NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

Linear Separability and Connectionist Categorization: A Study of Speed and Generalization of Two Connectionist Networks.

BY 

Kevin S. Shamanski

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

DEPARTMENT OF PSYCHOLOGY

Edmonton, Alberta

Spring, 1994

ISBN 0-612-11366-3

Canada

UNIVERSITY OF ALBERTA

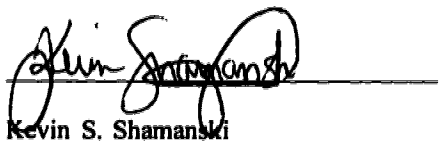
RELEASE FORM

NAME OF AUTHOR: Kevin S. Shamanski

TITLE OF THESIS: Linear Separability and Connectionist Categorization: A Study of Speed and Generalization of Two Connectionist Networks.

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1994



Kevin S. Shamanski

Box 297

Carberry, MB

ROK OHO

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled Linear Separability and Connectionist Categorization: A Study of Speed and Generalization of Two Connectionist Networks submitted by Kevin S. Shamanski in partial fulfillment of the requirements for the degree of Master of Science.

Michael R.W. Dawson

Don Kuiken

Peter Van Beek

MARCH 22/94

Abstract

The explosion in connectionist research that has occurred in the past decade has produced a large number of learning rules, each designed to train a specific network architecture. The *generalized delta rule* proposed by Rumelhart, Hinton, and Williams (1986a, 1986b) has become one of the most prominent of these learning algorithms. The purpose of the research described is to systematically compare the performance of two rules used to train connectionist networks: the standard generalized delta rule devised by Rumelhart, Hinton, and Williams(1986a, 1986b); and an extension of this rule developed by Dawson and Schopflocher (Dawson and Schopflocher, 1992; Dawson, Schopflocher, Kidd, and Shamanski, 1992). In order to make this comparison, the paper proceeds as follows: First, activation functions and learning rules are described, and their interrelation is briefly explored. Second, in depth consideration is given to the logistic and Gaussian activation functions and how the generalized delta rule relates to each respective functions. Third, the results of a series of computer simulations are reported. The aim of these simulations was to provide a controlled and systematic comparison of the two rules.

The results of Experiment 1 demonstrated that the logistic architecture was more suited to solving a linearly separable problem than a linearly nonseparable problem when speed to convergence was considered. A network of Gaussian units, in contrast, had difficulty solving the linearly separable problem, but was well suited to solving the linearly nonseparable problem. The simulations of Experiment 2 add additional support to the network type-problem type interaction though a dependent measure of generalization. Overall, each network architecture generalized well on only one specific problem type: value units on the linearly nonseparable problem, and integration devices on the linearly separable problem. These results are discussed within the framework of cognitive science, and consideration is given to how they might contribute to a theory of categorization.

Acknowledgement

I would like to acknowledge my esteemed supervisor, Michael R. W. Dawson, whose patience and guidance allowed me to achieve this goal for without him who knows what I would have done. I would also like to acknowledge the members of the Biological Computation Project for their ideas, comments, and their gentle criticism of this research; their aid was invaluable. Acknowledgement should also be given to TLD, DMA, MAF, KS, who each in their own way kept me on the path to complete this journey, and never doubted that I could do it.

Table of Contents

List of Figures and Illustrations

Figures

Introduction

The basic instrument of cognitive science is a model and, in general, over the last 50 years the predominant form of these models has been that of a symbolic or classical nature (Fodor & Pylyshyn, 1988; Newell & Simon, 1981). It has not been until the last decade that a new approach to modelling has arisen called parallel distributed processing (PDP). This approach uses networks of simple processing units that communicate with one another simultaneously via weighted connections to model cognitive phenomena (see Anderson & Rosenfeld, 1988). An example of such a system is shown in Figure 1. The example network consists of three processing layers: an input unit layer, which receives environmental information; a hidden unit layer, which detects features in the input information; and an output unit layer, which records the network's response to the stimulus.

Each connection in a PDP network has a weight associated with it that scales information passing through it. The system's "knowledge" is stored as the pattern of connection weights in the network. "In this sense, the connection strengths play the role of the program in a conventional computer" (Smolensky, 1988, p. 1). A property of the connections in a PDP model that make them particularly attractive is that they are modifiable. The current connection weights of the system can be altered on the basis of environmental information the network receives. Such changes in connection strength are determined by a learning rule. To apply a learning rule to modifiable weights is to "teach" the network to perform some specific task (e.g. Smolensky, 1988).

The explosion in connectionist research that has occurred in the past decade has produced a large number of learning rules, each designed to train a specific network architecture. The *generalized delta rule* proposed by Rumelhart, Hinton, and Williams (1986a, 1986b) has become one of the most prominent of these learning algorithms, and has the ability to train multilayer networks such as the one depicted in Figure 1.

As will be elaborated below, the mechanics of the generalized delta rule depend upon the characteristics of individual processing units in a PDP network. As a result, variants of the generalized delta rule have been devised to train slightly different types of networks. The purpose of the research

described below is to systematically compare the performance of two of these rules: the standard generalized delta rule devised by Rumelhart, Hinton, and Williams(1986a, 1986b); and an extension of this rule developed by Dawson and Schopflocher (Dawson and Schopflocher, 1992; Dawson, Schopflocher, Kidd, and Shamanski, 1992). In order to make such a comparison, this paper will proceed as follows: First, activation functions and learning rules will be described in general, and their interrelation briefly explored. Second, in depth consideration will be given to two specific activation functions and how the generalized delta rule relates to each. Third, the results of a series of computer simulations will be reported. The aim of these simulations was to provide a controlled and systematic comparison of the two learning rules above.

Processors, Activation Functions, and Learning Rules

To understand how a PDP model works, one must understand the mechanics of an individual processor within the network. The first operation of a processing unit is to compute its *net input*. A unit's net input is simply the sum of all of the weighted signals that it receives from other processors. The second operation of a processor is to activate. The processor will adopt a new level of internal activity based on the net input that it has received. This level of activity is determined by the unit's *activation function*. The final operation of a processor is to output its activity to other processors. Each unit will send its signal over a weighted connection to be received by another unit in the network. Of these three operations, computing activation is the most important with regards to the learning rules to be described.

The simplest activation function for a processing unit is the linear threshold function (see Figure 2a). According to this function, if a unit's net input exceeds some threshold, 0, then the unit becomes activated (shown in Figure 2 as net output levels of 1.0). If the net input is less than 0, then the unit becomes inactive (activation value of 0.0). This function has the advantage of being neurologically plausible: it captures the "All or None Law" governing a neuron's generation of an action potential as a function of the signals that it receives (Levitan & Kaczmarek, 1991, p.38).

Training Integration Devices

Unfortunately, the linear threshold function is not continuous, and therefore does not possess a derivative. As a result, researchers were unable to develop a learning rule that could train multilayer networks that used this function (eg. a network like Figure 1). This is because the development of a learning rule usually requires using calculus to determine what changes in connection weights will reduce system error. If a derivative for an activation function does not exist, then the rules of calculus cannot be exploited, and a learning rule can not be derived. This nearly led to the death of connectionism, because without trainable hidden units, networks were extremely limited in their computational power (Rumelhart & McClelland, 1986; Minsky & Papert, 1988)

To overcome such limitations, modern connectionism began research into a continuous approximation of the linear threshold function. One such approximation is the logistic function illustrated

in Figure 2b. The limits of this sigmoid-shaped "squashing" function provide a good approximation of the two discrete states of the linear threshold function in Figure 2a. Ballard (1986) refers to processors using this activation function as *integration devices.*

The important difference between the linear threshold function and the squashing function is that the latter is continuous, and therefore possesses a derivative. As a result, a learning rule can be derived for multilayer architectures whose processors use this function. The ability to train hidden units with this new learning rule allowed networks to solve more complex problems. For example, Rumelhart, Hinton, and Williams (1986b) demonstrated that a multilayer network utilizing the logistic activation function is able to learn the exclusive-or (XOR) distinction, while a single layer network (ie. no hidden units) is unable to make this distinction.

In order to train multilayer networks of integration devices the standard generalized delta rule (Rumelhart, Hinton, and Williams, 1986a, 1986b) is used. The aim of the generalized delta rule is to minimize network error by modifying the network's connections, where network error (E_p) is given by the equation

$$E_p = \frac{1}{2}\sum (T_{pj}-O_{pj})^2 \qquad (1)$$

which represents the sum of squared differences between the desired (T_{pj}) and observed value (O_{pj}) of all output units when presented a specific pattern *p*. This error term is minimized as follows: First, the network is presented input pattern *p*. After processing, this creates a pattern of response activity in its output units. This response is compared to the desired response that the network is being taught to produce. Discrepancies between the desired output pattern and the observed output pattern create an error signal for the output units. This error signal is used to modify the weights of the output unit connections. Then, output unit error is sent to the hidden units through the modified connections. The hidden units then use the transmitted error to calculate their own error term. This error is used to change the weights of the next layer of connections. The sequence of (1) compute error, (2) modify weights, and (3) transmit error is repeated until all network connections have been updated. These modifications of connection weights dictated by the generalized delta rule are guaranteed to reduce system error as defined in

Equation 1. As a result, the next time the network is presented pattern p, its response will be closer to the desired response.

Training Networks of Value Units

The generalized delta rule has become a benchmark to which other algorithms are compared because it consistently produces extremely good pattern classifiers (e.g., Barnard & Casasent, 1989). However, it can often lead to networks comprised of a very large number of hidden units. As a result, network training does not necessarily generalize well to new patterns, and network structure becomes extremely difficult to interpret (Rumelhart & McClelland, 1986).

One approach to improving network interpretability is to adopt an activation function which makes processing units more powerful. As a result, fewer processors will be required in a trained network's architecture. For example, Dawson and Schopflocher (1992) developed a multilayer perceptron whose units used the Gaussian activation function. This activation function is illustrated in Figure 2c. Ballard (1986) has referred to units possessing this kind of activation function as *value units*, because such units only activate strongly to a small range of net input values.

The generalized delta rule was derived under the assumption that the activation function of the network's units would be monotonic. For example, generic units using the logistic function are monotonic, meaning that their output value always increases as a function of increasing net input. In contrast, the Gaussian is nonmonotonic because its value can decrease as net input increases. As a result, the standard form of the generalized delta rule cannot be used as a practical method for training networks of value units, as shown by Dawson and Schopflocher (1992). They used the standard version of the generalized delta rule to train a network built from units that had a Gaussian activation function. They found that the network was almost always trained into a local minimum, and did not learn to respond correctly to all inputs.

To solve this problem Dawson and Schopflocher (1992) developed a variant of the generalized delta rule that minimized an elaborated error term:

$$C_p = \frac{1}{2}\sum (T_{pj}-O_{pj})^2 + \frac{1}{2}\sum T_{pj}\bullet(net_p-\mu_j)^2 \qquad (2)$$

The first component of Equation 2 is identical to Equation 1, where T_{pj} represents the desired response of output unit *j* for pattern *p*, and O_{pj} is the observed response of the unit. The second component measures the failure of the system to set $net_{pj} = \mu_j$ when the desired output is equal to 1.0, where net_{pj} is the net input to unit *j* when pattern *p* is presented, and μ_j is the 'bias' value of the activation function. As defined, this second component requires that T_{pj} be either 0.0 or 1.0; this limits the network to (ideally) generating binary outputs after training. With the minimization of equation (2) the goal of learning, the local minimum problem described above is avoided, because the second term in C_p prevents the weights being changed such that all the net inputs are drawn towards infinity. In essence, the additional term in Equation 2 applies heuristic information about the Gaussian to prevent patterns where T_{pj} is equal to 1.0 from being pulled into the tail of the activation function's distribution. This modified version of the generalized delta rule has the ability to train multilayer networks of value units.

Comparison of The Generalized Delta Rules

Given that we have two distinct generalized delta rules for architectures that use different activation functions, how do the rules compare to one another? Does one of the networks always learn faster than the other? Does one architecture generalize better to novel instances after training than the other? Does network performance on one problem generalize to performance on other problems? Currently, little research has been done on any of these questions, and research that *has* been performed is inconclusive.

For instance, Dawson and Schopflocher (1992) compared the performance of the two rules on a variety of small problems (XOR, symmetry, encoder, parity). The results of this comparison showed that there were definite advantages for using value units in a network: in general, value unit networks learned to solve these problems much faster, and failed to converge significantly less frequently, than did networks of integration devices (see also Dawson, Schopflocher, Kidd, & Shamanski, 1992; Dawson, Shamanski, & Medler, 1993). Unfortunately, this comparison lacked systematic control over important network parameters and design variables. For example, insufficient attention was paid to the relation between the network's activation function (Gaussian or logistic) and the problem type used for training.

In essence, their results could lead to the mistaken conclusion that the value unit architecture is always better than the generic architecture. One must consider that all of the problems studied by Dawson and Schopflocher (1992) were linearly nonseparable, and therefore their conclusions may not apply to linearly separable problems. In fact, it is argued below that networks of integration devices should perform better than networks of value units on a particular type of problem -- called a linearly separable problem. However, networks of value units are arguably better suited to solve more complicated problems.

Pattern Classifiers, and Pattern Spaces

PDP networks are known to be good classifiers. This means that a trained network is able to make a distinction between whether a pattern belongs, or does not belong, to a specific category. In essence, when the network is given an input pattern, it will produce the name of the category to which the input pattern belongs. To do this the network must "carve" the pattern space into differently named classification regions.

A pattern space is an N-dimensional space, where N represents the number of coordinates required to locate any pattern as a point in this space. The network's task is to carve this pattern space into different regions. Patterns that belong to one category will fall into some of these regions, and patterns that do not belong to the category will fall into other regions.

From this perspective, some pattern classifications may be very simple. For example, a *linearly separable class* may be defined by carving a single hyperplane through the respective pattern space (Note: In two-dimensions this hyperplane would be a straight line). In this case, patterns on one side of the hyperplane belong to one category, while patterns on the opposite side of the plane belong to a different category. An example of a linearly separable class is the *majority class.* If a majority of the pattern's units are on, then the pattern belongs to the majority class, otherwise the pattern does not belong to the majority class. For example, the pattern [1 0 0 1 1] belongs to the majority class because 3 of its 5 units are active, where 1 represents an input unit being active. The pattern [0 1 0 1 0] does not belong to the majority class as less than one half of its units are active.

In contrast, some pattern classifications are very complicated, and almost arbitrary decision

regions must be cut in the pattern space to provide a solution (Lippman, 1987). For example, consider the *parity class* (referred to as a Type II problem by Minsky & Papert, 1988). If an odd number of units in the pattern are active, then these points will belong to the class of odd parity, while all other possible patterns will not belong to the class of odd parity. For example, the pattern [1 1 1 0 0] would be classified as an odd parity pattern as an odd number of units in the pattern are active, while the pattern [0 1 1 0 0] would not be an odd parity pattern as an even number of units in the pattern are active.

Note that the difference between simple and difficult classifications is not the size of the pattern space, but is instead the complexity of the decision regions required to carve the space correctly. For example, one can take exactly the same set of input patterns (and thus exactly the same pattern space), and define either the majority class or the parity class for this set. This is important, because it allows a researcher to manipulate problem complexity, while at the same time controlling other factors, such as the number of input units and therefore the dimension of the pattern space.

PDP Networks Carve Pattern Spaces

As previously mentioned, PDP networks can learn to classify patterns. This ability is a result of the activation function in the network having a "receptive field" that carves the pattern space into decision regions. The type of activation function controls how this space is partitioned. For instance, the logistic activation function carves a single plane through the pattern space. This is shown by considering the receptive field of this function that is shown in Figure 3a. As shown, the logistic divides the decision region into two areas: one area (shaded) represents inclusion in the class, while the other (white) represents exclusion from the class.

In contrast, the Gaussian activation function carves a "hyperbar" through the pattern space (eg. Hartman & Keeler, 1991). Figure 3b shows the receptive field of the Gaussian. This band creates three decision regions: one area within the boundaries of the band (shaded) representing inclusion, and two areas on either side of the band representing exclusion from the class. This allows a more complex division of the pattern space than the single hyperplane of the logistic. As shall be discussed, this complex partitioning is not always beneficial in solving particular problems.

For example, consider the linearly separable majority problem. To solve this problem, a network must learn to activate its output unit when more than half of its input units have been activated. Figure 4a shows that a small (2 input unit) version of the majority problem can be solved by an integration device with no hidden units. For this problem, the activation function of the output unit carves a single line through the pattern space, which after weight manipulations within the network learns to separate the pattern with a majority of its bits on from the other patterns. A value unit network given this same problem would also be able to partition the pattern space correctly by carving two parallel lines that differentiate this one pattern from the others (Figure 4b).

However, when the dimensionality of the pattern space is increased, a difference between the two architectures emerges. As Figure 4c shows, an integration device network with no hidden units can still solve a larger (3 input unit) version of the majority problem by carving a single plane through this larger pattern space, because of the problem's linear separability. In contrast, this is *not* true of the value unit network. Because the "ON" region of its receptive field has limited width (see Figure 3), it is too narrow to capture all of the "majority-on" patterns in the larger pattern space. To compensate, another unit (providing an adjacent "ON" region) is required to correctly solve the problem (Figure 4d).

However, the simple division of the pattern space performed by the logistic activation function can encounter problems when complex solutions are required. For instance, the parity classification requires a more complicated partitioning. Figure 5a shows that a hyperplane through the pattern space is unable to correctly partition the patterns for the parity classification as an area to one side of the line encompasses patterns of both classes (inclusion and exclusion in odd parity). In order to make this classification another hidden unit is required that adds a second hyperplane to the pattern space. Placed correctly, the second hyperplane will form a band around one class of patterns, and the other class will be outside of the band. This leads to the conclusion that as the number of input units, and thus the dimension of the pattern space, increases the number of hidden logistic units must also undergo an analogous increase. However, smaller increases in the complexity of network structure may occur when other activation functions are used.

Figure 5b shows how an output unit using a Gaussian activation function carves a hyperbar through the 2-bit parity pattern space. In this case, the bar is able to encompass the even parity patterns between the two parallel lines of the bar which represents class inclusion. Due to the Gaussian's ability to carve two lines in the pattern space simultaneously, fewer units are required to create the same decision regions as those created by a logistic network. This should allow a smaller network structure (ie. fewer hidden units and connections), as well as create a more interpretable network structure.

Figure 5c and 5d show each respective network's ability to carve a 3-dimensional parity problem. Figure 5c shows how an integration device network is able to correctly partition the parity pattern space given 3 planes to position. It is interesting to note that the logistic units try to position their planes parallel to one another in order to create a hyperbar much like the shape of the value unit partition. Figure 5d shows the correct partitioning of the parity pattern space carved by value units. In this case, the value unit network has 2 hyperbars to position, which it manipulates to capture the odd parity instances within the hyperbars. So, as the parity problem is scaled up to higher dimensions the simplest network structure to solve the parity classification is drastically different for each network type: integration devices require more hidden units in order to create hyperband like partitions, while value units already possess a hyperbar structure that allows them to easily encompass the odd parity patterns.

Based on this information, one would have to assume that the logistic function is better able to make a linear discrimination such as the majority classification. This does not mean that the value unit architecture is unable to make this classification, but that its performance will be poorer than that of the logistic *if* an analogous network structure is used. A parallel difference in performance may not exist for networks tested on different pattern classifications. For instance, incorporation of value units into an architecture learning the parity classification would allow fewer units to perform the same partitioning of the pattern space as a larger network of logistic units. This change in performance based upon the networks structure (ie. number of units) has been explored theoretically, but no empirical testing has been done to verify this theory. The simulations below attempt to provide this empirical evidence.

Experiment I

The purpose of this set of simulations was to compare the learning performance of two network architectures, integration devices and value units, on two types of classifications: the linearly separable majority problem and the linearly non-separable parity problem. The argument illustrated in Figure 2 suggests that linearly separable problems, particularly those set in a high-dimensional pattern space, will be more complicated (i.e., require more hidden units) for a network of value units than for a network of integration devices. Our dependent measure was the number of training epochs required by a network to correctly classify all patterns for a particular problem. These simulations were designed to test two specific hypotheses: (1) Networks of integration devices should learn the majority problem much faster than networks of value units, and should utilize a simpler network structure; (2) Networks of value units should learn the parity problem much faster than networks of integration devices, and should utilize a simpler network structure.

Method

Problem Types

Networks were trained on two different problems: majority and parity. Each to-be-classified pattern was represented as a binary pattern of input unit activity, where 1 represented activation of an unit, and 0 represented unit inactivity. Problem size was manipulated by using different numbers of input units in a network. Eight different sizes of the majority problem were studied, consisting of 2, 3, 4, 5, 6, 7, 8, or 9 input units (bits). As a great deal of difficulty was encountered in training integration devices to convergence on the parity problem, only six different sizes of it were studied for both architectures (2, 3, 4, 5, 6, or 7 input units). For a specific problem of a particular size, a network was trained on all possible patterns where half of the possible patterns belong to one class, and the other half did not.

Network Architectures

Networks of integration devices and networks of value units were both trained on the problems described above. All of the networks had a single output unit, and their number of input units defined

by problem size, and had massively parallel sets of connections between adjacent layers of processing units. The number of hidden units used for each network was determined by a combination of pilot studies and theoretical analyses. Our goal was to equate the different architectures to one another with respect to computational power by using the simplest network, in principle, possible to solve the problem.

Considering the two distinctly different architectures, one must also address the numerous free parameter within the architectures that may be varied to allow a network to learn. In these simulations the parameters were not equated for each architecture type, rather these parameters were systematically varied to allow the specific network to learn the classifications. It was felt that this approach would allow a more holistic evaluation of network properties, where the emphasis was placed on equating the two architectures with respect to their computational power. In essence, each network was given only enough computational power to solve the problem by restricting the number of hidden units used in the network architecture.

For all sizes of the majority problem, no hidden units were used for the integration device networks. For the value unit networks, the 2-bit majority problem was solved with no hidden units, the 3-bit through 7-bit versions of this problem were solved with one hidden unit, and the 8- and 9-bit versions of this problem were solved with two hidden units. For the parity problem, networks of integration devices with N input units also had N hidden units. This design decision was taken because previous work had shown that generic networks with this structure can arrive at a solution to the parity problem (e.g., Rumelhart, Hinton & Williams, 1986a, 1986b). For the value unit networks, no hidden units were used for the 2-bit parity problem (see also Dawson & Schopflocher, 1992). For the remainder of the problems, $N/2$ hidden units were used when N was even, and $(N + 1)/2$ hidden units were used when N was odd.

Training Procedures

In order to compare the average performance of the two architectures on the different problems, each network was trained on every combination of problem type and problem size 25 different times; at the outset, all of the networks had a random starting configuration. Networks of integration devices

were trained with the generalized delta rule (Rumelhart, Hinton & Williams, 1986a, 1986b). The initial connection weights were randomly set in the range from -0.3 to 0.3. Biases were initialized to zero. Connection weights and biases were updated after the presentation of each pattern; order of pattern presentation was randomized every epoch. The network was trained until a "hit" was recorded for every output unit for every pattern in the training set. We operationalized a hit as being an activation of 0.9 or greater when the desired output was 1.0, and as being an activation of 0.1 or less when the desired output was 0.

For all sizes of the majority problem, the networks of integration devices were trained with a learning rate of 0.5 and a momentum term of 0.9. These parameters were also used for the 2- and 3-bit parity problems. For all other sizes of the parity problem, the learning rate was reduced to 0.1, as this tended to give the best learning performance of these networks to the problem.

As is described in the results section, the performance of the integration device networks on the parity problem was very poor, particularly as problem size increased. As a result, we decided to end training if convergence had not been achieved after 20,000 epochs. If a network failed to converge after this much training, we recorded its total sum of squared error to the training set for later comparison with value unit networks.

The networks composed of value units were trained with the Dawson and Schopflocher (1992) extension of the generalized delta rule. Network weights were randomly set in the range of -0.3 to 0.3, while biases were initialized to 0.0. Once again, connection weights and biases were updated after every pattern presentation; order of pattern presentation was randomized every epoch. The same hit criterion and criterion for a failure to converge was used as described above for integration device networks.

For all sizes of the majority problem, the value unit networks were trained with a learning rate of 0.1 and with no momentum. These parameters were also used for the 2- and 3-bit parity problems. However, with larger versions of the parity problem, this large a learning rate led to dramatic oscillations in the error term being minimized by the learning rule. As a result, as problem size increased we used a smaller learning rate: 0.01 for 4-, 5- and 6-bit parity, and 0.001 for 7-bit parity. In addition to

recording the number of epochs to convergence for the parity problem, we also recorded total sum of squared error to enable comparisons between value unit networks and integration device networks that had failed to converge.

Results & Discussion

Majority Problem

Figure 6 shows that, as predicted, the integration device architecture learned the majority problem significantly faster than did the value unit architecture. For each problem size the integration device architecture's median epochs to convergence was at least two standard deviations lower than the same measure for value unit networks, as indicated by nonoverlapping standard error bars.

The two functions plotted in Figure 6 also illustrate qualitatively different effects of problem size (i.e., the number of input units) on network type. For networks of integration devices, the median number of epochs to convergence monotonically *decreased* as the number of input units increased. At first glance, this result seems counterintuitive. However, recall that to solve each of these problems, an integration device network is only required to correctly position a single hyperplane in the pattern space. For the majority problem, only a relatively small amount of information (i.e., a relatively small number of patterns) is required for this hyperplane to be correctly placed (e.g., Ahmad & Tesauro, 1988). As problem size increases, the amount of information provided to the network per epoch (i.e., the number of patterns) increases exponentially. Thus, it is not surprising that with very large numbers of patterns, fewer epochs are required to provide the network the information required to solve the problem.

The function for networks of value units in Figure 6 is, in contrast to the function discussed above, decidedly nonmonotonic. This nonmonotonicity reflects the fact that this particular curve displays the speed to learn of three different network sizes -- one with no hidden units (for the 2-bit majority problem), one with one hidden unit (for the 3- through 7-bit problems), and one with two hidden units (for the 8- and 9-bit problems). For any one of these network types, it appears that smaller versions of the problem can be solved relatively quickly, presumably because the network has available more than enough computational power. As problem size increases, and network architecture is held constant,

convergence times increase because the problem places more demands on the architecture. This trend continues until problem size reaches a point that the architecture cannot discover a solution without using an additional hidden unit. Once this hidden unit is added, a dramatic decrease in median learning speed occurs, as illustrated by comparing learning speeds for the 2- and 3-bit problems, and by comparing learning speeds for the 7- and 8-bit problems.

Parity Problem

While the results indicated a significant advantage for integration device networks over value unit networks when the majority problem was learned, in general, the reverse was true for the parity problem. However, the comparison between the two types of architectures was less straightforward for this problem than was the case for the majority problem, because of the difficulty in obtaining solutions to the parity problem when the generic architecture was used.[1]

This is illustrated in Figure 7, which indicates the frequency of "failures to converge" obtained over the 25 runs of each architecture on each size of the problem. On the basis of pilot simulations, we operationally defined a "failure to converge" as being a network that did not record a "hit" for each pattern after 20,000 epochs. Figure 7 shows that for the integration device network, once the problem size increased to 5-bits or higher, failures to converge were the norm: 24 of the networks failed to converge for the 5-bit problem, and all of the networks failed to converge for the 6- and 7-bit problems. In contrast, the value unit architecture was able to converge several times for all sizes of the parity problem.

Figure 8 presents the median epochs to convergence for the two architectures for simulations in which correct solutions were achieved. For these runs, the value unit architecture was significantly faster than the integration device architecture, as indicated by the nonoverlapping standard error bars. The functions plotted in Figure 8 also suggest that increasing problem size produces an exponential increase in convergence times for both architectures. However, the rate of growth of the value unit

---

[1] Difficulty in training generic architectures on the parity problem have been reported by other researchers (e.g. Tesauro & Janssens, 1988).

function appears to be substantially less than that of the integration device function.

One problem with comparing speed to converge on the parity problem, and with speculating about data trends as problem sizes increase, is the fact that convergence was extremely rare for the generic networks when problem size increased. A fairer approach to comparing network performance under these circumstances is to examine the total error of the network to the pattern set. Figure 9a illustrates the sum (over the entire pattern set) of squared error for the output unit for both types of network for all 25 runs at each size of problem. This figure indicates that for the smaller versions of the problem, the error for both network types was about the same, which is not surprising as both networks were able to solve the smaller versions of the parity problem. However, as problem size increased to 6 or more bits, the amount of error produced by the integration device networks was significantly larger than the amount of error produced by the value unit networks. Figure 9b presents the same error data as Figure 9a, but does so only for the simulations that failed to converge to a solution. For this select set of simulations, the two networks generate the same amount of error for all problem sizes except the largest. For the 7-bit problem, when the value unit networks failed to converge, their total error to the stimulus set was substantially lower than that generated by the generic networks.

However, one drawback of Experiment 1 was that networks were trained by presenting every pattern required to define the class. This kind of training presents three different problems. First, to the extent that PDP networks are intended to be psychologically relevant (cf. Dawson & Shamanski, 1992), such training is not psychologically plausible -- humans learn categories without being presented every possible instance and noninstance. Second, such training does not examine one of the presumed strengths of artificial neural networks, their ability to generalize what they have learned from a small sample of instances to novel stimuli (e.g, Zurada, 1992, p. 54). Third, as is detailed below, one can plausibly argue that additional architecture by problem-type interactions should be revealed when the ability of a network to generalize to new patterns is measured.

Consider the majority class. In principle, one would expect networks of integration devices to learn this class when only presented a very small number of instances, even if the network has no hidden

units. This is because all the network has to do to learn the classification is position a single hyperplane through the pattern space. As previously mentioned, this should only require a small number of patterns adjacent to the border in the pattern space between members and nonmembers of the class. Indeed, Ahmad and Tesauro (1988) found that network of integration devices learned the entire majority class very quickly when only presented a small number of "border patterns".

In contrast, networks of value units should require a much larger number of instances to learn the majority problem. This is because value units are limited in their ability to utilize information about border patterns. The increased complexity of the value unit receptive field (i.e., the fact that it carves more than one hyperplane) requires the network to receive information about nonborder patterns to correctly place the receptive fields of hidden units. In short, because the value unit must position a number of "nonborder" receptive fields to learn the majority problem, it must receive information from a large number of points away from the class boundary.

The ability of a PDP network to generalize also depends upon the regularity of the pattern space that is being sampled. In particular, one would expect excellent generalization when the pattern space is considered "smooth" in the sense of Marr (1982): namely, neighbouring patterns should belong to the same class. As the majority class only violates this condition at the border between majority and nonmajority members, it is extremely smooth. In contrast, complex classifications like parity are not smooth, because neighbouring patterns usually belong to different classes. For this reason networks generally have extreme difficulty learning such classes, and have poor generalization. For example, Moody and Darken (1989) note that radial basis function networks are poor at learning the parity class due to its pattern space being so irregular. From this, it would be expected that both networks of integration devices and network of value units would have poor generalization for the parity class.

A second set of simulations was conducted to investigate this possibility. Once again, the performance of the two different types of networks on the majority problem and on the parity problem was compared. However, in this case the networks were only trained on 75% of the patterns. The dependent measure of interest was the total sum of squared error of the trained networks on the 25% of

the patterns that had not been presented during training. This experiment was designed to test two hypotheses: (1) Networks of integration devices should generalize better than network of value units for the linearly separable majority problem. (2) Both network types should have poor generalization for the linearly nonseparable, and extremely irregular, parity problem.

Experiment II

Method

Problem Types

The problem types and problem sizes that were studied were the same as in Experiment I. The only difference was that prior to training, 25% of the problems were randomly deleted from the training set. At the end of training, these deleted patterns were presented to the network. This random deletion was performed every time a network was trained on the same problem (to allow later comparisons of average performance). For example, a 7 input unit classification creates 128 different stimuli for presentation to the network. If 75% of the patterns were used as training patterns (96 stimuli), then the remaining 25% of the pattern set (32 stimuli) were used as test patterns to assess the network's ability to generalize to novel instances. In presenting the withheld stimuli after training, a measurement of whether the network had learned the relation desired, or whether the network had learned some subset of the relationship, was assessed. Our dependent measure was the total sum of squared error generated by the network to these novel patterns.

Network Architectures and Training Procedure

The network sizes and training procedures described in Experiment I were also employed in this second experiment. The only exception was the fact that all networks had to be trained to convergence on the 75% of the pattern set used during learning before generalization was tested. Thus, if a network did not converge to a solution on the training set after 20,000 epochs it was discarded from the experiment, and a new network was trained in its place. Unfortunately, we were still unable to collect generalization data for integration devices on the 7-bit parity problems because we could not train a network to convergence. Once the network had learned to correctly classify the training set, the network was then presented the group of test patterns. Each of these patterns was randomly presented to the network, and the network produced an output activity based on the pattern. The network's output and the desired output were compared to determine the network's ability to generalize to these novel test patterns.

Results and Discussion

In examining the ability of the two types of networks to generalize, two different issues must be addressed. The first is the relative ability of the networks: for a particular problem, does one architecture offer significantly better generalization than the other? The second is the absolute ability of the networks: how well do either of the networks generalize in comparison to what would be expected by chance?

To deal with this second issue, let us define an *ignorant network* as some network that has no knowledge whatsoever about a novel set of patterns. In other words, by definition an ignorant network has zero ability to generalize whatever knowledge is encoded in its connection weights to these new patterns. For a pattern set to which binary outputs are to be generated, the expected output (from a statistical perspective) of an ignorant network to one of the novel patterns must be 0.5, because this network can at best be viewed as generating a random output between 1 and 0 for this pattern. From this it follows that the expected value of the squared error for an ignorant network to a single pattern must be 0.25. For any set of N novel patterns, the expected value for the ignorant network's total sum of squared error is therefore $0.25N$.

Figure 10a illustrates the results of Experiment II for the majority problem, giving the actual sum of squared error for the two different types of networks that were trained as well as the expected sum of squared error for an ignorant network. This figure clearly shows that both types of networks were better generalizers than an ignorant network. This figure also clearly shows that when the problem size was greater than 3-bits, the networks of integration devices were significantly better generalizers than networks of value units.

Figure 10b illustrates the results of Experiment II for the parity problem, and presents a markedly different pattern than that found in Figure 10a. First, for all sizes of this problem, the value unit architecture provided significantly better generalization than did the network of integration devices. Second, the network of value units was a poorer generalizer than the ignorant network for the 2-bit parity problem, but then provided better generalization than the ignorant network for all other sizes of the

problem. In contrast, the networks of integration devices were poorer than the ignorant network for all sizes of the problem.

The results of value units being able to generalize better than that of an ignorant network were intriguing. A plausible account of this result is that value units only activate themselves for very narrow ranges of net input, and in all other cases are inactive. Given this it is conceivable that the network is able to generalize well to patterns requiring outputs of 0.0 simply due to their functional properties: for example, probabilistically the chance a value unit will generate an output of 0.0 is much higher than generating 1.0, therefore the units should be very good at generalizing to patterns that require an output of 0.0.

In the case of integration devices, one might question how a trained network could give a performance level less than that of chance. In this case, one must evaluate what responses a network may make to allow such a result. For example, in the ignorant network case an output of 0.5 is assumed for both expected outputs of 0.0 and 1.0, and therefore allows a squared error of .25 for each pattern presented as described above. In contrast, a trained network of integration devices is able to produce any value along the continuum between 0.0 and 1.0 for an output. If the network is being trained to produce a 1.0 for a specific pattern, and the network instead produces an output of .25 giving a squared error for the pattern greater than that of an ignorant network. If such a discrepancy happens for a number of patterns presented to the network, then the network is able to generate a squared error in excess of that created by an ignorant network. The problem of having the observed output pulled towards the opposite expected output is a major contributor to local minima experienced in integration device networks learning the parity problem.

Once again an iteration of problem type and architecture type is revealed. Consider that the integration device networks are able to generalize better than value unit networks, in general, when trained on the linearly separable majority problem. The reverse of this relationship is found when the networks are trained upon the linearly nonseparable parity problem. This adds additional support, in the form of a second dependent measure for the networks, to the concept that each network type is best

suited to one particular type of problem.

General Discussion

A comparison of the generalized delta rule of Rumelhart, Hinton, and Williams (1986a, 1986b) and the modified generalized delta rule for Gaussian units created by Dawson and Schopflocher (1992) has revealed interesting interactions between network type and problem type. In Experiment I it was shown that the logistic architecture was more suited to solving a linearly separable problem than a linearly nonseparable problem. A network of Gaussian units, in contrast, had difficulty solving the linearly separable problem, but was well suited to solving the linearly nonseparable problem. In addition, Experiment II showed that networks of integration devices had better generalization than networks of value units for linearly separable problems, but had poorer generalization for linearly nonseparable problems. So, given two different dependent measures of network performance each architecture has shown itself to be best suited for only one distinct problem type.

What does this evidence tell us about connectionist networks? Does this mean that we should adopt one specific unit type for a network given a particular problem type? What application do these results have for connectionist models in general? What does this research have to offer to cognitive science and psychology as a whole? The sections below attempt to address these questions and provide the bridge that is required between the evidence given above and cognitive science theory.

Engineering Perspective

First, let us consider the results and their implications for PDP modelling in general. For instance, what does this evidence mean to someone constructing a generic connectionist network, whether they be an engineer, computer scientist, or perhaps a cognitive scientist? It has been shown that each activation function is particularly suited to a specific problem type. Given this evidence, a re-evaluation of *a priori* network design is required. Designers must now attempt to understand the nature of the problem to be learned, in order to make an educated decision about which particular architecture to use: i.e. given such and such a problem, you would use this type of network unit. It must be noted that even though a specific architecture performs well on many problems, it may not perform well for *all* types of problems. In essence, PDP modelling must become more open to utilizing activation functions other

than the logistic function (for example, value units, Dawson & Schopflocher, 1992; radial basis functions, Moody & Darken, 1989). Such a perspective raises the possibility of utilizing many different unit types within a network to best solve a particular range of problems (i.e. hybrid systems; Dawson, Schopflocher, Kidd, & Shamanski, 1992). Future research must be conducted to ascertain which activation functions are problem specific and good at only one certain type of problem, which architectures offer more flexibility by performing consistently well on differing problems, and how these different functions may work in tandem to create multi-functional networks.

Psychological Perspective

Models: PDP models have been developed for a diverse range of phenomena, as a survey of almost any journal related to cognitive science will show. For example, in recent years *Psychological Review* has published connectionist models concerned with aspects of reading (Hinton & Shallice, 1991; Seidenberg & McClelland, 1989), classical learning theory (Kehoe, 1988), automatic processing (Cohen, Dunbar, & McClelland, 1991), apparent motion (Dawson, 1991), and dreaming (Antrobus, 1991). In addition, many basic connectionist ideas are being directly implemented in hardware (e.g., Jabri & Flower, 1991) under the assumption that increases in computer power and speed require radical new parallel architectures (e.g., Hillis, 1985; Müller & Reinhardt, 1990, p. 17). "The neural network revolution has happened. We are living in the aftermath" (Hanson & Olson, 1991, p. 332). Such advances have resulted in a growing number of papers by anti-connectionist researchers devoted to the use of non-connectionist cognitive models (Besner, Twilley, McCann, and Seergobin, 1990; Fodor & Pylyshyn, 1988; Broadbent, 1985).

McCloskey (1991) has provided an important recent criticism of the value of connectionist modeling in psychology. McCloskey considers connectionist models to be neither theories, nor even simulations of theories. The reason for this claim is that, in general, connectionists are unable to interpret network structure, and therefore are unable to explain how a network arrives at its solution; no interpretation leads to no understanding. A second point is that PDP models fail to offer concrete and testable theories about the phenomena they describe. McCloskey argues that connectionist models and

simulations may be a stepping stone to the creation of theories, but in themselves fail to create a "workable theory".

Seidenberg (1993) has offered an alternative view of connectionism in a response to McCloskey (1991). Seidenberg considers connectionism to be separable into two distinct forms: (1) descriptive connectionism where a network is designed to describe or account for particular phenomena; and (2) explanatory connectionism which utilizes a small set of concepts that are independently motivated rather than phenomenon-specific to guide the creation of a model. The important difference to be stressed between these two forms of connectionism is that a descriptive model's function is to describe the data, and to show how an outcome may be achieved by a network. In contrast, an explanatory connectionist model can attempt to account for a particular outcome, but in addition, it may also generate other novel predictions about the system's functioning. Furthermore, in explanatory connectionism the model itself (and its building blocks) become legitimate objects of study in their own right.

Given this dichotomy, one would consider the experiments and results above to be a form of explanatory connectionism. Specifically, these experiments were designed to investigate fundamental differences between two connectionist architectures. In turn, the results obtained will allow us to generate new predictions about these networks for further research. In short, the goal of explanatory connectionism is to provide a basis for descriptive connectionism by providing a basic understanding of the abilities of each architecture.

In addition, such abilities can help a descriptive connectionist to make design decisions about their model. For example, consider how a descriptive connectionist would use this information when creating a psychological model of categorization. Experimental results show that humans do not typically find linearly separable classes easy to learn, and in some cases find them more difficult to learn than linearly nonseparable classes (e.g., Medin & Schaffer, 1978; Medin & Schwanenflugel, 1981; Wattenmacher, Dewey, Murphy, & Medin, 1986). This indicates that the value unit architecture may be more appropriate than the integration device architecture for a PDP model. If the explanatory connectionist research on architecture type had not been done, then a descriptive researcher would not

be able to make such a decision. The explanatory research provides the building blocks for the descriptive model.

Neuroscience: Seidenberg (1993) also considers the theoretical principles that connectionist models use to be biologically plausible. In general, artificial neural networks (ANNs) are considered to be functional representations of biological neural network. Only properties that are considered relevant to the model are incorporated. For example, neurons within the nervous system are known to be connected to one another by weighted synapses. ANNs emulate this constraint by joining network units together through the use of weighted connections. An additional example is the property of ANN processing units being homogenous.: i.e. each processing unit in the system uses the same activation function. The reason for this assumption is the predominance this view has gained in the neurosciences, and neural network modeling, over the last 20 years (Getting, 1989). According to this view, a network can be completely understood by determining the anatomy of its connections.

However, Getting (1989) reviews a considerable amount of evidence to suggest that this view is false, and that biological network components are extremely heterogeneous. "No longer can neural networks be viewed as the interconnection of many like elements by some simple excitatory or inhibitory synapses." (Getting, 1989, p.187). Why might biological networks be heterogeneous? Possibly this is because such networks are faced with a wide variety of information processing tasks. The results from Experiment I and Experiment II show that different architectures are better suited to solve different problems. Perhaps the heterogeneity of biological networks provides a sufficient range of architectures to deal with a diverse range of situations.

It should be stressed that the assumptions incorporated into ANNs represent, at best, the limited knowledge of how the actual neural system are believed to operate. In essence, the long term promise made by explanatory connectionism is that these basic artificial neural network principles will eventually evolve into the neurophysiological principles used within biological neural networks.

FIGURES

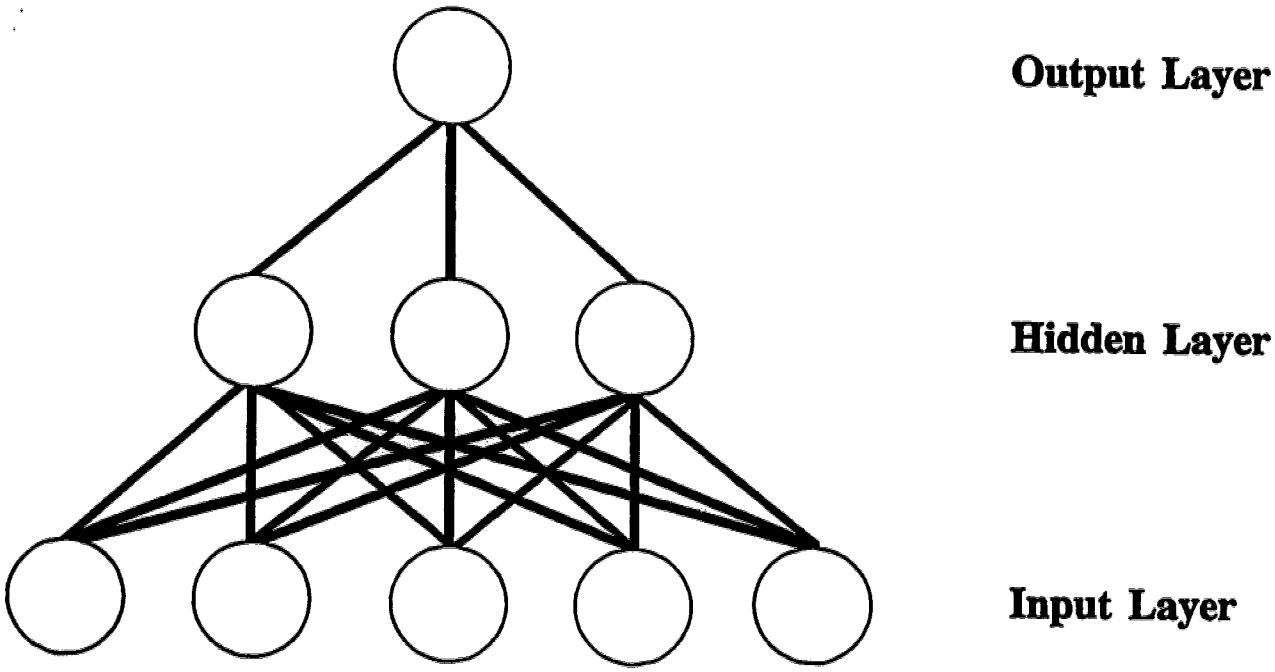
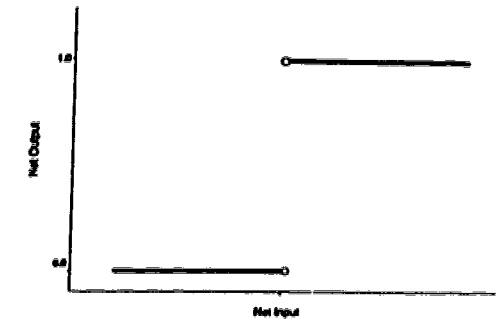
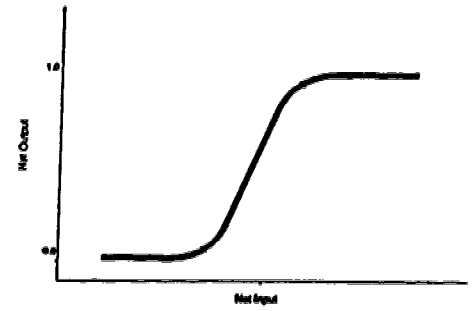


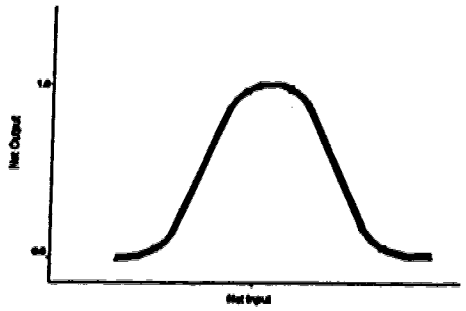
Figure 1 : A multilayer PDP network constructed from massively parallel connections between three layers of processing units.



a

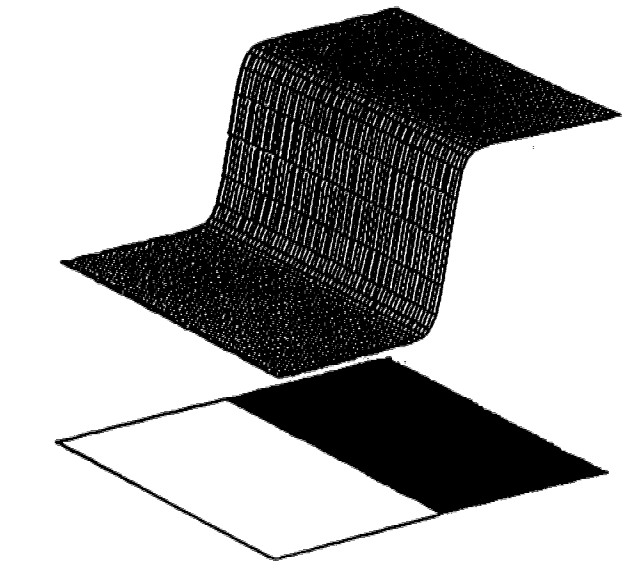


b

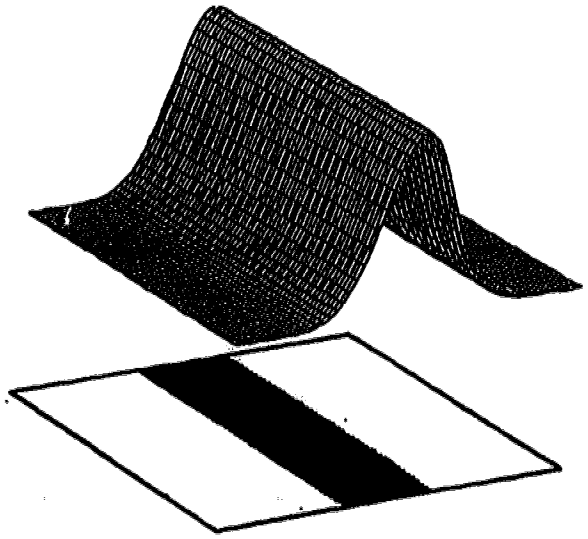


c

Figure 2 : Different network activation functions - (a) the discrete linear threshold function, (b) the continuous logistic function which approximates the linear threshold function, and (c) the continuous Gaussian function.



a



b

Figure 3 : (a) The receptive field carved by a processing unit if it is an integration device. Any input pattern that falls in the dark shadow will turn the processing unit on. (c) The receptive field carved by a processing unit if it is a value unit.

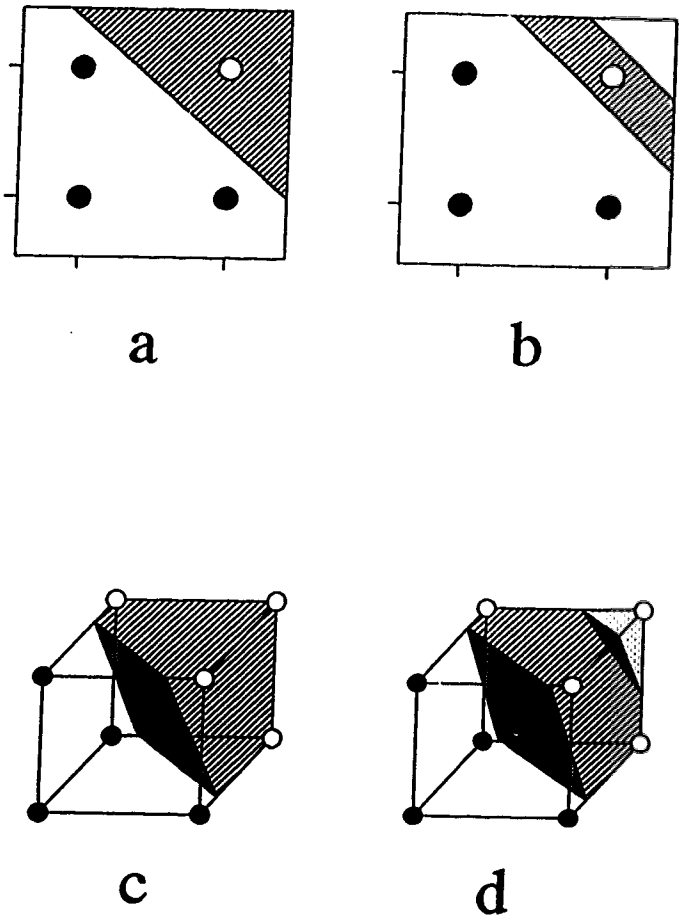


Figure 4 : A comparison of how two architectures might solve different versions of the majority problem. (a) A single integration device isolates the input pattern (1,1) from the other patterns in a two-dimensional version of the problem. (b) A single value unit can also isolate this pattern. (c) For a three-dimensional version of the majority problem, a single integration device can still carve a single plane that separates "ON" from "OFF" pattern. (d) For the larger dimensional majority problem, the limited width of the value unit's receptive field requires that an additional hidden unit be added to capture all of the "ON" patterns. One unit turns "ON" to the patterns that fall between the two planes, while the other unit turns "ON" to the remaining majority pattern that does not fall between the two original planes.

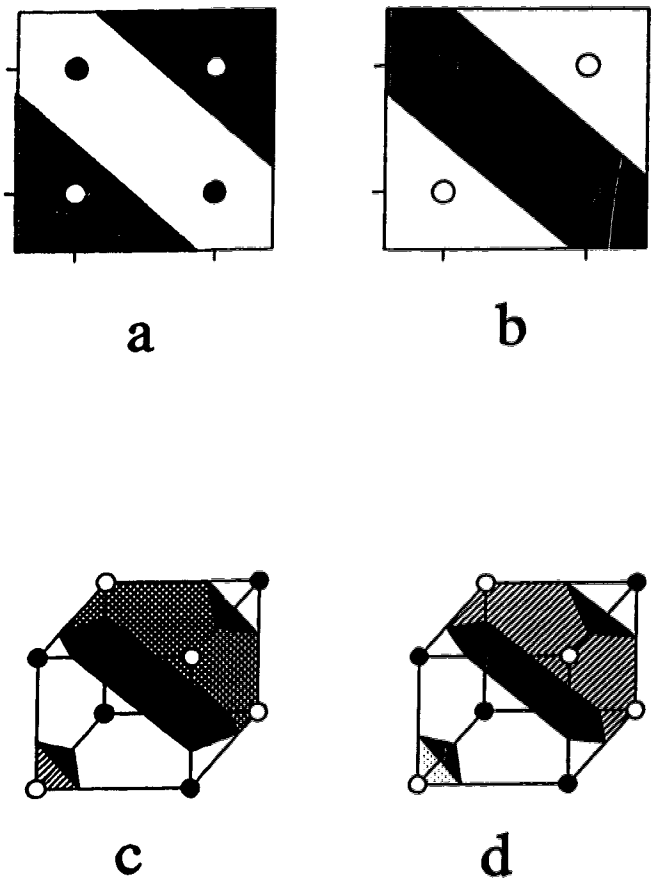


Figure 5 : A comparison of how two architectures might solve different versions of the parity problem. (a) Two integration device isolate the input patterns (0,1), and (1,0) from the other patterns in a two-dimensional version of the problem. (b) A single value unit can also isolate this pattern. (c) For a three-dimensional version of the parity problem, three integration devices are required to carve three different planes to separates "ON" from "OFF" patterns. (d) For the larger dimensional parity problem, the value unit's more complex receptive field allows the network to require only one additional hidden unit to capture all of the "ON" patterns.

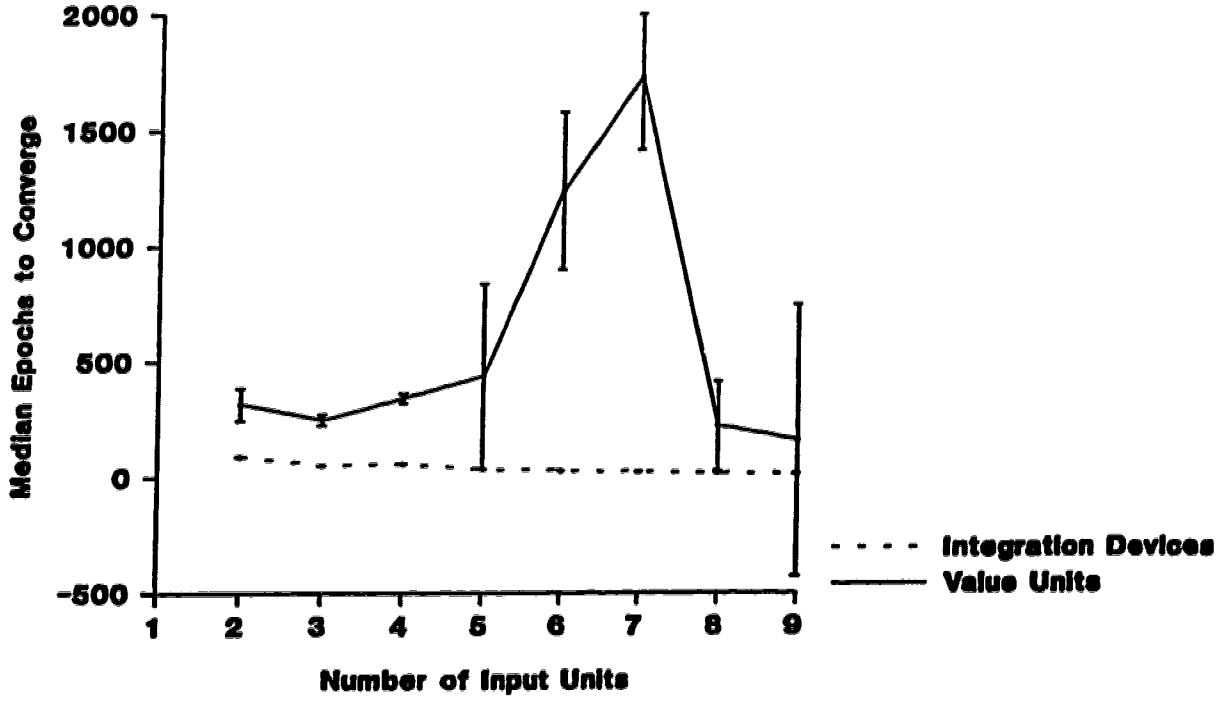


Figure 6 : Median iterations to convergence on the majority problem (with standard errors of the mean) for both architectures.

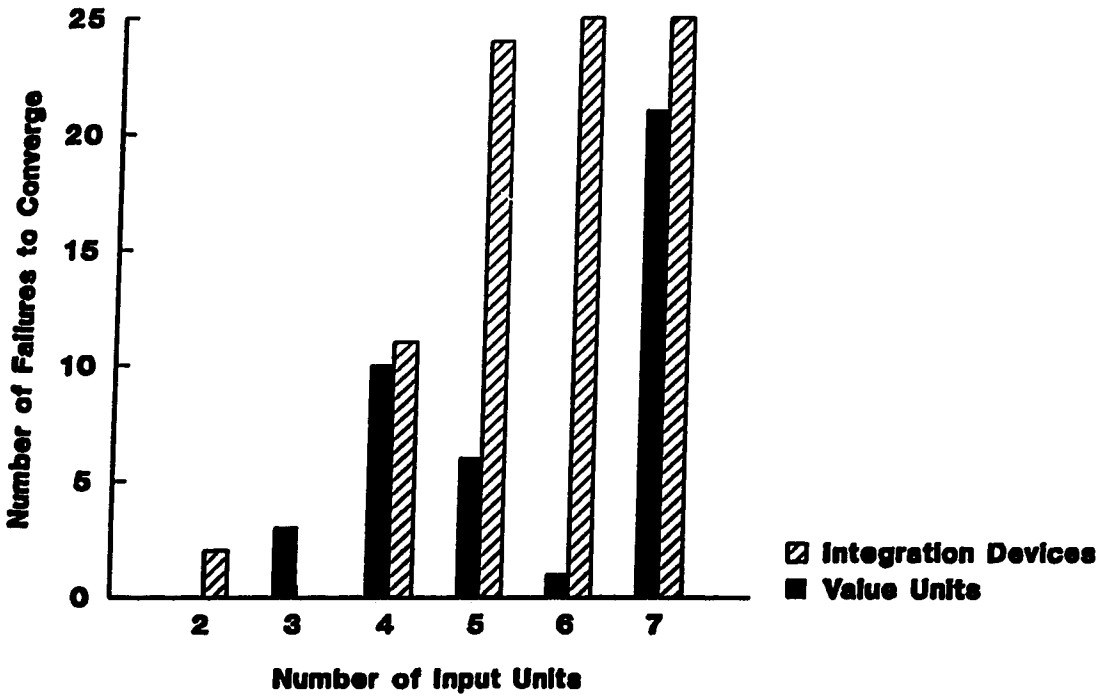


Figure 7 : The frequency of failures to converge, over 25 independent runs, for both architectures on different sizes of the parity problem.

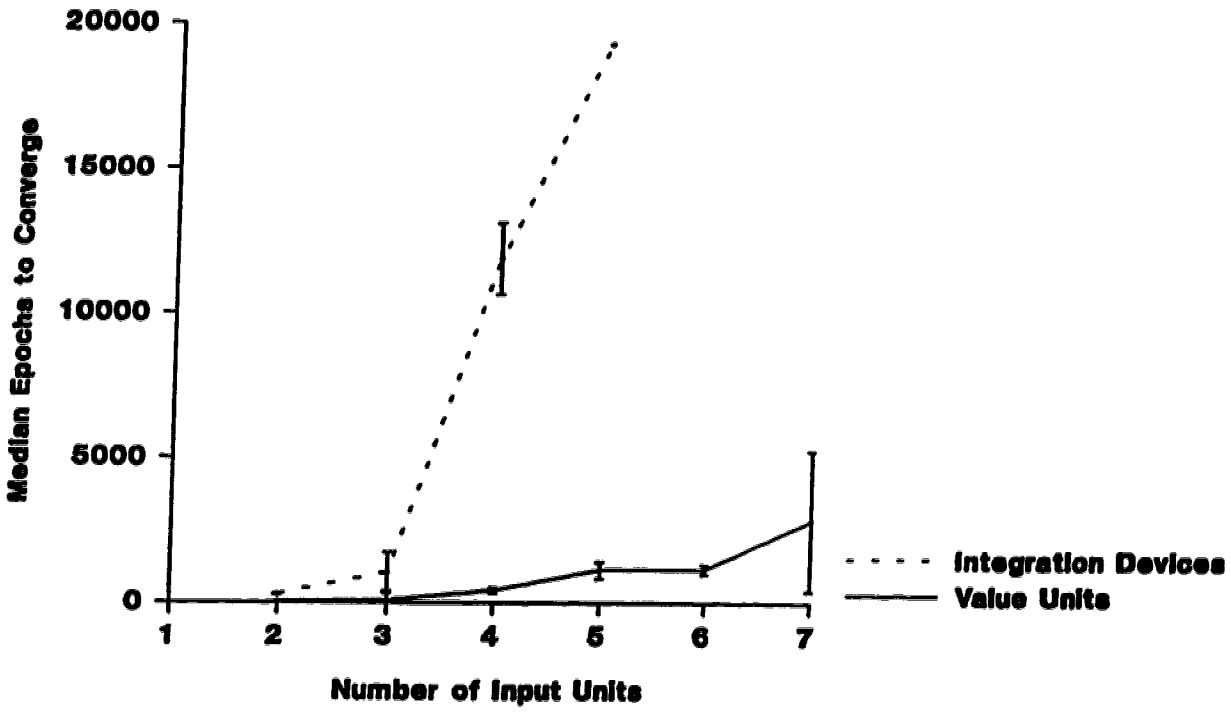


Figure 8 : Median iterations to convergence on the parity problem for both architectures, including simulations which did solve the problem within 20,000 epochs.

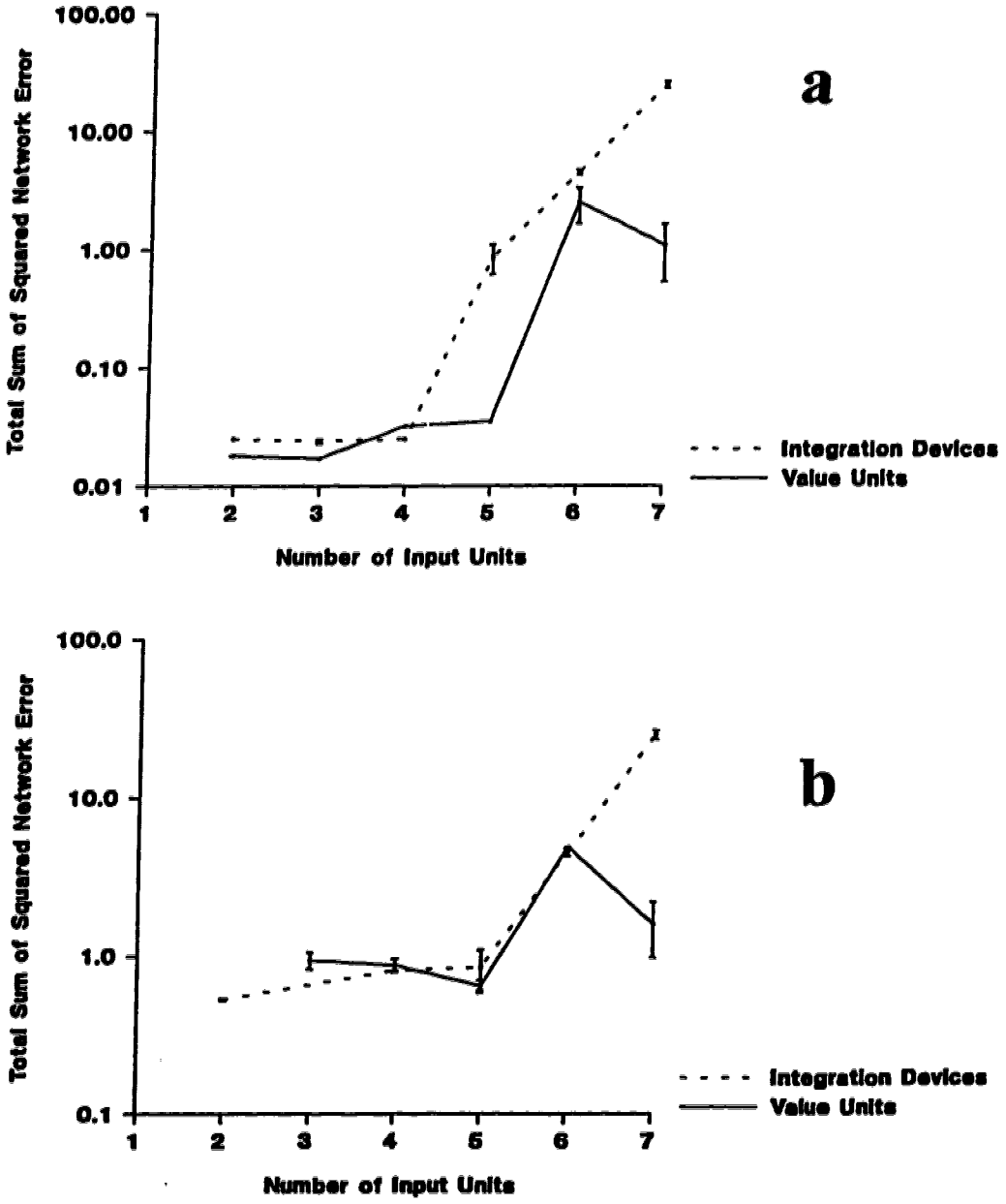


Figure 9 : (a) Total sum of squared error for both architectures on the parity problem using a logarithmic scale. Each data point is a median based on 25 independent runs. (b) Median total sum of squared error for both architectures on the parity problem, including only those runs that failed to converge, using a logarithmic scale.

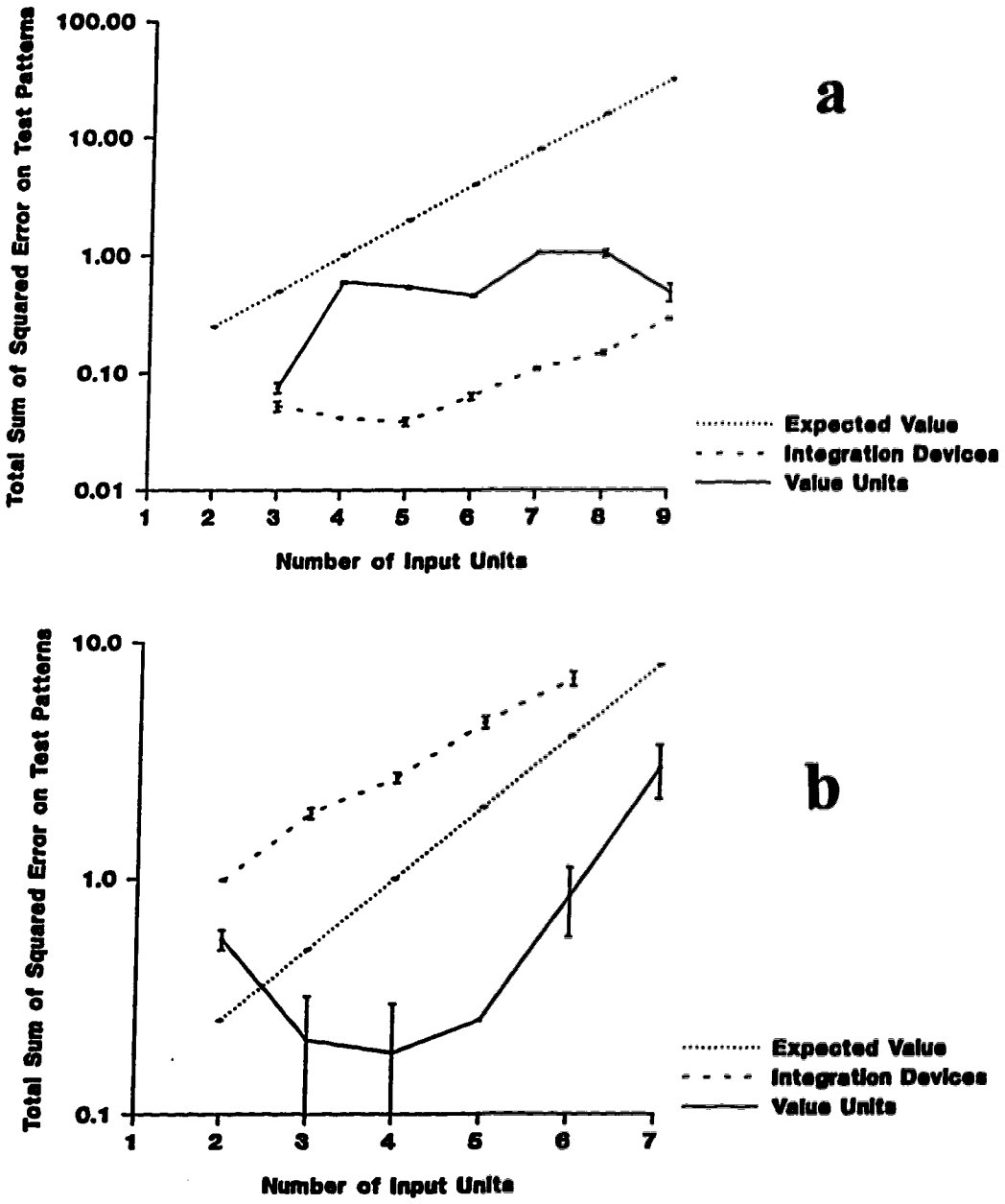


Figure 10 : (a) Median total sum of squared error for networks on the 25% of the patterns not presented during training on the majority problem using a logarithmic scale (b) Median total sum of squared error for networks on the 25% of the patterns not presented during training on the parity problem using a logarithmic scale.

Bibliography

Ahmad, S., & Tesauro, G. (1988). Scaling and generalization in neural networks: A case study. In G.E. Hinton, T.J. Sejnowski, & D.S. Touretzky (Eds.) *Proceedings 88 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann.

Anderson, J.A., & Rosenfeld, E. (1988). *Neurocomputing: Foundations of research.* Cambridge, MA; MIT Press.

Antrobus, J. (1991). Dreaming: Cognitive processes during cortical activation and high afferent thresholds. *Psychological Review, 98,* 96-121.

Ballard, D.H. (1986). Cortical connections and parallel processing: Structure and function. *Behavioural and Brain Sciences, 9,* 96-121.

Barnard, E., & Casasent, D. (1989). A comparison between criterion functions for linear classifiers, with an application to neural nets. *IEEE Transactions on Systems, Man, and Cybernetics, 19,* 834-846.

Besner, D., Twilley, L., McCann, R.J., & Seergobin, K. (1990). On the association between connectionism and data: Are a few words necessary? *Psychological Review, 97,* 432-446.

Broadbent, D. (1985). A question of levels: Comment on McClelland and Rumelhart. *Journal of Experimental Psychology: General, 114,* 189-2.

Cohen, J.D., Dunbar, K., & McClelland, J.L. (1991). On the control of automatic processes: A parallel distributed processing account of the Stroop effect. *Psychological Review, 97,* 332-361.

Dawson, M.R.W. (1991). The how and why of what went where in apparent motion: Modeling solutions to the motion correspondence problem. *Psychological Review, 98,* 569-603.

Dawson, M.R.W., & Schopflocher, D.P. (1992). Modifying the generalized delta rule to train networks of nonmonotonic processors for pattern classification. *Connection Science, 4,* -31.

Dawson, M.R.W., Schopflocher, D.P., Kidd, J., & Shamanski, K.S. (1992). Training networks of value units. In J. Glasgow, & R. Hadley (Eds.) *Proceedings of the Ninth Canadian Artificial Intelligence Conference,* Palo Alto, CA: Morgan Kaufman Publishers Inc.

Dawson, M.R.W., & Shamanski, K.S. (1992). Connectionism, confusion, and cognitive science. *Journal of Intelligent Systems*, in press.

Dawson, M.R.W., Shamanski, K.S., & Medler, D.A. (1993). From connectionism to cognitive science. In L. Goldfarb (Ed.) *Proceedings of the Fifth University of New Brunswick Artificial Intelligence Symposium* (pp245-305). Fredericton, NB: UNB Press.

Fodor, J.A., & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition, 28,* 3-71.

Getting, P.A. (1989). Emerging principles governing the operation of neural networks. *Annual review of neuroscience, 12,* 185-204.

Hanson, S.J., & Olson, C.R. (1991). Neural networks and natural intelligence: Notes from Mudville. *Connection Science, 3,* 332-335.

Hartman, E., & Keeler, J.D. (1991). Predicting the future: Advantages of semilocal units. *Neural Computation, 3,* 566-578.

Hinton, G.E., & Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review, 98,* 74-95.

Hillis, W.D. (1985). *The connection machine.* Cambridge, MA: MIT Press.

Jabri, M., & Flower, B. (1991). Weight perturbation: An optimal architecture and learning technique for analog VLSI feedforward and recurrent multilayer networks. *Neural Computation, 3,* 546-565.

Kehoe, E.J. (1988). A layered network model of associative learning: Learning to learn and configuration. *Psychological Review, 95,* 411-433.

Levitan, I.B., & Kaczmarek, L.K. (1991). *The neuron: Cell and molecular biology.* New York, NY: Oxford University Press.

Lippmann, R.P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine, April,* 4-22.

McCloskey, M. (1991). Networks and theories: The place of connectionism in cognitive science. *Psychological Science, 2,* 387-395.

Marr, D. (1982). *Vision.* San Francisco, CA: W.H. Freeman.

Medin, D.L., & Schafer, M.M. (1978). Context theory of classification learning. *Psychological Review, 85,* 207-238.

Medin, D.L., & Schwanenfluegal, P.J. (1981). Linear separability in classification learning. *Journal of Experimental Psychology: Human Learning and Memory, 7,* 355-368.

Minsky, M., & Papert, S. (1988). *Perceptrons, 3rd Edition.* Cambridge, MA: MIT Press.

Moody, J., & Darken, C.J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation, 1,* 281-294.

Müller, B., & Reinhardt, J. (1990). *Neural networks.* Berlin: Springer-Verlag.

Newell, A., & Simon, H. (1981). Computer science as empirical inquiry. In J. Haugeland (Ed.) *Mind Design.* Montgomerty, VT: Bradford Books.

Rumelhart, D.E., McClelland, J.L., & the PDP Group (1986). *Parallel Distributed Processing, V.1.* Cambridge, MA: MIT Press.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986a). Learning representations by back-propogating errors. *Nature, 323,* 533-536.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986b). Learning internal representations by error backpropogation. In D. Rumelhart , J. McClelland, & the PDP Group (Eds.) *Parallel Distributed Processing V.1.* Cambridge, MA: MIT Press.

Seidenberg, M.S. (1993). Connectionist models and cognitive theory. *Psychological Science, 4,* 228-235.

Seidenberg, M.S., & McClelland, J.L. (1989). A distributed, developmental model of word recognition and naming. *Psychological Review, 96,* 523-568.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioural and Brain Sciences, 11,* 1-74.

Tesauro, G. & Janssens, B. (1988). Scaling relationships in back-propogation learning. *Complex Systems*, 2, 39-44.

Wattenmaker, W.D., Dewey, G.I., Murphy, T.D., & Medin, D.L. (1986). Linear separability and concept learning; Context, relational properties, and concept naturalness. *Cognitive Psychology, 18,* 158-4.

Zurada, J.M. (1992). *Introduction to artificial neural systems.* St. Paul, MN: West Publishing Company.