i > U_i \\ -1 & x_i \leq U_i \end{cases} \quad [2]$$

where  $V_i$  is the output of neuron  $i$ ,  $T_{ij}$  is the weight associated with input  $j$  to neuron  $i$ , and  $U_i$  is the threshold of neuron  $i$ . In this dissertation the thresholds will usually be set to zero. The weight matrix  $T$  is sometimes referred to as the *synaptic weight matrix* or *synaptic connection matrix* since weight  $T_{ij}$  represents the strength of the synapse from neuron  $j$  to neuron  $i$ . A positive weight indicates an excitatory synapse, and a negative weight indicates an inhibitory synapse.



The simple Hopfield model assumes each of the  $N$  neurons of the neural net has inputs from each of the other  $N-1$  neurons and a synaptic weight associated with each input. There is no direct feedback ( $T_{ii} = 0$ , for all  $i$ ). The state of the neural network is simply the vector made up of the outputs of the  $N$  neurons. Updating the network is viewed as being synchronous if the outputs of the neurons are updated simultaneously. It is viewed as being asynchronous if the outputs are updated one at a time.

To *input* a pattern, one sets the outputs of the neurons to the binary sequence of the pattern. If one inputs a pattern that is not a stable state, the outputs continually change based on the above update equations until a stable state is reached. To *store* a pattern (or *learn* a pattern), one must make that pattern a stable state by appropriately modifying the synaptic connection matrix  $T$ .

## 2.2 Energy

Hopfield showed that updating a neuron's output using Equations 1 and 2 will always decrease the energy of the neural net if the weight matrix  $T$  is symmetric about the main diagonal and all elements of the main diagonal are 0. The elements of the main diagonal will be zero if direct feedback of neurons to themselves is not allowed. The weight matrix will be symmetric if the weight representing input  $i$  to neuron  $j$  is always equal to the weight representing input  $j$  to neuron  $i$ . In other words, the effect that neuron  $i$  has on neuron  $j$  is always equal to the effect that neuron  $j$  has on neuron  $i$ . Since the energy function Hopfield defined produces a local minimum when the network is stable and updating a neuron will always decrease the energy, the network will

always reach a stable state if the neurons are updated asynchronously using the update equations, Equations 1 and 2.

Using the following measure of the energy of the neural network, Hopfield showed that, if  $T_{ij} = T_{ji}$  and  $T_{ii} = 0$ , then Equations 1 and 2 will always lead to a stable state:

$$E = -\frac{1}{2} \mathbf{V}^t \mathbf{T} \mathbf{V} + \mathbf{U}^t \mathbf{V} \quad [3]$$

$$= -\frac{1}{2} \sum_i \sum_j T_{ij} V_j V_i + \sum_i U_i V_i \quad [4]$$

He showed that, if any neuron  $V_k$  changes when applying Equations 1 and 2, then the above energy function will not increase. The change in energy  $\Delta E$  due to updating  $V_k$  can be calculated as follows. First, separate out all terms of Equation 4 that involve  $V_k$ .

$$E = -\frac{1}{2} \left[ \sum_{\substack{i \neq k \\ j \neq i}} \sum_{\substack{j \neq i \\ j \neq k}} T_{ij} V_j V_i + \sum_{j \neq k} T_{kj} V_j V_k + \sum_{i \neq k} T_{ik} V_k V_i + T_{kk} V_k V_k \right] + \sum_{i \neq k} U_i V_i + U_k V_k \quad [5]$$

Assuming  $T_{ij} = T_{ji}$  for all  $i$  and  $j$ , this equation can be rewritten as:

$$E = -\frac{1}{2} \sum_{\substack{i \neq k \\ j \neq i \\ j \neq k}} \sum_{\substack{j \neq i \\ j \neq k}} T_{ij} V_j V_i - \sum_{j \neq k} T_{kj} V_j V_k - \frac{1}{2} T_{kk} + \sum_{i \neq k} U_i V_i + U_k V_k \quad [6]$$

Notice that  $V_k V_k$  in Equation 5 will always equal +1 since  $V_k$  must equal +1 or -1.

$\Delta E$  in response to  $\Delta V_k$  can now be calculated:

$$\Delta E = -\sum_{j \neq k} T_{kj} V_j \Delta V_k + U_k \Delta V_k = -\Delta V_k (\sum_{j \neq k} T_{kj} V_j - U_k) \quad [7]$$

If  $T_{kk} = 0$ , this can be written as

$$\Delta E = -\Delta V_k (x_k - U_k) \quad [8]$$

According to Equation 2,  $V_k$  will only change if  $x_k > U_k$  and  $V_k = -1$  or if  $x_k \leq U_k$  and  $V_k = 1$ . In either case the change in energy will be non-positive. Only if  $x_k = U_k$  and  $V_k = 1$  will  $\Delta E$  be 0. The neural network will, therefore, always reach a stable state if one uses Equations 1 and 2 to asynchronously update the neural net and if all  $T_{ii}$  are required to be 0. Notice that the change in energy,  $\Delta E$ , will be the same whether or not  $T_{kk}$  is required to be 0. A non-zero  $T_{kk}$  will only affect  $E$  (see Equations 6 and 7). If  $T_{kk} > 0$ ,  $E$  will be lower for each state. This will lead to more stable states of the network. Some of the states, however, may not be local minima. (In the extreme case,  $T$  would be the identity matrix, and all states would be stable). On the other hand, if  $T_{kk} < 0$ ,  $E$  will be higher for each state. This will tend to reduce the number of stable states; however, updating a neuron may lead to an increase in energy. If one requires that updating the network always leads to a stable state, then each  $T_{ii}$  must be non-negative.

### 2.3. The Hebbian Rule

As mentioned previously, to store a pattern, one must make the pattern a stable state by appropriately modifying  $T$ . Hopfield utilizes an information storage algorithm inspired by Hebb. If one wishes to store the set of  $m$  patterns  $V^s$ ,  $s = 1, \dots, m$ , each  $T_{ij}$

for which  $i \neq j$  is calculated as

$$T_{ij} = \sum_{s=1}^m V_i^s V_j^s \quad [9]$$

The diagonal elements are set to zero to avoid direct feedback of a neuron to itself:

$$T_{ii} = 0 \quad [10]$$

Notice that this equation will produce a symmetric connection matrix with  $T_{ij} = T_{ji}$ .

The number of nonredundant connections is thus  $N(N-1)/2$  (or  $\binom{N}{2}$ ).

*Example 1:* Let the patterns to be stored be  $V^1 = (+ + + +)$ ,  $V^2 = (+ + - -)$ , and  $V^3 = (- - + +)$  and assume each  $U_i = 0$ . The Hebbian connection matrix is:

$$T = \begin{bmatrix} 0 & 3 & -1 & -1 \\ 3 & 0 & -1 & -1 \\ -1 & -1 & 0 & 3 \\ -1 & -1 & 3 & 0 \end{bmatrix} \quad [11]$$

A simple check using Equations 1 and 2 will reveal that this  $T$  does indeed successfully store all three patterns. Notice that  $V^3$  is the complement of  $V^2$ . Usually the complement of a stored state is also a stable state. This follows from the fact that, if pattern  $V^s$  produces  $x^s$ , then the complement of  $V^s$  will produce  $-x^s$ . Only if one or more  $x_i^s = 0$  can the complement of  $V^s$  not be a stable state. Also, notice that pattern  $(- - -)$  is also a stable state. Such a state is often referred to as a *spurious* state since it was not intentionally stored, but arose in the process of making other states stable.

This spurious state is, of course, the complement of pattern  $V^1$ . Spurious states that are not complements of intentionally-stored states will be referred to as *non-complement spurious* states. Such states seem to occur with about the same frequency as complement spurious states.

#### 2.4 Capacity Using the Hebbian Rule

The capacity of neural networks that use a Hebbian T is rather easy to calculate. McEliece *et al.*<sup>2</sup> showed that, if  $m$  patterns are chosen at random, the maximum asymptotic value of  $m$  in order that *most* of the  $m$  patterns are exactly recoverable is

$$m = \frac{N}{2 \ln N}. \quad [12]$$

With the added restriction that *every* one of the  $m$  patterns be recoverable exactly,  $m$  can be no more than

$$m = \frac{N}{4 \ln N}. \quad [13]$$

Our experimental results show that Equation 13 gives an accurate prediction of  $C_{100\%}$ . This can be seen in Table 1. The data in this table was obtained as follows. For each value of  $N$ , 100 attempts were made to store  $m$  randomly-generated patterns, each bit of each pattern having an equal probability of being +1 or -1. Checks were made to ensure that no two patterns differed by less than two bits or by more than  $N - 2$  bits; otherwise, it would not be possible to store the set of patterns, as will be proved in Section 3.2. The table gives the number of successes in 100 attempts to store  $m$

patterns of  $N$  bits each. For example, of 100 attempts to store 3 random patterns of 25 bits each, 96 were successful. When  $m$  was less than the value calculated using Equation 13, all 100 attempts were successful.

N	Number of Patterns											$\frac{N}{4 \ln N}$
	1	2	3	4	5	6	7	8	9	10	11	
25	100	100	96	81	46	35	12	2	0	0	0	1.9
30	100	100	97	90	77	38	22	2	1	0	0	2.2
35	100	100	99	98	87	57	29	10	2	0	0	2.5
40	100	100	100	97	93	57	38	8	4	0	0	2.7
45	100	100	100	100	86	77	64	39	15	3	0	3.0

Table 1. Successes in 100 trials, using Hebbian weight matrix.

### CHAPTER 3. Neural Network Learning

The number of patterns one can store in a neural network can be greatly increased by using learning algorithms other than the Hebbian rule such as the delta rule.<sup>6</sup> Abu-Mostafa and St. Jacques<sup>3</sup> indicated that the number of arbitrary patterns that one can store in Hopfield network of  $N$  neurons is bounded above by  $N$  regardless of the method of obtaining the weight matrix. Their definition of capacity  $m$  is that *every* set of  $m$  patterns that one wishes to store has an associated zero-diagonal weight matrix  $T$  (and threshold vector  $U$ ) such that each pattern is a fixed point. Their proof is as follows. Given  $K$  patterns of  $N$  bits each, for each of the  $2^K$  choices of the first bit of each pattern, one must find a different threshold function of  $N - 1$  variables with  $K$  points in the domain. Using the patterns of Example 1, notice that, for the first bit of each pattern to remain the same upon updating requires

$$\begin{aligned} T_{12} + T_{13} + T_{14} &> 0 \\ T_{12} - T_{13} - T_{14} &> 0 \\ -T_{12} + T_{13} + T_{14} &\leq 0 \end{aligned} \tag{13b}$$

Each of the eight choices of the first bit requires a different set of weights,  $\{T_{12}, T_{13}, T_{14}\}$ . Let  $B_{N-1}^K$  be the number of different threshold functions of  $N - 1$  variables with  $K$  points in the domain. One must have

$$B_{N-1}^K \geq 2^K. \tag{14}$$

Abu-Mostafa and St. Jacques give a proof, which they attribute to Cameron<sup>7</sup> and Winder,<sup>8</sup> that an upper bound to  $B_{N-1}^K$  is

$$B_{N-1}^K \leq 2 \sum_{i=0}^{N-1} \binom{K-1}{i}. \quad [15]$$

If  $K > N$ , however, then

$$B_{N-1}^K \leq 2 \sum_{i=0}^{N-1} \binom{K-1}{i} < 2 \sum_{i=0}^{K-1} \binom{K-1}{i} = 2 \times 2^{K-1} = 2^K. \quad [16]$$

This, obviously, contradicts the condition of Equation 14. If  $m > N$ , there will exist some sets of  $m$  patterns which cannot be stored.

There are two problems with this bound. First, if  $m < N$ , there will still be sets of  $m$  patterns that cannot be stored due to the fact that, if two patterns differ by 1 or  $N - 1$  bits, they cannot both be stored. Second, For  $m > N$ , the probability of storing  $m$  randomly-generated patterns can still be very high. For large  $N$ , the probability of storing, say,  $1.5N$  patterns will be greater than 99%, as our experimental results show.

Another important contribution to understanding the capacity of neural networks is that of Venkatesh.<sup>4</sup> Venkatesh calculated what he calls the "epsilon capacity" of neural networks. He introduces error-tolerance into the retrieval mechanism by specifying components in the retrieved memory which are treated as don't-cares. The epsilon capacity  $C_\epsilon(N)$  is defined to be "the largest rate of growth of the number of associations that can be stored such that, with high probability, the retrieved memory after one synchronous step differs from the desired associated memory in no more than (essentially) a fraction  $\epsilon$  of components." He shows that for large  $N$ , and with  $0 \leq \epsilon < 1/2$ , the epsilon capacity  $C_\epsilon(N)$  is at most  $2N/(1 - 2\epsilon)$ . Thus, for perfect recall,  $C_0(N) = 2N$ . Venkatesh does not provide the proofs of his assertions (he refers the



reader to his PhD thesis), but mentions that they utilize "large deviation Central Limit Theorems, very large deviation estimates, and function counting theorems in combinatorial geometry." He concludes from his results and those of Abu-Mostafa and St. Jacques<sup>3</sup> that, if  $N < m < 2N$ , then there are guaranteed to be choices of  $m$  patterns which cannot be stored regardless of the choice of weight matrix  $T$ , but that such choices of patterns will constitute an asymptotically negligible proportion of the total number of choices as  $N$  approaches infinity.

In this chapter, I show that, with the delta rule, the capacity is between  $N$  and  $2N$ . I show that certain sets of patterns cannot be stored in a neural network with a zero-diagonal weight matrix and how to recognize such sets of patterns. In particular, if two patterns differ by either 1 bit or  $N - 1$  bits, they cannot both be stable.<sup>9</sup> For this reason, one can only define capacity in terms of the likelihood of storing a set of  $m$  patterns rather than in terms of the ability to store every possible set of  $m$  patterns. My definition of capacity  $C_{p\%}$  is that one can store a random set of  $C_{p\%}$  patterns in a neural network of  $N$  neurons with a probability of  $p\%$ . For example, if the probability of storing a random set of  $m$  patterns is 90%, then the  $C_{90\%}$  capacity is  $m$ .

### 3.1 The Delta Rule

To store the  $N$ -bit pattern  $V^s$ , first calculate the next state of each neuron using Equations 1 and 2 with pattern  $V^s$  as the input for each calculation, and then compare this to the desired next state, pattern  $V^s$  itself. If the next state of neuron  $i$  as calculate from Equations 1 and 2 is not equal to the desired next state of the neuron,  $V_i^s$ , modify

the weight matrix  $T$  as follows:

Case 1. The desired output,  $V_i^s = 1$  but the actual output,  $V_i = -1$ : Increment each term in Equation 1, thereby increasing  $x_i$ . For each  $j \neq i$ , if  $V_j^s = 1$ , increment  $T_{ij}$ . If, on the other hand,  $V_j^s = -1$ , decrement  $T_{ij}$ . If the amount of change is sufficient, Equations 1 and 2 will produce an output of  $V_i = 1$  the next time  $V_i$  is calculated with  $V^s$  as input.

Case 2. The desired output,  $V_i^s = -1$  but the actual output,  $V_i = 1$ : Decrement each term in Equation 1, thereby decreasing  $x_i$ . For each  $j \neq i$ , if  $V_j^s = 1$ , decrement  $T_{ij}$ . If, on the other hand,  $V_j^s = -1$ , increment  $T_{ij}$ .

These two cases can be combined using the algorithm: if  $V_i \neq V_i^s$ , then, for each  $j \neq i$ , increment  $T_{ij}$  by  $V_i^s V_j^s$ . This will always result in incrementing  $x_i$  if it is negative but should be positive, and it will always result in decrementing  $x_i$  if it is positive but should be negative. This delta rule can be written as: if the actual output,  $V_i$  is not equal to the desired output,  $V_i^s$ , then, for each  $j \neq i$ ,

$$\Delta T_{ij} = V_i^s V_j^s \quad [17]$$

or simply: for each  $j \neq i$ ,

$$\Delta T_{ij} = c (V_i^s - V_i) V_j^s \quad [18]$$

where  $c$  is a constant. Notice that the corrections to the weight matrix due to  $V_i \neq V_i^s$  cause *only row  $i$*  of the weight matrix to be changed; thus, if the amount of change in

the weights of row  $i$  of  $T$  is sufficient, these weights need only be adjusted once. For Equation 18,  $c = \frac{1}{2}$  will produce a change of  $\pm 1$  in each  $T_{ij}$  and a change of  $N - 1$  in  $x_i$ . This is often enough to change the sign of  $x_i$ . If it is not, Equation 18 can be repeated for pattern  $V^s$  until the pattern is stored. This algorithm is guaranteed to store any  $N$ -bit binary pattern.

Notice that "multiplying"  $V_i^s$  by  $V_j^s$  is equivalent to performing the 'exclusive nor' operation. The arithmetic required for this delta rule (Equation 18) is extremely simple if  $c = \frac{1}{2}$  since  $T_{ij}$  is either incremented by 1 or decremented by 1 depending on the signs of  $V_i^s$  and  $V_j^s$ .

### 3.2 Capacity Using the Delta Rule

As mentioned previously, even if  $N$  is very large, there will be pairs of patterns that cannot both be stored simultaneously:

*Theorem 1.* If two patterns differ by only one bit, they *cannot* be stored simultaneously in a network that does not allow direct self-feedback of neurons.

*Theorem 2.* If two patterns differ by  $N - 1$  bits, they *usually cannot* be stored simultaneously in a network that does not allow direct self-feedback of neurons.

The proofs are as follows. If two patterns differ in bit  $i$  only, the calculation of  $x_i$  (see Equation 1) will be identical regardless of which of the two patterns is applied to the network. Since  $x_i$  will be identical for both patterns, so will  $V_i$ . If one of the two patterns is a stable state, the other cannot also be a stable state. By a similar argument,

if two patterns differ in  $N - 1$  bits, they usually cannot be stored simultaneously. In this case, let bit  $i$  be the only bit in which the two patterns match. If application of one pattern produces  $\hat{x}_i$ , application of the other pattern will produce  $-\hat{x}_i$  (see Equation 1). Only if  $x_i = 0$  can both patterns be stored simultaneously. Note that, as  $N$  increases, the likelihood that two random patterns differ by 1 or  $N - 1$  bits decreases exponentially.

Since, for any neural network requiring a zero-diagonal weight matrix, there exist pairs of patterns that cannot be stored simultaneously, the definition used by Abu-Mostafa and St. Jacques must be revised.

We ran several tests to determine experimentally the capacity of the model described by Equations 1 and 2 using the delta rule of Equation 18. Attempts were made to store randomly generated patterns (each bit with equal probability of being 1 or -1) of length  $N = 10$  and  $N = 30$ .

For  $N = 10$ , 100 attempts were made to store 1 pattern, 100 attempts were made to store 2 patterns, and so on up to 15 patterns. Each time a pattern was randomly generated, it was rejected if it differed by less than 2 bits or more than  $N - 2$  bits from any pattern already in the set. The results, Table 2, indicate that the probability of storing  $m$  randomly-generated patterns in a neural network of 10 neurons is greater than 90% if  $m$  is less than 9, provided no pair of patterns in the set differ by less than 2 bits or more than  $N - 2$  bits. If capacity  $C_{90\%}$  is defined as  $\frac{m}{N}$  such that the probability of successfully storing  $m$  randomly generated patterns of  $N$  bits each is 90%,

then  $C_{90\%}$  for  $N = 10$  is approximately 0.9.

Number of Patterns	4	5	6	7	8	9	10	11	12	13	14
Trials Successful	100	99	96	99	91	80	67	34	17	11	0

Table 2. Number of successes in 100 trials for  $N = 10$ .

Table 3 shows results using  $N = 30$ . Since there are  $2^{30}$  possible patterns, it is extremely unlikely that two randomly-generated patterns differ by 1 or  $N - 1$  bits; and so, checking the distance between patterns is not necessary. Notice that, for  $N = 30$ , the success rate does not drop below 90% until the number of patterns reaches 37. For  $N = 30$ ,  $C_{90\%}$  is about 1.2.

Number of Patterns	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
Trials Successful	100	98	98	99	93	86	85	73	62	48	30	29	15	9	3	1	1	0

Table 3. Number of successes in 100 trials for  $N = 30$ .

Often we were able to store sets of more than  $N$  patterns. Because of the unexpected nature of this phenomenon, checks were made by hand to confirm that one can store more than  $N$  patterns in this type of neural network.

*Example 2:* Using  $N = 9$ , the delta rule generates the matrix

$$T = \begin{bmatrix} 0 & 2 & 0 & -2 & 4 & -2 & 4 & -4 & 2 \\ 3 & 0 & -5 & 3 & -3 & -3 & -5 & 1 & -5 \\ 4 & -2 & 0 & 6 & -6 & -2 & 2 & 0 & -4 \\ -2 & 4 & 4 & 0 & 2 & -6 & -2 & -2 & 4 \\ 6 & -4 & -2 & 2 & 0 & 0 & -2 & 4 & -2 \\ 1 & -1 & -1 & -3 & 1 & 0 & -3 & -5 & 1 \\ 3 & -3 & 3 & -3 & -1 & -5 & 0 & 1 & -3 \\ -3 & 3 & 1 & -3 & 3 & -5 & 1 & 0 & 5 \\ 4 & -6 & -4 & 6 & -6 & 2 & -4 & 2 & 0 \end{bmatrix} \quad [19]$$

for the twenty patterns

(1)	-1	-1	-1	-1	-1	1	-1	-1	1
(2)	-1	1	-1	1	-1	1	-1	-1	1
(3)	-1	1	1	1	-1	-1	-1	1	1
(4)	1	-1	-1	-1	1	1	-1	-1	1
(5)	1	-1	1	1	1	-1	1	-1	-1
(6)	-1	-1	1	-1	-1	1	1	-1	-1
(7)	-1	-1	-1	-1	1	1	-1	1	1
(8)	-1	-1	-1	-1	-1	-1	1	1	1
(9)	1	-1	1	-1	-1	1	1	-1	1
(10)	1	1	-1	-1	1	-1	1	1	-1
(11)	1	-1	1	1	1	-1	1	1	1
(12)	-1	-1	1	1	-1	-1	1	1	1
(13)	-1	-1	1	1	-1	1	-1	-1	1
(14)	-1	1	-1	-1	-1	1	-1	-1	-1
(15)	1	-1	-1	-1	1	1	1	-1	-1
(16)	1	-1	1	-1	1	-1	1	1	-1
(17)	1	1	-1	-1	1	1	-1	-1	-1
(18)	1	1	-1	1	1	1	-1	-1	1
(19)	1	1	1	1	-1	-1	1	-1	-1
(20)	1	1	1	1	1	-1	1	1	-1

Using Equations 1 and 2, one can check that these twenty patterns are indeed all stable

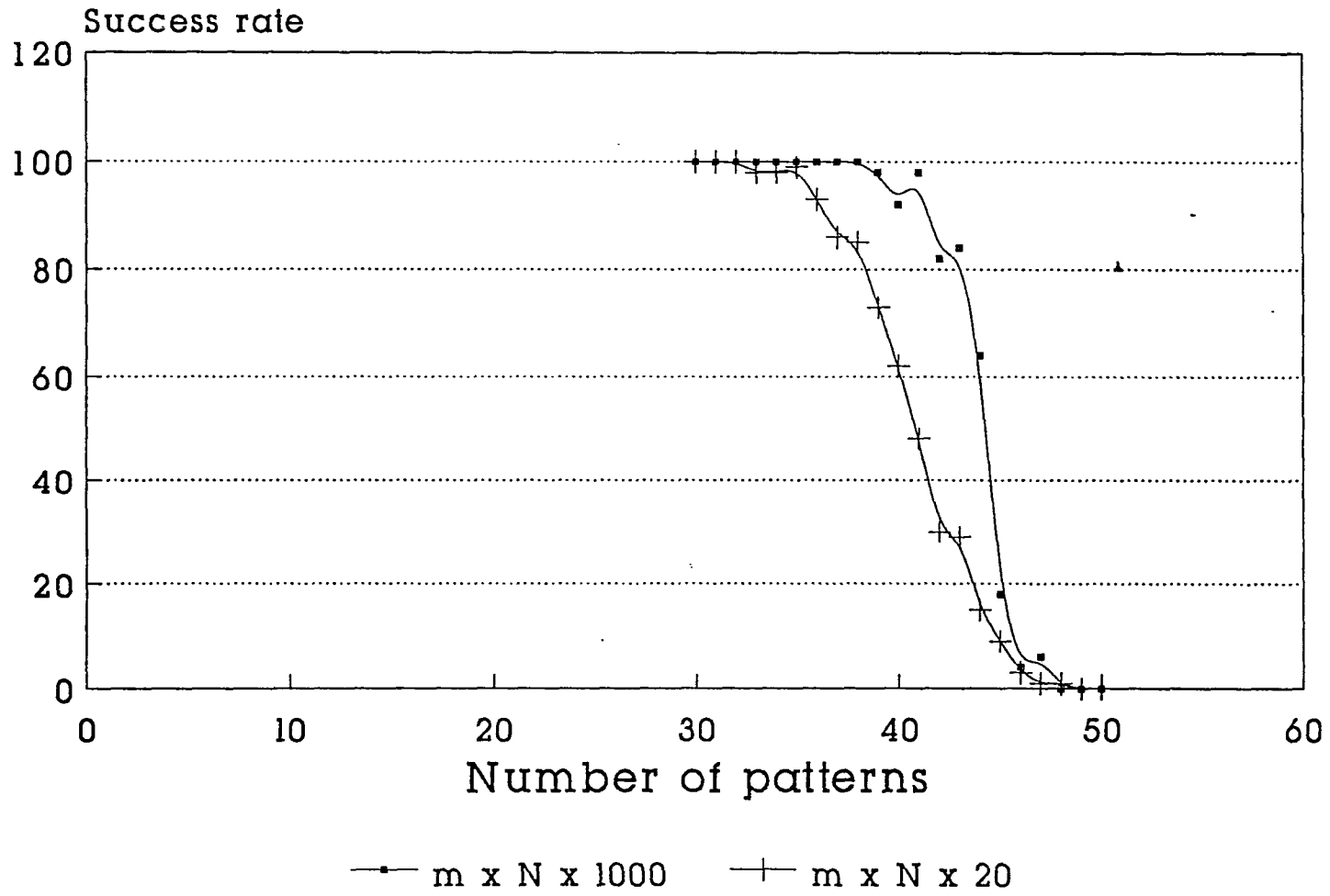
states of the neural network.

Note that some limit must be set on the number of iterations that one runs the learning algorithm. Simply because the algorithm "failed" to store a set of patterns does not mean that the set could not have been stored if the algorithm had been allowed to continue beyond the chosen limit. If one defines an iteration as successfully storing each pattern once (note that each time one is stored others may be forgotten), then a good choice for the limit on the number of iterations is  $N$  times  $m$  for the case of  $m$  less than  $N$ . If  $m$  is greater than  $N$ , however, the number of iterations required increases rapidly as  $m$  increases. The results of Table 3 use  $m \times N \times 20$  iterations, as do most of the results in presented in this dissertation. To find out how much improvement can be obtained using a very high limit on the number of iterations, the test was repeated using a limit of  $m \times N \times 1000$  iterations. Figure 2 shows a comparison between the results shown in Table 3 and the results using this very high limit on the number of iterations. Notice that complete failure occurred at about the same point, but that the drop-off is much sharper using a very high limit on the number of iterations.

There are two ways of storing a set of  $m$  patterns using the delta rule. In both cases, if the actual output  $V_i$  is not equal to the desired output,  $V_i^s$ , row  $i$  of the weight matrix  $T$  is modified. This is done using Equation 18.

The first method is to store the patterns one at a time. Store the first pattern, then store the second pattern, and so on until the last pattern has been stored. As each pattern is learned, however, other patterns may be forgotten. After learning the last

Figure 2. Limit comparison,  $N = 30$





pattern, one must go back to see if the first is still stored (still a stable state). If it is not, it will be necessary to re-learn it. One can continually iterate through the set of patterns, checking if each pattern is still stored and re-learning it if it has been forgotten, until either all patterns have been successfully learned or a limit on the maximum number of iterations has been reached.

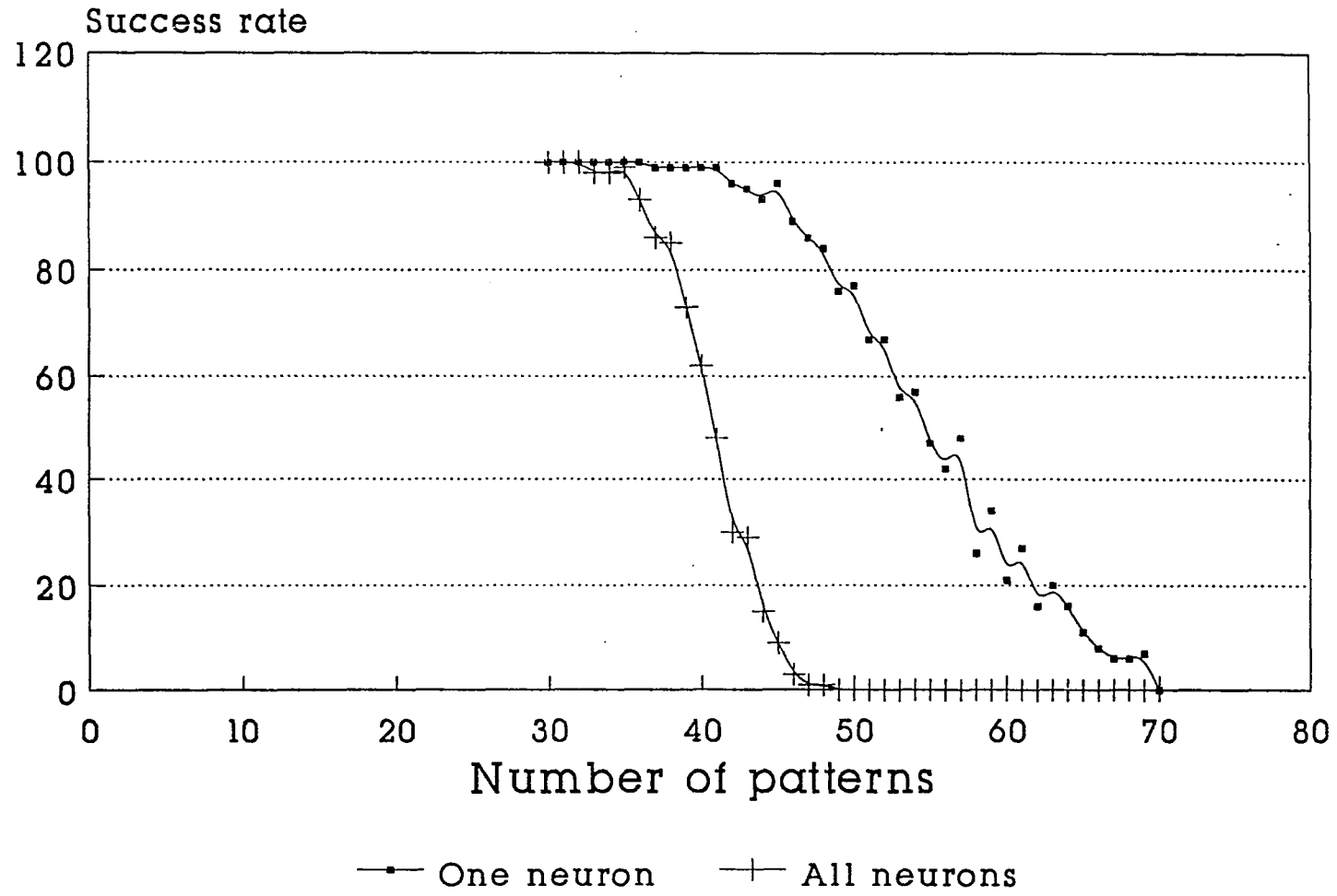
The second method tends to be much faster. Notice that the calculation of the output of neuron  $i$ ,  $V_i$ , depends only on row  $i$  of the weight matrix  $T$  (see Equations 1 and 2). A pattern is stored if the output of each neuron does not change when that pattern is input to the net. The set of  $m$  patterns is stored, of course, if the output of each neuron does not change when any of the  $m$  patterns is input to the net. One can therefore store the set of patterns one bit at a time rather than one pattern at a time. If the first row of the weight matrix is modified successfully, the state of the first neuron will remain unchanged if any of the patterns are input to the net. One can then move on to the second neuron. Modifying the second row of the weight matrix such that the state of second neuron does not change when any of the patterns is input to the net does not change the first row. One can modify the weight matrix one row at a time from top to bottom. When one has successfully modified the bottom row of the matrix, the set of patterns is stored. If failure occurs at any row, the set of patterns has not been successfully learned. If, for example, one is unable to modify the first row of the weight matrix in such a way that the first neuron will remain unchanged when any of the patterns is applied to the net, one will be unable to store the set of patterns and the algorithm fails. When the algorithm fails, it tends to fail much more quickly than does the

first algorithm. Also, when it succeeds, it tends to succeed more quickly.

Neuron  $i$  must separate the set of patterns into two groups, each pattern in the first group having a value of 1 for neuron  $i$ , and each pattern in the second group having a value of -1 for neuron  $i$ . Row  $i$  of the weight matrix  $T$  must be modified in such a way that  $x_i > 0$  when a pattern of the first group is applied to the net and  $x_i \leq 0$  when a pattern of the second group is applied to the net. Since  $T_{ii} = 0$ , each neuron must be able to separate  $m$  vectors of length  $N - 1$ . If, for any neuron  $i$ , the set of vectors obtained by removing neuron  $i$  from the set of patterns is not linearly separable, the set of patterns cannot be stored.

If the set of patterns has been randomly generated, each neuron has an equal probability of separating the set of patterns. If the probability of one neuron being able to separate the set of patterns successfully is very close to 100%, the probability of successfully storing the set of patterns will be high. Figure 3 shows results of a test to experimentally determine how many randomly-generated patterns of length  $N = 30$  one can store. The top curve shows the ability of one bit to separate a set of randomly-generated patterns. Every time a set of less than 37 patterns was generated, the set of patterns was successfully separated. Of 100 attempts to separate sets of 37 patterns, 99 were successful. The success rate gradually declined until all attempts to separate a set of 71 patterns were unsuccessful. The bottom curve shows results of attempting to store entire sets of randomly-generated patterns. When the number of patterns is increased to the point where 1 bit can no longer successfully separate the sets of patterns virtually 100% of the time, the ability to successfully store the sets of patterns

Figure 3. Delta rule,  $N = 30$



begins to decrease very rapidly. All attempts to store 49 patterns failed even though 1 bit could successfully separate 49 patterns 76% of the time. This occurs because every one of the 30 bits must successfully separate the set of patterns for the set of patterns to be successfully stored. Figure 4 shows a repeat of the above test using a very high limit on the number of iterations. Again, when the number of patterns is increased to the point where 1 bit can no longer successfully separate the sets of patterns virtually 100% of the time, the ability to successfully store the sets of patterns begins to decrease very rapidly.

The results of tests such as the one discussed above suggest that one can obtain interesting results on the capacity of neural nets by studying the ability of one neuron to separate sets of patterns. Our analysis requires defining the energy of the neural network as did Hopfield. (The constant  $\frac{1}{2}$  is left out for simplicity).

$$E = -\sum_i \sum_j T_{ij} V_j V_i \quad [20]$$

Using Equation 1, one can express the energy of state  $V^s$  as

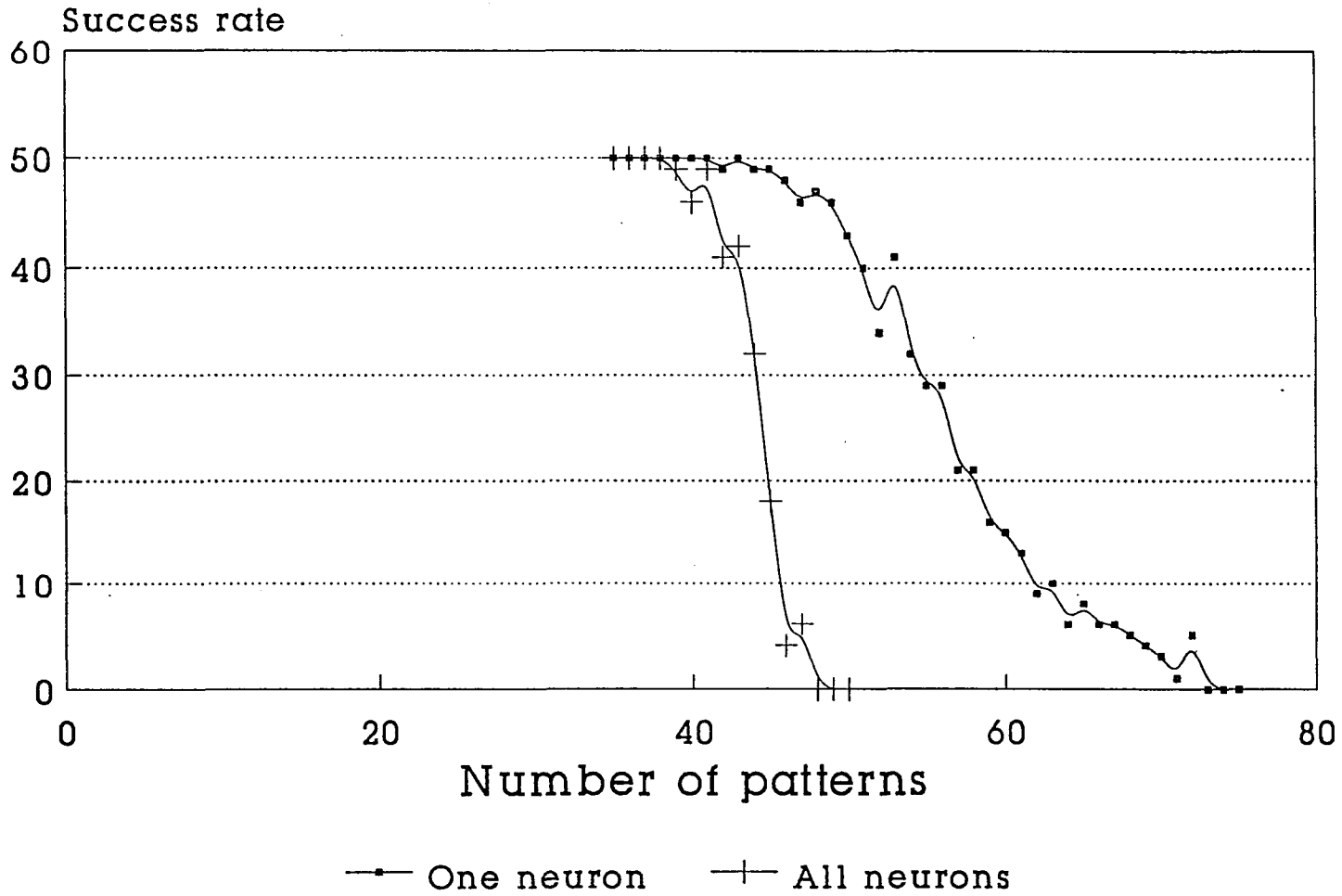
$$E^s = -\sum_i x_i^s V_i^s \quad [21]$$

One can decompose  $E^s$  to obtain

$$E^s = \sum_{i=1}^N E_i^s \quad [22]$$

where  $E_i^s$  is the component of  $E^s$  in the  $i$ th direction (the contribution of the  $i$ th neuron

Figure 4. High limit,  $N = 30$



to  $E_i^s$ ).  $E_i^s$  can, of course, be written as

$$E_i^s = -x_i^s V_i^s. \quad [23]$$

where

$$x_i^s = \sum_j T_{ij} V_j^s \quad [24]$$

For state  $V^s$  to be stable, none of the  $E_i^s$  can be positive. If  $x_i^s$  is nonzero and the sign of  $x_i^s$  is equal to  $V_i^s$ , then the energy  $E_i^s$  will be negative. A negative  $E_i^s$ , therefore, implies that, when  $V^s$  is input to the net (using Equations 1 and 2),  $V_i^s$  does not change and row  $i$  of matrix  $T$  does not need to be changed. If  $x_i^s = 0$ , then  $E_i^s = 0$ , and  $V_i^s$  changes sign only if it is positive (see Equation 2). If any  $E_i^s$  is positive (or zero when  $V_i^s = 1$ ), then pattern  $V^s$  is not a stable state and has not been successfully stored. If one then applies the delta rule, Equation 18, row  $i$  will be changed and  $E_i^s$  will be decreased by  $N - 1$  (assuming  $c = 1/2$  in Equation 18).

As noted earlier, each time a pattern is stored, other patterns may be forgotten, since storing a pattern involves changing  $T$ . Similarly, modifying row  $i$  of  $T$  to reduce  $E_i^s$  may increase, decrease, or leave unchanged each of the  $E_i^s$ ,  $s \neq s$ . The amount of change can be easily calculated. If  $V_i \neq V_i^s$  and  $c = 1/2$ , applying the delta rule of Equation 18 will change each weight in row  $i$  of  $T$  (except  $T_{ii}$ , which we will assume will remain zero) by

$$\Delta T_{ij} = V_i^s V_j^s \quad [25]$$

After the change in T,

$$E_i^s = -V_i^s \sum_{j \neq i} (T_{ij} + V_i^s V_j^s) V_j^s \quad [26]$$

$$= -V_i^s \sum_{j \neq i} T_{ij} V_j^s - V_i^s \sum_{j \neq i} V_i^s V_j^s V_j^s \quad [27]$$

$$= -V_i^s x_i^s - V_i^s V_i^s \sum_{j \neq i} V_j^s V_j^s \quad [28]$$

The first term in Equation 28 is the value of  $E_i^s$  prior to the change in T. The change in  $E_i^s$  is

$$\Delta E_i^s = -V_i^s V_i^s \sum_{j \neq i} V_j^s V_j^s \quad [29]$$

The summation in this equation is approximately the dot product of patterns  $V^s$  and  $V^s$ :

$$\sum_{j \neq i} V_j^s V_j^s = V^s \cdot V^s - V_i^s V_i^s \quad [30]$$

The dot product has limits

$$-N \leq V^s \cdot V^s \leq N \quad [31]$$

and, assuming the patterns are randomly generated with each bit having an equal probability of being +1 or -1, has an expected value of zero. Combining Equations 29 and 30, one obtains:

$$\Delta E_i^s = - V_i^s V_i^s (V^s \cdot V^s - V_i^s V_i^s) \quad [32]$$

$$= 1 - V_i^s V_i^s (V^s \cdot V^s) \quad [33]$$

An example is given in Table 4. The data are from a failed attempt of neuron 1 to separate the following set of 8 patterns of length 7:

(1)	-1	-1	-1	-1	-1	1	1
(2)	1	-1	1	-1	-1	1	1
(3)	1	1	1	-1	1	-1	1
(4)	-1	-1	1	-1	1	1	-1
(5)	-1	-1	1	1	-1	-1	1
(6)	1	-1	-1	-1	-1	1	-1
(7)	1	1	-1	-1	-1	1	1
(8)	-1	1	1	-1	1	-1	-1



Pattern count s	Energy of bit l of pattern s							
	$E_1^1$	$E_1^2$	$E_1^3$	$E_1^4$	$E_1^5$	$E_1^6$	$E_1^7$	$E_1^8$
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	4	-6	0	2	2	-2	-2	-2
4	2	-6	-6	2	2	2	-2	2
5	2	-4	-6	-4	4	4	-4	0
6	2	-2	-6	-2	-2	2	-6	2
7	6	-4	-2	0	-4	-4	-8	0
8	6	-4	-2	0	-4	-4	-8	0
1	6	-4	-2	0	-4	-4	-8	0
2	0	0	-4	0	-4	0	-4	4
3	4	-6	-4	2	-2	-2	-6	2
4	4	-6	-4	2	-2	-2	-6	2
5	4	-4	-4	-4	0	0	-8	0
6	4	-4	-4	-4	0	0	-8	0
7	8	-6	0	-2	-2	-6	-10	-2
8	8	-6	0	-2	-2	-6	-10	-2
1	8	-6	0	-2	-2	-6	-10	-2
2	2	-2	-2	-2	-2	-2	-6	2
3	2	-2	-2	-2	-2	-2	-6	2
4	2	-2	-2	-2	-2	-2	-6	2
5	2	-2	-2	-2	-2	-2	-6	2
6	2	-2	-2	-2	-2	-2	-6	2
7	2	-2	-2	-2	-2	-2	-6	2
8	2	-2	-2	-2	-2	-2	-6	2
1	6	-4	2	-4	0	-4	-8	-4
2	0	0	0	-4	0	0	-4	0
3	4	-6	0	-2	2	-2	-6	-2
4	2	-6	-6	-2	2	2	-6	2
5	2	-6	-6	-2	2	2	-6	2
6	2	-4	-6	0	-4	0	-8	4
7	6	-6	-2	2	-6	-6	-10	2
8	6	-6	-2	2	-6	-6	-10	2
1	10	-8	2	0	-4	-8	-12	-4
2	4	-4	0	0	-4	-4	-8	0
3	4	-4	0	0	-4	-4	-8	0
4	2	-4	-6	0	-4	0	-8	4
5	2	-4	-6	0	-4	0	-8	4
6	2	-4	-6	0	-4	0	-8	4
7	6	-6	-2	2	-6	-6	-10	2
8	6	-6	-2	2	-6	-6	-10	2
1	10	-8	2	0	-4	-8	-12	-4
2	4	-4	0	0	-4	-4	-8	0
3	4	-4	0	0	-4	-4	-8	0
4	2	-4	-6	0	-4	0	-8	4
5	2	-4	-6	0	-4	0	-8	4
6	2	-4	-6	0	-4	0	-8	4
7	6	-6	-2	2	-6	-6	-10	2
8	6	-6	-2	2	-6	-6	-10	2

Table 4.  $E_1^s$  as neuron 1 attempts to separate 8 patterns of 7 bits each.

The data were generated as follows:

1. Set all weights of T to 0. Set s, the pattern count, to 1.
2. Calculate and print each  $E_1^s$ ,  $s = 1, 2, \dots, 8$ . If all are negative, bit 1 has successfully separated the set of patterns, no further modifications to the weights of row 1 of T will be necessary, and the algorithm exits.
3. For bit 1 of pattern s: If  $E_1^s > 0$  (or if  $x_1^s = 0$  and  $V_1^s = 1$ ), modify the first row of T according to the delta rule, thereby, decreasing  $E_1^s$  by  $N - 1$ .
4. Increment s (or reset s to 1 if  $s = 8$ ), and go to step 2.

The algorithm iterates through the set of patterns until either all the patterns are stored or the limit on the number of iterations is reached.

Since all weights of T are initially zero and the first bit of pattern 1 is -1, the weights of the first row of T are not changed after applying pattern 1 to the net. When pattern 2 is applied to the net,  $x_1^2 = 0$  but  $V_1^2 = 1$ ; therefore, the weights of the first row of T are changed according to Equation 18. This reduces  $E_1^2$  by  $N - 1$  and has side affects of increasing  $E_1^1$ ,  $E_1^4$ , and  $E_1^5$ , while decreasing  $E_1^6$ ,  $E_1^7$ , and  $E_1^8$ , as can be seen in Table 4. Since  $E_1^3$  does not change, remaining 0, and  $V_1^3 = 1$ , row 1 of T is changed again, reducing  $E_1^3$  by  $N - 1$ , increasing  $E_1^6$  and  $E_1^8$ , while decreasing  $E_1^1$ , and leaving the rest unchanged.

The main purpose of showing this example is to examine more closely the learning algorithm and to see how the algorithm can fail. Notice that  $E_1^1$  remains non-negative throughout the test. Reducing  $E_1^2$  by  $N - 1$  increased  $E_1^1$  from 0 to 4;

reducing  $E_1^3$  by  $N - 1$  decreased  $E_1^1$  by 2, bringing it to 2; reducing  $E_1^4$  by  $N - 1$  and reducing  $E_1^5$  by  $N - 1$  did not affect  $E_1^1$ ; reducing  $E_1^6$  by  $N - 1$  increased  $E_1^1$  by 4, bringing it to 6; and  $E_1^7$  and  $E_1^8$  did not have to be reduced. The value of  $E_1^1$  now being 6, applying the delta rule of Equation 18 brings it back down to 0. After a second cycle through the set of patterns,  $E_1^1$  has been increased to 8, since decreasing  $E_1^2$  and  $E_1^6$  each increased  $E_1^1$  by 4. After several cycles, the algorithm enters an infinite loop: reducing  $E_1^3$  from 0 to -6 decreases  $E_1^1$  by 2, but reducing  $E_1^6$  from 0 to -6 and reducing  $E_1^8$  from 2 to -4 each increase  $E_1^1$  by 4. The net affect is to increase  $E_1^1$  by 6, thus offsetting the reduction of  $E_1^1$  by  $N - 1$ .  $E_1^1$ , therefore, remains positive; and pattern 1 is never successfully stored.

Note that, whenever  $E_i^s$  is reduced by  $N - 1$ , the amount of change in  $E_i^s$  is always the same (Equation 33).

*Example 3:* Using Equation 33, one can calculate  $\Delta E_1^1$  resulting from decreasing  $E_1^2$  by  $N - 1$ :

$$\Delta E_1^1 = 1 - (-1)3 = 4. \quad [34]$$

Notice that, in Table 4, whenever T is modified because  $E_1^2$  is positive,  $E_1^2$  is decreased by  $N - 1$  (this also follows from Equation 33) and  $E_1^1$  is increased by 4. In fact, this occurs the very first time T is modified.

If one uses the Hebbian outer-product weight matrix, one can calculate  $E_i^s$  from Equation 33:

$$E_i^s = \sum_{s=1}^m 1 - V_i^s V_i^s (V^s \cdot V^s) \quad [35]$$

$$= m - \sum_{s=1}^m V_i^s V_i^s (V^s \cdot V^s) \quad [36]$$

$$= m - N - \sum_{s \neq 1} V_i^s V_i^s (V^s \cdot V^s) \quad [37]$$

Writing this in the form

$$E_i^s = -(N-1) + (m-1) - \sum_{s \neq 1} V_i^s V_i^s (V^s \cdot V^s) \quad [38]$$

allows one to see the contribution of pattern  $V^s$ ,  $-(N-1)$ , along side of the contributions of the other  $m - 1$  patterns. One can see from this equation that, if  $m = N$ , each neuron should have a 50% chance of successfully separating the set of patterns. The probability of storing the set of patterns, however, will be  $(1/2)^N$  since each of the  $N$  neurons must separate the set of patterns.

If, on the other hand, one uses the delta-rule algorithm rather than the Hebbian, during each iteration through the set of patterns, only some of the patterns will require  $T$  to be modified. For the example of Table 4, if every pattern  $V^s$ ,  $s \neq 1$ , caused a change in  $T$ , the total change in  $E_1^1$  would be 10 (this follows from Equation 33). When the algorithm is in the infinite loop, only 3 of the other  $N - 1$  patterns change  $T$ , causing a change in  $E_1^1$  of +6. This is enough to offset exactly the reduction in  $E_1^1$  of 6 that results from applying the delta rule to  $V_1^1$ .

Suppose one uses the delta-rule algorithm to store  $m$  randomly-generated patterns of  $N$  bits each. During each iteration through the set of patterns, only some of the patterns will cause  $T$  to be modified. Suppose a fraction  $p$  of the other  $m - 1$  patterns initially cause a change in  $E_i^s$ . The change in  $E_i^s$  will be approximately

$$\Delta E_i^s = -(N-1) + p(m-1) - p \sum_{s \neq s} V_i^s V_i^s (V^s \cdot V^s) \quad [39]$$

If  $p(m-1) < N-1$ ,  $\Delta E_i^s$  is more likely to be negative than positive since the dot product of two random patterns has an expected value of zero. If each  $\Delta E_i^s$ ,  $s = 1, \dots, m$ , is more likely to be negative than positive, then one would expect  $p$  to be less than  $1/2$ . If  $p$  is decreased, the chance of each  $E_i^s$  being negative increases, and  $p$  falls further. Therefore, if  $p(m-1) < N-1$ , then one can expect neuron  $i$  to be able to successfully separate a set of  $m$  patterns with probability greater than 50%. Since  $p \leq 1$ , one can expect neuron  $i$  to separate a set of  $m$  patterns with high probability if  $m \leq N$ . Since the expected value of  $p$  is initially  $1/2$  (given a random  $T$  or an all-zero  $T$ ), if  $m = 2N$ , there will be about an equal probability that  $\Delta E_i^s$  will decrease as increase during the first iteration. Since this is true for each  $E_i^s$ , the expected value of  $p$  will remain approximately  $1/2$ . If  $p$  drops below  $1/2$ , each  $E_i^s$  will tend to decrease; whereas, if  $p$  rises above  $1/2$ , each  $E_i^s$  will tend to increase. One should, therefore, expect, at best, a 50% chance of separating a set of randomly-generated patterns if  $m = 2N$ . Since each neuron will have, at best, a 50% chance of separating the set of patterns, the probability of storing the set of patterns will be very low.

Our experimental results confirm that, if  $m = 2N$ , each neuron has, at best, a 50% chance of successfully separating the patterns. Figure 5 shows results of a study on the ability of one neuron to *separate* randomly-generated patterns. The number of neurons,  $N$ , was varied from 20 to 150; and the number of patterns,  $m$ , was varied from  $\frac{6}{5}N$  to over  $2N$ . For each value of  $m$ , 50 attempts were made to separate  $m$  randomly-generated patterns of  $N$  bits each. The top line shows the lowest value of  $m$  for which all 50 attempts failed, and the second line from the bottom shows the lowest value of  $m$  for which 1 or more of 50 attempts failed. The lines  $m = N$  and  $m = 2N$  are shown for comparison. Our results showed about a 20% success rate for  $m = 2N$ . It must be noted, however, that a limit had to be set on the number of iterations to run the delta-rule algorithm. Often, failure to store a set of patterns occurred because the algorithm was not allowed to run long enough. The limit on the number of iterations for these tests was set to  $m \times N \times 20$ . As can be seen in Figure 2, increasing the limit on the number of iterations can significantly improve the results. Also, as  $N$  increases, the success rate at  $m = 2N$  improves significantly, as will be shown shortly.

Figure 6 shows results of a study on the ability to *store* randomly-generated patterns. The number of neurons,  $N$ , was varied from 20 to 100; and the number of patterns,  $m$ , was varied over a suitable range. For each value of  $m$ , 50 attempts were made to store  $m$  randomly-generated patterns of  $N$  bits each. One line shows the lowest value of  $m$  for which all 50 attempts failed, another line shows the lowest value of  $m$  for which one or more of 50 attempts failed, and the lines  $m = N$  and  $m = 2N$  are again shown for comparison. Notice that the slope of both curves is about  $1.5N$ .

Figure 5. Delta rule, one neuron

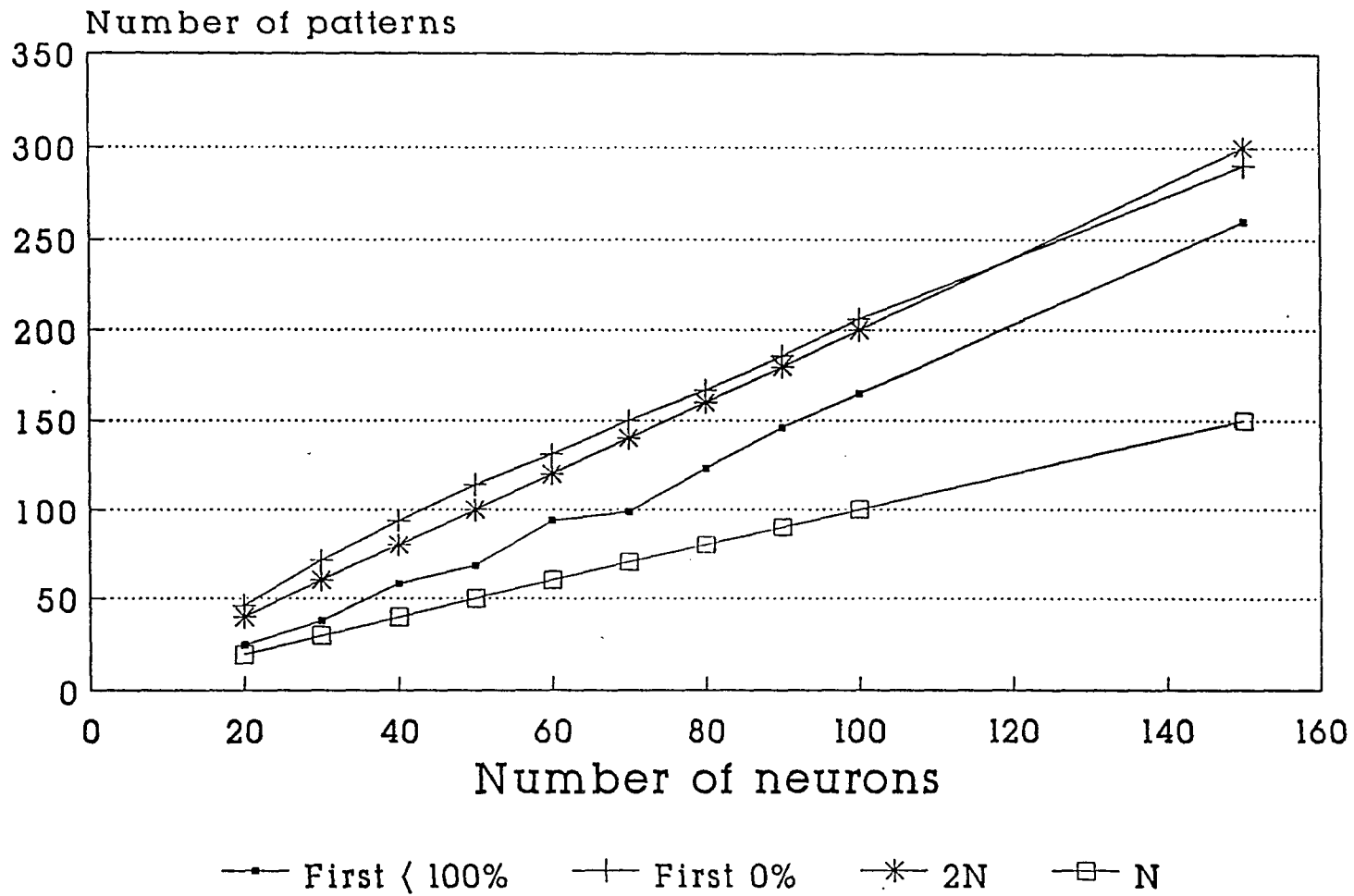


Figure 6. Delta rule, all neurons

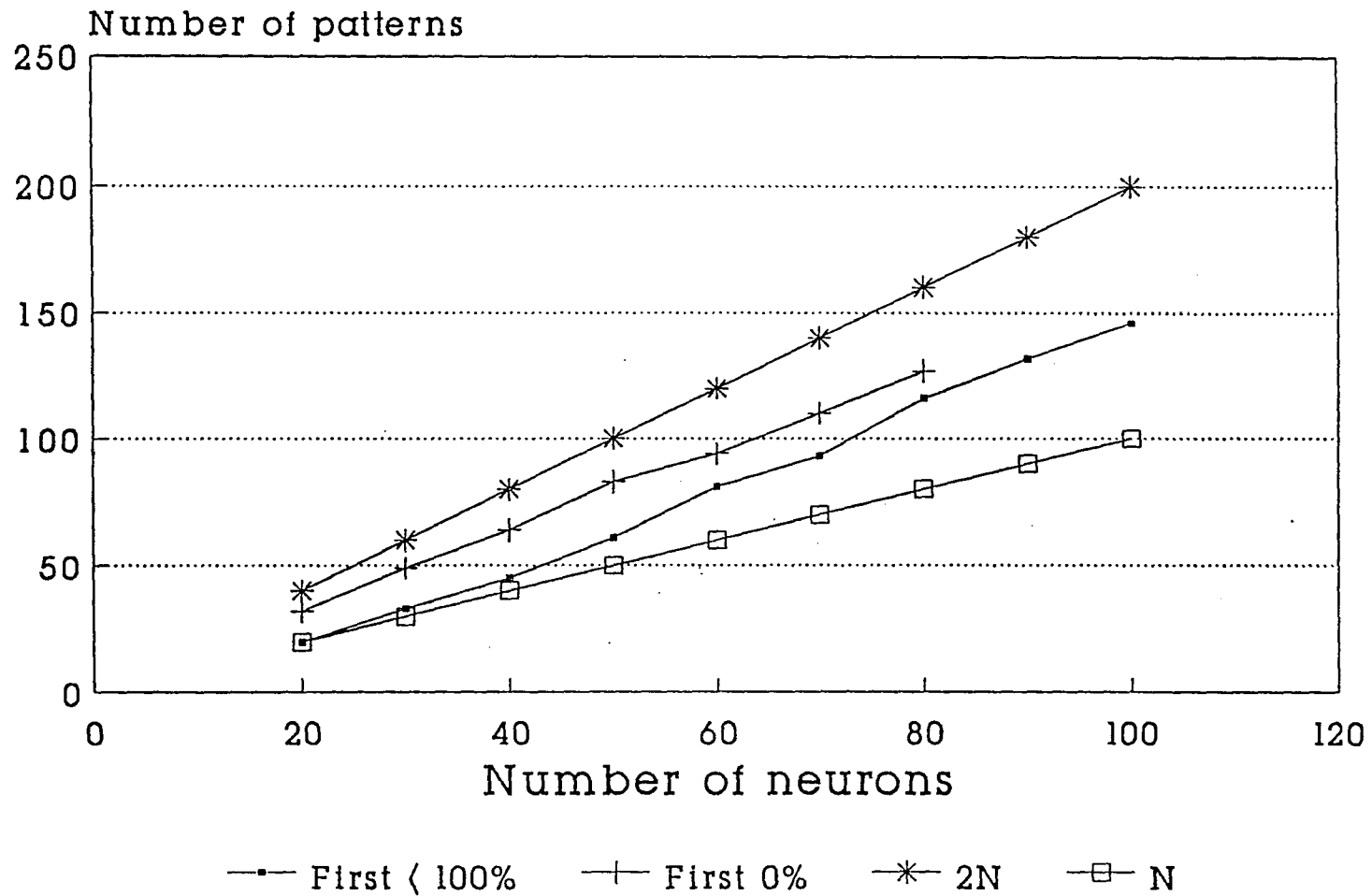




Figure 7 shows the relationship between the ability of one neuron to separate a set of patterns and the ability to store the set of patterns. One curve shows the minimum  $m$  for which 1 neuron was unsuccessfully at least once at separating the set of patterns, and the other curve shows the lowest value of  $m$  for which at least one of ten attempts to store the set of patterns failed. Although one neuron can successfully separate  $2N$  patterns of  $N$  bits each about 20% of the time, it must be able to separate the set of patterns successfully virtually 100% of the time in order for the probability of storing the set of patterns to be high. In Figure 7, this limit, the point at which one neuron can no longer separate the set of patterns virtually 100% of the time, is only slightly above the point at which the set of patterns can no longer be stored virtually 100% of the time. The drop-off from about 100% success to 100% failure at storing sets of patterns can more easily be seen in Figure 8. Notice that, for  $N = 20$ , the drop-off begins at about  $m = 20$ ; for  $N = 40$ , the drop-off begins at about  $m = 50$ ; and, for  $N = 70$ , the drop-off does not begin until about  $N = 100$ . The drop-off from about 100% success to 100% failure at separating sets of patterns can be seen in Figure 9. As  $N$  increases, the drop-off appears to approach  $2N$ . Tests are currently under way using much larger  $N$  to determine if this is indeed the case.

### 3.3 Hetero-Associative Neural Networks

To store a pattern using the Hopfield model, one adjusts the weights in such a way as to make that pattern a stable state. Since the pattern is associated with itself, this procedure is known as auto-associative learning. In contrast, the hetero-associative model

Figure 7. One neuron and all neurons

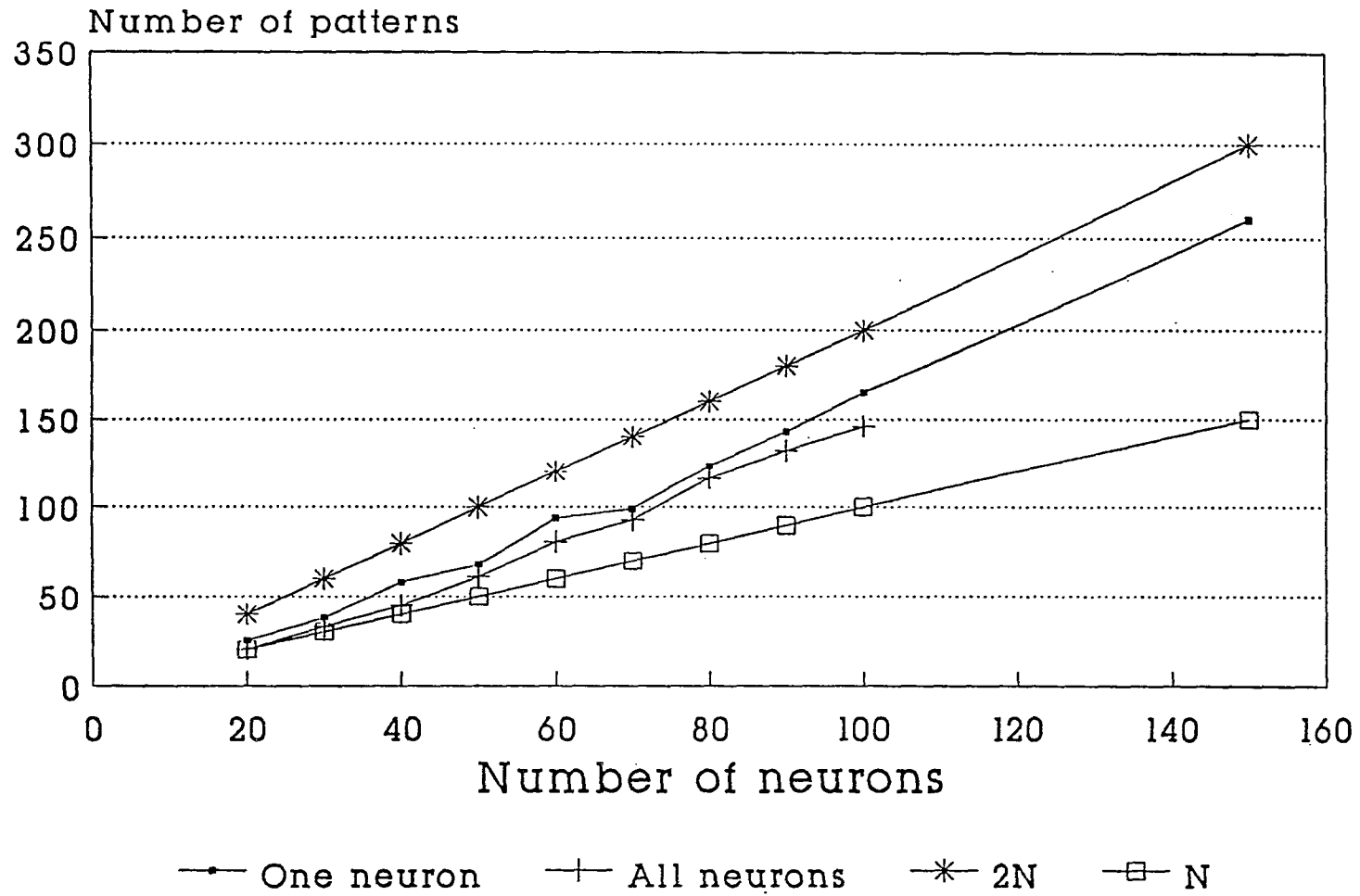


Figure 8. Success, 50 trials,  $N = 20-90$

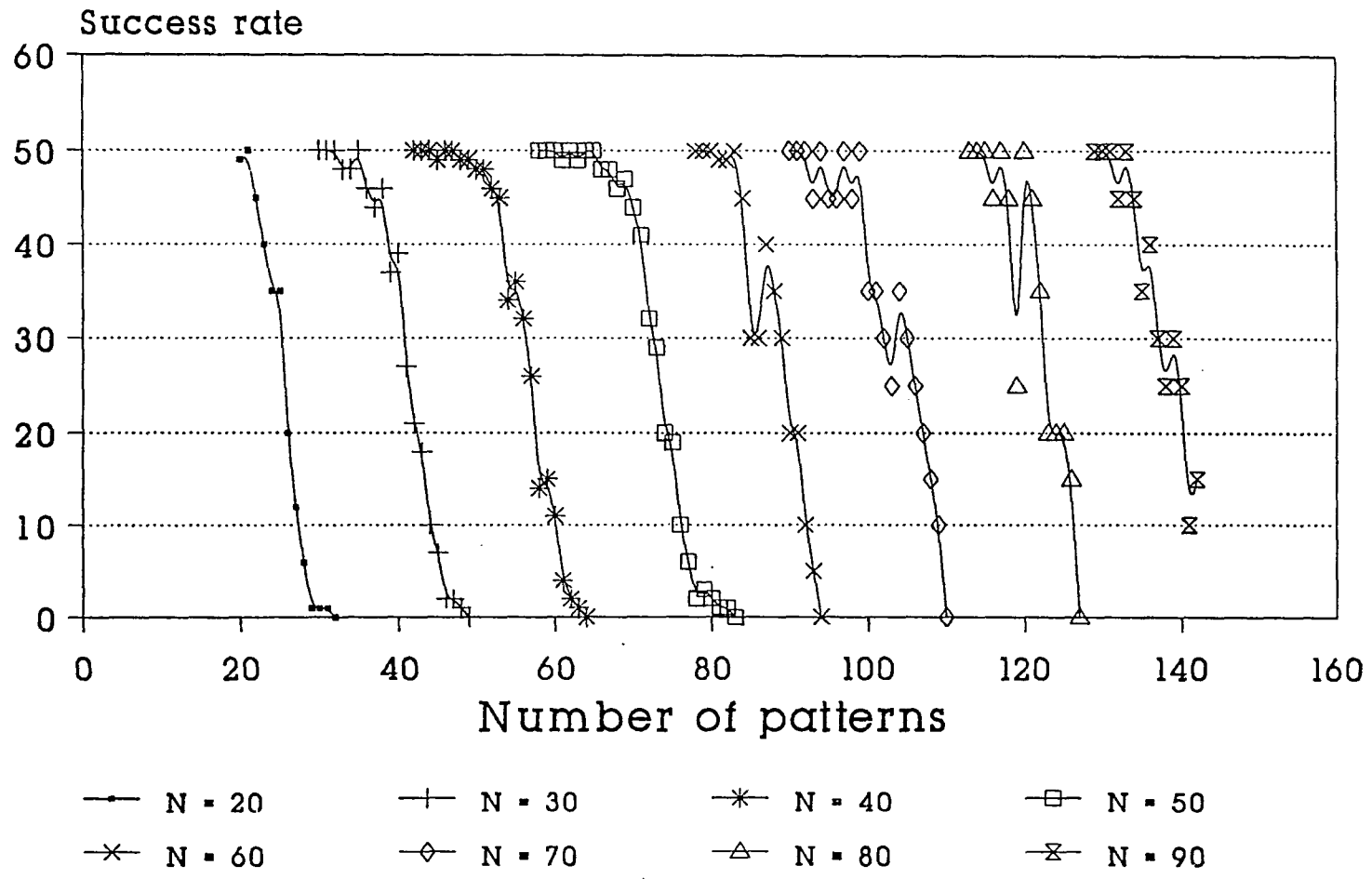
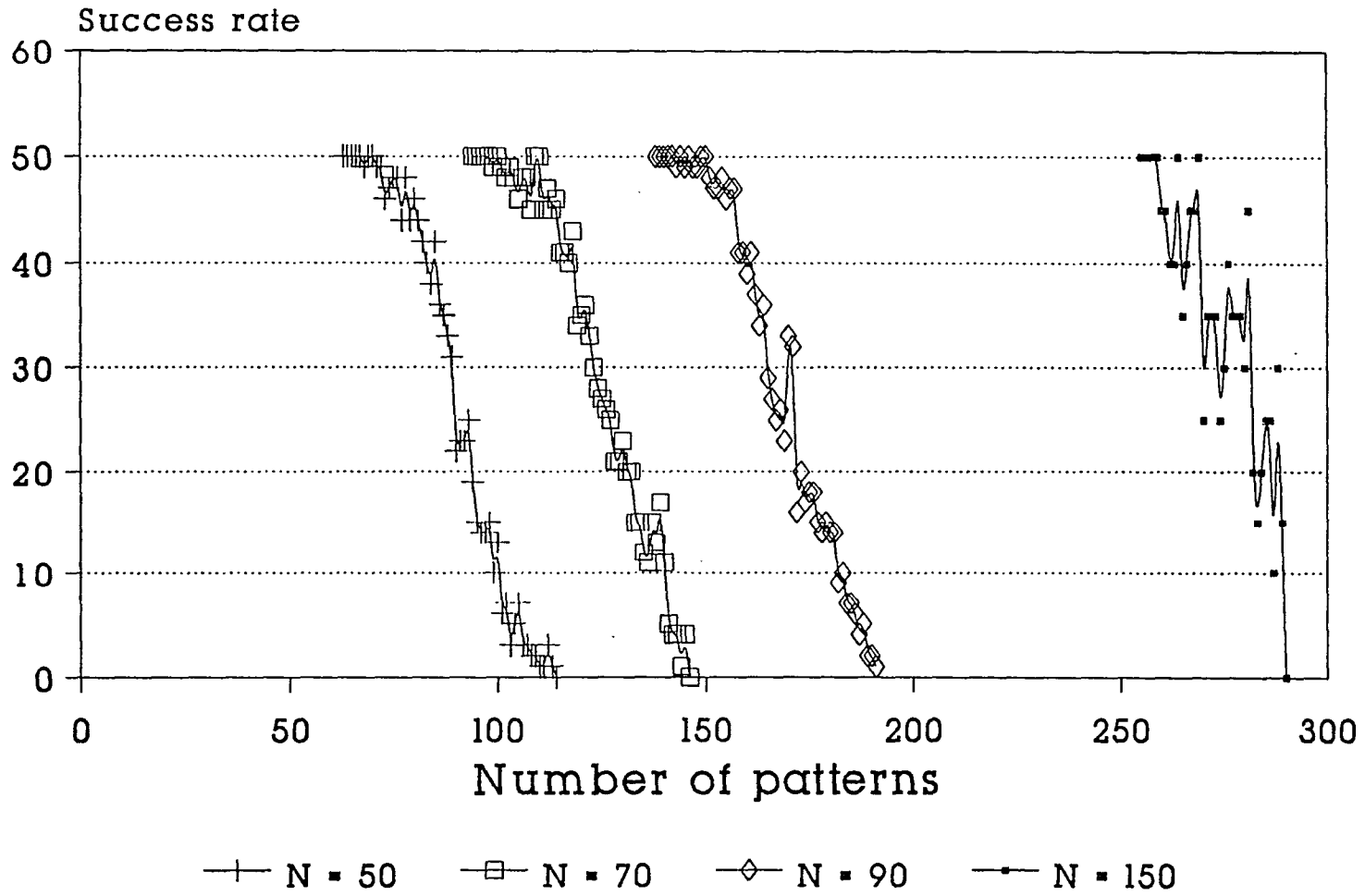


Figure 9. One neuron,  $N = 50-150$



learns to associate a given input pattern with a corresponding output pattern. In hetero-associative learning, the weights are adjusted in such a way that the given input pattern will always produce the corresponding output pattern when input to the net. The neuron update equations and learning algorithms for the hetero-associative model are generalizations of the update equations and learning algorithms of the auto-associative model.

$$x_i = \sum_j T_{ij} I_j \quad [40]$$

$$O_i = \begin{cases} 1 & x_i > 0 \\ -1 & x_i \leq 0 \end{cases} \quad [41]$$

where  $I_j$  is bit  $j$  of the input pattern and  $O_i$  is bit  $i$  of the output pattern. Notice that there is no requirement that the weight matrix  $T$  be a zero-diagonal matrix.

To store the  $s$ th input/output pair  $I^s \rightarrow O^s$ , first calculate the output of each neuron using Equations 40 and 41. Then compare this output,  $O$ , to the desired output  $O^s$ . For each bit of the actual output,  $O$ , that does not match the corresponding bit of the desired output,  $O^s$ , modify the weight matrix  $T$  as follows:

Case 1. The desired output,  $O_i^s = 1$  but the actual output,  $O_i = -1$ : Increment each term in Equation 40, thereby increasing  $x_i$ . For each  $j$ , if  $I_j^s = 1$ , increment  $T_{ij}$ . If, on the other hand,  $I_j^s = -1$ , decrement  $T_{ij}$ . If the amount of change is enough, Equations 40 and 41 will produce an output of  $O_i = 1$  the next time  $O_i$  is calculated with  $I^s$  as input.

Case 2. The desired output,  $O_i^s = -1$  but the actual output,  $O_i = 1$ : Decrement each term in Equation 40, thereby decreasing  $x_i$ . For each  $j$ , if  $I_j^s = 1$ , decrement  $T_{ij}$ . If, on the other hand,  $I_j^s = -1$ , increment  $T_{ij}$ .

These two cases can be combined using the algorithm: if  $O_i \neq O_i^s$ , then, for each  $j$ , increment  $T_{ij}$  by  $O_i^s I_j^s$ . This will always result in incrementing  $x_i$  if it is negative but should be positive, and it will always result in decrementing  $x_i$  if it is positive but should be negative. This hetero-associative delta rule can be written as

$$\Delta T_{ij} = c ( O_i^s - O_i ) I_j^s \quad [42]$$

If the diagonal elements of the weight matrix are not required to be zero, each neuron must separate  $m$  patterns of  $N$  bits each; whereas, each neuron of an auto-associative neural network must separate  $m$  patterns of  $N - 1$  bits. Thus, the probability of a hetero-associative neural network of  $N$  neurons storing  $m$  patterns of length  $N$  is approximately equal to the probability of an auto-associative neural network of  $N + 1$  neurons storing  $m$  patterns of length  $N + 1$ . As  $N$  approaches infinity, the capacity of the hetero-associative model is equal to the capacity of the auto-associative model.

### 3.4. Higher-Order Terms

The next state of the simple neuron described previously (see Equations 1 and 2) is found by calculating a linear combination of the other  $N - 1$  neurons and thresholding. Since each of the  $N - 1$  terms of Equation 1 depends on only one neuron, the simple neuron has order 1. A neuron of order 1 can implement only linear functions. Each neuron must separate the set of patterns to be learned into two groups--those with a 1

for that neuron and those with a -1 for that neuron--based on a linear combination of the values of the other  $N - 1$  neurons. This requirement of linear separability greatly reduces the sets of patterns that can be stored. The classic example of a nonlinear function is the exclusive-or function. If a two-input first-order neuron has an output of -1 for inputs (-1, -1), +1 for inputs (-1, +1), and +1 for inputs (+1, -1), then it will be +1 for inputs (+1, +1) because of its linear nature; thus, it cannot perform the exclusive-or function which would require an output of -1 for inputs (+1, +1). This neuron is not able to separate the 4 patterns into the proper two groups by calculating a linear combination of the other two neurons. By increasing the order to 2, the neuron is capable of performing the exclusive-or function.

The Hopfield model can be converted to a second-order neuron model by having a weight associated with each pair of neurons rather than, or in addition to, having a weight associated with each of the individual neurons. If one multiplies the outputs of two neurons, the product will be +1 for inputs (-1, -1) and (+1, +1) but will be -1 for inputs (-1, +1) and (+1, -1), thus allowing the neuron to perform the exclusive-or operation. Having a weight associated with each pair will require nearly  $N^3$  weights rather than the nearly  $N^2$  weights required by the first-order model. If symmetry is maintained in the weight matrices, the first order model requires  $\binom{N}{2}$  non-redundant weights, and the second order model requires  $\binom{N}{3}$ . Maintaining symmetry in the weight matrices requires that a modification be made to the learning algorithm; however, this modification does not significantly affect the capacity of the neural network.

The next state of a second-order neuron can be calculated using the equation

$$x_i = \sum_{j \neq i} \sum_{\substack{k \neq i \\ k \neq j}} T_{ijk} V_j V_k \quad [43]$$

in place of Equation 1, and using Equation 2 for thresholding.

One can derive a learning algorithm for this model in the same manner as before. To store pattern  $V^s$ , compare  $V_i^s$ , the desired output of each neuron, to  $V_i$ , the output calculated using Equations 43 and 2 with  $V^s$  as the input. For each bit that changes, modify the weight matrix  $T$  as follows:

Case 1. The desired output,  $V_i^s = 1$  but the actual output,  $V_i = -1$ : Increment each term in Equation 43, thereby increasing  $x_i$ . For each  $V_j^s$  and  $V_k^s$  such that  $j \neq i$  and  $k \neq i$ , if  $V_j^s V_k^s = 1$ , increment  $T_{ij}$ . If, on the other hand,  $V_j^s V_k^s = -1$ , decrement  $T_{ij}$ .

Case 2. The desired output,  $V_i^s = -1$  but the actual output,  $V_i = 1$ : Decrement each term in Equation 43, thereby decreasing  $x_i$ . For each  $j \neq i$  and  $k \neq i$ , if  $V_j^s V_k^s = 1$ , decrement  $T_{ij}$ . If, on the other hand,  $V_j^s V_k^s = -1$ , increment  $T_{ij}$ .

These two cases can be combined using the algorithm: if  $V_i \neq V_i^s$ , then, for each  $j \neq i$ , increment  $T_{ij}$  by  $V_i^s V_j^s V_k^s$ . This will always result in incrementing  $x_i$  if it is negative but should be positive, and it will always result in decrementing  $x_i$  if it is positive but should be negative. This delta rule can be written as

$$\Delta T_{ijk} = c (V_i^s - V_i) V_j^s V_k^s \quad [44]$$



Tests were run to determine experimentally the capacity of the second-order models. In the first set of tests, only the three-dimensional weight matrix was used (see Equation 43). In the second set of tests, both 2D and 3D weight matrices were used:

$$x_i = \sum_{j \neq i} T_{ij} V_j + \sum_{j \neq i} \sum_{\substack{k \neq i \\ k \neq j}} T_{ijk} V_j V_k \quad [45]$$

Each time that a pattern was randomly generated, it was rejected if it differed from one of the other patterns by less than 2 bits or by more than  $N - 2$  bits. Table 5 shows results for  $N = 10$  using only the 3D weight matrix. Notice that the probability of storing  $m$  randomly-generated patterns does not drop below 90% until  $m = 39$ . Thus  $C_{90\%}$  is about 3.8 for  $N = 10$ .

Number of Pattern	35	36	37	38	39	40	41	42	43	44	45	46	47
Trials Successful	50	49	48	46	42	33	25	29	17	15	10	4	0

Table 5. Successes in 50 trials, 3D binary model,  $N = 10$ .

The results using both 2D and 3D weight matrices with  $N = 10$  are shown in Table 6. In this case  $C_{90\%}$  is about 5.2. This compares to 0.9 for the model using the 2D weight matrix alone (Table 2).

Number of Patterns	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
Trials Successful	50	47	44	41	38	38	28	24	23	16	14	12	7	8	3	2	2	0

Table 6. Successes in 50 trials, binary model (2D and 3D),  $N = 10$ .

Note that symmetric matrices were used to obtain the results of Tables 5 and 6.

The delta rule was simply modified to maintain the symmetries  $T_{ijk} = T_{ikj} = T_{jik} = T_{jki} = T_{kij} = T_{kji}$ . Such symmetry reduces the number of weights to less than  $\frac{1}{6}$  the number required for the asymmetric case without significantly reducing the capacity. Our results are similar to those of Maxwell *et al.*<sup>10</sup> in that the increased capacity due to increasing the dimension of the weight matrices is proportional to the increase in the number of weights used.

### 3.5 Unlearning

Any state  $V^s$  can be made stable by using the delta rule to decrease each  $E_i^s$  that may be non-negative. Conversely, a stable state can be "forgotten" or "unlearned" by increasing the energy of the state appropriately.<sup>11</sup> To unlearn pattern  $V^s$ , simply apply the following equation to T:

$$\Delta T_{ij} = -kV_i^sV_j^s \quad [46]$$

where  $k$  is a positive constant. In this case

$$\Delta E_i^s = k(N-1). \quad [47]$$

If the new value of  $E_i^s > 0$ , the state will no longer be stable. The effect that the change in T has on stored pattern  $V^s$  can be calculated easily:

$$\Delta E_i^s = -k(1 - V_i^sV_i^s(V^s \cdot V^s)) \quad [48]$$

$$= -k + kV_i^sV_i^s(V^s \cdot V^s) \quad [49]$$

The expected value of this change is  $-k$ . It has been shown by Youn and Kak<sup>12</sup> that  $k = 0.1$  leads to unlearning of the spurious state without significantly affecting other stable points.

Notice that the ratio of the constant  $c$  in the delta rule (Equation 18) to the constant  $k$  in the unlearning equation determines the affect of forgetting a pattern on the other stored patterns. Choosing  $k = 0.1$ , as Youn and Kak did, requires the weights to be real numbers rather than integers. This significantly increases both the amount of memory required to store the weights and the speed of the neural network. All tests presented in this dissertation are from a neural network program that uses only integer arithmetic. If one requires  $k$  to be  $1/10$  of  $c$ , one must choose  $c$  to be at least 10 to keep the weights integers. This will significantly increase the amount of memory required to store the weights. Therefore, incorporating unlearning into a neural network simulation can lead to much greater memory requirements. Furthermore, using continuous unlearning significantly reduces the number of patterns one can store since the adjustments to the weight matrix necessary to unlearn a pattern can cause other patterns to also be forgotten.

### 3.6 Negative Feedback in Learning

The purpose of unlearning is to reduce the number of spurious states. When a pattern is unlearned, however, states which were previously unstable can become stable. An alternative to unlearning is to use direct feedback of neurons during the learning procedure. During learning, set each  $T_{ij} = -e$  where  $e$  is a positive constant. This will

cause each  $E_i^s$  to be increased by  $e$ . A positive  $E_i^s$  means that, during delta learning, the weights of row  $i$  will be adjusted to decrease  $E_i^s$ . If one uses a positive  $e$  during learning and resets  $e$  to 0 after learning, all complements of the stored patterns will be stable states since no  $E_i^s$  will be 0. The maximum value of  $E_i^s$  will be  $-e$ . Using larger  $e$  will, therefore, tend to improve the error-correcting ability of the net. Whereas unlearning serves to increase the energy of the spurious states, learning with negative feedback serves to reduce the energy of the non-spurious stored states. Reducing the energy of the non-spurious stored states, in turn, tends to also increase the energy of the spurious states.

Learning with negative feedback tends

1. to decrease the chance of learned patterns from being forgotten due to subsequent changes in  $T$ ,
2. to increase the average size of the attraction basins of the learned patterns,
3. to reduce the size of the attraction basins of the non-complement spurious states,
4. and to improve the performance of the neural network by improving the error-correcting ability of the neural network.

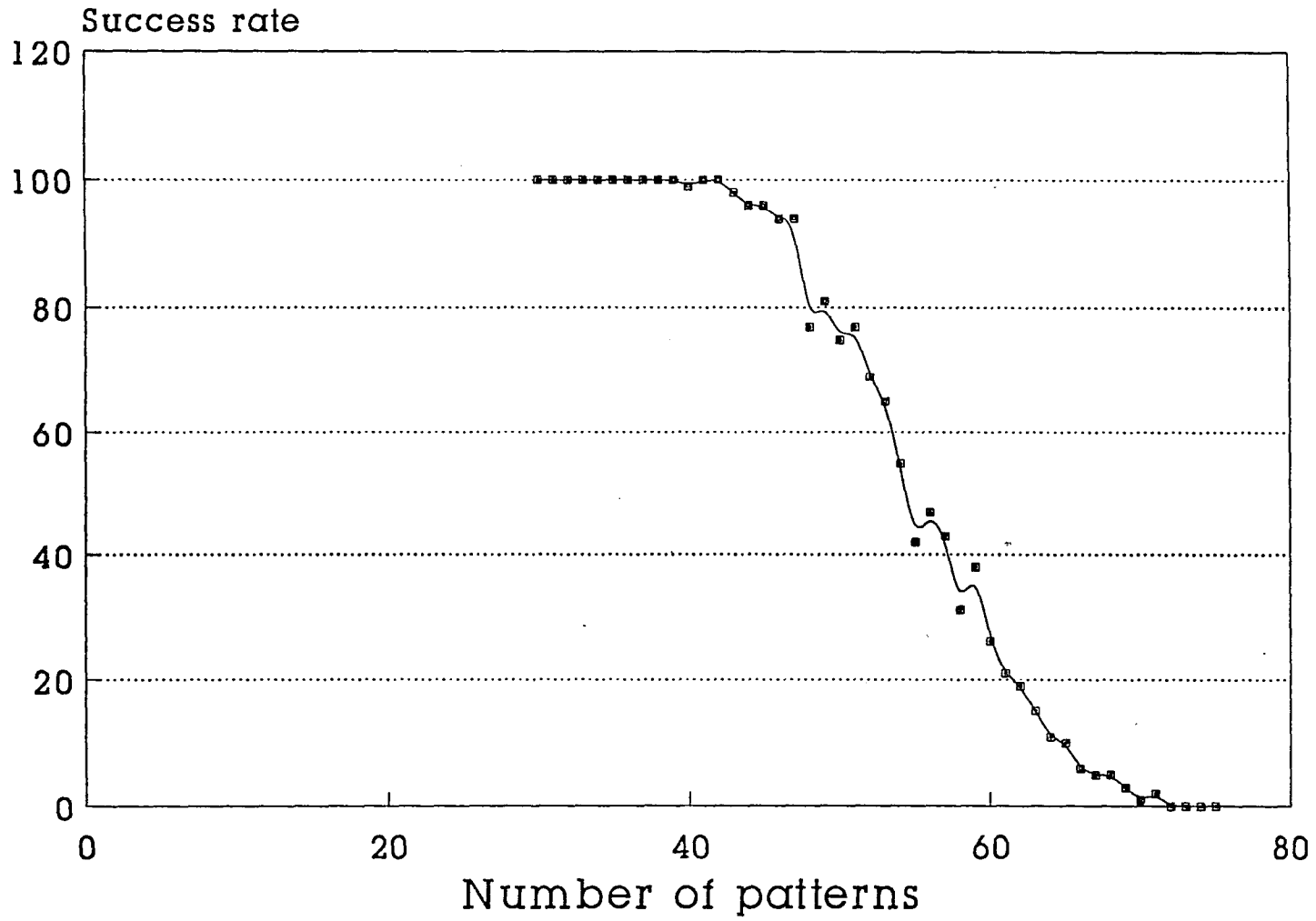
Furthermore, it does all of the above without significantly reducing the capacity of the model or substantially increasing the amount of memory needed to store the weights.

If one uses the delta rule to store a set of patterns, for some of the patterns one or more  $E_i^s$  may be 0 or close to 0. For example, if bit  $V_i^s = -1$  and  $E_i^s = 0$ , neuron  $i$  will not change when pattern  $V^s$  is applied to the net, and it will not be necessary to modify the weights of row  $i$ . Unfortunately, however, the error-correcting ability will be very poor. If a pattern that differs from  $V^s$  in only one bit is applied to the net, the net will have only about a 50% chance of correcting the error. Also, the complement of  $V^s$  will not be a stable state. If one uses  $e = 1$  during learning and resets  $e$  to 0 after learning, the maximum value of  $E_i^s$  will be  $-e$ . Using larger  $e$  will, therefore, tend to improve the error-correcting ability of the net.

Figure 10 shows the results of a study using  $e = 1$ . Notice that the probability of one bit separating a set of  $m$  patterns of  $N$  bits each is not significantly worse than when direct feedback was not used (see Figure 3).

Tables 7 and 8 show results of tests comparing the performance of the learning algorithm using various values of  $e$ . With  $e = 0$ , the number of stable complement states was less than the number of intentionally stored states. For  $e = 5$  and  $e = 10$ , all complement states are stable. Although the number of non-complement spurious states increases as  $e$  increases, the average size of the attraction basins of the non-complement spurious states is significantly reduced. Also, the percentage of states which settled to the stable state closest in Hamming distance increased significantly. Further studies are currently underway to compare more closely this method of using negative feedback in learning to the method of continuous unlearning.

Figure 10. Delta rule,  $N = 30$ ,  $e = 1$



e	0	5	10
Stored	100	100	100
Spurious	57	92	100
Complement	58	100	100
Stored Basins	398	346	363
Spurious Basins	355	118	99
Complement Basins	375	363	356
Correct	33054	37528	38466

Table 7. Performance for  $e = 0, 5,$  and  $10,$  using  $N = 10$  and  $m = 5.$

e	0	5	10
Stored	120	120	120
Spurious	69	71	85
Complement	71	120	120
Stored Basins	340	299	300
Spurious Basins	297	147	113
Complement Basins	288	295	297
Correct	28025	33642	35192

Table 8. Performance for  $e = 0, 5,$  and  $10,$  using  $N = 10$  and  $m = 6.$

### 3.7 Networks of Limited Connectivity

Because the number of weights in a neural network varies with the square of the number of neurons, a huge amount of memory is required if  $N$  is large. This can be a particularly serious problem if one is storing images. Images of  $256 \times 256$  pixels or even  $512 \times 512$  pixels are not uncommon. Even if the image is  $64 \times 64$  pixels, over 16 million weights are required if the weight matrix is asymmetric.

The number of weights can be greatly reduced by limiting the connectivity of the

network. For instance, each neuron can be connected only to those neurons within a defined neighborhood. One would expect that, if each neuron is connected to only  $d$  of the  $N$  neurons in the network, the capacity would be approximately equal to that of a network of  $d$  neurons.

Simulations indicate that this is, indeed, the case. Table 9 shows results from tests in which the total number of neurons  $N$  was varied from 30 to 90, but each neuron was connected to only 30 other neurons. The capacity is essentially the same in all three cases.

N	Number of Patterns															
	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
30	50	49	48	46	42	33	25	29	17	15	10	4	0	0	0	0
60	50	50	50	47	48	44	33	30	18	11	3	2	0	0	0	0
90	50	50	49	47	46	46	42	34	20	15	12	5	0	0	0	0

Table 9. Successes in 50 trials,  $d = 30$ .

In general, the capacity of a neural network in which each neuron is connected to only  $d$  of the total number of neurons seems to be approximately equal to the capacity of a neural network in which the total number of neurons is  $d$ . Figures 11 and 12 provide further evidence of this. In Figure 11  $d$  is set to  $\frac{N}{2}$ , and  $N$  varies from 30 to 70. In Figure 12  $d$  is set to  $\frac{2N}{3}$  and  $N$  varies from 30 to 60.

These results should be of particular interest to those using hardware implementations of neural networks. Such hardware implementations rarely have more than about one thousand neurons.<sup>13, 14, 15</sup> One could store large images in such networks only by



Figure 11. Delta rule,  $d = N/2$

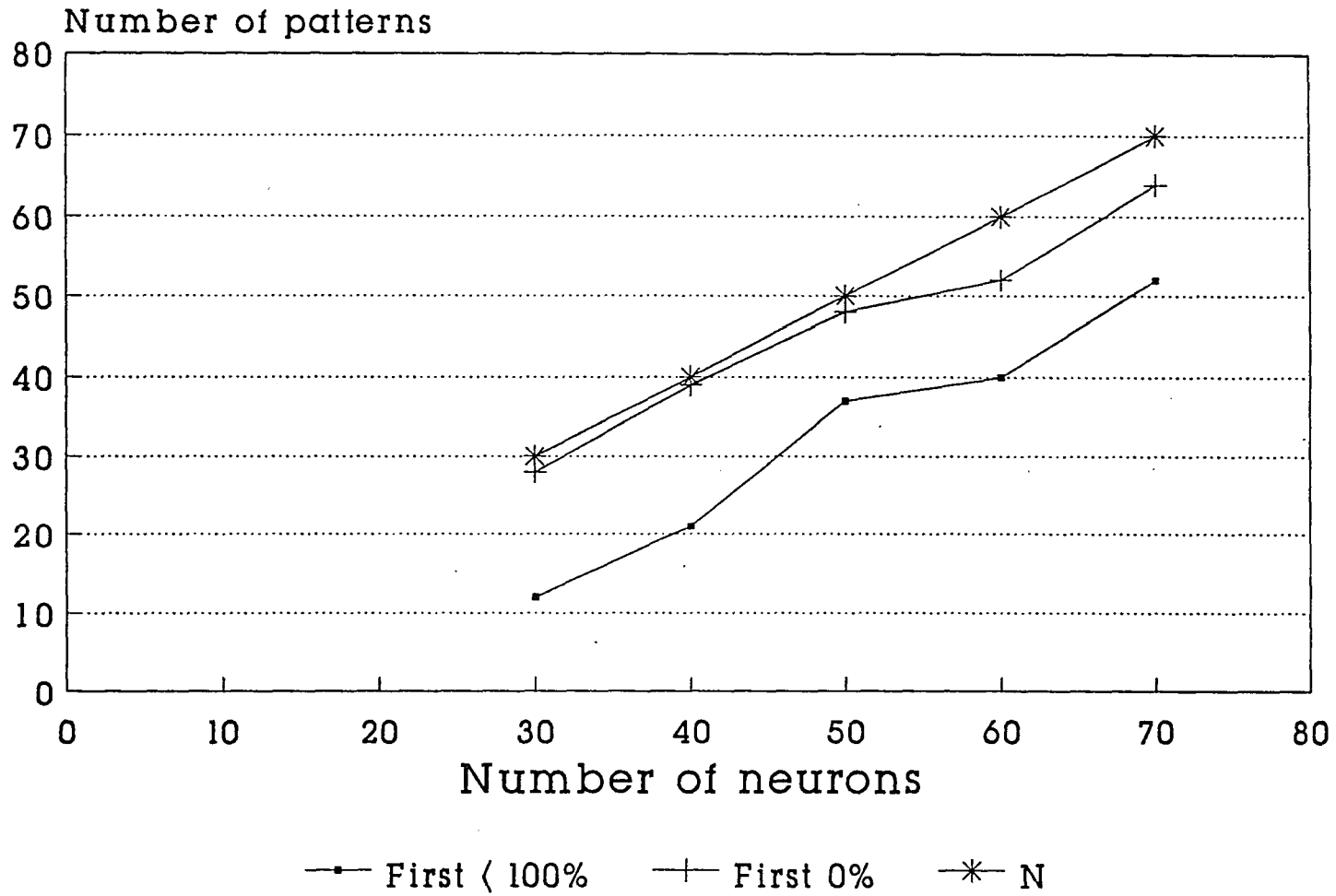
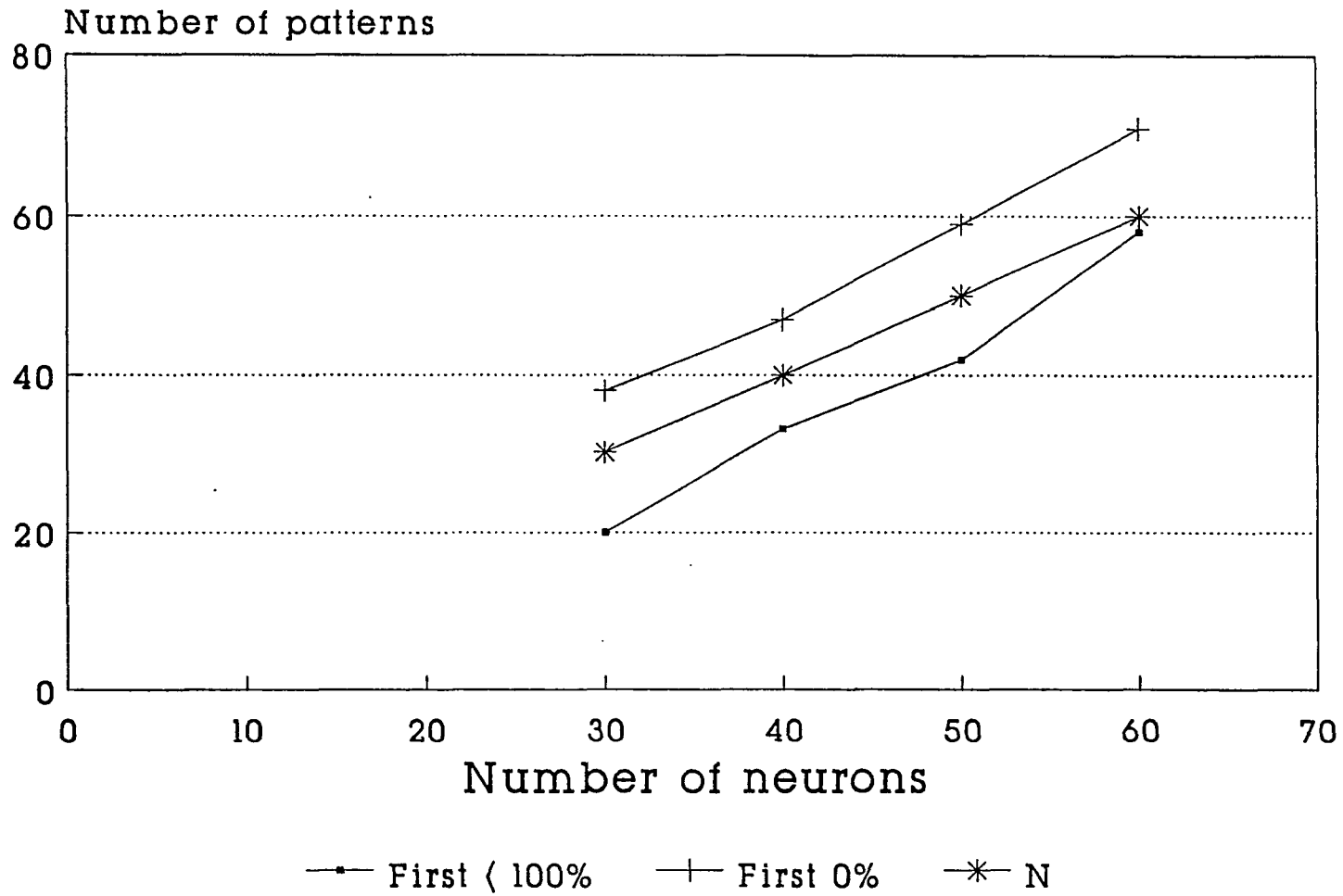


Figure 12. Delta rule,  $d = 2N/3$



limiting the connectivity of the model.

## CHAPTER 4. Non-Binary Neural Networks

The Hopfield model requires a pattern or image to be binary. Often, however, patterns and images are not binary and can lose much information when converted to a binary form. In addition, the manner in which one converts to binary significantly affects the amount of information lost. A generalized neural network model has been proposed that allows neurons to take on more than two values.<sup>16</sup> Very few changes to the simple binary model are required to obtain an n-ary neural network model that can successfully store and retrieve patterns

### 4.1. Update Equations for the Non-Binary Model

To convert a binary neural network to the more general n-ary neural network, Equation 1 need not be changed; only Equation 2 needs to be changed. For instance, for a quaternary neural network, Equation 2 can be modified to

$$V_i = \begin{cases} 3 & x_i > t \\ 1 & t \geq x_i > 0 \\ -1 & 0 \geq x_i > -t \\ -3 & -t \geq x_i \end{cases} \quad [50]$$

where  $t$ ,  $0$ , and  $-t$  are thresholds.

### 4.2. Delta Rule for the Non-Binary Model

One can derive a learning algorithm for this model using the same method we used to derive the binary delta rule. To store pattern  $V^s$ , first calculate the next state of each neuron using Equations 1 and 50 with  $V^s$  as the input, and then compare this to the

desired state of the neuron,  $V_i^s$ . If these two values are equal, no change in weight matrix  $T$  is necessary. For each neuron that changes, modify the weight matrix  $T$  as follows:

Case 1. The desired output,  $V_i^s$ , is greater than the actual output,  $V_i$ : Increment each term in Equation 1, thereby increasing  $x_i$ . For each  $j \neq i$ , if  $V_j^s$  is positive, increment  $T_{ij}$ . If, on the other hand,  $V_j^s$  is negative, decrement  $T_{ij}$ .

Case 2. The desired output,  $V_i^s$ , is less than the actual output,  $V_i$ : Decrement each term in Equation 1, thereby decreasing  $x_i$ . For each  $j \neq i$ , if  $V_j^s$  is positive, decrement  $T_{ij}$ . If, on the other hand,  $V_j^s$  is negative, increment  $T_{ij}$ .

In general we can say: if  $V_i \neq V_i^s$ , then, for each  $j \neq i$ , increment  $T_{ij}$  by an amount proportional to  $V_i^s V_j^s$ . This will always result in incrementing  $x_i$  if it is negative but should be positive, and it will always result in decrementing  $x_i$  if it is positive but should be negative. This is, of course, just a more general version of the binary delta rule--the very same equation is used (Equation 18) but the neurons are non-binary.

The only difference in the algorithm is that Equation 50 is used instead of Equation 2 for thresholding. For the algorithm to be successful, the relationship between the constant  $c$  in Equation 18 and the threshold  $t$  in Equation 50 must be chosen properly. We will refer to this ratio as the *convergence ratio* since, if it is too small, the algorithm will not converge. With a large enough convergence ratio, any quaternary pattern can quickly be stored using Equations 1, 18, and 50. Since it is the ratio of  $t$  to

c that is important, we can set c to 1 and choose an appropriate threshold t.

For the binary model, we required only that the change in  $x_i$  due to the changes in the weights (Equation 18) be large enough to change the sign of  $V_i$ . With non-binary networks of the type described, not only does the change in  $x_i$  have a minimum, but it also has a maximum. Normally, we would not want  $x_i$  to change by more than t, since such a change could overshoot the desired value. Suppose, for example, that the neurons can take on the values {3, 1, -1, -3}, as in Equation 50. If the desired value  $V_i^s = 1$  and the actual value  $V_i = -1$ , too large a change in the weights will result in too large a change in  $x_i$  causing an output of  $V_i = 3$  the next time  $V^s$  is applied to the network.

The change in  $x_i$  caused by application of Equation 18 can be calculated by combining Equations 18 and 1:

$$\Delta x_i = \sum_{j \neq i} c (V_i^s - V_i) V_j^s V_j^s \quad [51]$$

$$\Delta x_i = c (V_i^s - V_i) \sum_{j \neq i} (V_j^s)^2 \quad [52]$$

Using the {3, 1, -1, -3} model, the greatest change in  $x_i$  will occur if the pattern to be stored,  $V^s$ , consists of only the values +3 and -3. In this case Equation 52 can be written as

$$\Delta x_i \leq c (V_i^s - V_i) (N - 1) 3^2 \quad [53]$$

and the amount of change in  $x_i$  will depend on the difference between the desired

output,  $V_i^s$ , and the actual output,  $V_i$ . A reasonable choice of  $t$  (assuming  $c = 1$ ) is  $18(N - 1)$ . With such a choice, if  $V_i^s$  and  $V_i$  differ by 2, the change in  $x_i$  will be at most  $18(N - 1)$  and will never be so much that more than one threshold is crossed at a time. For the  $\{3, 1, -1, -3\}$  quaternary model, repeated application of Equation 18 with  $t/c = 18(N - 1)$  is guaranteed to quickly store any pattern.

For an  $n$ -ary neural network, it is easy to calculate a ratio  $t/c$  that is guaranteed to allow any pattern to be stored. Let  $V_{\max}$  be the maximum allowable magnitude for a neuron ( $V_{\max} = 3$  for the quaternary example given above). Let  $V_{\text{diff}}$  be the maximum difference between any two "adjacent" output values ( $V_{\text{diff}} = 2$  for the  $\{3, 1, -1, -3\}$  model). The following convergence ratio will allow any one pattern to be stored:

$$t/c = V_{\text{diff}} (N - 1) V_{\max}^2 \quad [54]$$

This is because

$$\Delta x_i \leq c V_{\text{diff}} (N - 1) V_{\max}^2 \quad [55]$$

For the quaternary model described above, this formula gives the minimum value of  $t/c$  that will guarantee that any pattern can be stored. For other  $n$ -ary models, there are values of  $t/c$  below the one calculated from this formula that will also allow any pattern to be stored, as will be seen shortly.

### 4.3. Capacity of the Non-Binary Model

Several tests were run to determine experimentally the optimal choice of the conver-

gence ratio and to compare the capacity of the quaternary model to that of the binary. Sets of patterns to be learned were generated randomly; however, each time a pattern was generated, it was compared to each of the patterns already in the set to be sure that it differed in more than one place and less than  $N - 1$  places from each of them. Tests were run on the following quaternary models  $\{3, 1, -1, -3\}$ ,  $\{2, 1, -1, -2\}$ , and  $\{4, 1, -1, -4\}$ . For each model,  $c$  was set to 1, and the value of  $t$  was varied over a large range. For each value of  $t$ , 50 attempts were made to store 1 random pattern, 50 attempts were made to store 2 random patterns, and so on up to 15 random patterns.

Table 10 shows results for  $N = 10$  using the values of  $t$  suggested by Equation 54 for each of the three quaternary models.

model	Number of Patterns									
	5	6	7	8	9	10	11	12	13	
$\{2,1,-1,-2\}$	50	50	50	49	40	19	9	0	0	
$\{3,1,-1,-3\}$	50	50	50	47	39	12	9	1	0	
$\{4,1,-1,-4\}$	50	50	49	50	36	17	3	0	0	

Table 10. Successes in 50 trials, 3 quaternary models,  $N = 10$ .

Note that these results support the claim that, with such a choice of  $t$ , any one random pattern can be stored successfully. For the  $\{3, 1, -1, -3\}$  model, whenever  $t$  was less than that suggested by Equation 54, there were individual patterns that could not be stored. For the  $\{2, 1, -1, -2\}$  and  $\{4, 1, -1, -4\}$  models, however, values of  $t$  less than



those suggested by Equation 54 produced slightly better results. For these two models, the best results were obtained with

$$t = (N - 1) V_{\max}^2. \quad [56]$$

These were also the minimum values of  $t$  with which one can be sure to be able to store any one random pattern successfully.

Table 11 shows results using quaternary model  $\{2, 1, -1, -2\}$  with  $N = 30$ .

Number of Patterns	26	27	28	29	30	31	32	33	34	35
Trials Successful	50	49	50	45	39	26	14	2	1	0

Table 11. Successes in 50 trials, quaternary model  $\{2, 1, -1, -2\}$ ,  $N = 30$ .

Comparing these results with those of the binary model, Table 3, one can see that, for the quaternary model, the success rate drops below 90% at about  $N = 29$ ; whereas, for the binary model, it drops below 90% at about  $m = 37$ . The information capacity of the quaternary model, however, is actually higher, since each pattern contains about twice as much information.

## CHAPTER 5. Shift-Invariant Neural Networks

It has long been known<sup>17</sup> that neural nets possess some of the properties of associative memory. Given a neural net trained to recognize certain patterns, as an associative memory it must be able to retrieve a pattern given only a part of it. Simple neural nets perform this type of memory recall when part of the pattern is simply missing. However, for a neural net to be of practical value as an associative memory, it must be able to recognize parts of patterns that are shifted spatially with respect to the ones that have been stored. This includes not only two-dimensional images that are shifted vertically or horizontally and patterns that have missing parts, but also patterns that have the remaining parts bunched together (for instance, a misspelled word with missing letters), or patterns with extra parts inserted within the pattern (such as extra letters in a word).

For the case of linearly shifted patterns, one can implement shift invariance with a pre-processor that performs a shift-invariant transformation such as a Fourier transformation. The associative memory is then used for storing the transformation coefficients. The recent speech recognition system of Kohonen<sup>18</sup> for example, uses an FFT pre-processor, as does the Adaptive Pattern Recognition (ART) system of Carpenter and Grossberg.<sup>19</sup> Alternatively, one may use a shift invariant neural network that accepts patterns directly rather than their transforms. With such a network, it may be easier to perform direct non-linear operations that allow recognition of patterns with missing or extra parts. Shift-invariant associative memory using neural networks has been described earlier by Maxwell *et al.*,<sup>10</sup> Widrow and Winter,<sup>20</sup> and Prados and

Kak.<sup>21</sup>

In 1947, McCulloch and Pitts<sup>22</sup> discussed the idea of recognizing an object, or *apparition*, that is a member of a group of equivalent objects. The group of objects share a common *figure*, and there exists a group of transformations that "take the equivalents into one another but preserve the figure invariant". They give the example of a square that can be recognized regardless of translations from one place to other places.

They derive general methods for designing neural networks which recognize figures (members of the group) in such a way as to produce the same output for every input belonging to the figure. The following method is presented.

A manifold  $M$  is described by a set of coordinates  $(x_1, x_2, \dots, x_n)$  constituting the point-vector  $x$ . Denote the distributions of excitation received in  $M$  by the function  $\phi(x, t)$  having the value unity if there is a neuron at the point  $x$  that has fired within one synaptic delay prior to time  $t$ , and otherwise zero. Let  $G$  be the group of transformations which carry the functions  $\phi(x, t)$  describing apparitions into their equivalents of the same figure. In the simplest case, the only case discussed in detail by McCulloch and Pitts, the transformation  $T$  of  $G$  can be generated by linear transformations  $t$  of the underlying manifold  $M$ , so that  $T \phi(x) = \phi[t(x)]$ . For example, if  $G$  is the group of translations, then  $T \phi(x) = \phi(x + aT)$ , where  $aT$  is a constant vector depending only upon  $T$ . If  $G$  is the group of dilations,  $T \phi(x) = \phi(\alpha T x)$ , where  $\alpha T$  is a positive real number depending only upon  $T$ .

The simplest way to construct invariants of a given distribution  $\phi(x, t)$  of excitation, they claim, is to average over the group  $G$ . Let  $f$  be an arbitrary function that assigns a unique numerical value, in any way, to every distribution  $\phi(x, t)$  of excitation in  $M$  over time. Form every transformation  $T\phi$  of  $\phi(x, t)$ , evaluate  $f[T\phi]$ , and average the result over  $G$  to derive

$$a = 1/N \sum_{\text{all } T \in G} f[T\phi]. \quad [57]$$

Now, let the original manifold  $M$  be duplicated on  $N - 1$  sheets, a manifold  $MT$  for each  $T$  of  $G$ , and connected to  $M$  or its sensory afferents in such a way that whatever produces the distribution  $\phi(x)$  on  $M$  produces the transformed distribution  $T\phi(x)$  on  $MT$ . Therefore, separately for each value of  $\zeta$  for each  $MT$ , the value  $f[T\phi\zeta]$  is computed by a similar net, and the results from all the  $MT$ 's are added by convergence on the neuron at the point  $\zeta$  of the mosaic. The output of this neuron will be invariant to any transformation  $T$  of  $G$ . This technique is very similar to the technique recently discussed by Widrow and Winter.<sup>20</sup>

There appears to be a much simpler method of obtaining shift-invariant pattern recognition. Pitts and McCulloch hint at such a method; however, they discuss it from the biological point of view, and their mathematics is quite complicated. They suggest that the brain may compute the "center of gravity of the distribution of brightness" and give the following example: "If the square should appear anywhere in the field, the eyes turn until it is centered, and what they see is the same, wherever the initial position of the square." Using this idea of focusing the center of attention, one can obtain a very powerful, but simple, shift-invariant neural network. Such a neural network will

be presented in the next section.

### 5.1. Update Equations for the Shift-Invariant Model

The conventional neural network model needs to be modified only slightly to provide such a shift-invariant neural network model. Instead of having a different set of weights for each neuron, a neural net can use the same set of weights for each neuron. This focusing of the "center of attention" of the neural network requires that the connection matrix depend only on relative coordinates. This technique of implementing shift invariance has been used by Maxwell *et al.*<sup>10</sup>

Suppose one wishes to store the following two patterns in a conventional neural network memory:  $V^1 = (-1 -1 -1 +1 +1 +1)$  and  $V^2 = (-1 +1 -1 +1 -1 +1)$ . Associated with each of the 6 neurons is a set of weights corresponding to the effect of each of the other neurons on it. For example, neuron 3 should have a positive weight reflecting an excitatory effect of neuron 1 on it and negative weights reflecting inhibitory effects of neurons 4 and 6 on it. The Hopfield model using the Hebbian storage algorithm would assign weights of  $T_{20} = +2$  and  $T_{23} = T_{25} = -2$ . A shift invariant neural network, on the other hand, should assign weights in accordance with the relative distances between neurons. The weight  $T_2$ , for instance, can be calculated as the average of  $T_{20}$ ,  $T_{31}$ ,  $T_{42}$ , and  $T_{53}$ .

To obtain an update rule, Equation 1 can be modified to obtain

$$x_i = \sum_{j \neq i} T_j V_{i-j} \quad [58]$$

where  $T_j$  represents the connection strength between neuron  $i$  and neuron  $i - j$ . Since there are only  $N - 1$  weights, the capacity is extremely low. By using higher-order terms, the capacity can be increased substantially. Weight  $T_{jk}$  can represent the affect of  $V_{i-j}V_{i-k}$  on neuron  $V_i$ ; and weight  $T_{jkl}$  can represent the affect of  $V_{i-j}V_{i-k}V_{i-l}$  on  $V_i$ :

$$x_i = \sum_{j \neq i} T_j V_{i-j} + \sum_{j \neq i} \sum_{\substack{k \neq i \\ k \neq j}} T_{jk} V_{i-j} V_{i-k} + \sum_{j \neq i} \sum_{\substack{k \neq i \\ k \neq j \\ l \neq k}} \sum_{\substack{l \neq i \\ l \neq j \\ l \neq k}} T_{jkl} V_{i-j} V_{i-k} V_{i-l} \quad [59]$$

## 5.2. Delta Rule for the Shift-Invariant Model

We can apply our method for obtaining a delta rule to this model as follows. To store pattern  $V^s$ , we must input  $V^s$  to the neural network. If each bit of the output  $V$  (next state of  $V^s$ ) is equal to the corresponding bit of  $V^s$ , no changes are necessary. For each bit that changes, modify  $T$  as follows:

Case 1.  $V_i^s = 1$  but  $V_i = -1$ : Increment each term in Equation 59. If  $V_{i-j}^s = 1$ , increment  $T_j$ ; otherwise, decrement  $T_j$ . If  $V_{i-j}^s V_{i-k}^s = 1$ , increment  $T_{jk}$ ; otherwise, decrement  $T_{jk}$ .

And, if  $V_{i-j}^s V_{i-k}^s V_{i-l}^s = 1$ , increment  $T_{jkl}$ ; otherwise, decrement  $T_{jkl}$ .

Case 2.  $V_i^s = -1$  but  $V_i = 1$ : Decrement each term in Equation 59. If  $V_{i-j}^s = 1$ , decrement  $T_j$ ; otherwise, increment  $T_j$ . If  $V_{i-j}^s V_{i-k}^s = 1$ , decrement  $T_{jk}$ ; otherwise, increment  $T_{jk}$ . And, if  $V_{i-j}^s V_{i-k}^s V_{i-l}^s = 1$ , decrement  $T_{jkl}$ ; otherwise, increment  $T_{jkl}$ .

In general, this algorithm can be written as:

$$\Delta T_j = c ( V_i^s - V_i ) V_{i-j}^s \quad [60]$$

$$\Delta T_{jk} = c ( V_i^s - V_i ) V_{i-j}^s V_{i-k}^s \quad [61]$$

$$\Delta T_{jkl} = c ( V_i^s - V_i ) V_{i-j}^s V_{i-k}^s V_{i-l}^s \quad [62]$$

where  $c$  is the learning constant. If the subtracting of indices is performed modulo  $N$ , all cyclic shifts of a pattern will be stable if that pattern is stable.

### 5.3. Capacity of Shift-Invariant Model

Tests were first run using the second-order terms alone, the third-order terms alone, and the second- and third-order terms combined. Table 12 is a comparison of the second-order model and the third-order model. Surprisingly, the second-order model had higher capacity. When the two were combined, the results were far superior to the results of either alone. Table 13 shows results combining first- and second-order models. Notice that the success rates are only slightly greater than using the second-order model alone. Table 14 shows that, using both second-order and third-order terms, one can store as many as 16 patterns with a 90% success rate. Keep in mind that, if all 16 patterns are successfully stored, all cyclic shifts of these 16 patterns will also be stored. This leads to 160 stable states!

order	Number of Patterns											
	1	2	3	4	5	6	7	8	9	10	11	12
2D	48	49	43	39	34	26	8	1	0	0	0	0
3D	49	40	33	19	19	7	3	8	3	1	3	0

Table 12. Successes in 50 trials, 2D and 3D shift-invariant models,  $N = 10$ .

Number of Pattern	1	2	3	4	5	6	7	8	9	10	11
Trials Successful	48	48	46	39	43	36	14	6	1	1	0

Table 13. Successes in 50 trials, shift-invariant model (1D and 2D),  $N = 10$ .

Number of Pattern	9	10	11	12	13	14	15	16	17	18	19	20	21
Trials Successful	50	48	47	49	49	49	49	46	36	18	11	5	0

Table 14. Successes in 50 trials, shift-invariant model (2D and 3D),  $N = 10$ .

Table 15 shows results combining 1D, 2D, and 3D weight matrices. Again, including the 1D weight matrix does not significantly improve the results.

Number of Pattern	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Trials Successful	50	50	50	50	50	48	50	42	30	17	13	2	2	0

Table 15. Successes in 50 trials, shift-invariant model (1D, 2D, and 3D),  $N = 10$ .

We have seen that the Hopfield neural-network model requires  $\binom{N}{d}$  nonredundant connections. With the shift-invariant model there are  $\binom{N-1}{d}$  nonredundant connection weights. This is still a very large number for large  $N$ . For a  $64 \times 64$  binary image, one would still need over 8 million nonredundant connections, about the same as the number required for the Hopfield model.

By limiting the connections to a neighborhood surrounding the neuron, this number can be reduced significantly. For example, if each neuron is connected to only those neurons within a  $15 \times 15$  grid surrounding it, less than 25,000 connections would



be required (for a first order network). A shift invariant neural network of this kind could be used as a low-level classifier as part of a hierarchical system. Objects smaller than the neighborhood could be stored in the connection matrix. These small objects could then be recognized by the low-level classifier and application of a higher-level classifier could follow.

Note that, although there are estimated to be about a trillion neurons in the nervous system, each neuron has, on average, about 100 inputs converging on it while it in turn diverges to 100 other neurons.<sup>23</sup> Also, when one observes an image, for example a written page, one does not take in the entire image at once but focuses on only a small area of the image at a time. By limiting the connectivity of a neural network as described above, relatively small objects could be stored and could be recognized regardless of their location in the image without using an astronomically large number of connection weights. For a neural-network-based associative memory to be feasible such a reduction in the number of connection weights is essential. In addition, it is more realistic from a biological perspective to have each neuron connected to no more than a few hundred other neurons.

## CHAPTER 6. Conclusions

This dissertation has presented both analytical and experimental results on the number of patterns one can expect to be able to store in Hopfield neural networks using a modification of the delta rule. Analytical results indicate that the probability of one neuron successfully separating a set of  $m$  random patterns of  $N$  bits each should fall below 50% at about  $m = 2N$ . The following table shows experimental results on the ability of one neuron to separate sets of patterns for  $C_{100\%}$ ,  $C_{90\%}$ ,  $C_{50\%}$ , and  $C_{0\%}$  for  $N$  between 20 and 150.

Success Rate	N									
	20	30	40	50	60	70	80	90	100	150
$C_{100\%}$	1.2	1.23	1.42	1.34	1.55	1.4	1.52	1.58	1.64	1.73
$C_{90\%}$	1.4	1.47	1.65	1.6	1.73	1.62	1.75	1.76	1.68	1.79
$C_{50\%}$	1.85	1.87	1.85	1.84	1.88	1.83	1.9	1.87	1.94	1.91
$C_{0\%}$	2.3	2.37	2.35	2.28	2.18	2.08	2.14	2.13	2.06	1.93

Table 16.  $C_{100\%}$ ,  $C_{90\%}$ ,  $C_{50\%}$ , and  $C_{0\%}$  for  $N = 20$  to 150. (Separating)

The ability of one neuron to separate a set of  $m$  patterns of  $N$  bits each is directly related to the ability of the neural network to successfully store  $m$  patterns of  $N$  bits each. Once the probability of one neuron separating a set of patterns falls below about 99%, the ability to store the set of patterns falls very quickly. Table 17 shows experimental results on the ability to store sets of patterns for  $C_{100\%}$ ,  $C_{90\%}$ ,  $C_{50\%}$ , and  $C_{0\%}$  for  $N$  between 20 and 100.

Success Rate	N								
	20	30	40	50	60	70	80	90	100
$C_{100\%}$	0.95	1.07	1.1	1.2	1.3	1.31	1.44	1.46	1.45
$C_{90\%}$	1.10	1.27	1.32	1.40	1.42	1.42	1.51	1.50	1.49
$C_{50\%}$	1.30	1.40	1.45	1.48	1.50	1.51	1.54	1.56	1.52
$C_{0\%}$	1.60	1.63	1.60	1.66	1.57	1.57	1.59	1.61	1.54

Table 17.  $C_{90\%}$ ,  $C_{50\%}$ , and  $C_{0\%}$  for  $N = 20$  to 100. (Storing).

Figures 13 and 14 show the same results graphically.

There are two reasons why the experimental results are not as good as that predicted by the analysis. First, because of time limitations, some limit must be set on the number of iterations one can run the delta rule. The higher the limit is, the sharper the drop-off from nearly 100% success to nearly 0% success. Second, as  $N$  is increased, the beginning of the drop-off, in relation to  $N$ , is increased.

A method of improving the performance of the delta rule has been presented. This method involves using direct negative feedback of neurons ( $T_{ii} < 0$ ) during the learning process and removing the negative feedback afterwards. A thorough comparison between this method and the continuous unlearning method<sup>12</sup> still remains to be performed. It has been shown that using negative feedback during learning can reduce the average size of the basins of attraction of the non-complement spurious states and improve the chance that a random input will converge to the attractor closest in Hamming distance. This leads to better performance of the neural network as a content-addressable memory.

It has also been shown that the Hopfield model needs to be modified only slightly to be able to store and retrieve non-binary patterns. A learning rule for such a model

Figure 13. Capacity--separating

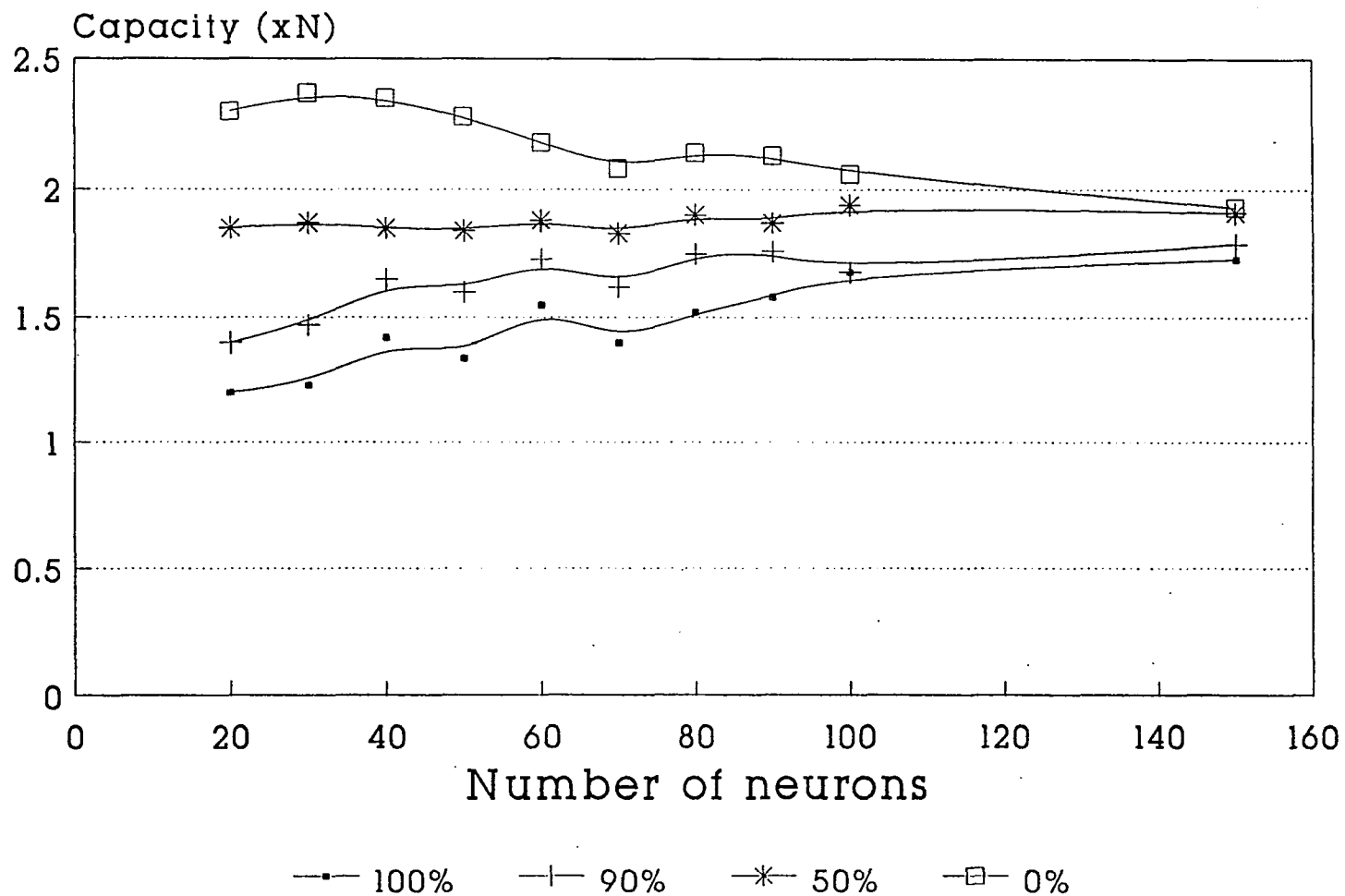
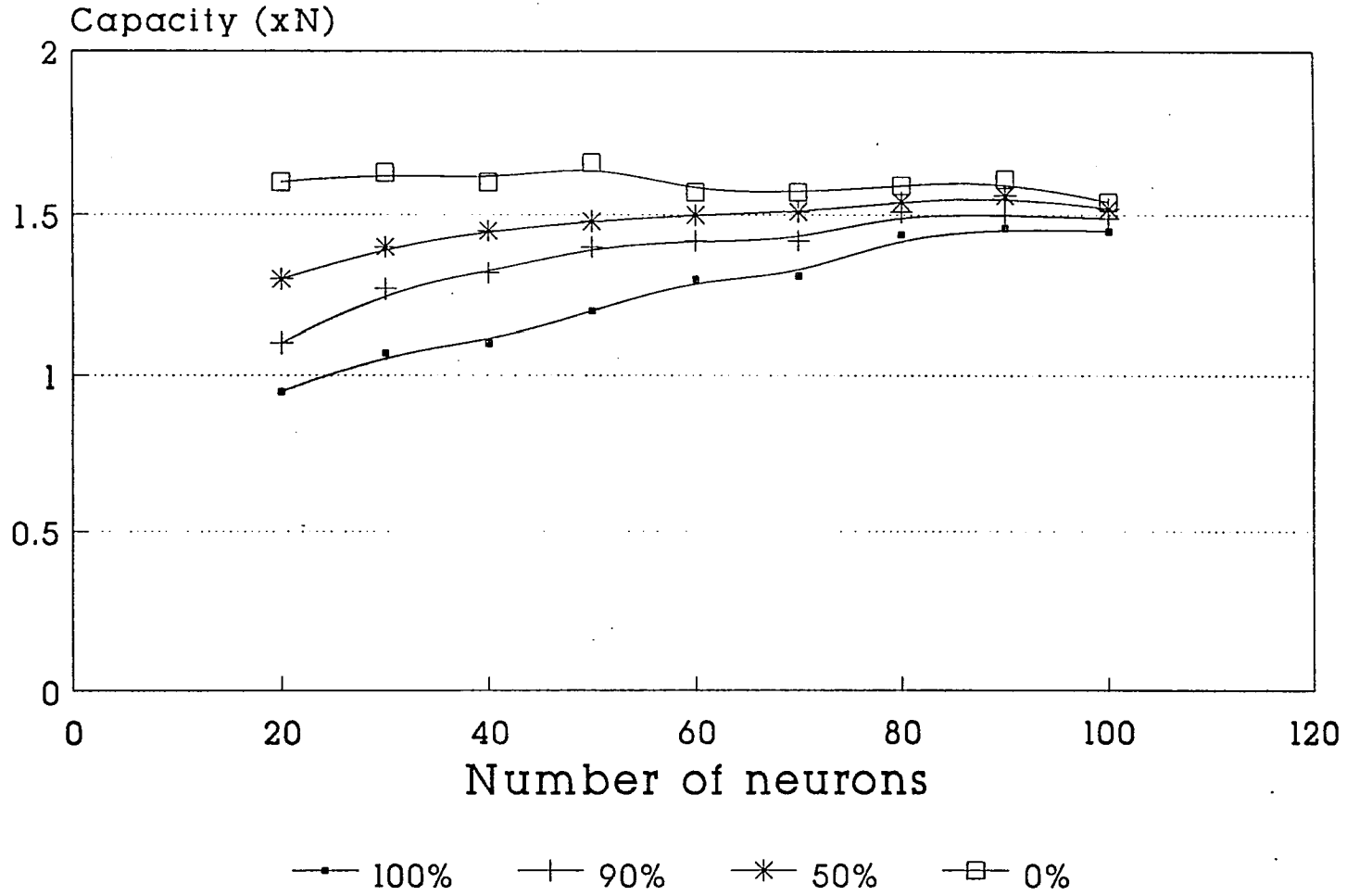


Figure 14. Capacity--storing



has been presented. Although such a model cannot store as many patterns as the binary model, each pattern contains twice as much information as binary patterns.

A learning rule for shift-invariant neural network models has also been presented. Such a network requires using a three-dimensional weight matrix in addition to a two-dimensional weight matrix; however, each time a pattern is stored, all of its cyclic shifts are also stored. For example, if  $N = 10$ , although the success rate falls below 90% when the number of patterns reaches about 17, one can store, say, 15 patterns with all their cyclic shifts (150 patterns in all) with about a 99% success rate. If one were to try storing all 150 patterns using the Hopfield model with both second-order and third-order terms, the probability of success would be much less.

Since the number of weights in a shift-invariant neural network that uses both a three-dimensional and a two-dimensional weight matrix can grow quite large for large  $N$ , limiting the connectivity of such a model has been proposed. It appears that the capacity of a neural network in which each neuron is connected to only  $d$  of the other  $N - 1$  neurons is approximately equal to the capacity of a neural network of  $d$  neurons. We propose that a limited-diameter shift-invariant neural network could be used as a low-level classifier as part of a hierarchical system. Objects smaller than the neighborhood could be stored in the connection matrix. These small objects could then be recognized regardless of their location in the image with any noise or other variations in the shape of the images being removed. Application of a higher-level classifier could follow.

## References

1. J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. National Academy of Science, USA*, vol. 79, pp. 2554-2558, 1982.
2. R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh, "The Capacity of the Hopfield Associative Memory," *IEEE Transactions on Information Theory*, vol. IT-33, pp. 461-482, 1987.
3. Y. S. Abu-Mostafa and J. St. Jacques, "Information Capacity of the Hopfield Model," *IEEE Transactions on Information Theory*, vol. IT-31, no. 4, pp. 461-464, July, 1985.
4. S. S. Venkatesh, "Epsilon Capacity of Neural Networks," *Proc. AIP Conference on Neural Networks for Computing*, pp. 440-445, Snowbird, UT, 1986.
5. David H. Hubel, *Eye, Brain, and Vision*, Scientific American Library, New York, 1988.
6. D. L. Prados and S.C. Kak, "Neural Network Capacity Using the Delta Rule," *Electronics Letters*, vol. 25, no. 3, pp. 197-199, 2nd February, 1989.
7. S. H. Cameron, "An Estimate of the Complexity Requisite in a Universal Decision Network," Bionics Symposium, Wright Airforce Dev. Div. (WADD) Rep. 60-600, pp. 197-212, 1960.
8. R. O. Winder, "Threshold Logic," PhD Dissertation, Princeton University, Princeton, NJ, 1962.

9. D. L. Prados, "The Capacity of a Neural Network," *Electronics Letters*, vol. 24, pp. 454-455, 1988.
10. T. Maxwell, C. L. Giles, and Y. C. Lee, "Transformation Invariance Using High Order Correlations in Neural Net Architectures," Plasma Preprint UMLPF #88-125, University of Maryland, 1988.
11. J. J. Hopfield, "Unlearning Has a Stabilizing Effect in Collective Memories," *Nature*, vol. 304, pp. 158-159, July, 1983.
12. C. H. Youn and S. C. Kak, "Continuous Unlearning in Neural Networks," *Electronics Letters*, vol. 25, no. 3, pp. 202-203, February 2, 1989.
13. L. Akers, M. Walker, D. Ferry, and R. Grondin, "A Limited-Interconnect, Highly Layered Synthetic Neural Architecture," in *VLSI for Artificial Intelligence*, ed. J.G. Delgado-Frias and W.R. Moore, pp. 218-226, Kluwer Academic Publishers, 1989.
14. Z. Butler, A. Murray, and A. Smith, "VLSI Bit-Serial Neural Networks," in *VLSI for Artificial Intelligence*, ed. J.G. Delgado-Frias and W.R. Moore, pp. 201-208, Kluwer Academic Publishers, 1989.
15. M. Verleysen, B. Sirletti, and P. Jespers, "A New CMOS Architecture for Neural Networks," in *VLSI for Artificial Intelligence*, ed. J.G. Delgado-Frias and W.R. Moore, pp. 209-217, Kluwer Academic Publishers, 1989.
16. D. L. Prados and S.C. Kak, "Non-Binary Neural Networks," in *Advances in Computing and Control*, ed. W.A. Porter, S.C. Kak, and J.C. Aravena, pp. 97-



- 104, Springer-Verlag, New York, 1989.
17. F. Rosenblatt, "The Perceptron: A Probabilistic Model For Information Storage and Organization in the Brain," *Psychological Review*, vol. 65, pp. 386-408, 1958.
  18. T. Kohonen, "The Neural Phonetic Typewriter," *Computer*, vol. 21, no. 3, pp. 11-22, 1988.
  19. G. A. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," *Computer*, vol. 21, no. 3, pp. 77-88, 1988.
  20. W. Widrow and R. Winter, "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition," *Computer*, vol. 21, no. 3, pp. 25-39, 1988.
  21. D. L. Prados and S.C. Kak, "Shift Invariant Associative Memory," in *VLSI for Artificial Intelligence*, ed. J.G. Delgado-Frias and W.R. Moore, pp. 189-197, Kluwer Academic Publishers, 1989.
  22. W. Pitts and W. S. McCulloch, "How We Know Universals: The Perception of Auditory and Visual Forms," *Bulletin of Mathematical Biophysics*, vol. 9, pp. 127-147, 1947.
  23. W. F. Ganong, *Review of Medical Physiology*, Lange Medical Publications, Los Altos, CA, 1979.

**VITA**  
Donald L. Prados  
1515 Aztec Ave., Apt. #8  
Metairie, LA

**JOB OBJECTIVE** Teaching Position in Artificial Intelligence, Neural Networks, Expert Systems, Image Processing.

**EDUCATION**

8/84 - 8/89 Louisiana State University, Baton Rouge, LA 70803  
Ph.D. in Electrical Engineering, August, 1989 (expected)  
Dissertation: "The Capacity of Artificial Neural Networks Using the Delta Rule"  
M.S. in Electrical Engineering, May, 1986.

8/83 - 5/84 University of New Orleans, New Orleans, LA  
No Degree, Electrical Engineering

8/77 - 8/82 Tulane University, New Orleans, LA 70118  
M.S. in Biomedical Engineering, August, 1982  
Thesis: "Studies on the Sensation and Pain Thresholds of Electrotactile Stimulation"  
B.S. in Biomedical Engineering, May, 1981.

*Electrical Engineering:* Educational emphasis on Neural Networks, Artificial Intelligence, and Pattern Recognition. Research has involved studying the capacity of neural networks and various learning algorithms for neural networks.

*Biomedical Engineering:* Educational emphasis on Rehabilitation Engineering, Biomechanics, and Biomaterials. Research involved studying the sensation and pain thresholds of electrotactile stimulation.

**EXPERIENCE**

5/87 - 8/89 *Teaching Assistantship*  
Electrical and Computer Engineering Dept., Louisiana State University  
Instructor: Digital Logic II  
Instructor/Laboratory Supervisor: Digital Logic Design Lab

8/86 - 5/87 *Instructorship*  
Electrical and Computer Engineering Dept., Louisiana State University  
Instructor: Digital Logic II and Microprocessor Systems

8/85 - 8/86 *Teaching Assistantship*  
Electrical and Computer Engineering Dept., Louisiana State University  
Instructor/Laboratory Supervisor: Digital Logic Design Lab

8/84 - 8/85 *Teaching Assistantship*  
Electrical and Computer Engineering Dept., Louisiana State University  
Laboratory Assistant: Digital Logic Design Lab, Microprocessor Lab

11/83 - 5/84 *Laboratory Technician*  
Physiology Dept., Tulane University Medical Center  
New Orleans, LA

- PUBLICATIONS**
1. D. Prados and S.C. Kak, "Neural Network Capacity Using the Delta Rule," *Electronics Letters*, Vol. 25, No. 3, 2nd February, 1989, pp. 197-199.
  2. D. Prados and S.C. Kak, "Non-Binary Neural Networks," in *Advances in Computing and Control*, W.A. Porter, S.C. Kak, and J.C. Aravena (Eds), New York, Springer-Verlag, 1989, pp. 97-104.
  3. D. Prados and S.C. Kak, "Shift Invariant Associative Memory," in *VLSI for Artificial Intelligence*, J.G. Delgado-Frias and W.R. Moore (Eds), Kluwer Academic Publishers, 1988, pp. 189-197.
  4. D. Prados, "Capacity of a Neural Network," *Electronics Letters*, Vol. 24, No. 8, 14th April, 1988, pp. 454-455.
  5. M. Solomonow and D. Prados, "Further Evidence of Learning in the Tactile Sense," *IEEE Frontiers of Engineering in Health Care*, 1982, pp. 82-84.
  6. M. Solomonow, D. Lazar, and D. Prados, "Advances in Electrotactile Stimulation for Prosthetic Sensory Feedback," *Advances on External Control of Human Extremities*, D. Popovic (Ed) published by Etan, Belgrade, Yugoslavia, 1981, pp. 69-84.

#### CONFERENCE PRESENTATIONS

"Non-Binary Neural Networks" (see publication 2), presented at *ComCon*, Baton Rouge, LA, October, 1988.

"Shift Invariant Associative Memory" (see publication 3), presented at *The International Workshop on VLSI for Artificial Intelligence*, Oxford University, England, July, 1988.

**MEMBERSHIPS** Institute of Electrical and Electronic Engineers Computer Society

**INTERESTS** Basketball, tennis, science fiction, music

**PERSONAL DATA** Male, Age: 30, Single, Excellent Health

**REFERENCES** Dr. Alan H. Marshak, Chairman, Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, phone: 504-388-5243.

Dr. Subhash C. Kak, Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, phone: 504-388-5552.

Dr. William C. Van Buskirk, Head, Department of Biomedical Engineering, Tulane University, New Orleans, LA 70118.

Dr. Moshe Solomonow, LSU Medical Center, New Orleans, LA.

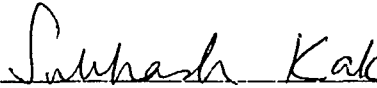
**DOCTORAL EXAMINATION AND DISSERTATION REPORT**

**Candidate:** Donald Louis Prados

**Major Field:** Electrical Engineering

**Title of Dissertation:** The Capacity of Artificial Neural Networks Using the Delta Rule

**Approved:**

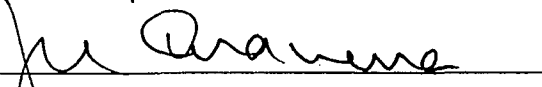
  
Major Professor and Chairman

  
Dean of the Graduate School

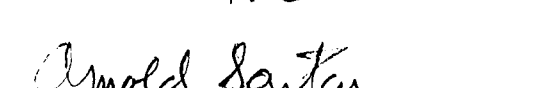
**EXAMINING COMMITTEE:**

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_

**Date of Examination:**

July 14, 1989