1-1-1987

Learning Object-Centered Representations

Peter Anthony Sandon
Dartmouth College

LEARNING OBJECT-CENTERED PRESENTATIONS

Peter A. Sandon

Technical Report PCS-TR88-139

LEARNING OBJECT-CENTERED REPRESENTATIONS

by

PETER ANTHONY SANDON

A thesis submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN — MADISON

1987

This work is dedicated to

Maureen Hervey Sandon

Acknowledgements

I am happy to have the opportunity to thank a number of people who have influenced me during my five years in Madison. My advisor, Len Uhr, has spent many hours discussing issues of learning and vision with me. It was he who convinced me of the importance of learning in models of intelligent behavior. I thank him for his support and advice on this research, and for his careful reading of the dissertation. Chuck Dyer introduced me to the field of computer vision. I have benefited from the many discussions we have had on vision and machine intelligence. I thank him for sharing his views on these topics of mutual interest, and for his many useful comments on the dissertation. Gregg Oden's course on knowledge representation sparked my interest in connectionist modelling. Gregg has helped me to understand a number of issues involved in connectionist and non-connectionist forms of representing knowledge. I thank him for his several worthwhile courses, his encouragement of my work, and his insights on important issues. I would also like to thank two other members of my dissertation committee, Larry Travis and Lola Lopes, for their encouragement and helpful discussion of the work reported here.

I have been influenced by people in a number of discussion groups over the past few years. I thank my fellow advisees, Seng-Beng Ho, Ze-nian Li and Ayse Unat, for their friendship and for the stimulating discussions we shared. I would also like to thank the members of the AI seminar group, the vision group, and the CALM discussion group for many beneficial interactions. In addition, I would like to thank

the organizers and fellow participants in the 1986 CMU Connectionist Models Summer School, where I received a cram course on the who, what and why of connectionism.

Finally, the support of my wife, Maureen, and of both our families, has provided needed encouragement during a time that has been both exciting and difficult. I thank them all for their part in seeing this work to its conclusion.

Peter A. Sandon

August 1987

Abstract

LEARNING OBJECT-CENTERED REPRESENTATIONS

Peter Anthony Sandon

Under the supervision of Professor Leonard Uhr

When we look at a familiar object from a novel viewpoint, we are usually able to recognize it. In this thesis, we address the problem of learning to recognize objects under transformations associated with viewpoint. Our vision model combines a hierarchical representation of shape features with an explicit representation of the transformation. Shape features are represented in a layered pyramid-shaped subnetwork, while the transformation is explicitly represented in an auxiliary subnetwork. The two connectionist networks are conjunctively combined to allow object-centered shape features to be computed in the upper layers of the network. A simulation of a 2-D translation subnetwork demonstrates the ability to learn to recognize shapes in different locations in an image, such that those same shapes can be recognized in novel locations.

Two new learning methods are presented, which provide improved behavior over previous backpropagation methods. Both methods involve competitive interactions among clusters of nodes. The new learning methods demonstrate improved learning over the generalized delta rule when applied to a number of

network tasks.

In the first method, called error modification, competition is based on the error signals computed from the gradient of the output error. The result of this competition is a set of modified error signals representing a contrast enhanced version of the original errors. The error modification method reduces the occurrence of network configurations that correspond to local error minima.

In the second method, called error augmentation, competition is based on the activations of the nodes in the cluster. Network changes resulting from this competition augment those specified by the error gradient computation. This competition is implemented by the trace comparison rule, a new self-organizing mechanism that is effective in developing highly discriminating features within the cluster. The error augmentation method improves learning in the lower network layers when backpropagated error is weak.

Table of Contents

Chapter 3 - Learning in Fine-grained Parallel Networks

Chapter 4 - Learning Experiments

Chapter 1

Introduction

When we look at a familiar object from a novel viewpoint, we are usually able to recognize it. In a computational model of vision, we must specify how the information in the image is represented to allow this recognition process to occur. In addition, we must specify some means of acquiring such representations, either through direct programming in the case of a restricted domain computer vision system, or through an automated knowledge acquisition process, in the case of biological vision and adaptive computer vision.

In this thesis, we concern ourselves with adaptive computer vision systems. In particular, we are interested in developing a model of vision which can efficiently represent the invariant information required to recognize objects from various viewpoints, and which is capable of acquiring this information through experience. In the work described here, we will restrict our vision model to consider only the shape of an object in attempting to identify it. In representing shape features, it is efficient to decompose shapes into those sub-shapes that are shared among many shapes. A hierarchical representation of the shape features of an object is used in the model developed here. This decomposition allows the detection of many primitive shape features to occur simultaneously. Similarly, shape features composed of these primitives can be detected in parallel once the primitives are detected. This suggests

a fine-grained parallel layered structure for the vision model. The model is implemented in a connectionist network, in which nodes, singly or in groups, represent shape features and links represent evidential relations among shape features.

In developing the set of shape features to be used in the recognition process we use a learning by example approach. That is, from presented examples of a shape to be recognized and information identifying which shape is being presented, the network acquires an appropriate hierarchical representation of the shape. The two major aspects of this work are the development of this highly parallel model of viewpoint-invariant vision, and the development of network learning algorithms that are sufficient for the vision task to be learned from examples.

1.1. Viewpoint-invariant vision

Vision involves the extraction of useful information from an array of light intensities. Machine vision is just one sub-area of the more general field of artificial intelligence (AI). The study of visual processes is particularly relevant to AI because it has much in common with the more general study of cognitive function [Witkin & Tenenbaum 1983]. Many of the processes involved in the perception of visual stimuli, such as pattern matching, search, deduction and planning, have counterparts in cognitive processing. Like other forms of perception, the acquisition of visual knowledge and processing of visual data is a subconscious component of the cognitive repertoire. As such, human visual perception is an ability taken for granted

by the majority who possess it.

The problem of viewpoint invariant vision can be stated as follows: The shape of an object as seen from an arbitrary viewpoint is some rigid-body transformation of visible shape features of some canonical shape of the object. Given a particular shape in an image, the vision system must recognize the object in the image as an instance of the canonical shape.

One approach to obtaining invariant features from an image is to extract features that are insensitive to the transformation. Any feature that does not change under transformation can be compared directly with the corresponding feature in a canonical description of the object. This leads to a straightforward pattern matching scheme for recognizing objects. Use of Fourier descriptors and chain coded boundaries are examples of this approach. Another approach to obtaining invariant descriptions is to extract features that are sensitive to the transformation. Invariant features are then computed by inverting the transformation. This inversion procedure requires that the transformation information be made explicit.

Two related models that have been described in recent years [Hinton 1981, Ballard 1984] attempt to cooperatively identify the transformation and recognize the canonical shape simultaneously. A partial identification of the transformation can be used to constrain the possible interpretations of the object shape and vice versa. In this work, we combine the representation of transformation information suggested in the Hinton and Ballard models with the hierarchical representation of shape used by

Uhr [1972]. A major constraint on the resulting model is that it be amenable to learning of both shape and transformation.

Vision involves the processing of large amounts of information in a short period of time. A single image of moderate resolution contains millions of bits of data. A real-time processing task might require thirty such images to be processed per second. For these reasons, various special purpose computer architectures with a high degree of fine-grained parallelism have been developed to implement machine vision systems. One class of highly parallel architectures is referred to as connectionist models.

1.2. Connectionist models

There are two different models of machine intelligence that are currently receiving much attention. The first is based on the von Neumann serial stored program model of computation. In this model, knowledge is represented in data structures and procedures that are stored in a central repository. Knowledge is accessed through indices and addresses that allow many levels of indirection. One or more action sequences executed by central processing elements operate on this knowledge to produce appropriate behavior by the system. The process of acquiring new knowledge involves adding new data structures and procedures, or modifying current data structures and procedures, and providing access to this new knowledge.

The second model is based on some of the properties of computation in biological systems. Here, both processing and stored knowledge are decentralized at

a fine granularity. Each simple processing element can access only a small fraction of the knowledge base. It is the coordinated processing of all elements that allows all of the knowledge to be used in producing appropriate system behavior. In this model, knowledge acquisition involves modifications to the small amount of data associated with each of the large number of processing elements.

The first type of model is that conventionally studied in AI. It is characterized by serial computation and symbolic processing. The current emphasis in the use of these models is a so-called strong approach, based on domain-specific knowledge-intensive computation. Expert systems, for instance, are programs consisting of a large database of facts and rules pertaining to some limited domain of application. The goal in developing such a program is to produce behavior that mimics that of an expert, by making efficient and effective use of the knowledge contained in the database.

The second type of model, due to its dominant architectural feature, is often referred to as a connectionist model. It is characterized by massively parallel computation using numerical data. Connectionist models emphasize the combining of detailed evidential knowledge in intelligent decision making. For instance, in classifying written words using visual features, the individual characteristics of the component letters, as well as letter pair frequencies and the identities of surrounding words might all be brought to bear on the identification of a given word.

The usefulness of these models is a popular topic of debate, not only in the field of artificial intelligence, but within all other areas of cognitive science. A common theme in the debate is the disparity between the low-level representations of the connectionist networks and the high-level processing required for many cognitive tasks. Those who study high level, conscious cognitive behavior, such as natural language understanding or problem solving, may find a serial, symbolic approach to intelligence appropriate. For lower-level cognitive functions, like perception, the fine-grained parallelism of connectionist networks is appropriate. Our view is that a synthesis of these two approaches will develop as we gain a better understanding of each.

The following sub-sections provide some background information on connectionist models. The processing element and network architecture descriptions are assumed in later chapters. The history is provided to put the current interest in these models into perspective.

1.2.1. Architecture

Connectionist models are characterized by fine-grained massively parallel computation. A connectionist network consists of a large number of simple processing elements linked together by weighted connections. Specification of a particular network requires a definition of the computation performed by a single element and a description of the connectivity pattern that links each element to others in the network. For an adaptive network, an algorithm for adjusting connection

weights must be specified as well.

Each processing element (PE) computes a single scalar output value that is some simple function of its inputs (see Figure 1.1). An m input PE has $m+1$ *weights* associated with it. One weight is associated with each input, and a *bias* weight is treated as if associated with a constantly active input. The *net* input to the PE is typically computed by taking the weighted sum of all inputs:

$$net = \sum_{i=0}^{m} w_j i_j$$

The output is computed by applying some *activation function*, f, to this net input:

$$o = f(net)$$

Three commonly used activation functions define the linear, threshold and logistic processing elements:

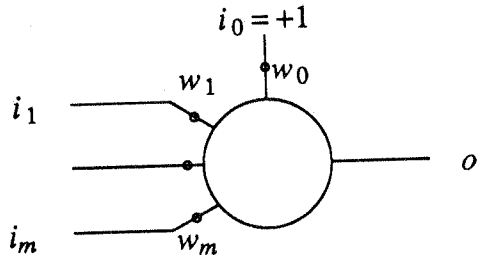


Figure 1.1

$$f(net) = net \qquad \textit{linear element}$$

$$f(net) = \begin{cases} 0 & net < 0 \\ 1 & otherwise \end{cases} \qquad \textit{threshold element}$$

$$f(net) = \frac{1}{1 + e^{-net}} \qquad \textit{logistic element}$$

The linear element is the simplest to use when mathematically characterizing a network. It is, however, limited in applicability to single layer networks, since any multi-layer network of linear elements can be reduced to a single layer equivalent network. The threshold element is a discrete decision making device. It classifies all input patterns into two sharply distinguished sets. The logistic function is a continuous version of the threshold element (see Figure 1.2 for its input-output characteristic). One advantage of using a logistic function is that it enhances contrast between patterns in a graded fashion. Another is that it is everywhere differentiable, a property required by some learning algorithms (see section 3.2.4).

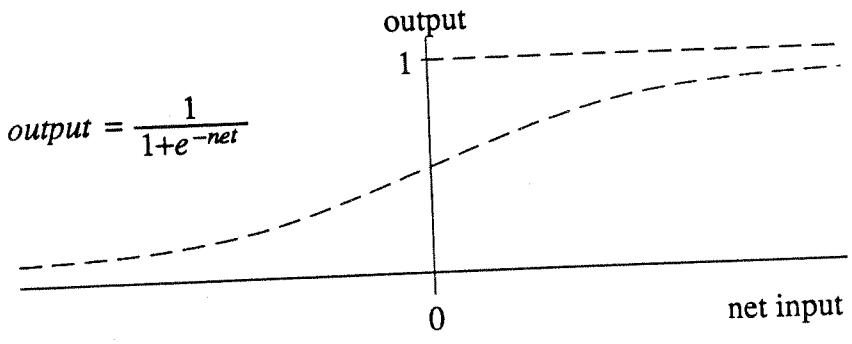


Figure 1.2

Complex functions are represented by connecting many processing elements together into a network such that the output of one PE provides an input to many other PEs. Since each element computes its output value independently of computations by many of the other elements, a high degree of parallelism can be achieved.

The elementary function performed by a processing element is a simple kind of pattern matching. The weighted sum of inputs to a given node will be maximized by an input pattern that correlates exactly (and positively) to the corresponding weights. This weighted sum will decrease as the correlation between inputs and weights diminishes.

Figure 1.3 shows a single linear PE taking inputs from a 3×3 subarray of an image. The weight vector **w** is shown in registration with the image input. For input i_1 the PE computes an output value ($o = net$) of -2, indicating a poor match between

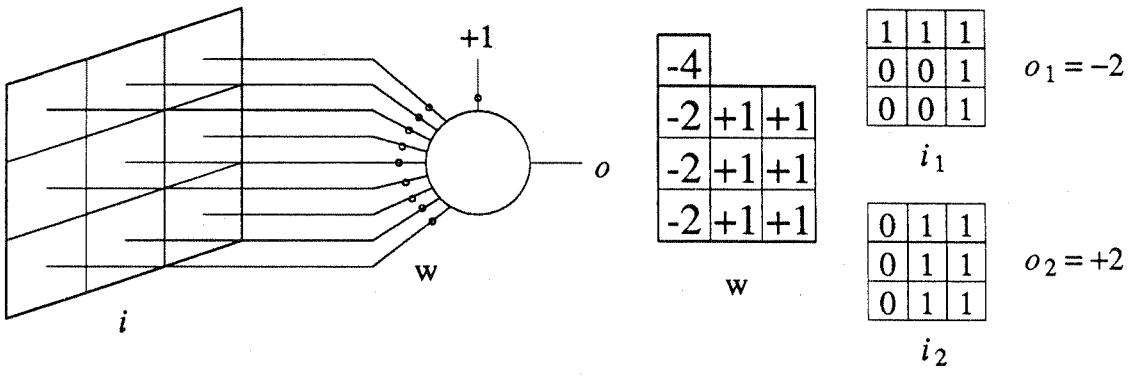


Figure 1.3

weights and input pattern. For input i_2 the PE computes an output value of +2, indicating a strong match. This PE detects vertical edges in the intensity values of the image elements within its receptive field.

By compounding many of these elementary pattern matching processes together, a complex pattern matcher can be constructed. For instance, Figure 1.4 shows schematically how a layered network can be used to detect complex pattern features by combining the outputs of simple feature detectors. By combining evidence in a graded manner from many feature detectors at the layer below, a robust feature can be represented. Missing evidence due to noise or occlusion can be compensated by using a redundant set of features and detecting closeness of a match rather than an exact match. This pattern matching behavior can be used as a basic component not only of perceptual functions such as visual pattern recognition, but of higher cognitive functions such as language understanding as well.

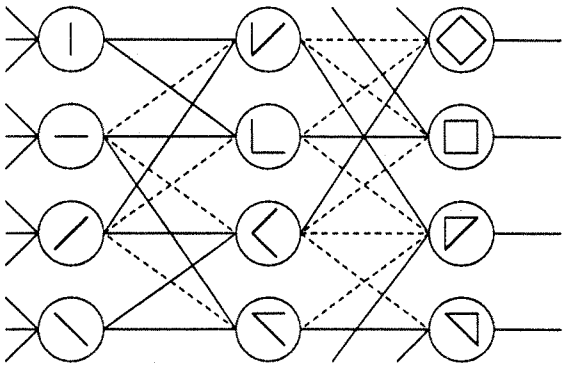


Figure 1.4

For a particular PE, the relationship between its output and its inputs depends on the connection weights. Thus the *knowledge* in the network, which determines what function is performed, is contained in the weights. An adaptive connectionist network must have the capability to modify its knowledge base, that is, its connection weights. Learning algorithms for these networks base the modification of weights on the correlations between the activity value of the corresponding input, and some measure of the current network behavior. For unsupervised forms of learning, the input and output activities of each node are correlated. In the case of error-correction type learning algorithms, network error values are used to drive the modification of weight values.

Figure 1.5 shows a typical processing element with an explicit indication of one of the element inputs, i_j. The feedback to the element, t_k, is a value indicating to what degree the PE response to the current input is correct. This feedback may be provided by an external training agent, or may be developed internally through an estimate of the desired behavior of the element. A typical rule for adjusting the connection weight shown in the figure has the form:

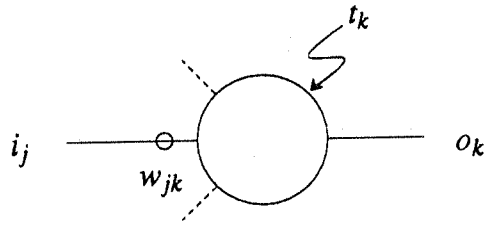


Figure 1.5

$$\Delta w_{jk} = \eta\, i_j\, \delta_k(o_k, t_k)$$

where η is a constant of proportionality called the learning coefficient, and δ_k is an error term that is a function of the feedback and current output. Learning algorithms are discussed in greater detail in Chapter 3.

We summarize this description of connectionist models by listing their important characteristics. First, these networks exhibit a large-scale fine-grained parallelism. This allows a detailed representation of knowledge while providing fast access to that knowledge. Second, each processing element computes a very simple function of its inputs. It is the parallel execution of a large number of these individual computations that provides the high level of processing power. Third, the knowledge of the network is contained in the weights. This distributed form of knowledge representation makes the entire knowledge base accessible to active processors at all times. Fourth, computation at each processing element is local. An element has access to only a small part of the knowledge of the system, and can only handle a small part of the overall task being performed. It takes the interactions of a large number of such local computations to provide the global result required by the task. Fifth, learning is an iterative process based on individual examples. This use of on-line learning methods gives the network the potential to adapt to unforeseen changes in the environment, and provides a means of initially acquiring knowledge.

1.2.2. A history of connectionist modelling

The study of massively parallel models of intelligence goes back at least to 1943, when McCulloch and Pitts published their influential paper describing sufficient conditions on their neuron to support the desired characteristics for a unit psychic event, or psychon [McCulloch & Pitts 1943]. McCulloch later described the importance of that paper as follows [McCulloch 1965]:

> Turing had produced a deductive machine that could compute any computable number, although it had only a finite number of states, and although it could move only a finite number of steps forward or backward, look at one spot on its tape at a time, and make, or erase, 1 or else 0. What Pitts and I had shown was that neurons that could be excited or inhibited, given a proper net, could extract any configuration of signals in its input. Because the form of the entire argument was strictly logical, and because Gödel had arithmetized logic, we had proved, in substance, the equivalence of all general Turing machines -- man-made or begotten.

This was an era when mathematicians such as Gödel, Turing and von Neumann were studying the properties of automata and the computability of functions. McCulloch and Pitts were interested in revealing the properties of functions that could be computed using their neural model. In analogy to the theory of computable functions, their work might be characterized as developing a theory of thinkable and unthinkable thoughts.

The next important step in the study of neural networks came in 1949, when Hebb presented his theory of how the physiology of the brain supports observable behavior. [Hebb 1949]. Hebb suggested that learning is based on the modification of synapse strengths, according to a rule now called the correlational synapse. For each synapse between two neural elements, this rule specifies a change in strength that is

proportional to the correlation between the activity levels of the two elements. The correlational synapse is a simple, neuron-level mechanism which results in the sensitization of neurons to recurring patterns in the network. Whereas McCulloch envisioned memory as resulting from closed loops of neural connections, Hebb's idea was that knowledge is retained in the connectivity of the neurons. Thus, by adjusting those connections as a function of input patterns, those patterns can be learned and subsequently recognized by the network when they recur.

During the 1950's the field of AI began to develop along with computer science in general. Early papers by Turing, describing his famous 'intelligence test' [Turing 1950], and by Shannon, describing an approach to computer chess playing [Shannon 1950], set the tone for future work. By the late 1950's, symbolic processing was identified as the key to implementing high-level cognitive processes in machines. This led McCarthy to develop LISP as a symbol manipulation language with which intelligent programs could be written [McCarthy 1960, McCarthy 1978].

During this time, Samuel [1959] used a checkers playing program as a vehicle for the study of learning. Samuel's work is interesting to study as it relates to the computer technology of the time. His investigation of heuristic functions and of the convergence of learning procedures and the hill climbing problem were revealing and useful for future work. However, in reading his papers, it becomes obvious that Samuel spent a good deal of his time solving the systems problems of efficient access to and storage of large amounts of data. This work emphasizes the dependence on

current and predicted technology of prevailing theories of intelligence.

Others in the late 1950's continued to pursue the non-symbolic neural models of intelligence [Selfridge 1959, Rosenblatt 1957,1962]. Complementing this work was that of the neurobiologists [Lettvin, et. al. 1959, Hubel & Wiesel 1962] who were discovering meaningful functions of sets of neurons in the visual paths of frogs and cats. The primitive feature detectors found in these biological systems inspired similar networks for detection of visual features by machine. Rosenblatt's perceptrons (described in section 3.2.1) resembled the earlier neural models in having an all-or-none output determined by the weighted sum of inputs and a threshold. However, the algorithm for adjusting weights of inputs was a supervised one, in which the response of the network was used to determine how weights should be adjusted. Thus, rather than the local correlational synapse procedure, a global determination of the effectiveness of connection strengths was made, and the connections modified accordingly. In fact, the algorithm was proven to be capable of producing a satisfactory set of weights for correctly executing a given pattern discrimination task, in cases where such a set of weights existed.

Exaggerated claims regarding the importance and power of the perceptron model led to heated debate on the subject, which culminated in Minsky and Papert's criticism of the model. [Minsky & Papert 1969]. This algebraic and geometric analysis of perceptron theory was taken by many to disprove the usefulness of perceptrons once and for all. Some debate followed as to whether the elegant

mathematics of the book did indeed demonstrate the inadequacies of the general theory of perceptrons [Block 1970], but the result was a very marked turning away from such models for some years afterward.

Meanwhile, the more symbolic approaches to AI were progressing, with the feeling that intelligent machines were just around the corner. The major areas of research were in natural language understanding, perception with an emphasis on vision, problem solving spurred on by chess playing, following Shannon's lead, and representation of knowledge. Learning was studied very little during this time.

The serial computer of the 1960's was used to implement the algorithms to solve the various AI problems. Processor and device speeds, as well as storage capacities, restricted the performance of programs, motivating the search for shortcuts. Heuristics became increasingly popular as performance enhancers, despite a general lack of theory in their use.

During the 1970's, parallel models of computation gained popularity for many applications areas. Having long recognized a need for large amounts of computing power, AI researchers, and in particular vision researchers, have been at the forefront in trying to apply a parallel processing model to the solutions of their problems. One particular architecture that has been found useful for image processing is the highly parallel, fine-grained array processor [Duff 1976, Flanders, et. al. 1977, Batcher 1980]. Typically, such a design will consist of a large two dimensional array of simple processing elements, each connected to the four other processing elements

nearest to it. Each processor is capable of only simple calculations on a small subset of the image data, but due to the large number of such processors, a great deal of computational power is realized. A pyramid processor is an extension of the array processor, in which layers of processing arrays are used to represent hierarchical data structures. Like the array processor, the pyramid model is useful for machine vision applications. Many other topologies have been suggested as massively parallel computational models for various applications. The Connection Machine [Hillis 1985] is one example that combines the array structure and an n-cube structure in a single system.

In the 1980's, the advent of these massively parallel models of computation has been responsible in part for the renewed interest in massively parallel models of intelligence. For those intent on building intelligent machines, advances in VLSI technology are making feasible the implementation of these computational models in hardware. For those interested in modelling human intelligence, a model that is based on the structure of the brain potentially provides a link to the neurobiological substrate of behavior.

In recent work, these models have been applied to problems in vision [Nass & Cooper 1975, Ballard 1984, Feldman 1985, Sabbah 1985], speech [Sejnowski & Rosenberg 1986, McClelland & Elman 1986], representation of semantic knowledge [Hinton 1981, Shastri & Feldman 1984, Ackley et. al. 1985], language [McClelland & Rumelhart 1981, Cottrell 1985, Grossberg & Stone 1986], problem solving

[Touretzky and Hinton 1985] and learning [Sutton and Barto 1981, Feldman 1982, Rumelhart et. al. 1986].

To summarize, the early work on neural networks as models of intelligent behavior is today being reconsidered and extended for a number of reasons. First, advances in VLSI technology are making fabrication of highly parallel networks feasible. Second, serial computation appears inadequate to process the large amounts of data in the short periods of time that are required to support intelligent behavior. Third, advances in neurobiology are providing clues to the design of these networks. And finally, as the limitations of the early models become better understood, the means of avoiding these limitations are being found.

1.3. Learning

Learning is an important component of intelligent behavior. One reason is the necessity for an intelligent system to adapt to a changing environment. Even within a fixed environment, learning mechanisms are useful, if not necessary, in automating the acquisition of knowledge of that environment. As the study of machine intelligence proceeds, the increasing complexity of the models of intelligent behavior makes it imperative that knowledge acquisition be automated. This is particularly true in neural models of behavior, where knowledge is represented at a fine-grain of detail.

The representation of visual information in a hierarchical, layered structure is advantageous for the parallel implementation of object recognition (see section

2.2.4). One difficult problem that this structure presents to a learning algorithm is that of identifying which part of the representation to change when new information is made available. This credit assignment problem is particularly difficult when each change must be based on a separate training experience, and when the change to a particular part of the representation cannot take into account changes that may take place in many other parts of the representation. We refer to this as iterative, localized learning.

Learning in massively parallel models requires a solution to this difficult credit assignment problem. For simple representations, called single-layer representations, where each object is defined directly in terms of non-adaptive image features, assignment of credit to each feature involved in a particular recognition is straightforward. Furthermore, a change in the definition of one object does not change the representation of other objects. This allows a learning algorithm to proceed directly to the development of an appropriate representation of the object, if such a representation exists in terms of the image features and the operations for combining features.

The single-layer representation is inadequate for many tasks because it is not capable of representing complex functions of its inputs. In addition, the lack of sub-feature sharing in these representations leads to storage inefficiencies and lack of knowledge transfer across objects during learning.

A hierarchical representation in which objects are defined in terms of sub-objects that are shared among many objects, and these sub-objects are in turn defined in terms of less complex features, is commonly found in models of vision (eg. [Uhr 1972]). Such a multi-layered structure, however, complicates the learning task. In order to learn this hierarchical representation of objects, the sub-objects and simpler features must be learnable. If the definition of one sub-object is modified to improve the representation of one object, it may degrade the representation of another object that shares that sub-object. In addition, it is a difficult problem to decide, when an object definition is incorrect, whether this is due to the way in which sub-objects are combined to define the object, or due to the definition of one of the sub-objects.

In a connectionist model, these decisions on how to change the representation are implemented as rules for changing connection weights. The correlational synapse of Hebb provides one answer to the credit assignment problem by defining each weight change in terms of the activities of the elements connected by that weight. This unsupervised method is not sufficient for task specific learning, however, since it involves no task specific feedback.

Error-correction methods allow task specific behavior to be learned. These learning algorithms calculate changes to connection weights based on estimates of the marginal effect of each weight on the output behavior. The standard heuristic used to iteratively refine the weights in a network for the performance of a particular task is to change each weight to reduce the error on the current iteration. Although

this somewhat greedy approach works well in some cases, it has a number of undesirable characteristics. The most important of these is that the system may become stuck in a configuration where none of the marginal weight changes reduce the non-zero error. Current work in multi-layer learning is aimed at improving this general approach to eliminate this and other drawbacks.

1.4. Summary of the present research

One goal of this research is to develop an adaptive model of transformation-invariant object recognition for machine vision. In particular, we are interested in the problem of recognizing novel transformations of known shapes. This requires a model that generalizes its knowledge of shapes across transformations. Our second goal is to develop improved learning algorithms for multi-layer networks.

In our approach to these problems, we assume and emphasize two aspects. First, any feasible model of machine vision must have a high degree of parallelism and an ability to represent information at a fine level of detail. Second, a robust vision model must be capable of adapting to its environment.

Our vision model is implemented as an adaptive hierarchical, feedforward connectionist network. Although the lack of top-down data and control paths limits the capability of the model, it provides a useful model for studying the representational and learning problems of interest. The network consists of two sub-networks which separately represent the object shape and transformation information. To provide support for the hierarchical representation of shape information, a

pyramid-based sub-network is used. Transformation information is represented in a layered auxiliary network that controls the inversion of the transformation to produce object-centered shape features in the upper layers of the pyramid.

In this thesis, we use the problem of letter recognition as a vehicle for the study of machine vision and machine learning. This task has received significant attention in the past, due to its practical nature. It is chosen for this work because it is intrinsically a two-dimensional vision task. In addition, it can pose a wide range of problems at varying levels of difficulty, from highly stylized block letters, to handwritten text.

When viewing an object in a scene, a change in viewpoint corresponds to a rigid-body transformation of the object shape. For two-dimensional shapes, such as alphabetic characters, the possible transformations are translation, rotation and magnification. The focus of this work is on the task of visually recognizing letters that have undergone such transformations. While the regularity of the problem involving rigid-body transformations may suggest that automating the knowledge acquisition process is unnecessary, the model we describe is applicable to more general transformations as well. For instance, recognition of handwritten characters would proceed by representing the regularities in the character generating process to compute a description of the corresponding canonical character. These regularities would be represented in the auxiliary network. In this case, a direct programming approach is likely to be too complex to consider, and learning methods will be

needed.

We use an error correction learning method to train the network by example. The most commonly used multi-layer algorithm is the generalized delta rule. Since this algorithm is weak in a number of ways, a number of modifications to this rule have been developed and applied to this task. These modifications involve the addition of local constraints to the global error reduction constraint normally used to drive the learning.

The main results reported in this thesis are the following:

(1) An adaptive network model for transformation-invariant shape recognition is presented. A particular instance of this model, a 2-D translation-invariant network, is simulated to demonstrate the representational and adaptive characteristics of its various sub-networks.

(2) New learning algorithms are presented which combine global error information with local representational constraints to obtain a more effective assignment of credit. These learning mechanisms are demonstrated and compared in a number of simulations including the 2-D translation network.

(3) A hierarchical structure for combining two knowledge sources is presented. The hierarchical structure reduces the connectivity that would otherwise be required when using context information to modify the computations applied to sensory data. This reduced connectivity facilitates learning by reducing the space of connection weights to be searched.

1.5. Overview of the thesis

In this chapter, we have introduced the subject matter of the thesis and the three fields of research that relate to the problem posed and the search for its solution. The topic is adaptive visual recognition that is invariant to transformations of the shapes to be recognized. The research fields into which this work falls are computer vision, machine learning and connectionist modelling. The vision and learning problems are described only briefly here, since they are discussed in more detail in subsequent chapters. The section on connectionist models provides some background on terminology and assumptions as well as some historical perspective.

In the next chapter, we discuss the aspects of computer vision related to this work. Previous work is reviewed in the areas of computation of shape descriptors, parallel models of vision and object-centered representation. We then present a network model for transformation-invariant shape recognition that is capable of learning the recognition task. This model combines aspects of Hinton's representation of transformation information with Uhr's representation of shape. In particular, a specific network for performing 2-D translation-invariant recognition is described.

In Chapter 3, the problem of machine learning is discussed. For our purposes, we define learning as the process of developing and maintaining an accurate model of the current environment. Learning in a connectionist network involves the modification of connection strengths in a network of simple processing elements. We

review previous work on single- and multi-layer network learning algorithms, discussing backpropagation methods in particular. Two ways of extending these methods are suggested. In one extension, nodes compete locally for weight modifications based on backpropagated errors. In the second extension, nodes compete for weight modifications based on their ability to represent a particular feature.

Chapter 4 describes a number of simulations of small networks. These simulations demonstrate some weaknesses of the generalized delta rule, the most popular of the backpropagation learning algorithms. Extensions to the algorithm are then tested, and shown to improve the learning behavior in these networks.

Chapter 5 presents the results of simulating various sub-networks of the 2-D translation network. These simulations demonstrate the representational and adaptive characteristics of these networks. In particular, the development of object-centered shape features and explicit representations of location information is shown to lead to generalization of recognition across transformations of objects.

Chapter 6 presents our conclusions and suggestions for extensions to this work.

Chapter 2

Network Models for Object Recognition

The purpose of this chapter is to motivate and describe a connectionist model of object recognition that is capable of learning invariant features with respect to viewpoint. After describing the recognition problem, we motivate the design of the network model by reviewing previous work in the areas of representation of shape, parallel models of vision and intrinsic properties of objects.

The final section of the chapter presents the network model itself. The discussion is based on invariance to transformations in the two-dimensional plane, and we later demonstrate the network using only translations in the plane. The model is intended, however, to allow more general invariant information to be represented, such as that due to rubber sheet deformations of a non-rigid object. This is discussed further in Chapter 6.

The characteristics of the model are its:

(1) hierarchical representation of shape features

(2) parallel feature extraction and matching

(3) use of object-centered descriptions of objects

(4) explicit representation of transformation information

(5) conjunctive combination of shape and transformation information

(6) ability to learn to recognize objects from examples

2.1. Perceptual recognition

A photographic image captures the description of a scene in a collection of light intensity values. The scene itself is composed of a number of objects in some spatial relationship to the imaging device and to one another. Each of the objects in the scene can be described in terms of these relationships and additional characteristics such as surface features and composition by sub-parts.

A description of the scene is contained implicitly in the light intensity values of the image, though some information is lost in the imaging transformation. The goal of vision is to extract information from the image intensity values, that is useful for describing or interacting with the scene.

2.1.1. Scene understanding

Scene understanding requires the recognition of objects and their relationships, including spatial, temporal, functional and sub-part relations. The extraction and representation of such relations involves problems of short term memory, goal-directed control of recognition and focus of attention. Feldman [1984] has provided a sketch of what is involved in the general scene understanding problem. A great deal of earlier work has been concerned with the segmentation of images, which is presumed to be a prerequisite for recognizing objects in a scene [Tenenbaum &

Barrow 1976, Hanson & Riseman 1978, Chen & Pavlidis 1979, Levine 1980, Barrow & Tenenbaum 1981].

A determination of what information is useful in understanding a scene will depend on the task to which the vision system is to be applied. In any case, the description will be one with semantic content, ascribing meaning to the various syntactic components of the image. The ability to recognize individual objects is a prerequisite to solving the overall scene understanding problem. Object recognition involves finding the descriptions of known semantic entities in a syntactic (structural) description of the image.

The keys to solving the object recognition problem are in identifying the syntactic features to be extracted from the image, in defining the semantic objects to be recognized in terms of these extracted features and in providing an efficient and flexible mechanism for matching.

An object in a scene may have many different representations in the image intensity values, due to various aspects of the imaging process, the environment, and the state of the object. For instance, changing the viewpoint from which an object is observed changes the projection of its shape onto the imaging surface. Similarly, the shape of a non-rigid object may change through different relative motions of sub-parts or stretching and bending. Defining each object as a disjunction of all of its possible image instantiations leads to a representation whose size is exponential in the number of imaging and deformation parameters. To overcome this problem, it is

desirable to identify features of each object that are invariant to changes in these parameters.

2.1.2. Viewpoint invariance

An important subset of these parameters is that related to the spatial relationship between the imaging device and the object. For rigid objects in scenes where the effects of perspective projection can be ignored, it is the transformation of the object due to viewpoint that causes a single object to have multiple image shapes. To solve the recognition problem, viewpoint invariant features must be computed from the image data. Such features we refer to as being object-centered, since they have meaning relative to the object itself rather than to their image appearance.

Much effort through the years has been devoted to solving the problem of recognition under transformation. For two-dimensional images, for instance, the viewpoint transformations are translation, rotation and scale. In this case, objects must be modeled to allow recognition independent of location, orientation and size in the image. Viewpoint invariant features include Fourier descriptors [Kabrisky et. al. 1970, Persoon & Fu 1974], chain codes [Freeman 1974] and central moments [Rosenfeld & Kak 1976]. These features can be computed without any knowledge of the viewpoint.

Given the task of recognizing tools placed on a table, a computer vision system that uses this type of invariant feature to characterize each tool could answer the question: Is tool X on the table? A related task would require an answer to the

question: Where on the table is tool X? Similarly, a computer vision / robot arm system might be designed to: Pick up tool X from the table.

Invariant features of the type just described have been used for tasks related to the first question. These features contain no information spatially relating the object to the imaging device. Since they are invariant under rigid-body transformations, we refer to these as semi-topological features. An alternative approach is one in which the viewpoint transformation is identified and this information used in the computation of invariant features. Marr [1982] suggested that 'natural' axes such as elongation or symmetry be used to impose a frame of reference on an object prior to recognition. One advantage to this type of approach is that it involves explicit representation of spatial information describing the object. This information is needed when applying the vision system to the second and third tasks just mentioned.

2.1.3. Learning and generalization

Designing an object recognition system requires the specification of each object in terms of information that can be extracted from the image. An adaptive system has the advantages of automating this initial definition of objects and of changing those definitions as the task demands change. By appropriately choosing the representation structure of the system, knowledge acquisition through learning can be made efficient. This efficiency comes from generalizing the information in specific image examples to allow some degree of recognition in unknown image patterns. Such generalization might allow the recognition of a known object under a transformation

corresponding to a viewpoint from which the object has never been seen. This generalization capability would eliminate the necessity of exposing the system to all objects under all transformations in order for it to acquire a representation that would allow it to recognize all objects under all transformations.

2.2. Shape

There are many characteristics of objects that can be computed from the image intensity array. These include surface characteristics such as texture, color and reflectance, the shape of the object including the spatial relations among its parts and surfaces, and motion of the object relative to other parts of the scene. The usefulness of a given characteristic to the task of visual recognition of objects depends on the difficulty of extracting the characteristic from the image data, as well as the ability of that characteristic to discriminate different objects. Of the characteristics that can be used to recognize objects, shape is often the most useful. For this reason, it is the shape of an object that provides the primary cues for identification. For example, a line drawing of an object, containing only shape cues, is often as recognizable as a photograph of the object. Other object properties, such as color, texture and motion, are useful for segmenting an image prior to extraction of shape properties. In addition, these other properties can serve as secondary recognition cues when edges are ambiguous or of insufficient detail.

2.2.1. Segmentation

The shape of an object is defined by the spatial extent of its surfaces. One aspect of the shape that is most easily extracted from an image is the contour of the object comprised of all the surfaces normal to the line of sight. This bounding contour separates the image region corresponding to the object from similar regions corresponding to other objects. It is the goal of image *segmentation* to group the image elements into meaningful clusters. In a perfectly segmented image, the image area corresponding to each different scene object is identified. One important component of the segmentation process is the detection of edges. Edges in the image are detected as sharp spatial changes in the light intensity which often correspond to object boundaries in the scene. Psychological [Attneave 1954] and physiological [Hubel & Wiesel 1962] evidence suggests the importance of edge detection in the perception of objects by animals. The segmentation process is confounded by the presence of image edges where no corresponding object boundary exists in the scene, and by a lack of image edges corresponding to some object boundaries. Various approaches have been suggested for improving image segmentation. The IGS system of Tenenbaum and Barrow [1976] used a top-down scene interpretation component to refine the otherwise bottom-up segmentation from edge features. The VISIONS system of Hanson and Riseman [1978] performed two segmentations of the image, one based on edge detection, one on region growing. These two imperfect components complemented each other, so that together they represented a good quality segmentation. Kelly [1971] used the interactions of multiple resolution image

representations to fill in boundary gaps cooperatively. Levine [1980] used multiple segment properties to find high quality segment boundaries. Use of intrinsic properties of scene objects has been suggested (cf. [Marr 1978]) for use in segmentation, but these properties have proven difficult to extract from the image (cf. [Barrow & Tenenbaum 1981]).

Given the representation of all object boundaries in the image there are a number of ways to represent the outline of the object. One method is to choose a family of parametric curves that can be fitted to the shape boundary. This allows the shape to be described in a compact way using a relatively small number of parameter values. This approach is useful to the extent that the family of curves can be made to fit closely the shapes of interest, and to the extent that the corresponding shape parameters can be extracted from the edge data. Chain coding is a piecewise form of parametric curve fitting in which very small pieces described by a single parameter are used collectively to describe a larger contour.

2.2.2. Whole-template matching

Another method of extracting shape features from an image is *template matching*. In its simplest form, template matching consists of specifying an intensity array (the template) that describes the shape feature of interest, then searching the image for that template pattern. Due to the many possible ways in which a given object may appear in an image, an exact match between the template and image is not a useful operation. The matching process is, therefore, usually

somewhat fuzzy. One commonly used way to search the image for the shape of interest is to convolve the image with the template. Peaks in the resulting array indicate locations of the best match between the template and the image.

Two major drawbacks to using whole-templates are the noise sensitivity of these procedures, and the requirement of a large number of templates for recognition of complex objects or objects under various transformations.

The problem of sensitivity to noise is generally handled by using inexact matching and including redundancy in the feature extraction process. For example, the use of 'spring-loaded' templates [Davis & Rosenfeld 1976] allows inexact matching of the template with the image. One variation of the template matching approach which gives good noise immunity is the use of the Hough transform [Ballard 1981].

2.2.3. Transformation invariance

The problem of providing transformation invariant recognition may be handled in various ways. One is to use a procedural method to adjust templates for different presumed transformations. In fact, the use of the convolution operation to search for a pattern at every location in an image [Rosenfeld & Kak, 1976] is just such a procedure for handling translation invariance. This is equivalent to the specification of multiple templates, one for each possible transformation. This is discussed further in section 2.3.

2.2.4. Hierarchical representation

When a complex object can vary in appearance in many ways, a large number of templates is required to represent each variation. Using a template hierarchy reduces the number of individual templates needed to represent all variations of all objects. The template for a given object is defined in terms of sub-templates. These sub-templates can be shared by many different objects, and may themselves be defined in terms of still simpler sub-templates. To the extent that the object shapes can be described in terms of a relatively small number of common shape features, this hierarchical representation will be efficient. For instance, Block, Nilsson and Duda [1964] present a simple example in which 6×10^5 templates are needed to classify a set of patterns into 20 categories using a non-hierarchical representation. A two-layer hierarchy reduces the number of templates needed to 2500 and a three layer hierarchy requires only 56 templates.

In addition to the storage economy associated with hierarchical representations, the sharing of common sub-templates allows generalization of learned knowledge from one object to another. Evidence for the importance of hierarchical structure is also provided by the vertebrate visual system [Hubel & Wiesel 1962, Van Essen & Maunsell 1983].

One well known hierarchical approach to vision is that described by Marr [1982]. At the lowest level, the shape features extracted from the image are edges. These edge features, extracted at various levels of resolution, are clustered into

groups of similar features to form the primal sketch. The shape features composing the primal sketch are further compounded to form intrinsic features. Intrinsic features are related to the properties of the objects in the scene, rather than simply features of the image of the object. Intrinsic features comprise the 2½-D sketch which contains information on object surface orientation and surface discontinuity and on depth magnitude and depth discontinuity.

Some approaches, such as the recognition cone [Uhr 1972], use a part-subpart hierarchy to represent objects (see section 2.3.1). Other approaches base their hierarchy on multiple levels of resolution. The use of multiple resolution images allows transformations of scale to be handled in a natural way. In addition, course to fine control of the recognition process is facilitated. Hierarchies based on multi-resolution representations include quad-trees [Dyer, et. al. 1980], strip-trees [Ballard 1981], the difference of low pass transform [Crowley 1983], scale-space filtering [Witkin 1983] and Davis' [1977] representation of angles and sides of a closed curve.

Extraction of 3-D surface shape features is more difficult than that of 2-D shapes, since the imaging process maps depth information into the 2-D image plane. The general approach has been to use specific cues in the image and interpret them according to what they most likely represent in the scene. Such an interpretation depends on known constraints on the scene domain. Thus, work with the blocks world [Waltz 1975] used the configurations of corners to suggest a 3-D interpretation of the edges of an object. This approach was later extended to a more complex

domain, referred to as the 'origami world' [Kanade 1981]. For less constrained environments, more powerful algorithms and heuristics are required. There have been a number of 'shape from' algorithms which use shading [Horn 1975, Ikeuchi 1980], texture [Kender 1979, Ikeuchi 1980, Witkin 1981] and contour [Augusteijn & Dyer 1985] as cues to 3-D surface orientation. Once such features are extracted, the resulting patterns must be matched with known models just as in the 2-D case.

2.3. Parallel models

Despite the efficiencies gained by using a hierarchical representation of shape features, the extraction of information from an image and matching this information to object models still requires the processing of large quantities of data. Extraction of low-level features requires the matching of many small features in all parts of the image. Array processor architectures [Duff 1976, Flanders et. al. 1977, Batcher 1980] were designed to efficiently support such operations. An array processor consists of a two-dimensional array of simple processing elements. Each processing element can directly access only a small subset of the image data. Due to the large number of processors, however, a great deal of computational power is realized. Use of a large number of simple processors to execute a single task is referred to as the massively parallel approach to computation.

The array processor has a flat two-dimensional structure. The pyramid model [Tanimoto & Pavlidis 1975, Hanson & Riseman 1978] is an extension of the array model that imposes a hierarchy on the processing elements. A pyramid processor

consists of layers of two-dimensional processor arrays. The largest array processes the image data directly. Successively smaller arrays above the first process the results of previous arrays. One use of this pyramid structure is in implementing 'recognition cone'-based object recognition systems [Uhr 1978, Uhr & Douglass 1979, Uhr & Schmitt 1982]. In the recognition cone (see section 2.3.1), the lowest level array extracts low-level features directly from the image. Higher level arrays extract more and more abstract features by compounding the features in lower layers. In this way, the pyramid structure directly supports a hierarchical representation of visual information.

Pyramids have also been applied to the representation of multi-resolution images, which are useful for coarse-to-fine processing [Kelly 1971], region growing [Cibulskis & Dyer 1984], segmentation [Levine 1980, Hong et. al. 1982] and template matching [Tanimoto 1978]. Other work on pyramids has been concerned with efficient hardware implementation [Ahuja & Swamy 1982, Sandon 1985] and formalization of pyramid processing operations [Tanimoto 1983].

2.3.1. Uhr's Recognition Cone

The recognition cone [Uhr 1972] is an example of a parallel-hierarchical vision model with a pyramid structure. The recognition cone consists of layers of *transforms*. Each transform produces an output value that is some characteristic function of its input values. In general, the output values of each transform are used as input values to other transforms. The transforms in the lowest layer compute

properties of the image pixels, those in the next lowest layer compute properties of these lowest layer transform properties, and so forth. Higher layers in the recognition cone are logarithmically smaller than lower layers, giving a hierarchical, pyramidal structure. Information is converged as it proceeds up the cone, so that higher level transforms, performing local operations on the information below, compute more global properties of the image than do transforms at lower layers. The properties computed by any given transform are stored locally, or passed up to be accessible to transforms at higher layers.

The recognition cone can be implemented in a straight forward manner using a pyramid architecture. Each processing element in the pyramid has a set of transforms that it computes. Since the element must apply its transforms serially, transforms can be made to execute conditionally. This allows a simple focus of attention mechanism to be realized. In addition to the bottom-up processing, the property computed by a transform can be passed down the pyramid to provide feedback to earlier layers.

The learning mechanism suggested for the recognition cone requires a global processor to define new transforms, test them out in various parts of the network, and make decisions about when to keep new transforms and when to discard useless transforms. Useful low-level transforms can be broadcast to all processing elements, providing a means of learning translation- and magnification-invariant properties. Local processor memory can support such capabilities as motion detection, which requires features from previous images to be saved for comparison with current

image features.

The recognition cone has much in common with the connectionist paradigm. It can be described as a connectionist network by making the following changes. Each processing element in the pyramid that computes n transforms is replaced by n simple processing elements each computing a single transform. A given transform accesses the result of another transform through a connection. The difference between this model and the original recognition cone is that non-connectionist transforms can be more complex functions of their inputs than weighted sums. The local nature of these complex functions, however, makes them amenable to a neural interpretation. The original recognition cone can also make use of local memory and can produce more complex output signals than single scalar values. In addition, the recognition cone has a control structure that allows complex connectivity patterns to be realized by using processors to forward information between arbitrary points in the network.

Neocognitron

A related vision network that is nearly connectionist is Fukushima's *neocognitron* [1980]. This is a layered hierarchical network intended to mimic the structure of the vertebrate early visual system. The network consists of three pairs of layers. One layer in each pair is comprised of elements with modifiable synapses. Elements in the other layer of each pair have fixed connections. An unsupervised learning algorithm is used to train the network to discriminate various patterns

presented to it. To account for translation of the pattern within the image, a mechanism is added to broadcast the connection weights learned at one location in the image to nodes in that plane at all other locations. This broadcasting of features is similar to the mechanism used in the recognition cone. There are two disadvantages to the method used in the neocognitron to obtain translation invariant recognition. One is that a centralized control mechanism is needed to choose a set of weights to be broadcast and to perform the broadcast function. It is difficult to interpret this mechanism in a connectionist way. The second disadvantage is that information about the location of the object within the image is discarded. This makes the represented features semi-topological (see section 2.1.2), eliminating their potential use in locating, tracking or grasping of objects.

2.3.2. The Hinton / Ballard model

An approach which *explicitly* represents both the transformation invariant shape and the transformation itself is that proposed by Hinton [1981] and extended by Ballard [1984]. In this connectionist model, processing units are grouped into three distinct sets (see Figure 2.1), referred to as the retina-based frame, the object-based frame and the mapping units. The retina-based units represent features extracted from the image with spatial relations represented relative to the imaging device. The object-based units represent spatial relations relative to the object, without regard to the particular image representation. The mapping units represent the transformation between the retinal shape and the canonical coordinate frame represented in the

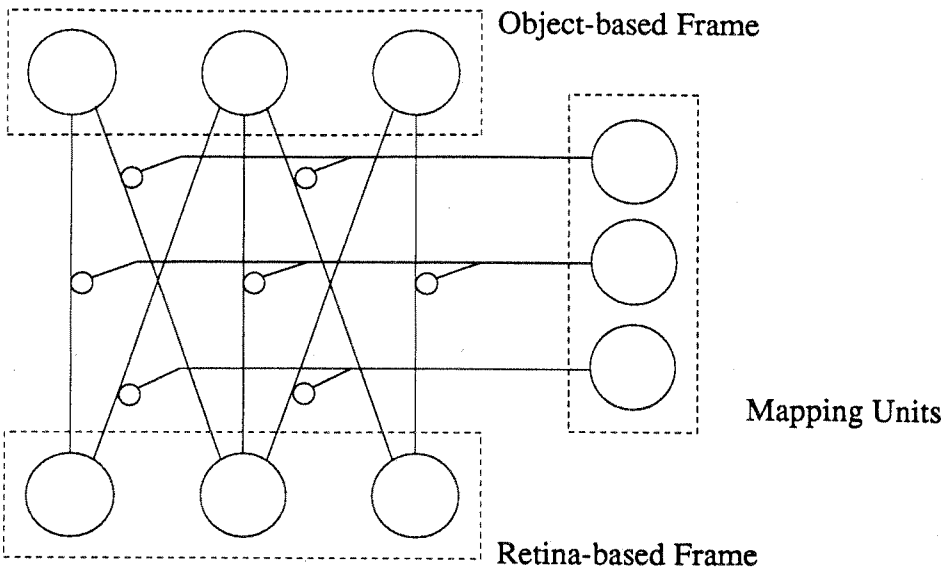


Figure 2.1

object-based frame.

Retina-based units compute features based on properties of the image itself. Object-based units combine retina-based and mapping unit features to compute object-centered features. The circles in Figure 2.1 represent the conjunctive modification of the retina-based to object-based connection by the mapping units. Similarly, mapping units use information from both retina-based and object-based units to compute the current object transformation. This computation is not represented in the figure. The interdependence of the object-based and mapping units requires a cooperative computation in which a partial result in one set of units improves the result in the other set.

Developing object-centered features has a number of advantages. First, object-centered features potentially simplify the recognition process by representing aspects of the object that do not depend on imaging parameters. In this sense, object-centered features are similar to intrinsic features of objects, such as surface reflectance, color, texture, optical flow, surface orientation and depth [Barrow & Tenenbaum 1978, Marr 1982, Ullman 1983, Terzopoulos 1984]. Since intrinsic features are attributes of the physical objects in the scene, rather than properties of the image intensity values, they are more useful in identifying those objects.

Another advantage to object-centered representation is that it allows more powerful generalization capabilities due to the similarity of representation within an object class. This same advantage applies to the explicit representation of transformation, allowing generalization across objects having the same transformation. This capability is demonstrated in the simulations described in section 5.2. A less powerful form of generalization, based on syntactic similarities in the input, would not provide the same learning efficiencies in a problem involving different transformations of a given shape. By representing shape and transformation in this way, the network correctly generalizes across both shape and transformation. This allows knowledge acquired from one input pattern to be applied to patterns having similar shapes or transformations. Without this generalization property, a learning system would require all objects under all transformations to be presented to it to become completely trained.

In computing the object-centered features, object-based units combine information from the retina-based and mapping units *conjunctively*. Since a given retina-based feature may correspond to many possible object-centered features, each retina-based feature by itself provides only weak evidence for any given object-based feature. For example, in a character recognition task, a horizontal line in the center of an image can be taken as evidence for many different characters. If the character is known to be centered and upright, the horizontal line provides evidence for the characters H and E, among others. If the character is known to be above center, the horizontal line gives evidence for the characters L and E, among others. If the character is known to be rotated 90°, the horizontal line gives evidence for the character T, and so forth. Without the transformation information, this single feature provides only weak evidence for almost any character.

Similarly, the transformation information represented in the mapping units gives no evidence at all by itself for the existence of any object-centered features in the image. Conjunctively combining these two sources of information provides strong evidence for object-centered features. A pair of features, one retina-based, one representing transformation, now gives evidence for a given object-centered feature only if there is evidence for *both* that retina-based feature and that transformation. This same conjunctive combining is used to compute the transformation from the retina-based and object-based units.

Since the computations of transformation and of shape information are mutually dependent, the network performs a constraint satisfaction procedure in order to cooperatively find a consistent interpretation for both shape and transformation. This is similar to the cooperative computation of shape and viewpoint found in the 'shape from contour' method of Witkin [1981].

By explicitly representing the one transformation associated with an object, the network implements what Hinton refers to as the single viewpoint constraint. In mapping the retina-based features to object-centered features, many mappings of each individual feature are possible. Each mapping corresponds to a different transformation of the object. Without the use of the transformation information, all of these possible mappings would be used as weak evidence for the object-centered features. The result would be partial evidence for most object-centered features, making object recognition difficult. By explicitly representing the transformation, and combining this information conjunctively with the retina-based features, only those mappings that correspond to the single active transformation are used as evidence for object-centered features. This can provide a much greater disparity of evidence between probable and improbable object-centered features, improving the recognition process.

In addition to the computational advantages of this structure, there is a representational efficiency gained in representing image information in this way. If there are O objects and T possible transformations of each object, then O object-

centered units are needed, and T transformation units. An alternative approach is one in which each object is defined in terms of retina-based features. In this case, $T \times O$ nodes are required to represent each object at every possible transformation. The first approach allows only a single object to be represented at any time, however, requiring further mechanisms for the processing of multi-object scenes. We briefly discuss this problem below.

2.4. Hierarchical transformation model

We now propose a vision model that addresses the problem of viewpoint-invariant recognition. The behavioral requirements on the system to be described are (1) that it be capable of recognizing objects under various two-dimensional transformations, (2) that it be capable of learning this recognition task from examples and (3) that it be capable of generalizing from examples to allow recognition of unknown patterns. Our objective in designing the network, then, is to provide a structure in which shape features can be shared among many object representations and transformation information is represented explicitly. These are the characteristics which will support the generalization of learned knowledge to novel transformations of known objects.

We will assume that each image contains a single object. The network task is to classify each object into one of n possible classes. We do not consider the problem of rejecting non-objects, though the network could be easily trained to do this by specifying that all output nodes be inactive in response to non-objects. We avoid the

problem of representing multiple objects in the scene, as it requires additional control mechanisms and a means of storing scene-based information [Hinton 1981].

2.4.1. Overview

The network model is based on the pyramid-structured recognition cone. This provides efficient support for the shape hierarchy that is used to define the objects to be recognized. We use a connectionist implementation of the recognition cone, in which each transform is represented by a logistic processing element. Previous work on connectionist learning provides a starting point for the study of learning in this network.

In order to handle the problem of transformation invariant recognition, the network explicitly represents transformation as well as shape information. The transformation is represented in a subnetwork that is separate from the shape pyramid, as previously discussed. The information in this subnetwork is conjunctively combined with that in the shape network to map retina-based shape features to object-centered ones.

One question that needs to be answered is: At what point in the shape hierarchy do the representations become object-centered? By providing the transition at a low-level, we maintain a high degree of sharing of sub-features in the object-based representation. That is, more complex features are shared among fewer objects. This sharing of object-based features facilitates generalization of learned knowledge across different objects. On the other hand, by deferring the transition to higher

layers, each retina-based feature is more complex and therefore maps to fewer object-based features. For instance, a pixel has only a translation associated with it, an edge has a translation and an orientation associated with it, and a line has translation, orientation and length.

An alternative to this single level transition is a multi-level transition where retina-based features are mapped to what we might call 'locally object-centered' features at each level. This is shown in schematic form in Figure 2.2. These features retain some of their retina-centered quality, in that they are associated with a particular region of the image. However, within that region, the feature represents a concept that is independent of transformations. This gradual transition from strictly retina-based to strictly object-based features has the advantage of spreading the

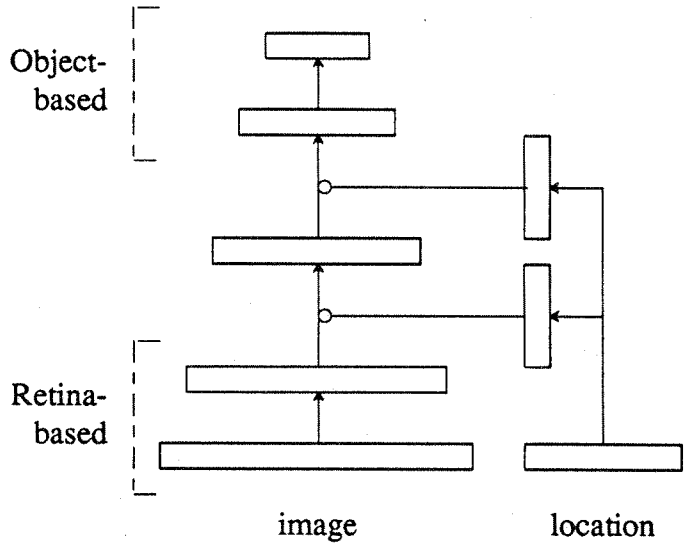


Figure 2.2

increased connectivity due to conjunctive combination over several layers. This reduces the number of connections needed at any given layer, and thus reduces the search space for the learning process. Another advantage to a gradual transition is that the local connectivity that is desirable in the recognition cone is maintained.

To summarize, the basic structure of the network model is that of a pyramid-like primary network in which shape features are hierarchically represented, augmented with a secondary network in which transformation information is represented. The transformation information is conjunctively combined with shape features, at various levels of the pyramid, to produce more object-centered representations of shape in higher layers of the network. We refer to all layers of the shape pyramid below the point where transformation information is introduced as retina-based layers. All layers above any use of transformation information are object-centered layers. Those layers in between are transition layers.

2.4.2. 2-D translation network

A more detailed description of a specific network of the type just described is now presented. This 2-D Translation Network demonstrates the characteristics and capabilities of the general design by recognizing various stick-figure patterns under all translations within a small image plane. A simulation of this network is presented in Chapter 5, where further details of its structure can also be found.

Shape features are organized in a five layer pyramid-like hierarchy (see Figure 2.3). The lowest two layers represent strictly retina-based shape features. The

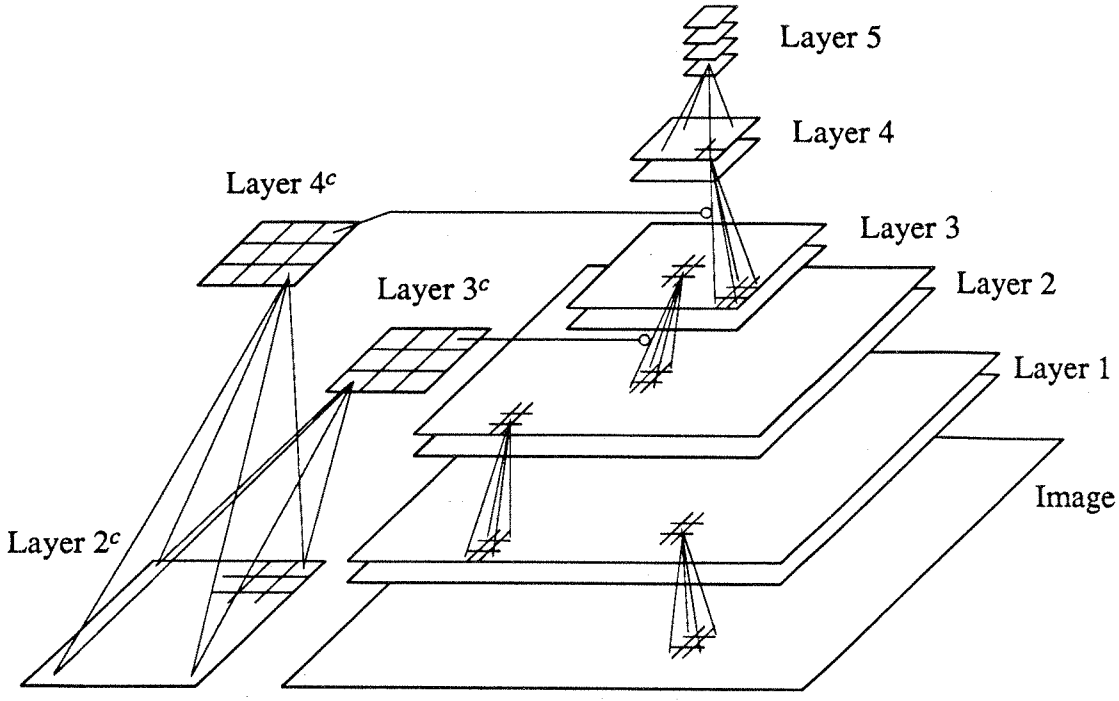


Figure 2.3

highest two layers represent object-based features. The middle layer represents features that provide a transition between retina-based and object-based features. The transformation (in this case, translation) of the object is represented in a two layer network, whose output is used to conjunctively combine with shape features at layers 3 and 4 in the shape hierarchy.

In the Hinton model, it is suggested that both shape and transformation can be extracted from the image. This requires that the identity of the shape be used to define the transformation, and that the identity of the transformation be used to define the shape. These mutually dependent definitions of shape and transformation require

feedback paths and a relaxation process for computation by a network. In order to maintain the feedforward structure of the network, we do not include the feedback term from the object-centered features in computing the transformation. However, by providing separate training signals to the context network, the transformation information can be learned as a function of the image pattern (see section 5.3.1).

The learning capabilities of this network are demonstrated by simulations described in Chapter 5. Here we describe qualitatively how the network is able to generalize its recognition capabilities across transformations

As training proceeds, we imagine the network learning the following sequence of patterns. First, a letter *E* centered in the image is presented to the network. We

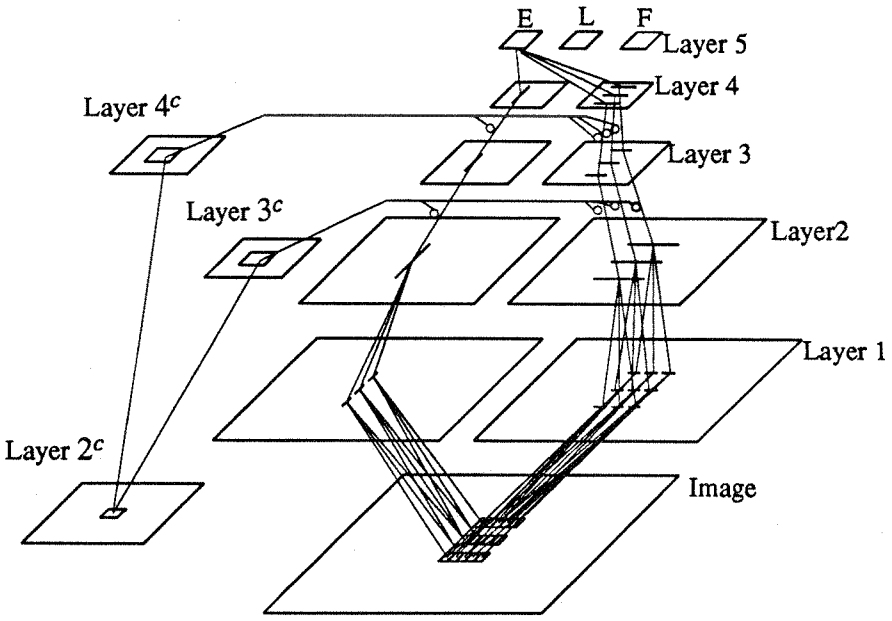


Figure 2.4

use simple horizontal and vertical lines to represent the types of features that might be learned to represent this object. Figure 2.4 shows the network with active nodes drawn as the features they represent and only active connections drawn between the corresponding nodes. The network has had to learn retina-based features in the lower layers, object-based (and transition) features in the upper layers and an activity pattern in the auxiliary network that adequately represents the transformation information provided. If a letter F centered in the image is now presented, the network can take advantage of shared sub-features and will have to learn only the connections in the top layer that define the object in terms of its object-based features. Similarly, a letter L centered in the image can be quickly learned.

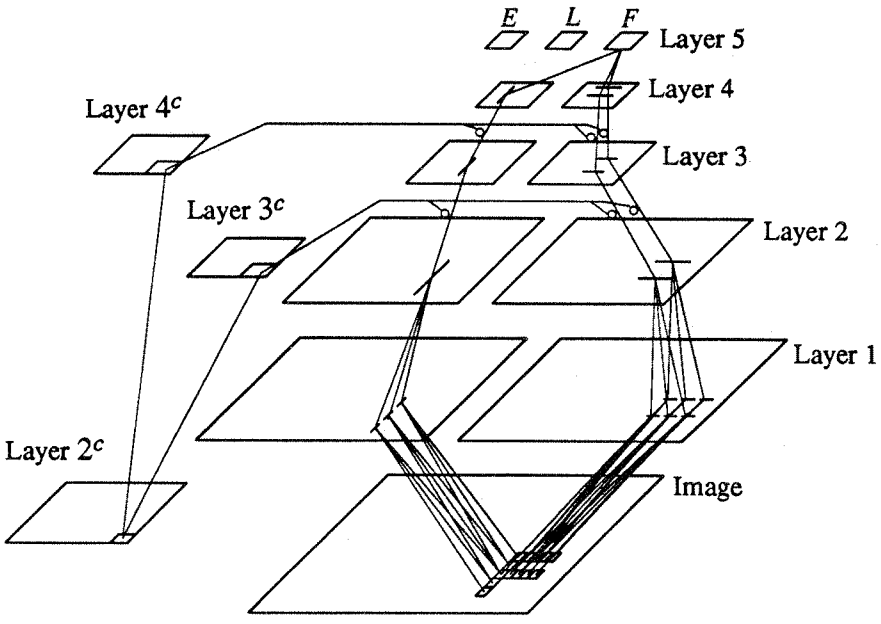


Figure 2.5

Now, if a letter F offset from center is presented (Figure 2.5), a new set of retina-based features must be learned, but the connections in the top layer are those that were previously learned. Similarly, presenting the letter L at this same offset requires some new retina-based features to be learned. Once the network has learned to recognize these five patterns, it is presented with a sixth. If the letter E is presented to the network at the same offset that L and F were presented (Figure 2.6), no new features are needed for recognition. Due to the overlap in sub-features, and the explicit representation of the translation, the correct object-centered features that define E will be active for this pattern, leading to recognition.

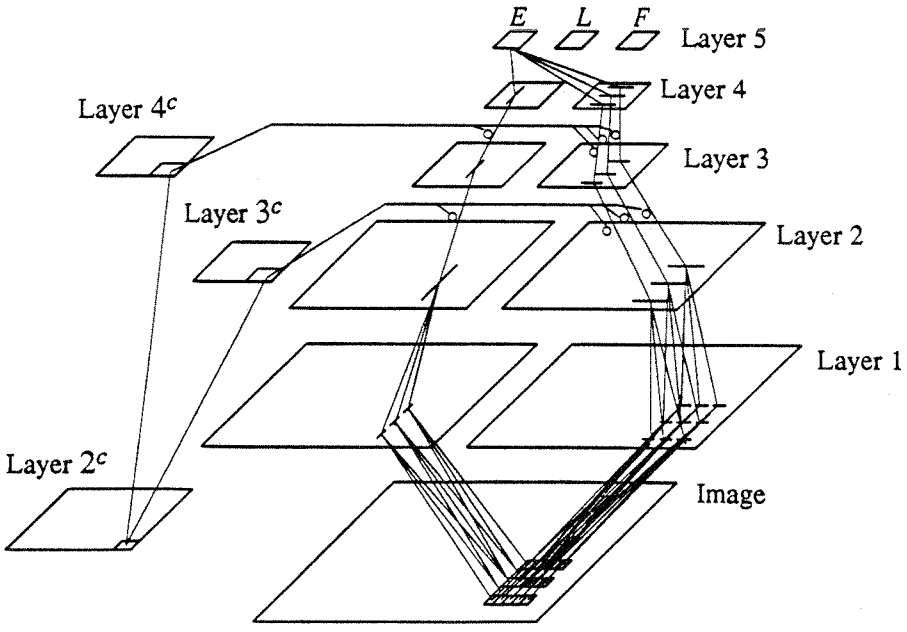


Figure 2.6

Simplicity of exposition requires that this example be somewhat contrived. It does, however, demonstrate that the network structure that has been described in this chapter is capable of taking advantage of shared features of objects and explicit representation of transformation to generalize its knowledge to novel transformations of known objects.

Chapter 3

Learning in Fine-grained Parallel Networks

3.1. Introduction

The characteristics of connectionist networks were described in a previous chapter. To reiterate the points that are useful to this chapter: the *knowledge* in the network is distributed throughout, being represented by the connection weights of each processing node. Since learning is a process by which the knowledge base is modified, learning in a connectionist network consists of modifying these connection weights. The problem is to determine which weights to modify, and in what fashion, in order for the network to learn to perform the task of interest.

3.1.1. Feature formation and credit assignment

Two aspects of this learning task which have often been treated separately are the structural credit assignment problem and the feature formation problem. Feature formation involves the development of a set of functions of the network inputs that can be used in the definitions of the network output functions. The problem is to find a set of functions, also called features, that sufficiently characterizes each input pattern for the solution of the task. Credit assignment involves the adjustment of the network output function definitions in terms of the feature set just described, which we will refer to as the 'formed features'. The problem in this case is to decide, when

the network behavior is incorrect, which of the formed features should be more or less associated with a given output function. In a network, credit assignment involves the adjustment of connection weights which define the associations between output functions and formed features.

The two sub-problems were considered separately, traditionally, because there was no known means of solving the overall problem with a single mechanism. Thus, the two steps for solving a classification task were first to choose a set of features, that is, functions of the inputs, and second, to define outputs in terms of these chosen features. With the introduction of promising approaches to credit assignment in multi-layer networks, the feature formation problem can be solved by treating it as a credit assignment problem at the lower layers of the network. This allows for a single mechanism to solve both sub-problems. On the other hand, the difficulty of the multi-layer credit assignment problem makes the multiple mechanism methodology worthy of further consideration. As we discuss below, current algorithms for multi-layer credit assignment are fairly weak, so additional mechanisms that improve feature formation and intermediate level credit assignment will be useful.

This distinction between feature formation and credit assignment bears on the question of the power of learning in neural nets. For a single layer network that has a fixed set of input features, the learning algorithm is limited to finding the right combination of those features to express the outputs of interest. This simply involves

adjusting the parameters until the best combination of pre-defined features is found. Such a network is incapable of discovering something new, that is, anything that cannot be expressed in terms of the fixed set of features.

Since learning in multi-layer networks also involves the adjustment of parameters, it might appear that a similar criticism applies. However, the capability of forming new features at all levels of the representation allows very complex functions of the sensory primitives to be developed. For this reason, the parameter adjustment methods of neural network learning algorithms provide sufficient power, in theory, to support complex knowledge acquisition and discovery behaviors.

3.1.2. Training

Learning methods are commonly identified according to the type of training information that is provided to the system. An *unsupervised* learning method is one in which no training information is provided. Improvements to system behavior are limited to the performance of similar behaviors in response to similar input patterns. In *Reinforcement* learning, the system receives a single scalar value in response to a particular behavior. This reinforcement signal indicates the correctness of the response but provides no information on how the behavior might be improved. An *error–correction* learning method is one in which each of the outputs receives an individual training signal. This training signal indicates the desired response of the corresponding output, or equivalently, the error in the output signal.

Error-correction learning requires a strong training input based on accurate knowledge of how the network should respond to each input pattern. Its use is restricted to those tasks in which a trainer has sufficient knowledge of the network and the task to provide the necessary feedback. In particular, it is difficult to use error-correction learning algorithms whenever the observable system behavior is something other than the direct network outputs. When it is appropriate to apply an error-correction learning algorithm, such an approach is more efficient than the weaker forms, due to the higher quality of the feedback information.

In this work, we restrict our consideration to error-correction tasks. However, as an enhancement to an error-correction learning algorithm, we will demonstrate the usefulness of an unsupervised algorithm as well.

Since network elements that are not in the output layer do not receive direct training information, indirect methods must be used. One way to provide training signals to these *hidden* elements is to estimate them from the training provided to the output layer. This is the approach of so-called backpropagation techniques. Direct training of hidden elements is generally not considered because it would require detailed knowledge of the network behavior, as well as direct access to every node in the network.

Learning in a multi-layer feedforward network proceeds as follows: A stimulus pattern is first presented to the network inputs. Each element in the first layer computes its output value based on the values of network inputs and the connection

weights corresponding to those inputs. Elements in the second layer then compute output values based on activity in the first layer. Each successive layer of elements, up to the output layer, computes a set of outputs based on the activity in the layer below. The response of the network is defined by the output values of the elements in the output layer. This response is compared with the training information, to produce an error vector for the output elements. This error vector is used to drive the modification of weights in the network, according to a particular learning rule.

3.2. Review of previous work

We review here some past approaches to solving the credit assignment and feature formation problems. We begin by discussing single layer algorithms for credit assignment, where the feature formation problem is deferred or handled separately. Multi-layer network learning algorithms are discussed next. In these algorithms, a single mechanism is used to solve both the feature formation and the credit assignment problems. Note that the distinction we are making between single layer and multi-layer approaches has to do with the learning method and not necessarily with the computational abilities of the network. For instance, in the Uhr & Vossler work, complex features, comparable to network functions requiring several layers to compute, were developed by the feature formation function. However, the adjustment of weights representing object-feature associations occurred at only a single level.

3.2.1. Single layer methods for error-correction

Perceptron

We begin with a well-known model of network adaptation. The *perceptron* is a class of network models studied by Rosenblatt [1962]. The *simple*-perceptron which is the best known and most studied is shown in Figure 3.1. This perceptron uses a layer of input nodes and two layers of processing nodes to classify input patterns into n classes. The weights on the connections from the input nodes to the first layer of processing nodes are fixed, that is, not modifiable by a learning algorithm. Only the single layer of weights between the two layers of processing nodes are adaptive, leading to the characterization of this model as a single-layer perceptron. The processing element used in this perceptron model is a linear threshold element (LTE), defined by:

$$o = \begin{cases} 1 & \text{if } net > 0 \\ -1 & otherwise \end{cases}$$

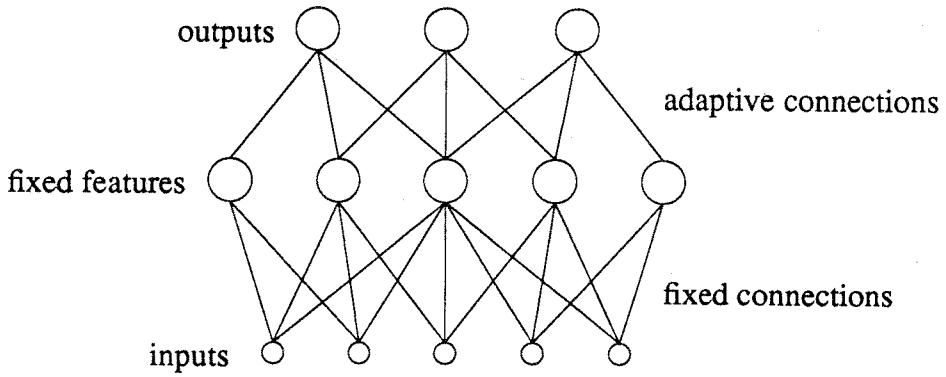


Figure 3.1

The network is to perform a classification task in which, for each input pattern presented, the output element corresponding to the correct class fires ($o = 1$) while all other output elements remain inactive ($o = -1$). The learning algorithm prescribes the modification to be made to each weight in the network, in order to improve the response to the current input pattern. For each output node, if the node response agrees with the desired output, the input weights of that node remain unmodified. If the actual output is different from the desired output, the weight is modified in such a way that the desired response is more likely to result the next time the current input pattern is presented. This modification rule is simply stated in the following formula:

$$\Delta w = \eta(t - o)i$$

where t is the desired output for a given element, i is one of the element inputs, and Δw is the change to the weight associated with that input.

The most interesting aspect of this learning algorithm is Rosenblatt's proof that it accomplishes the desired task. The perceptron convergence theorem states that if a set of weights exists that provides perfect classification of all patterns, the above learning algorithm will find such a set in a finite amount of time. However, if there exists no weight configuration that satisfies the desired response of the network, the weights may oscillate among a set of low error configurations.

The perceptron model handles the feature formation and credit assignment problems separately. Features are chosen by randomly setting the weights in the lower layer of the network to fixed values. If the resulting random features are

satisfactory, the perceptron learning algorithm is guaranteed to accomplish the task of developing appropriate classifiers. This is a simple but weak solution to feature formation. (In fact, random features were chosen for the perceptron model exactly because they are weak [Block 1970]).

Credit assignment is performed by the learning algorithm. Since the lower layer features are fixed, only the output layer weights can receive credit or blame for the network performance. In fact, it is blame that is assigned to each weight associated with an active input for each node that produces the wrong response. More generally, we can describe the assignment of credit as being a function of the difference between actual and desired output, and of the current input pattern. This is the basis for error-correction learning.

Widrow-Hoff

Another network model having an interesting learning algorithm associated with it is the Widrow-Hoff *adaline* [Widrow & Hoff 1960, Widrow 1962]. This model uses linear elements having continuous, rather than binary outputs. The Widrow-Hoff learning rule is the continuous counterpart of the perceptron learning rule, having an identical form:

$$\Delta w = \eta(t-o)i$$

Like the perceptron learning rule, the Widrow-Hoff rule, later called the delta rule, can be proved to converge to a correct solution state if such a state exists. Whereas the form of the perceptron learning algorithm was chosen to satisfy the

convergence proof, the form of the Widrow-Hoff rule was chosen because it minimizes the mean squared error of the network in classifying patterns. For this reason the algorithm is also referred to as the Least Mean Squares (LMS) rule. In fact, the weight changes specified by the algorithm represent a gradient descent in weight space along the error surface associated with the current input pattern. To see this, we express the error associated with pattern p as (see Figure 3.2):

$$E^p = \tfrac{1}{2} \sum_{k=1}^{n} (t^p{}_k - o^p{}_k)^2$$

A weight change along the error gradient is one proportional to the partial derivative of E^p:

$$\Delta w \propto -\frac{\partial E^p}{\partial w_{jk}}$$

Using the activation rule for a linear processing element:

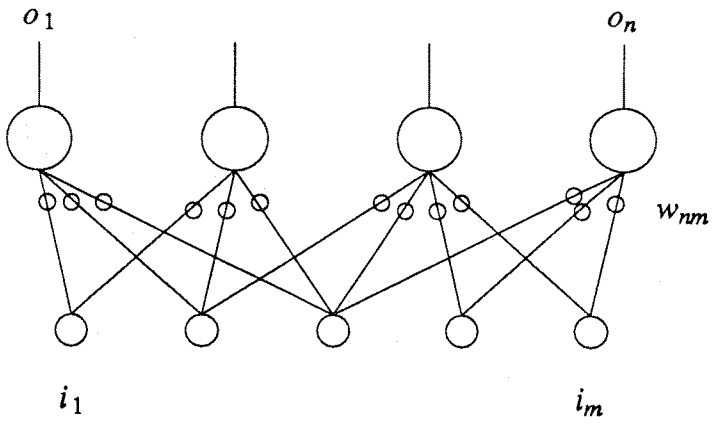


Figure 3.2

$$o_k = \sum_{j=0}^{m} w_{jk} i_j$$

yields:

$$\Delta w_{jk} = \eta\,(t_k - o_k) i_j$$

Notice that on each step, the error gradient associated with the *current* pattern is used to adjust the weights. In the single layer network considered by Widrow & Hoff the error as a function of any given weight is parabolic with a single minimum. This means that a gradient descent method minimizes the error associated with individual patterns considered separately. By choosing the learning coefficient, η, to be sufficiently small, the mean error over all patterns can be minimized. If this minimum error is zero, the network will reach a stable state which correctly implements the desired task. If the minimum error is non-zero, the network will reach a stable state in which its performance is as good as possible. This characteristic of the continuous model demonstrates an advantage over the binary perceptron model.

Like the perceptron, the adaline model does not attempt the adaptive formation of features. In fact, the single layer of output nodes in the adaline is directly connected through weighted connections to the network inputs. If the relation between input patterns and output patterns to be learned is too complex, that is, not linearly separable, the learning algorithm is powerless to overcome this deficit.

Pandemonium

Another model that allowed learning at only a single layer is Selfridge's [1959] Pandemonium network. In this model, a layered network of processors called *computational demons* are prewired to extract what are presumed to be useful features from the input pattern (See Figure 3.3). An adaptive layer of *cognitive demons* connects to the outputs of the feature network. Each of these elements is intended to represent one class of patterns. The response of the network is determined by a single *decision demon* which connects to all cognitive demons and simply chooses the maximally responding element. Though the algorithm is not

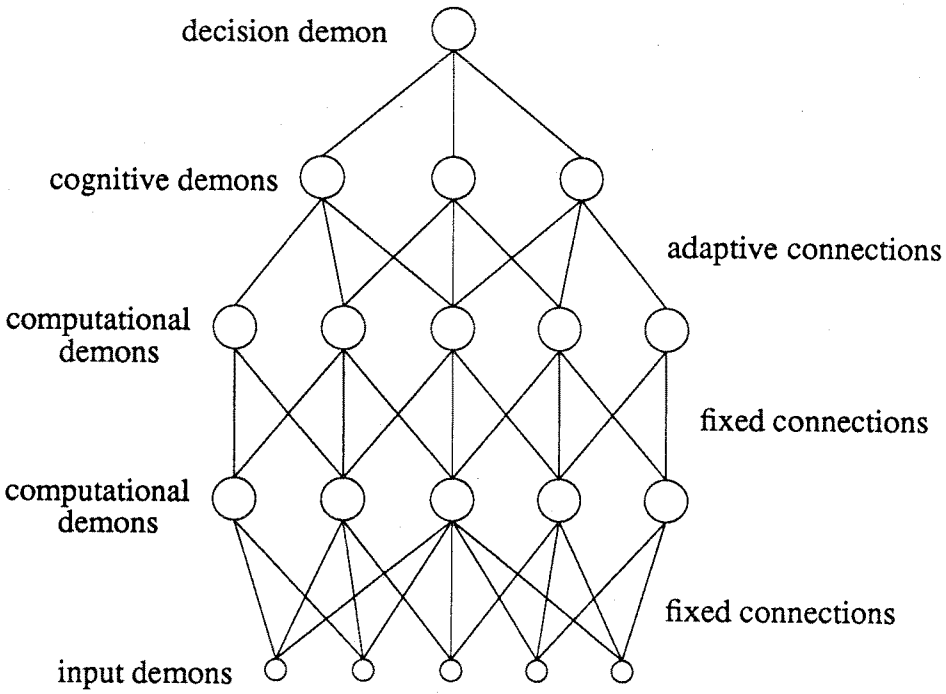


Figure 3.3

explicitly described, the learning procedure would modify the input weights of the cognitive demons so as to increase the *score* of the network, which is a measure of correct responses.

This structure is similar to that of the perceptron, but the problem of feature generation is more fully addressed in Pandemonium. Selfridge suggests that output weights of computational demons can be used as a measure of their usefulness. When the usefulness measure gets low, the feature is replaced with a new feature that is generated by *mutated fission* or *conjugation*. Mutated fission involves small random changes to the weights of one element to obtain a set of weights for a second element. Conjugation involves a logical combination of the weights of two elements to obtain a new third weight set. The intent of these operations is to produce new features that have some of the properties of other features that have already proven useful.

Samuel

Another well known learning model is that used in Samuel's checker playing program [Samuel 1959]. Although Samuel took a more symbolic approach to problem solving and learning than is taken by network models, some of his work is directly relevant to the network models. First, the problem of feature formation is given a good deal of attention. The next checkers move is chosen by the program based on an evaluation of resulting board configurations. The evaluation function is implemented as a weighted sum of features describing the board configuration.

Rather than use raw input data, or random functions of the input, a predefined a set of 38 features is used. These were carefully chosen for their presumed significance to the evaluation of good checkers moves. Since 38 features are too many to work with at once, only a subset of 16 are actively used in the computation at any given time. A measure of usefulness of each feature is maintained to determine when an active feature should be inactivated and replaced by another feature from the predefined set of 38. This measure of usefulness is generated by a credit assignment algorithm. Evaluation of board configurations by a weighted sum is similar to the computation of node outputs in a network. Using a tree search and min-max backup of evaluation values to provide desired evaluation results, the feature weights are updated in a manner similar to previously described learning algorithms. That is, weights are modified to reduce the difference between actual and target evaluation values. Large positive weights correspond to features that positively correlate to a high evaluation value, while large negative weights correspond to negatively correlated features. A weight close to zero indicates a feature that contributes little information to the determination of the evaluation value. Such a feature is a candidate for replacement by one of the inactive features.

Uhr & Vossler

The program described by Uhr and Vossler [1961] is also not a network simulation, though much of the mechanism described can be implemented in a network. The extraction of features for use in classifying patterns, and the method

used to modify the features is more complex than the usual network procedures. Features are defined by extracting sub-patterns, called windows, from the images presented to the system. Each window is convolved with the entire image while statistics are collected on where in the image a match occurs. The set of statistics is used as the 'output value' associated with the window. These window features are logically combined to build a hierarchy of complex features.

Once computed, these feature values are compared with lists of features for each class of pattern that has been learned. The weighted differences between the stored features and the extracted features are summed to produce a score for that classification. The lowest score classification is chosen as the class of the current pattern.

Based on these classifications, four types of modification occur. First, the weights of all characteristics of all classes may be modified through pairwise comparison with the correct class features. Each characteristic is evaluated on whether it helped or hindered the current classification, and its corresponding weight is increased or decreased accordingly. This is the credit assignment portion of the learning algorithm. Second, each feature value of the current class is replaced by the current pattern value with a probability of .25. Third, the set of window features can be modified by extracting new windows from the current pattern and eliminating those that are deemed no longer useful. This extraction of windows from the current pattern provides a particularly useful set of features. Replacement of windows is

based on a success count which uses the weights associated with a feature to evaluate its contribution toward successful classification of the patterns. Fourth, features can be logically combined to produce more complex features. This program was successfully applied to a number of classification tasks, including letter classification.

Non-linear elements

A means of obtaining the power of a multi-layer network of linear elements using a single layer is to use non-linear elements [Maxwell et. al. 1986]. Instead of using each input individually, the PE can use products of input pairs, input triples etc. as weighted input terms. Use of these higher order terms allows more complex functions to be computed. For instance, if the three features supplied as inputs to a network are two independent values and their product, then the Exclusive-OR (XOR) of the two independent inputs can be computed in a single layer network. On the other hand, the general non-linear unit requires 2^n terms to represent any possible function of n inputs. If a smaller set of terms can be identified as adequate for the task *a priori*, this approach has the benefit of being a single layer network, for which learning can efficiently converge. A particular non-linear unit that has been studied in some detail is the *sigma–pi* unit [Feldman and Ballard 1982, Williams 1986] which conjunctively combines terms prior to computing weighted sums.

Summary of single layer error correction

We can relate these single layer methods in terms of the structure suggested by Rosenblatt for his *Perceptron* model (See Figure 3.1). In the perceptron, the set of fixed features is pre-determined by the fixed connections to the inputs. For n input values, 2^n fixed feature elements are required to represent each possible combination of (binary) input values. If the network has all 2^n fixed features, then any desired mapping from inputs to outputs can be learned by single-layer learning methods. Since providing 2^n fixed elements is unacceptable, the problem is to provide a sufficient subset of those 2^n features to allow the problem to be solved. Rosenblatt suggested the use of a random set of fixed features. Widrow chose the inputs themselves as the fixed features. Uhr & Vossler used patterns that appear in the stimulus patterns as features. Selfridge, Samuel and Maxwell pre-define the fixed features based on what is known of the requirements of the task. In addition, the feature formation procedures of Selfridge, Uhr & Vossler, and Samuel provide means of replacing 'fixed features' as the learning task proceeds.

The convergence proofs for single layer networks make clear the dependency of successful learning on the appropriate choice of fixed features. Learning is 'easy' in single layer networks if the correct set of fixed features is chosen. However, there is no way to obtain perfect performance if a required feature is left out of the set. A number of the approaches just described provide discrete mechanisms for replacing one 'fixed feature' with another. Multi-layer learning algorithms take a more

continuous approach to the problem of replacing fixed features with more useful ones. Using extensions of the credit assignment mechanisms just described, adaptive features are developed to fit the needs of the output layer in its performance of the task.

3.2.2. Unsupervised methods

Error correction learning is one form of supervised learning. In a supervised approach, some form of feedback is supplied to the network as a means of evaluating the current response with respect to a desired task to be learned. Unsupervised learning is a method of adjusting network weights when no external evaluation of the network response is supplied. The first suggestion for such a learning scheme was Hebb's correlational synapse [Hebb 1949]. In his attempt to explain how learning might occur in the brain, Hebb suggested a simple mechanism for adjusting synapse efficacies that could lead to interesting organization. The rule for the correlational synapse can be simply stated as:

$$\Delta w = \eta o i$$

This rule specifies that if a node fires (has non-zero output), each of its weights that is associated with an active input (non-zero input) is increased. Consequently, that node is more likely to fire the next time the same pattern of inputs is presented. The overall result of applying the correlational synapse algorithm is that each node in the network tends to become sensitive to commonly occurring patterns, and a given node will respond to a set of patterns that are all similar to one another. This aspect of the

response is referred to as clustering, since a given node responds to a set of patterns that are clustered in a particular area of abstract feature space. Since the network is able to develop this organization without external feedback, this form of learning has been called self-organization. Much work on self-organizing systems has occurred since Hebb suggested the correlational synapse. A common aspect of this work is that nodes are often combined into groups of mutually competing nodes, in order that they not all develop a sensitivity to the same pattern. Examples of self-organizing networks include a simulation of the retina and primary visual cortex [von der Malsburg 1973], where a self-organizing mechanism develops oriented edge detectors; the Cognitron and Neocognitron models [Fukushima 1975, 1980], which develop a hierarchy of visual feature detectors; Adaptive Resonance Theory [Carpenter & Grossberg 1986], which uses activity at higher layers of a network as feedback to guide the development of features at lower layers; and the competitive learning paradigm [Rumelhart & Zipser 1985], which solves classification tasks.

In the case of supervised methods of learning, algorithms exist which can be proved successful in solving the credit assignment problem for a single layer network. Since the desired response is not specified in unsupervised learning, other criteria are used to decide the effectiveness of an algorithm. For instance, it is desirable for the weight configuration to converge to a stable configuration after a fixed set of patterns has been presented sufficiently many times [Amari 1977]. It is also desirable to have different nodes become sensitive to different patterns in the input such that pattern space is evenly covered by the categories associated with

different nodes [Nass & Cooper 1975]. Another desirable feature of self-organizing networks is to have nearby nodes come to represent similar patterns. This is referred to as a topological network organization [Kohonen 1982, Amari 1983].

An aspect of unsupervised learning algorithms that is of interest to this work is the method of handling the credit assignment problem. In one sense, we can say that an unsupervised algorithm assigns credit at all layers of a network, since it does specify how to modify the weights of each node in the network. This assignment of credit is based on the assumption that the goal of the network is to optimally categorize input patterns according to the similarity of their appearance. To the extent that this assumption supports the true goal of an adaptive network, this form of credit assignment will be effective. On the other hand, since the assignment of credit is completely independent of the task to be performed, unsupervised learning is an insufficient solution to the credit assignment problem.

Unsupervised learning mechanisms provide a feature formation capability that is better than many of those previously described. Instead of random or prespecified features, a self-organizing set of nodes can develop features that commonly occur in the stimulus patterns. For instance, in one modification to the simple perceptron, the fixed weight layer of processing elements was replaced with a layer that could self organize [Block et. al. 1964]. Although there is no guarantee that such features will be superior to random features, we would expect the additional organization to be helpful in solving the learning task.

3.2.3. Multilayer learning methods

We have discussed various solutions to the credit assignment and feature formation problems in single layer networks. Unfortunately, single layer networks are limited in the tasks they are capable of performing [Minsky & Papert 1969]. For instance, using linear sum elements, a single layer network can correctly classify a set of patterns only if the pattern classes are linearly separable. By designing multiple layer networks, we extend the representational power of the connectionist formalism. The task then is one of solving the credit assignment and feature formation problems in a multi-layer network. In their book outlining the limitations of the single layer perceptron model, Minsky and Papert predicted that there would be no useful extension of Rosenblatt's single layer results to multi layer networks. This prediction has proven true to the extent that no algorithm has been developed for which a convergence theorem like that for the single layer perceptron has been found. On the other hand, a number of algorithms for multi-layer learning have been proposed and empirically demonstrated with varying degrees of success.

The problem in assigning credit at multiple layers is to provide appropriate feedback to each node in the network. If the feedback is a reinforcement signal, one means of distributing this information is to broadcast it to all nodes. The A_{r-p} algorithm [Barto & Anandan 1985] is one such reinforcement learning procedure that can be extended in a straight forward manner from single to multiple layers.

Error correction is a more powerful form of feedback for learning, but it complicates the multi layer credit assignment problem. For error correction learning, each node must be provided with a target value or the difference between the target and actual output. Generally, such error information is only known for the output nodes, and must somehow be computed or estimated for nodes in lower layers.

Competitive learning

Unsupervised learning, like reinforcement learning, does not require differential feedback to each element in the network. Rumelhart & Zipser [1985] have demonstrated how an unsupervised learning mechanism can be used to train a two layer network to perform a classification task that cannot be represented in a single layer. The method uses a stimulus pattern that contains training information for the training period itself. Once training is complete, patterns without the training information can be correctly classified.

Boltzmann machine

One method for propagating error signals through the network is that used in the Boltzmann learning algorithm [Ackley et. al. 1985]. The Boltzmann machine is a stochastic network of symmetrically connected elements. Given an initial state, defined by the activation levels of some non-hidden elements, the network relaxes into a final state by a simulated annealing process [Kirkpatrick et. al. 1983]. The network can be trained so that applying a particular pattern of activation levels to the

input nodes (a subset of the non-hidden elements) leads to a desired pattern over the output elements (the remaining non-hidden nodes) in the final state. A measure of the difference between the current configuration of the network, and that needed to produce the desired behavior is the information gain G. The Boltzmann learning algorithm is a gradient descent search of weight space along the surface of G. Individual weight changes for each connection can be computed locally, by comparing the probabilities of activation of the two elements associated with the connection under two situations: when the output elements are clamped to their desired values, and when the output elements are unclamped. The propagation of errors occurs from the output units back through the network over the course of successive iterations, as elements close to the output units learn the appropriate weights needed to support the desired output behavior, and in turn constrain the elements they are connected to. In the actual simulations reported, the gradient of G was not followed exactly, rather fixed size steps were taken in each weight according to the sign of the gradient. The need for an inner relaxation loop in addition to the usual outer pattern iteration loop makes this algorithm relatively slow.

Holland

Holland's bucket brigade algorithm [1986] involves a 'usefulness' measure for each node, called its *strength*. The strength of a node is used as it competes with other nodes for the opportunity to become active. Once activated, the node passes some of its strength back to the nodes that caused it to fire. In addition to this credit

assignment method, Holland suggests a feature formation method called genetic learning which can be applied at all layers of a network. Genetic algorithms use the strength of a node to evaluate its usefulness. Elements having low strength are replaced by features that are 'genetic combinations' of high strength features. These combinations are formed by 'crossovers' of feature pairs involving the use of parts of the specification of each feature in the pair as the specification of the new feature. This is similar to the feature formation method used in Pandemonium.

3.2.4. Backpropagation methods

In error-correction learning, each of the output nodes of a multi-layer network receives a direct training signal from an external agent. This training signal is sufficient to guide the adaptation of the weights of the output nodes by using a single layer error-correction algorithm. One way to provide training signals to hidden nodes is through backpropagation.

Backpropagation uses the training signals for nodes at one layer of the network to estimate a set of training signals for the layer below. The connection weights between the layers are used in the backward direction to weight the training signals in the upper layer when computing estimates for the lower layer. The flow of training information is top-down since training is only explicitly supplied at the output layer. Once a training signal is available to a hidden node, any of the weight modification rules used in single layer networks can be applied.

Generalized Delta Rule

The most well known of the back propagation techniques is called the generalized delta rule [Rumelhart et. al. 1986]. As the name indicates, it is an extension of the delta rule (of Widrow & Hoff) that can be applied to multi layer networks. Like the Widrow-Hoff rule, the generalized delta rule performs a gradient descent search in the squared error of all output nodes in the network averaged over all patterns. There is no guarantee that the globally minimal error configuration will be reached, since the error function may have local minima. We discuss this problem in further detail below.

For output nodes, the Widrow-Hoff rule for adjusting weights can be applied directly (see section 3.2). For hidden nodes, the relationship between output error and weight value is somewhat more complex. Consider the small network presented in Figure 3.4. The form of the error expression is the same as that used in deriving the Widrow-Hoff rule. In this case, we have:

$$E = \tfrac{1}{2}(t_d - o_d)^2 + \tfrac{1}{2}(t_e - o_e)^2$$

Again, the weight change is specified so that it causes a decrease in error along the gradient:

$$\Delta w_{ca} \propto -\frac{\partial E}{\partial w_{ca}}$$

This partial derivative is:

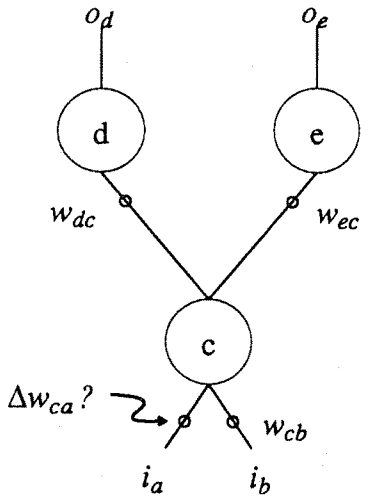


Figure 3.4

$$-\frac{\partial E}{\partial w_{ca}} = (t_d - o_d) f'_d w_{dc} f'_c i_a + (t_e - o_e) f'_e w_{ec} f'_c i_a$$

The term f'_x is the derivative of the activation function for node x. This term does not appear in the Widrow-Hoff derivation since linear elements were used. Linear elements are not useful in building multi-layer networks, however, because any such network would have a single layer equivalent. Instead, elements having non-linear, but differentiable activation functions, like the logistic function, must be used.

The expression of the partial derivative is not particularly useful in the form given above, since it describes a computation that involves terms that are not available to the node being adapted. Rumelhart, et. al. suggested that errors be propagated back through the network to allow local computation of these partial derivatives. Recalling that the error term associated with an output node, x, is:

$$\delta_x = (t_x - o_x) f'_x$$

we can rewrite the previous partial derivative as:

$$-\frac{\partial E}{\partial w_{ca}} = \delta_d w_{dc} f'_c i_a + \delta_e w_{ec} f'_c i_a$$

More generally, we have:

$$\Delta w_{jk} = \eta(-\frac{\partial E}{\partial w_{jk}}) = (\sum_l \delta_l w_{kl}) f'_k i_j$$

By analogy to the Widrow-Hoff rule, we define:

$$\delta_k = \begin{cases} (t_k - o_k) f'_k & \text{for } \textit{output nodes} \\ \sum_l \delta_l w_{kl} f'_k & \text{for } \textit{hidden nodes} \end{cases}$$

Then we have

$$\Delta w_{jk} = \eta\, \delta_k i_j$$

for all nodes in the network.

The estimated error signal for hidden nodes is computed by taking a weighted sum of the errors of all nodes that it connects to, using the same weights used in computing node activations. For a single layer network, the error as a function of a single weight is concave upward with a single point of inflection. Since a hidden weight can influence the error in more than one output node, the total error is a more complex function of that weight. It is the presence of multiple extrema in this function that leads to the problem of falling into local minima.

Parker

The generalized delta rule makes changes on each cycle that reduce the total squared error of the output layer values for the current input pattern. If small enough changes are made to the weights on each cycle, then the mean square error over all patterns will decrease. A related algorithm, called the 2nd order LMS rule [Parker 1987], makes changes on each cycle that reduce the total squared error averaged over all patterns. While the first derivatives of the weights in the LMS rule can be calculated from the error and input values, the first derivative of the weights for the 2nd order LMS rule is a function of the pairwise correlation of the inputs. In order to avoid computing the inverse of the input correlation matrix, R, Parker derives a formulation for the second derivative of the weights in terms of the error, the inputs, the previous weight change and R. Finally, to reduce the complexity of the algorithm ($O(m^2)$ due to calculating R) the average correlations are estimated by the current correlations. The resulting rule for changing weights can be specified as:

$$\Delta^2 w = \alpha(\mu\delta - \Delta o)i$$

where

$$\Delta o = \sum i \, \Delta w \quad \text{and} \quad \Delta w_{t+1} = \Delta w_t + \Delta^2 w_t$$

It is interesting to note that the paper introducing the generalized delta rule [Rumelhart et. al. 1986] also introduced a heuristic term called the momentum. Momentum is used to reduce the problem of ravines in the error function, where larger changes of some weights and small changes in other weights are desired. The

formula for including momentum in the weight change calculation is:

$$\Delta w_{n+1} = \eta \delta i + \alpha \Delta w_n$$

It is clear that the effect of using momentum is similar to that of using the 2nd order LMS rule: both weight and change in weight are stored values, while the change to the weight change is computed on each cycle.

LeCun

A third backpropagation method is that used in the Hierarchical Learning Machine (HLM) of LeCun [1986]. Rather than using the criterion of changing the weight along the steepest gradient of the error function, HLM loosens the constraint by insisting only that each weight change be in the direction that decreases error. That is, the sign of the weight change must match the sign of the gradient. One way to satisfy this constraint is to compute a desired output for each internal node, based on the desired output of its successors. This computation is exactly analogous to the propagation of errors in the generalized delta rule, that is:

$$t_k = \sum_l t_l w_{kl}$$

Given the target output for every node in the network, the weight changes can be computed using the Widrow-Hoff rule.

LeCun has used this algorithm successfully in training a six layer network to classify letters of the alphabet. He points out, however, that a good deal of manual control over the learning coefficient and the pattern of input presentations was required to avoid local minima while maintaining an acceptable learning rate.

3.3. Discussion

3.3.1. Summary of previous work

In a network learning task, the goal is to determine a set of connection weights between network elements such that the network produces a correct output response for any input pattern presented. Single layer learning algorithms that use a set of fixed features as inputs are guaranteed to converge to a configuration of weights that satisfies the task, if such a set exists. Although having a proof of convergence is desirable, the condition that the problem be a linearly separable one limits the application of this proof severely. One way to satisfy the condition is to choose input features which are known *a priori* to satisfy linear separability. More practically, a set of input features are chosen that are 'hoped' to satisfy the condition. The first problem with this approach is that it requires much of the difficult part of the learning task to be done before learning even begins. The other problem is that if these input features are not chosen correctly, the algorithm has no chance of successfully completing the task.

To avoid the problem of having to carefully choose the set of input features, a variety of feature formation methods have been suggested. In many of the earlier systems, a single layer learning algorithm was used, but input features could be replaced by new features. The replacement algorithms varied, but generally depended on some measure of usefulness associated with each feature. Uhr & Vossler use a success count. Holland uses a 'strength'. Selfridge uses a worth based

on output weights. Samuel uses a low-tally count. In all cases, a low usefulness for a feature marks it for replacement by a new feature.

Another way to form features is to apply the same weight modification schemes that are used in learning the output representations. Multi-layer networks accomplish feature formation by representing features as weighted sums of sub-features, which in turn may be weighted sums of more primitive sub-features. Various multi-layer learning algorithms have been developed, which form new features by assigning credit or blame to each weight in the network whenever training information is provided. This credit assignment problem is more difficult than the single-layer case because there is no simple heuristic, such as gradient descent, that will guarantee the success of learning, even when a satisfactory weight configuration is known to exist.

Of the methods that have been applied to multi-layer learning tasks, backpropagation of errors has been found to be the most effective [Anderson 1986]. Of the variations on backpropagation, the best known and most studied is the generalized delta rule.

3.3.2. Weakness of backpropagation

Despite the relative effectiveness of backpropagation techniques, they have a number of weaknesses. We address three major shortcomings of the generalized delta rule (GDR) in this work.

The strict generalized delta rule (without momentum) is a gradient descent search through the space of all network weights, using the sum of the squared error of

all output nodes as the optimization criterion. One problem with any gradient descent method is that non-optimal stable configurations may result. In fact, it is well known that the error surfaces of multilayer networks are not unimodal. Thus, although the algorithm may solve a given network learning task, there is no proof in general that the learning procedure will converge to a correct solution, even when it is known that such a solution exists. Gradient descent is a local, iterative optimization procedure, that fits well the needs of a highly parallel, on-line learning capability. The locality of information available to the algorithm has two aspects. First, each weight adjustment is based on activity in the network that results from the single current input pattern. The error gradient computation is therefore based on the error surface associated with a single pattern rather than the mean error over all patterns. The addition to the algorithm of the momentum term addresses this problem, by averaging the weight changes over multiple patterns. Second, the contribution to the error of each weight is computed based on the assumption that all other parts of the network remain constant. This extreme locality of information usage makes it difficult for related parts of the network to learn in a coordinated fashion. Our goal in extending this method is to overcome the deficiencies in information available to the learning mechanism while maintaining the local iterative character.

Another problem with GDR is that the error signals to lower layers in the network are weak. Since the top layer gets direct training information and the penultimate layer computes error estimates from these direct signals, the quality of the training information is relatively high at these layers. As each successively lower

layer computes estimates of the error based on previous estimates, the training information gets successively weaker. For this reason, backpropagation techniques have rarely been applied to networks with more than two adaptive layers.

A third drawback of GDR is the tendency of nodes to 'drop out' of the network. Since the error estimate for a hidden node is proportional to the weights connecting it to nodes at the next higher layer, a given node will receive small error signals if it is connected only weakly to higher level nodes [Sutton 1986]. The node receiving a small error signal will change its feature sensitivity very little. However, a low value for output weights is an indication that the node is not providing a useful function to the network. The effect of backpropagation is to allow useless nodes, those having low output weights, to remain useless due to lack of feedback.

We take these three deficiencies of backpropagation as the major obstacles to improving the performance of the technique. In the next section we discuss some means of overcoming these problems.

3.4. New learning algorithms

A major hypothesis of connectionism is that a large number of autonomous agents can work together in such a way that some desired global behavior is produced. The trick is to define the behavior of the elements in such a way that their interactions lead to interesting macro behavior.

In order to improve the behavior of the learning algorithms just described, we suggest an alternative to this hypothesis. The *cluster hypothesis* is that a more

powerful 'element', still working autonomously, can be more efficient in producing the desired global behavior. This more powerful element is a cluster of simple processing elements that interact in such a way that they behave in a coordinated manner. The clustering of processing elements is the basis for the competitive learning work of Rumelhart & Zipser [1985], and the winner-take-all networks of Feldman [1979]. Our view is that the cluster concept should be extended to include more complex behaviors. The learning algorithms described here take a step in this direction.

3.4.1. Error modification

A local minimum in the error function of the network corresponds to a non-optimal weight configuration in which a small change to any single weight in the configuration leads to an increase in output error. Under these conditions, a gradient descent procedure will specify that no changes be made to the network weights, and the configuration will persist indefinitely. There are various methods for avoiding this problem. One approach is to detect that a local minimum has been reached, and to escape by making a drastic change to some part of the configuration. Such an approach requires a certain amount of global control in detecting and correcting the condition, and shares with other random search methods the disadvantage that even incremental improvements in behavior over time cannot be guaranteed.

Another approach is to add noise to the changes that are specified by the gradient descent process. This eliminates the problem of detecting the local

minimum condition. In addition, by reducing over time the amount of noise added, more desirable convergence behavior is obtained (cf. [Barto & Anandan 1985]). This is the basis for the simulated annealing method [Kirkpatrick et. al. 1983] used in the Boltzmann machine learning algorithm [Ackley et. al. 1985]. Our approach is to avoid the occurrence of local minima by discouraging the types of network configuration that correspond to these minima. In particular, we address the problem of deadlocks.

The simplest form of deadlock occurs when two equipotential nodes develop a sensitivity to the same pattern. We say that two nodes are equipotential if they can represent the same function in the network. This is strictly the case only when both nodes have exactly the same set of inputs and same set of output destinations. The deadlock problem can also occur in nodes that only partially overlap in their potential functions. In a network where the number of nodes available to represent features is limited, a deadlock may leave a necessary feature unrepresented. Each node attempts to minimize its own individual contribution to the network error, preventing either one from representing the missing feature. These deadlocks are difficult to break for extremely local methods, because they cannot be detected nor overcome by individual nodes.

To reduce the occurrence of deadlocks within a cluster of nodes, we modify the error signals that are backpropagated to each node, such that duplication of function among the nodes is discouraged. Instead of using the error signals propagated to each

node directly, error modification methods compute a new set of error signals for each node in a cluster, based on the entire set of errors received by that cluster.

Error modifications within a cluster are based on a competition among the nodes for weight adjustment. This competition provides a means of enhancing the contrast among the error signals received by each node, and this in turn enhances the contrast among the features represented by the nodes. The competition is based on the magnitude of the backpropagated error signal that is initially computed for each node. For instance, each error can be computed in terms of the set of backpropagated errors as:

$$\delta_j = e_j - \frac{1}{n-1} \sum_{k=1}^{n} (e_k - e_j)$$

where e_k is the backpropagated error and δ_j is the modified error. This error modification mechanism can be interpreted as an additional local constraint which is added to the error minimization constraint that normally drives the learning procedure.

In the next chapter we consider related mechanisms that further enhance the contrast between nodes within a cluster. This contrast among the errors applied to nodes in a cluster prevents the assignment of identical sequences of error signals to multiple nodes in the cluster. The resulting error signals discourage multiple nodes from developing sensitivities to the same features of their input patterns.

Error modification works by adding local structure to the weight adjustment rule. This additional structure is gained at the expense of the guarantee that the

network error will be decreased at every step. On the other hand, decreasing the error on every step has clear disadvantages when the error surface contains local minima.

3.4.2. Error augmentation

Another weakness of backpropagation mentioned previously is that training signals are weak in the lower layers of the network. This is particularly true in the early stages of learning, due to the small random initial weights of the network. However, even as the network develops stronger weights in the upper layers, the lower layers must respond to the sometimes conflicting needs of nodes in the upper layers. The result is that error signals reaching the lower layers provide relatively weak information to the weight modification computation.

The goal of error augmentation methods is to improve the quality of the weight modifications occurring in the lower layers, by augmenting the error signals propagated down from the upper layers. The additional 'error' component is generated bottom-up, using a self organizing mechanism. The goal of self-organization is to develop features that are likely to be useful to other parts of the network. In particular, in the absence of any training signal, a useful function for a set of nodes to provide is the ability to discriminate the different input patterns that are presented to the nodes.

Just as with error modification, we can formulate an error augmentation mechanism as an additional constraint on the error minimization function performed by the backpropagation procedure. By maximizing the criterion function

$$D^k = \tfrac{1}{2}\sum_j \sum_l (o_j - o_l)^2$$

each node tends to develop a sensitivity to a different input pattern. The following weight change rule implements a change along the gradient of this measure:

$$\Delta w_{jk} = \gamma \sum_l (o_k - o_l) f'_k i_j$$

This corresponds to a form of competition among the nodes similar to that resulting from laterally inhibiting connections. Instead of competing for activation strength the nodes are competing for weight adjustment strength. There are many other ways to implement a self-organizing learning algorithm.

One way to combine this weight modification term with that due to error backpropagation is simply to add the two directly, as suggested above. The problem with this approach is that the top-down and bottom-up pressures may be inconsistent, causing a deadlock between layers. In order to blend the two components effectively, some means is needed of deciding when one component should prevail over the other. In Chapter 4, we simulate a number of combining mechanisms, all based on the *output weights* of the nodes.

We use the term output weight to refer to a measure of how strongly the output of a node is connected to node inputs in the next higher layer. Two potentially useful measures are the average weight magnitude of all outgoing connections, and the maximum weight magnitude over all outgoing connections. Any network learning algorithm adjusts the weights such that node A connects more strongly to node B in

the layer below to the extent that the activity of node B is useful in defining the function of node A. The significance of the output weight, then, is that nodes representing features that are useful to the next higher layer will develop high output weights. Nodes representing less useful features will have low output weights. Thus, output weights correspond to the usefulness measures used in earlier learning algorithms. The output weight provides a basis for combining the error correction and self-organization components.

When a node has a low output weight, it is not providing a significant contribution to the network computation. In addition, it is not receiving a strong training signal, since the training signal is propagated through the output weight. Such a node is a good candidate for self-organization. When a node has a high output weight, it is contributing significantly to the network function, and is receiving a strong training signal through its output weight. In this case, error correction is the appropriate rule for changing the weights.

Error augmentation, mediated by output weights, provides a solution to another problem with backpropagation. As previously described, a node whose output weight approaches zero receives very little training when using a backpropagation algorithm. Due to the small training signal, the node will not change the function it is performing. But the low output weight is an indication that the node is not performing a useful function. Therefore, the useless node will remain forever useless. Error augmentation addresses this problem by applying self-organization to

these useless nodes, until they develop a function that is useful enough to nodes at higher layers that they develop strong connections to the node.

3.5. Summary

The problem of learning can be decomposed into two major sub-problems, feature formation and credit assignment. Feature formation involves the determination of a set of features that can be used to adequately describe the concepts to be learned. Credit assignment involves the determination of how each concept is defined in terms of the feature set.

Single layer network learning algorithms treat these two problems separately. A variety of feature formation procedures have been studied, including the use of randomly defined features, raw input values, extractions of actual input patterns, predefined features, and numerous combining functions applied to these simpler features. Credit assignment in single layer algorithms adjusts the concept definitions by changing the strength with which a given feature is associated with a concept.

In multilayer network learning algorithms, the feature formation task is subsumed by credit assignment. Features are formed as weighted combinations of sub-features. These weights are modified in the same way that strengths of association are modified in defining concepts in terms of features. The most well known, and best performing of the multilayer learning algorithms is the generalized delta rule (GDR). This algorithm assigns credit and blame to network connection weights by computing the marginal network error contributed by each weight in the

network, and modifying each weight such that the error is reduced. This computation requires that error information be propagated to each node in the network, from the output nodes back toward the input nodes.

One problem with the gradient descent method of reducing errors used by GDR is that the network may reach a state where errors are not zero, but can no longer be reduced in this incremental fashion. We have presented a partial solution to this problem in the form of an error modification method. An error modification algorithm involves a competition among a set of nodes for the error signals being provided by the GDR. The set, or cluster, of nodes then learns as a single unit instead of as individual units. The resulting coordination of nodes within the cluster tends to avoid the deadlocks among nodes that lead to local error minima.

Another problem with propagating errors back through the network is that the error signal becomes weak as it propagates to the lower layers of the network. We have addressed this problem by introducing error augmentation methods. An error augmentation algorithm also involves competition within a cluster of nodes. In this case, the competition is based on the node activations themselves, rather than the error signal. A self-organizing mechanism computes useful weight changes based on input patterns received by the nodes in the cluster. These weight changes are combined with those dictated by the error signals based on the output weights of the cluster nodes.

The use of the output weight to mediate the combination of self-organized and error correction components also provides a solution to a third drawback of GDR. A node with a small output weight is ignored by a pure backpropagation method, because nodes receive their error signals through their output weights. Such a node provides no useful function to the network. The use of the output weight in the error augmentation algorithm allows these nodes with small output weights to change due to self-organization. Thus, otherwise useless nodes can be made useful again.

In Chapter 4, we present a number of network simulations that demonstrate the properties of these new learning methods.

Chapter 4

Learning Experiments

In this chapter, we present a number of experiments involving the applications of various learning algorithms to a number of network tasks. The purpose of these experiments is to demonstrate the behavior and shortcomings of the standard backpropagation algorithm (GDR), and to provide evidence of improved behavior through the use of error modification and error augmentation mechanisms. In the next chapter, we will apply the algorithms developed here to the vision problem described in the first chapter.

4.1. Background

4.1.1. Empirical study of learning

Network behavior can be studied analytically and empirically. Empirical approaches suffer from their dependence on the particular problems that are used in testing the algorithms and on certain details of the implementation. For instance, the connection weights may be initialized to random values at the beginning of an adaptive simulation. Since the simulation results depend on the initial weight configuration, multiple runs using different initial states must be used to ensure that conclusions are independent of these initial states. Other factors affecting simulation performance include values of various parameters affecting learning rates, details of the connectivity patterns between network layers, and the statistics and presentation

of input patterns.

Formal methods for analyzing the behavior of network learning algorithms are weak due to a lack of appropriate mathematical methods. Proofs of convergence are difficult to develop, and require severe restrictions on the algorithm and task. In some cases, it is known that convergence cannot be guaranteed in the general case. Here, average behavior under certain types of task characteristic is of more interest, but much more difficult to formalize. In fact, proofs of convergence are of little practical value, though of great theoretical value, when it comes to implementing 'useful' learning algorithms. The problem of choosing appropriate representations for the type of information that will 'typically' be represented is equally difficult to formalize.

In order for empirical studies to provide useful information, care must be taken in choosing appropriate tasks and avoiding conclusions based on details of the implementation that are not significant to the task.

Our approach has been to define a network for a specific task and to consider the performance of a class of learning algorithms with respect to the requirements of that network and task. To the extent that our vision problem has requirements similar to those of other tasks, our conclusions can be regarded as applicable to a wider range of problems than the particular ones addressed here.

To begin our study of learning, we take as our base method the generalized delta rule (GDR) of Rumelhart, Hinton and Williams [1986]. This algorithm is one

example of the use of backpropagation of errors that has been shown to perform better than a number of alternative approaches to the credit assignment problem in multi-layer networks [Anderson 1986]. Using this algorithm, learning experiments have been run on a number of small networks to demonstrate its behavior and to identify some of the ways in which it fails. This leads to a number of potential improvements to the algorithm, which are tested out on the small networks. The goal of the chapter is to develop an improved learning algorithm which can be applied to the larger network described in Chapter 5.

The simulations are written in the C programming language, and are run on a DEC Microvax II workstation under the UNIX operating system. The Xlib window support package is used to display graphic representations of the current values of some or all of the state variables in the network, as well as the time course of a small number of these variables.

4.1.2. General network structure

All of the networks described in this chapter have the same general structure, each being an example of a layered feedforward network. A given network consists of a primary network and possibly an additional context network. The primary network consists of $L+1$ layers, where layer 0 contains input units whose activations correspond to the current input pattern being presented to the network. For layers 1 to L, each processing element within the layer receives inputs from elements in the layer below. Each processing element maintains a single real-valued connection

weight for each of its inputs. Network output is provided by the activations of all elements in layer L. The context network consists of an input layer, layer 0^c, and a single layer of adaptive elements, layer 1^c. Layer 0^c consists of input units whose activations reflect context information provided externally. Layer 1^c receives inputs from elements in layer 0^c. The activation values of elements in the context network are used to modify the computations taking place in the primary network. This is accomplished by conjunctively connecting context elements and primary elements of one layer to provide input to the succeeding layer of the primary network. This general structure is indicated in Figure 4.1, where conjunctive connections are symbolized by the open circle connection.

4.1.3. Network processing element

The processing unit used in these simulations is the logistic element (see section 1.2). A processing unit must perform two types of computation. First, it computes its activation, or output value, based on the weighted sum of its inputs. Second, it

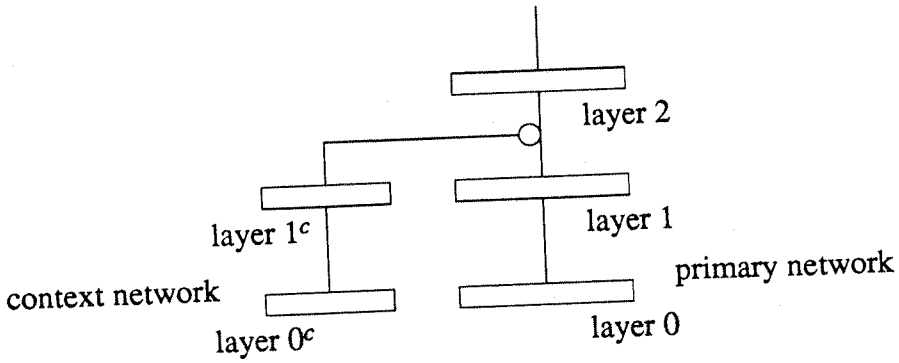


Figure 4.1

computes the changes to be made to its weights, based on input values, activation value and training information. More complex activation and learning behavior can be obtained by enhancing the computational capabilities of the element (see [Feldman & Ballard 1982]). One way to increase the capabilities of the PE is to use state variables that provide additional information to the PE computations. In particular, we will make use of an output trace variable in one of the learning algorithms described in this chapter. An output trace variable is used to track the recent activation history of an element. The output trace, o', is calculated as follows:

$$o'_{(t)} = \alpha\ o'_{(t-1)} + (1-\alpha) o_{(t-1)}$$

where o is the instantaneous activation value of the unit.

The trace value of the output has a number of uses. Sutton [1984] uses trace values in his approach to the deferred outcome problem, where training signals associated with each network behavior are not immediately available. In our simulations, the output trace is used to improve the self-organizing component of learning, and is described in section 4.3.

A network simulation consists of a series of network activations and weight modifications. A single multi-cycle network simulation proceeds as follows (see Figure 4.2).

At the beginning of a simulation run, all connection weights are initialized. Weights that are adaptive are initialized to random values chosen from the uniform distribution (-.25,.25). Weights that are not adaptive are initialized to fixed values

Simulation Program

```
Initialize weights; cycle = 0;
while (cycle < totalcycles) do
  foreach (pattern)
    foreach (layer) from 1^c,1 to L
        foreach (node)
          compute activation
    foreach (node in layer L)
        provide desired activation value
    foreach (layer) from L to 1,1^c
        foreach (node)
          compute error
          update weights
  increment cycle;
  next pattern;
end while;
```

Figure 4.2

according to the desired function of the corresponding nodes. In the simulations described in this chapter, all weights are adaptive. Some of the simulations described in Chapter 5 use fixed weights in some layers while demonstrating the learning characteristics of other layers.

Stimulus patterns are presented to the network as a pair of patterns of activation in the input layers (layer 0 and layer 0^c). Patterns are presented in a fixed sequence which is repeated throughout the simulation. This fixed sequence involves a change on every cycle to each of the patterns in layers 0 and 0^c such that all desired combinations of these patterns are included in the sequence.

Once a stimulus pattern is chosen, node activations can be computed. Since we consider only feedforward networks, computation of activation levels proceeds from

the bottom up. Activations in the context layer (1^c) are first computed, then activations in layer 1, then layer 2 then successively higher layers up to layer L. An error signal for each output node in layer L is computed as the difference between the actual node activation and the desired response corresponding to the stimulus pattern presented.

Based on these error signals, the weights in layer L are modified, and the error signals for layer L-1 computed. Modifications to other layers proceed from the top down as error signals are propagated back through both the primary and context networks. Following the weight modification phase, a new stimulus pattern is presented and the activate-update cycle repeats.

In addition to representing all the network variables, and controlling the sequence of pattern presentations, node activations, error computations and weight modifications, the simulator program also maintains the results of certain intermediate calculations and computes additional measures describing the network behavior. These values are useful in studying the overall behavior of the network structure and the learning algorithms. The measure that is used to report the results of simulations is performance. The performance of the network at any given time is computed as the number of correct responses in the most recent 100 pattern presentations. Node output values range from 0 to 1, and desired values of output nodes are chosen to be .1 or .9. A correct response is one in which all output nodes are within .2 of their desired values.

4.2. Error modification

In Chapter 3, error modification methods are proposed as a solution to the local minima problem exhibited by the GDR. Error modification uses competitive interactions among nodes in a cluster to modify the error signals received by those nodes. This leads to an improved local structuring of those nodes within the cluster. In this section, we demonstrate these methods by simulating three network tasks.

The first simulation involves a two-layer network that computes an Exclusive-OR (XOR) of its inputs. We use this simulation to study the interactions among adapting nodes that lead to the deadlock problem described in the previous chapter. This study motivates the application of a number of new learning mechanisms, whose behaviors we compare with each other and with the GDR. In the second simulation, the 3-bit rotate network, which includes a context layer, demonstrates the usefulness of error modification techniques in representing context information. The third simulation extends the application of error modification techniques to a somewhat larger network, which is related to the 2-D Translation network studied in Chapter 5. This more difficult task leads to the development of more effective error modification mechanisms.

4.2.1. XOR

We begin our detailed description of the simulations by considering a network for solving a two input Exclusive-OR (XOR) task. The desired behavior of the network is defined in the following table:

Inputs		Output
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

It is well known that despite the apparent simplicity of this function, a single layer network using linear input nodes is incapable of representing this computation. Of several possible two layer networks that we might choose to represent this problem, the smallest sufficient feedforward network in which only adjacent layers are connected is that shown in Figure 4.3.

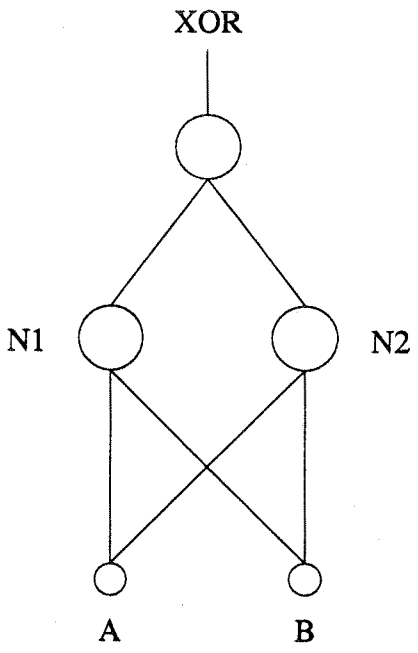


Figure 4.3

The XOR task has been chosen for study for a number of reasons. One, it uses a very small network. This allows the details of interactions to be studied more easily so that the behavior of the whole network can be understood in terms of the information being passed among individual nodes. Two, despite its apparent simplicity, this task proves difficult to learn for the types of adaptive algorithms in which we are interested. In particular, this network often exhibits the deadlock problem that we address in this section. Three, this task has become well-known and well studied by other workers in the connectionist learning field. It therefore provides a point of reference within this field for interpreting the results reported here.

We do not claim that the XOR task is one that is inherent in the problem of machine vision, and must therefore be solved. Rather, we are interested in the fact that deadlocks occur in this task as they do in the vision task described in the succeeding chapter.

In the network of Figure 4.3, layer 0 consists of input nodes representing the values of inputs A and B. Layer 1 consists of two logistic elements each having as inputs the activations of both layer 0 nodes. Layer 2 consists of a single logistic element having as inputs the outputs of both layer 1 nodes. The output of the network, XOR, is represented by the activation level of the single layer 2 node.

Since infinitely large weights are required to produce outputs of 0 and 1 in a logistic element, the desired values used to train the network are .1 (instead of 0) and

.9 (instead of 1). The error signal, δ, supplied to the output node is this desired value minus the actual activation value of the node. Error backpropagation with momentum is used, as described in [Rumelhart, et. al. 1986]. Except where indicated, we have used a learning coefficient, η, of .3 and a momentum coefficient, μ, of .9 in all of the simulations described. The weight adjustment rule is:

$$\Delta w_{ij}(n) = \mu \Delta w_{ij}(n-1) + \eta x_i(n) \delta_j(n)$$

We are ready now to present the results of simulating this network, first using the GDR learning algorithm, and then using a number of error modification algorithms. Figure 4.4(a) shows the initial random weights of the network for the simulation we will refer to as testcase A. Values inside the nodes represent bias weights, others represent connection weights. Figure 4.4(b) shows the weights after 300 cycles, where a single cycle corresponds to one pattern presentation. The input patterns are presented in the order 00,01,10,11 repeatedly. Figure 4.4(c) shows the weights after 600 cycles. Figure 4.4(d) shows the time course of the mean squared error (MSE) of the network, which is computed as:

$$MSE = \sum_{j \in O} (t_j - y_j)^2$$

where O is the set of all output nodes. Figure 4.4(d) also shows the time course of the network performance.

In this simulation, the learning algorithm was successful in finding a weight configuration that reduced the network error to zero within 600 cycles. The number of presentations needed to train this network (150 per input pattern) is an indication

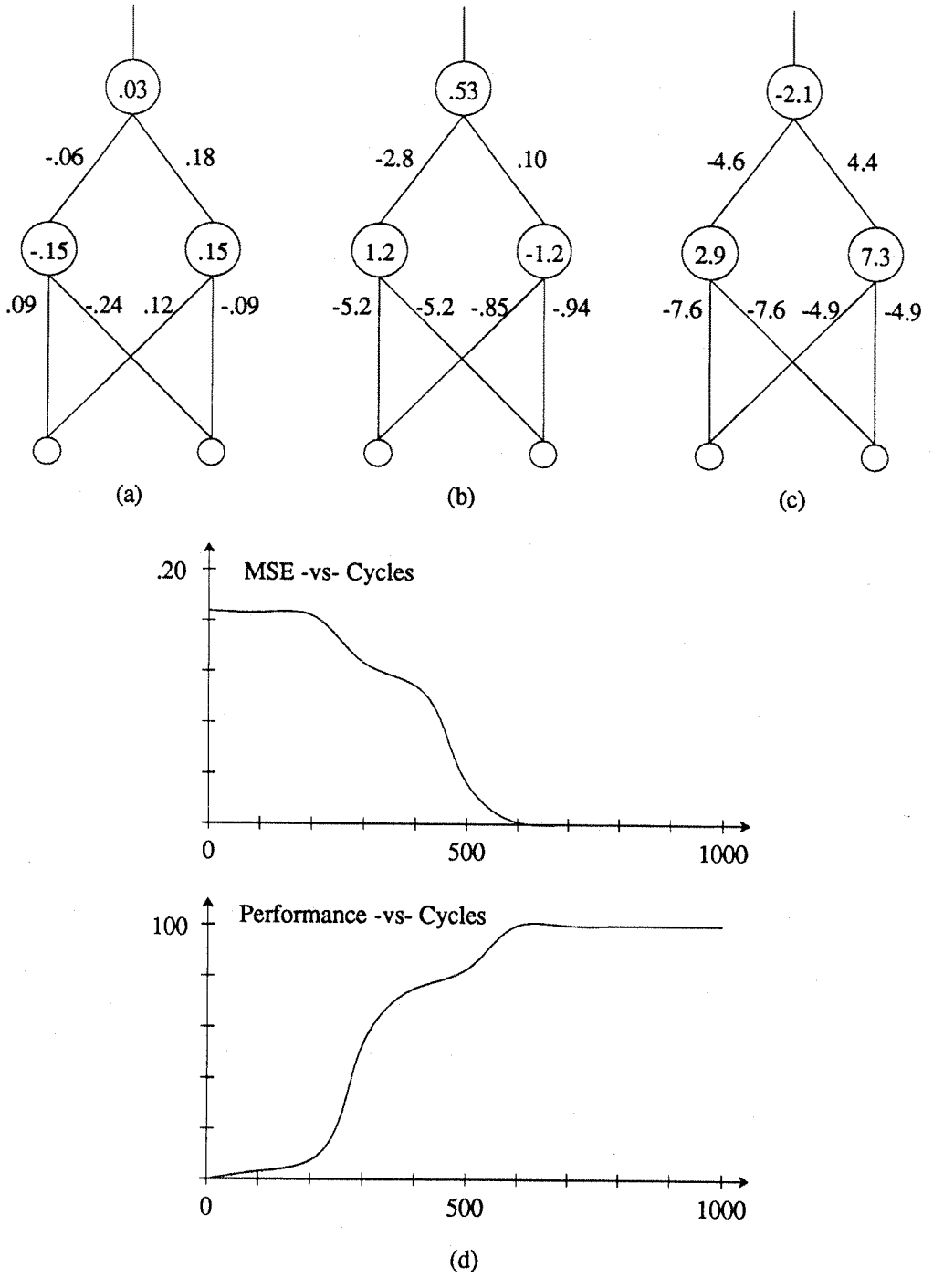
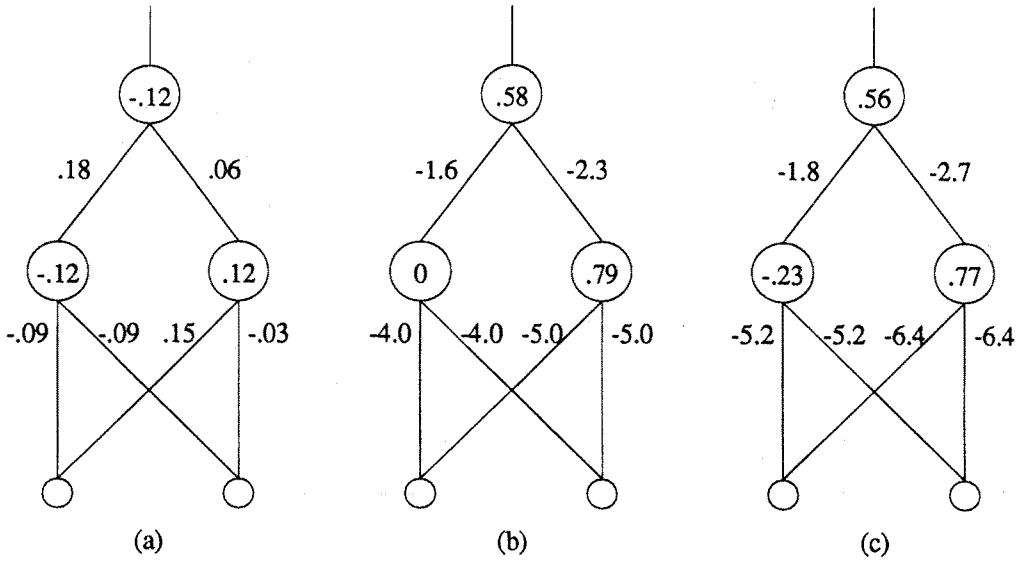

.03
-.06
.18
-.15
.15
.09
-.24
.12
-.09
(a)
.53
-2.8
.10
1.2
-1.2
-5.2
-5.2
-.85
-.94
(b)
-2.1
-4.6
4.4
2.9
7.3
-7.6
-7.6
-4.9
-4.9
(c)
.20
MSE -vs- Cycles
0
500
1000
100
Performance -vs- Cycles
0
500
1000
(d)

Figure 4.4

of the difficulty of the problem. Using the same network parameters, an OR task can be solved in under 150 cycles. By increasing the learning coefficient, which degrades the performance of the XOR task, the OR task can be solved using 10 presentations of each pattern.

The XOR problem is difficult because neither of its two inputs, taken alone, give any information on the desired network output. This results in weight changes that tend to cancel each other out when averaged over the four input patterns. The network then effectively performs a random search based on fluctuations due to initial weight values, until a particular combination of weight changes is found that reduces the network error. In the simulation just presented, these changes eventually lead to a decrease in error to zero. In other simulations, a weight change is found that reduces the error partially, but results in a configuration that allows no further error reductions.

Figure 4.5(a) shows the same network with a different initial configuration, which we refer to as testcase B. Figure 4.5(b) and (c) show the network weights after 400 and 600 cycles, respectively. Figure 4.5(d) shows the time course of the network error and performance. Here we see that after 600 cycles, the learning algorithm has been unable to reduce the network error to zero. In fact, running the simulation for 10000 cycles does not improve the network performance. This failure is known as falling into a local minimum, and can be understood by studying the error signals being propagated through the network, and the effects those error signals have on the


-.12
.18
.06
-.12
.12
-.09
-.09
.15
-.03
(a)
.58
-1.6
-2.3
0
.79
-4.0
-4.0
-5.0
-5.0
(b)
.56
-1.8
-2.7
-.23
.77
-5.2
-5.2
-6.4
-6.4
(c)

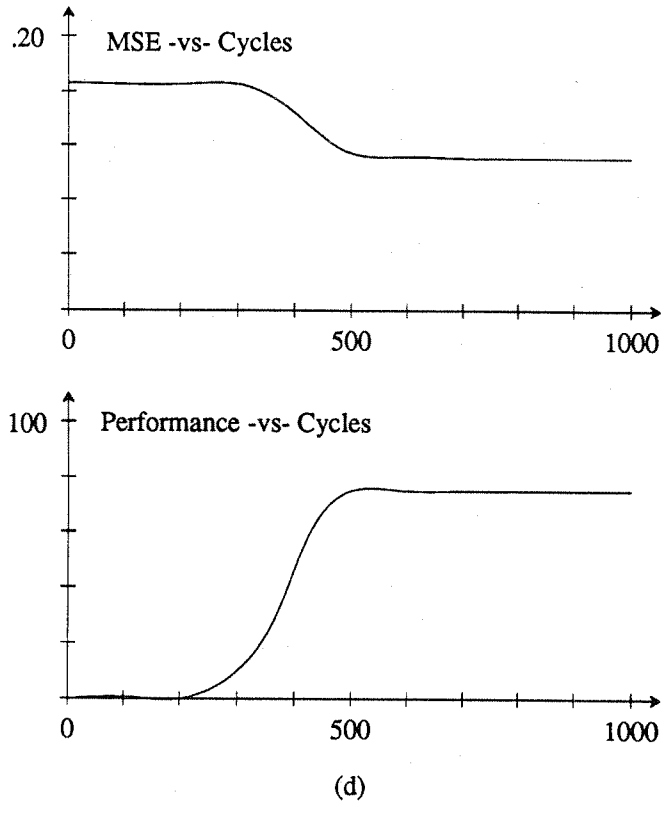

.20
MSE -vs- Cycles
0
500
1000
100
Performance -vs- Cycles
0
500
1000
(d)

Figure 4.5

weights.

For testcase A after 600 cycles, the response of each of the three nodes in the network to the four input patterns and the resulting error signals to the two hidden nodes are:

Inputs		Hidden		Error		Output
A	B	N1	N2	N1	N2	XOR
0	0	.94	.99	10	-9	.12
0	1	.01	.90	-19	19	.86
1	0	.01	.90	-20	20	.86
1	1	.0	.07	22	-21	.15

For testcase B after 600 cycles, the response of each node to the four input patterns and the resulting errors are:

Inputs		Hidden		Error		Output
A	B	N1	N2	N1	N2	XOR
0	0	.43	.67	3	5	.1
0	1	.0	.0	-45	-65	.63
1	0	.0	.0	-44	-64	.63
1	1	.0	.0	108	155	.65

In testcase B, where the local minimum is reached, the problem is that both hidden nodes have come to represent the same function of the inputs. This is an intra-cluster deadlock, as described in the previous chapter. The function represented is approximately a logical NOR. In testcase A, N1 takes on this logical NOR function, while N2 takes on the logical function NAND. Since a single node can compute any Boolean function of two variables except XOR and its inverse, these two functions in the second case are sufficient to allow the second layer node to

compute the XOR function: A XOR B = (A NAND B) AND_NOT (A NOR B).

As discussed in Chapter 3, the approach we take in trying to improve learning performance is to modify the error signals received by hidden nodes, in order to bias the representation in particular ways. In this first demonstration of error modification, we use a simple form of competition between the two hidden units to encourage them to develop different feature sensitivities. In particular, to avoid giving the same error signal to both nodes in layer 1, we impose the following modification to the error signal. Let

$$e_j = \sum_{k \in C_j} w_{kj}\, \delta_k$$

be the backpropagated error to node j. We compute the modified error as:

$$\delta_j = \begin{cases} e_{\max} & \text{if } j = \max \\ -1/4\, e_{\max} & \textit{otherwise} \end{cases}$$

where max is the node receiving the largest error in absolute value.

Thus, the two nodes always receive errors of different polarity, with the strongest error value unchanged.

If we now repeat the simulation for testcase B, using this modified weight update rule, we obtain 100% performance by cycle 800. In fact, starting in the local minimum represented by the weight configuration of testcase B after 600 cycles, the network is able to reach 100% performance within 400 cycles using the modified rule. Using this same error modification rule with testcase A leads to 100%

performance by cycle 600, as before.

The comparison of these two testcases is indicative of, and substantiated by the results obtained when running a set of 25 testcases, each with a different randomly assigned initial weight configuration. The 25 testcases were each run for up to 2000 cycles using each of four learning rules. Network performance is measured at the end of every 100 cycles. The results of these simulations are presented in Table 4.1.

The rule referred to as Mod#0 in Table 4.1 is that described in section 3.4.1. This rule implements a form of competition between the nodes based on mutual inhibitions. This is a fairly weak form of competition, since the enhancement of contrast is proportional to the actual difference between values. Two values that are close to each other will not be changed significantly by this rule. The results in Table 4.1 show that this rule is ineffective in reducing the deadlock problem.

The rule referred to as Mod#1 in Table 4.1 is that just described, where error polarities are forced to be different. This much stronger form of competition between the nodes results in nearly all testcases reaching 100% performance. The Mod#2 rule implements a somewhat weaker form of competition, where the magnitude of the non-maximum error signal is reduced by a factor of 4, but the polarity is not modified:

$$\delta_j = \begin{cases} e_j & \text{if } j = \max \\ 1/4\, e_j & \textit{otherwise} \end{cases}$$

This simplification of the first rule also breaks the symmetry of equal net error signals

Table 4.1 - Comparative learning performance on XOR task
# of cycles to reach 100% performance

Testcase	GDR	Mod#0	Mod#1	Mod#2	Mod#3
A	600	50%*	600	700	1000
B	75%*	700	800	900	75%*
C	600	50%*	500	600	1300
D	75%*	600	700	800	1100
E	600	50%*	500	600	2200
F	700	50%*	50%*	800	1300
G	600	500	500	700	75%*
H	75%*	700	700	800	1300
I	600	500	500	700	75%*
J	600	500	500	600	2200
K	75%*	600	700	700	1100
L	75%*	75%*	700	700	1100
M	75%*	600	700	800	1300
N	75%*	700	600	800	1900
O	600	50%*	600	700	1100
P	75%*	600	600	700	1900
Q	1100	500	600	700	75%*
R	600	50%*	500	600	75%*
S	75%*	75%*	600	700	1600
T	75%*	700	800	900	1600
U	600	50%*	500	700	75%*
V	500	50%*	500	600	1200
W	75%*	500	600	600	1000
X	600	500	500	700	75%*
Y	75%*	75%*	700	800	1200

* performance at cycle 2000

to the hidden nodes, but does not ensure that polarities differ. The third rule, Mod#3, is simpler yet. Here all non-maximum errors to the hidden layer are set to zero. From the results presented in Table 4.1, it appears that this third rule is much weaker than the other two. It yields more testcases that reach 100% performance within 2000 cycles than GDR, but converges much more slowly than all the other rules. The first

two error modification rules, on the other hand, are both effective in improving the convergence behavior of testcases that did not reach 100% performance using the GDR. Except for the single testcase in which 100% performance was not achieved by Mod#1, this rule resulted in a convergence time that was as good as or better than any of the other rules. These simple error modification mechanisms work well in this small network. In the following simulations we consider somewhat more complex networks, whose structural characteristics are similar to those of the network described in Chapter 5.

4.2.2. 3-bit Rotate

The 3-bit Rotate network consists of 3 input nodes and 3 output nodes in the primary network, plus 7 input nodes and 3 hidden nodes in the context network (see Figure 4.6). The desired behavior of this network is to produce a 3-bit output that is a

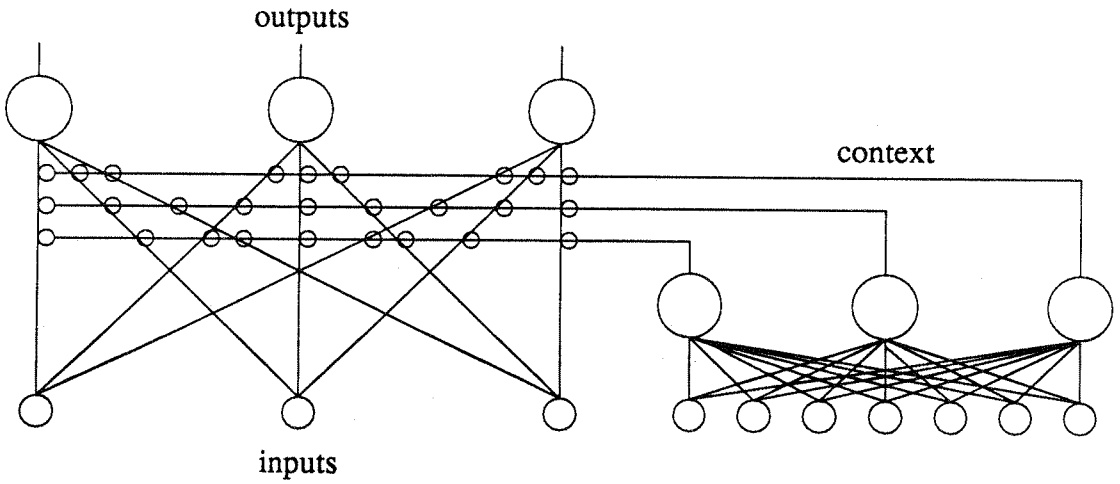


Figure 4.6

rotated version of the 3-bit input. The desired rotation of the input bits is determined by the context input. There are 3 possible rotations of the 3 input bits, which we can refer to as *rotate–left* (L), *rotate–right* (R) and *no rotation* (N). For this task, we will use the seven context inputs, in such a way that only one is active at any given time. The desired rotation associated with each of the seven context inputs is as follows:

L	N	L	N	R	N	R

This pattern has significance to the network described in the next chapter, but is not important to this discussion. We abbreviate the 1024 entry truth table describing the desired behavior of the network as follows:

Inputs			Context	Outputs		
A	B	C		O1	O2	O3
a	b	c	L	b	c	a
a	b	c	N	a	b	c
a	b	c	R	c	a	b

That is, call the activation level of input A, a. Then if the context indicates a left rotate (L), output node O3 should have that same activation level, a.

The use of conjunctive connections requires the following extension to the GDR [Rumelhart, et. al. 1986]:

$$\Delta w_{jkl} = \eta \delta_k i_j c_l$$

where c_l is the activation level of the conjunctively connected input. The rule for propagating errors to the conjunctive node is:

$$\delta_j = f'_j \sum_k \sum_l \delta_k w_{jk} i_l$$

Using this rule, we have simulated the network for a number of initial weight configurations and for three of the weight adjustment rules previously described. The results presented in Table 4.2 were obtained using a learning coefficient of .5. Again we see that the GDR leads to local minima in many cases. The Mod#2 rule again

Table 4.2 - Comparative learning performance on 3-bit Rotate task
# of cycles to reach 100% performance

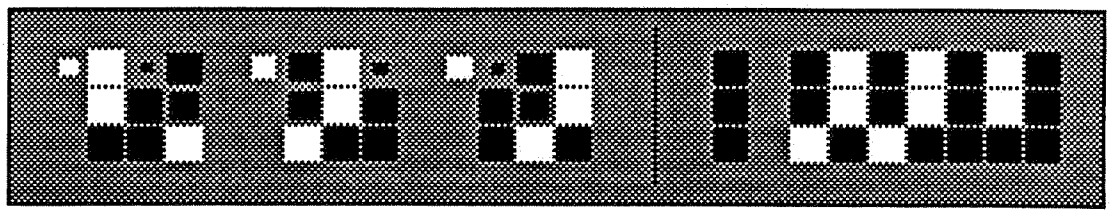
Testcase	GDR	Mod#1	Mod#2
A	71%*	328	646
B	43%*	288	729
C	856	276	71%*
D	71%*	351	1574
E	71%*	205	517
F	545	321	1532
G	71%*	316	1212
H	71%*	255	960
I	43%*	298	1843
J	71%*	302	71%*
K	71%*	307	71%*
L	498	268	554
M	1453	450	1292
N	43%*	342	566
O	783	312	655
P	456	312	71%*
Q	71%*	333	71%*
R	43%*	324	1492
S	869	372	71%*
T	571	270	723
U	814	289	632
V	1525	284	655
W	1779	337	1238
X	947	282	641
Y	71%*	254	613

* performance at cycle 2000

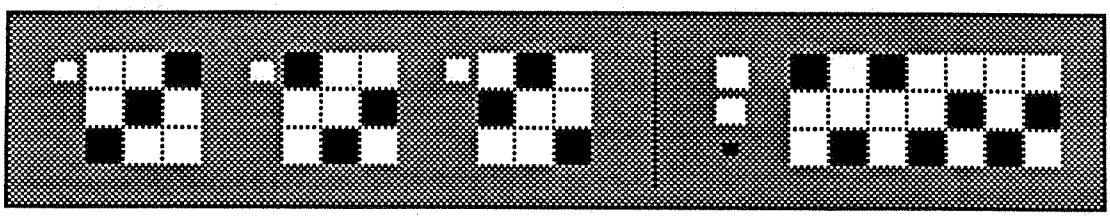
improves the learning performance of the network, but still is unable to solve the problem within 2000 cycles in some cases. The Mod#1 rule, however, performs extremely well, reaching 100% performance within 450 cycles in all cases.

These results demonstrate the applicability of error modification to the nodes representing context information. The reason these nodes are particularly susceptible to becoming deadlocked is that they completely overlap in their input and output connectivity, and are representing disjunctions of input patterns. This often leads to different nodes representing parts of a single function, or multiple nodes representing the same function. In either case, the nodes involved become deadlocked if the learning algorithm enforces a decrease in error at every node at each time step.

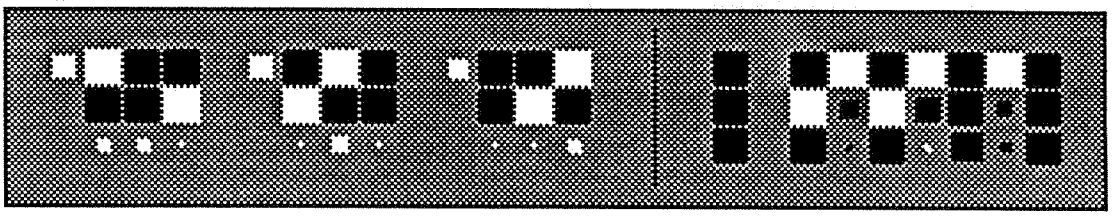
Figure 4.7 presents a schematic of the weights in both layers of the 3-bit Rotate network for a number of experiments. The left-hand set of weights are those of the three output nodes. The single square in the upper left is the bias weight, the three columns of squares for each node correspond to the three network inputs, and the three rows correspond to the three context units. The size of each square is proportional to the magnitude of the weight it represents, and the color corresponds to the sign of the weight. Black squares correspond to negative weights. Thus the square in the lower left corner corresponds to the conjunctive input to the leftmost output node combining the leftmost input and the lower-most context node. This is a negative weight having a magnitude of at least 1. The right-hand set of weights are those of the context nodes. The three rows correspond to each of the three context



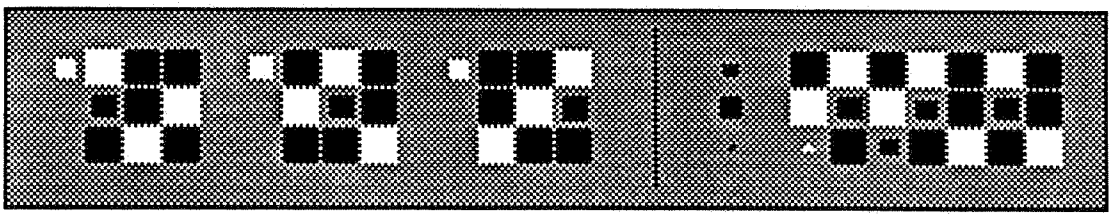
(a)



(b)



(c)



(d)

Figure 4.7

nodes. The single isolated column corresponds to the bias weights. The other seven columns correspond to the seven context inputs.

Figure 4.7(a) is an example of a local minimum. This is the weight configuration from testcase E after 600 cycles of applying the GDR learning algorithm. The context weights clearly show the source of the local minimum. The top two nodes have developed the same function of the context inputs (corresponding to context N - no rotation). The bottom context node represents Left rotation, but Right rotation is not represented. The 71% performance corresponds to correct behavior for Left and No rotation, and incorrect behavior for Right rotation.

Figure 4.7(b) corresponds to this same testcase when the Mod#1 algorithm is applied. After 600 cycles, the context weights have come to represent each of the three contexts differently. Figures 4.7(c) and (d) present the weights developed in testcase G after 500 cycles using GDR and Mod#1 respectively. In the first case, the Right rotation again is unrepresented, while the last figure shows the weights of a correctly behaving network.

To further study this error modification mechanism, we now present a more complex network.

4.2.3. 2-D Shift

In this third demonstration of error modification, we use a task involving a two-dimensional shift of the input pattern. The primary network consists of a 5×5 input layer, and a 3×3 output layer. Each output node is connected to a 3×3 subarray of the

input nodes (see Figure 4.8). In addition to this primary network, a context network, consisting of a 7×7 array of input nodes and a 3×3 array of hidden nodes is used. For each of the nine input nodes connected to each output node, there are 9 connections, each one a conjunction with one of the nine context nodes. Thus, each output node has 81 input connections.

Input patterns in the primary network consist of a single active node. The nodes in the hidden layer of the context network represent one of nine shifts to be applied to this input pattern. The desired output pattern is a single active node that represents the shifted input. Figure 4.9 (a) shows two sample input and output patterns. Figure 4.9 (b) presents the desired shift associated with each of the 49 context input nodes.

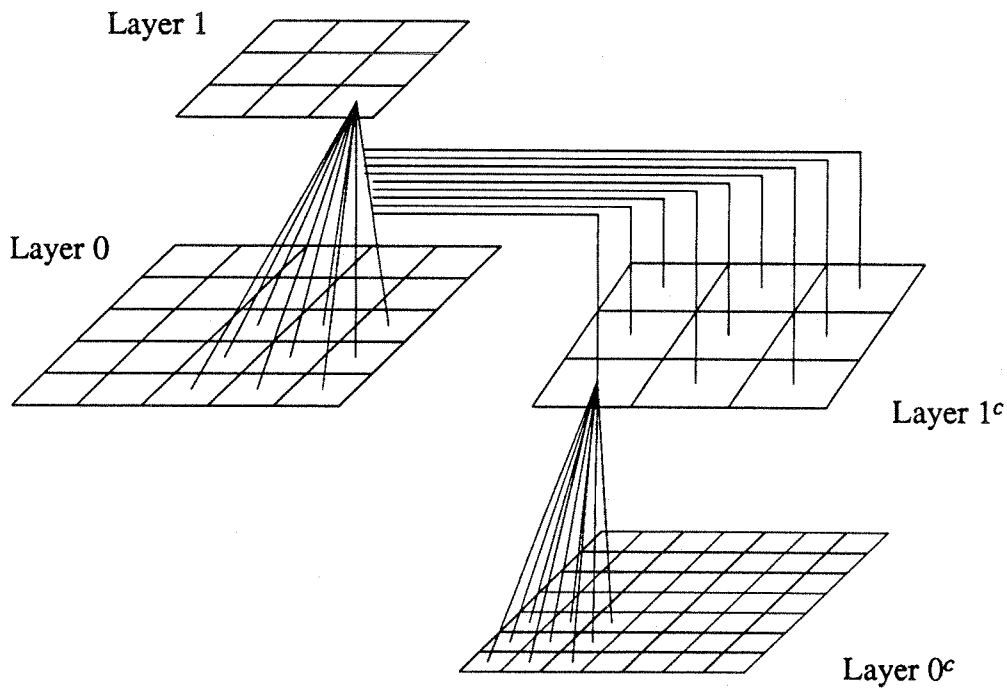
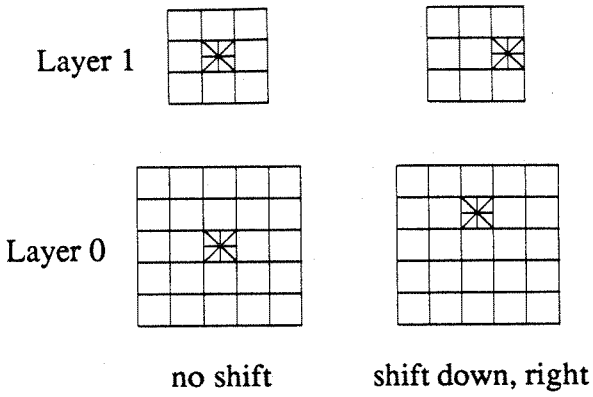


Figure 4.8



(a)

1	1	2	2	2	3	3
1	1	2	2	2	3	3
4	4	5	5	5	6	6
4	4	5	5	5	6	6
4	4	5	5	5	6	6
7	7	8	8	8	9	9
7	7	8	8	8	9	9

1 - shift down, right
2 - shift down
3 - shift down, left
4 - shift right
5 - no shift
6 - shift left
7 - shift up, right
8 - shift up
9 - shift up, left

(b)

Figure 4.9

The shift information is represented by a single active node in the 7×7 context input layer. This is intended as a high spatial frequency representation of the shift, while this task requires low spatial frequency information. The hidden context layer has the task of learning to represent this low frequency information so that the primary network can make use of it.

In the previous network (3-bit Rotate) the input pattern in the primary network provided no information as to the desired rotation, and so was completely ambiguous

without the context information. In this network, there is complete ambiguity only in the input pattern involving the central input node, since it can be shifted in any of nine directions. Since the patterns involving the four corner input nodes can each be shifted in only one direction there is no ambiguity in the patterns. The remaining patterns are partially ambiguous, since they can each be shifted in only some of the nine directions. This allows at least some of the patterns to be correctly learned without the use of the context information. Due to the ambiguity in other patterns, however, the network cannot attain maximum performance without making use of the context information. Of the 49 possible shifts that can be indicated in the context input, we use a sample of 25 nodes (indicated in Table 4.3) for these simulations.

This task is similar to the 3-bit rotate, but is more difficult due to the size of the network. This provides a greater challenge to the learning algorithm allowing us to investigate additional error modification mechanisms. The similarity between this task and that performed in the 2-D translation network (see Chapter 5) is, of course, intentional.

Table 4.3 - Context inputs

X		X		X		X
	X		X		X	
X		X		X		X
	X		X		X	
X		X		X		X
	X		X		X	
X		X		X		X

In Table 4.4 we present the results of a number of learning algorithms applied to 10 testcases, representing different initial weight configurations of the 2-D shift network. The first three columns correspond to the three algorithms discussed in the previous section. The only difference in the error modification algorithms is that an attenuation factor of $\frac{1}{10}$ rather than $\frac{1}{4}$ is applied to the non-maximum errors.

The results of applying these three mechanisms demonstrate the difficulty of this task. In no case did the performance reach 100%. For this reason, we simply report the maximum performance reached in 10000 input presentations. As in the previous task, the Mod#1 error modification mechanism leads to better learning performance than the GDR mechanism.

The four new error modification mechanisms demonstrate further refinements of the basic idea. In Mod#4, the maximum error is unchanged, as in previous

Table 4.4 - Comparative learning performance on 2-D Shift task
maximum performance (%) within 10000 cycles

Testcase	GDR	Mod#1	Mod#4	Mod#5	Mod#6	Mod#7
A	63	65	78	69	82	96
B	42	78	75	66	84	90
C	22	69	78	70	78	96
D	44	68	69	79	80	88
E	50	37	61	60	70	86
F	27	60	74	79	75	90
G	33	42	30	67	79	92
H	29	59	69	71	78	90
I	40	65	79	77	80	91
J	31	50	48	64	73	84
Average	38	59	66	70	78	90

mechanisms. Non-maximum errors are changed to values proportional to their outputs, retaining their signs. This rule is intended to introduce a component of unsupervised cluster organization, as a mechanism that is secondary to the contrast enhancement of the error signal. The modified errors for the Mod#4 rule are defined by:

$$\delta_j = \begin{cases} e_j & \text{if } j = \max \\ \beta\, o_j & \text{if } j \neq \max \;\&\; e_j > 0 \\ -\beta\, o_j & \text{if } j \neq \max \;\&\; e_j \leq 0 \end{cases}$$

In Mod#5, the error signals are modified to a form similar to that presented to the output layer nodes. That is, in the output, desired values for nodes are the extremes of the output range, 1 and 0. The corresponding error signals for these nodes are (desired - actual) 1-output and -output. The error modification for Mod#5 is:

$$\delta_j = \begin{cases} 1 - o_j & \text{if } j = \max \;\; AND \;\; e_j > 0 \\ -o_j & \text{if } j = \max \;\; AND \;\; e_j < 0 \\ (1/10)\,(1 - o_j) & \text{if } j \neq \max \;\; AND \;\; e_j > 0 \\ (1/10)\,(-o_j) & \text{if } j \neq \max \;\; AND \;\; e_j < 0 \end{cases}$$

One effect of this rule is to make the magnitude of the weight adjustments more constant over the entire learning period. Without this mechanism, the error signals are small at the beginning of learning, due to small weights on all connections in the network. As the weights begin to grow, error signals get larger and larger. Eventually, if learning is successful, the errors will again decrease due to the smaller errors actually occurring in the output layer.

The large range of weight magnitudes during the learning period makes it difficult to choose an appropriate learning coefficient. A small coefficient leads to extremely slow learning at the start of the learning period, due to the small weight values. A large coefficient may cause instabilities in the weight configuration once weights have grown, due to the much larger error signals. This idea of regulating the amount of change in the weights independently of the sizes of the weights is similar in effect to Sutton's suggestion that the sign of the weight, rather than its magnitude, should be used in propagating error signals [Sutton 1986]. This regulation of weight adjustments is advantageous to the error modification approach, as demonstrated by the improved performance of the Mod#5 rule over the two previous rules.

In each of these error modification techniques, a certain amount of error information is discarded in the interest of enhancing the effects of the most pronounced error. In contrast to the GDR, which minimizes the total weight change in the network for a given decrease in error, error modification has the effect of minimizing the number of weights that are significantly changed. This tends to place blame for incorrect behavior on a small number of nodes rather than on all the nodes in the network. Widrow [1962] used a similar philosophy in training his two layer *madaline* network.

The danger in discarding error information is that inefficiencies will result if too much information is discarded. The Mod#6 rule allows more of the original error information to be used in changing weights, while maintaining the contrast in the

cluster error pattern. This is accomplished by identifying the most positive and most negative error signals received by the cluster. The corresponding nodes, referred to as *maxpos* and *maxneg*, receive errors that are modified to mimic desired values of 1 and 0, respectively. All non-maximum error nodes receive an error signal of 0. No maxpos node is identified if all cluster errors are negative, and no maxneg node is identified if all cluster errors are positive. The Mod#6 rule is:

$$\delta_j = \begin{cases} 1 - o_j & \text{if } j = maxpos \\ -o_j & \text{if } j = maxneg \\ 0 & otherwise \end{cases}$$

This rule yields something of an improvement over the previous rules, but an even greater improvement occurs if we allow non-maximum nodes to receive an attenuated error signal as in previous rules. The Mod#7 rule combines the mechanisms of the previous two rules as follows:

$$\delta_j = \begin{cases} 1 - o_j & \text{if } j = maxpos \\ -o_j & \text{if } j = maxneg \\ (1/10)\,(1 - o_j) & \text{if } j \neq \max \ AND \ e_j > 0 \\ (1/10)\,(-o_j) & \text{if } j \neq \max \ AND \ e_j < 0 \end{cases}$$

These simulations demonstrate the improvements in learning behavior that can be obtained using error modification techniques. The maximum performance achieved using the GDR varies from 22% to 63% over the 10 testcases. Each maximum is reached within 3500 cycles, on the average, after which no further improvement occurs. The maximum performance achieved using the most effective

error modification mechanism (Mod#7) varies from 84% to 96% over those same 10 testcases. Each of these maxima are reached within 6000 cycles, on the average. Thus, the error modification rule does not lead to faster learning in the short run, but does lead to more learning in the long run. This is expected, since the descent along the gradient used by GDR is something of a greedy approach to reducing errors, while the error modification rule performs only some of the weight changes specified by GDR in the interest of developing useful local structure.

4.3. Error augmentation

The error modification techniques just described can be applied when an adequate error signal is being propagated to the cluster. They are used to impose more local constraints on the developing representations than are contained in the raw error signal. When the error signal is weak, due to small weights in higher layers and the noise of estimating errors at each layer, a different approach is required. Error augmentation is a means of obtaining useful weight adjustments when the error signal received by the cluster is weak.

Given that the error signal contains little information, we are left with unsupervised forms of learning. In this section, we first introduce a new self-organization algorithm called the trace comparison rule. This algorithm uses input normalization and output trace values to encourage different nodes to develop distinct feature detectors based on the patterns in the input. The behavior of this algorithm is demonstrated in a simulation of a single cluster network. In the second simulation of

this section, the new self-organization rule is combined with the GDR in the lowest layer of a 3-layer network. This simulation demonstrates the usefulness of self-organizing methods when applied to an error-correction task.

4.3.1. Self-organization

The important characteristics of an effective self-organizing mechanism are that it involves reinforcement based only on internal activity, and that it involves competition among nodes. The standard rule for modifying weights when no supervisory signal is available is the rule suggested by Hebb [1949], now called the correlational synapse:

$$\Delta w_{jk} = \eta\, o_k i_j$$

Using this rule, all weights are increased in proportion to the correlation between their corresponding input values and the node output value. By itself, this rule leads to unbounded increases in weight values, and fairly weak competition between nodes. One way to keep weights bounded, and to improve competition, is to normalize the input weights of all nodes. For instance, if the sum of the squares of all weights belonging to each node is normalized to a constant value, no single weight magnitude can exceed this constant, and the potential for each node to match an arbitrary input pattern is comparable to that of all other nodes. Since normalization of weights prevents certain types of desirable behaviors from occurring (e.g. classical conditioning [Grossberg 1982, p. 448]), normalization of input values is used instead. Input normalization involves shifting all input values down to reduce the mean input

to zero. The resulting inputs range both positively and negatively, reducing the problem of cluster dominance by a node having only large positive weights.

Competition among nodes can be enhanced by increasing the contrast between weight modifications applied to different nodes. While changes proportional to output provide some contrast, choosing a single *target* node to change provides much greater contrast. Various measures may be used to mediate the choice of which node in a cluster is to be modified. One way to choose a target node is to pick the node with the highest output. An alternative that is used here, is to choose the node whose output most exceeds a measure of previous activity. This alternative is less likely to lead to single nodes dominating the cluster.

We define the output trace, o', of a node as follows:

$$o' = \begin{cases} o & \text{if } o > o' \\ \gamma o' & \textit{otherwise} \end{cases}$$

where $\gamma < 1$ is a decay coefficient. This output trace is a measure of the maximum output of the node discounted over time. By comparing this trace value with the current node activation, we obtain a measure of how close a node is to giving its maximal response. This difference of actual and trace output values, $(o - o')$, is a useful criterion for choosing the winner among competing nodes in a cluster. Given an initially random configuration of weights, some nodes respond more strongly than others to a number of patterns, while others never respond more strongly than the others for any of the patterns. On the other hand, each of these non-responders does

give its maximum response for some pattern. The trace comparison rule (TCR) has the effect of enhancing the response of each node to the pattern to which it is most sensitive. This tends to make these non-responders respond.

The effect of the TCR is related to that of the direct extraction of input patterns used by Uhr & Vossler [1962]. In direct extraction, a feature is formed based on a single instance of an image subpattern. Duplication of features is avoided by comparing the new feature with the current set, and only adding it to the set if it is sufficiently different from the others. Using the TCR, a feature is formed by averaging multiple extractions of similar subpatterns. The competition between nodes, based on the trace comparison, causes a given subpattern to contribute to the definition of the current feature it most resembles. Duplication of features is avoided, in this case, by the automatic merging of similar features.

Instead of updating only the target node in each cluster, non-target nodes can be modified to a lesser degree. The following simulation uses this trace comparison rule to choose a target node in the cluster. The target node is updated according to the correlational synapse rule, while non-target nodes are updated according to:

$$\Delta w_{jk} = -\frac{1}{10}\,\eta\, o_k i_j$$

The network consists of a single cluster containing 10 nodes. A set of 10 patterns is presented in sequence, and this sequence is repeated throughout the simulation. Before learning begins, the response of each node to each pattern is recorded. These activation values are presented in Table 4.5. The maximal response to each pattern is

marked in boldface in the table.

After 2000 cycles (single pattern presentations) of adaptation, the response of each node to each pattern is again recorded. These activation values are presented in Table 4.6. With each node giving the maximal response for a different input pattern, this mechanism has achieved the best possible representation of the input patterns, in

Table 4.5 - Single cluster self organization
Initial Node Responses to 10 Patterns (×.01)

Pattern	Node									
	1	2	3	4	5	6	7	8	9	10
a	38	32	53	40	**66**	41	43	39	48	43
b	32	43	**53**	37	50	30	43	51	53	30
c	35	35	54	45	55	32	49	39	**58**	39
d	36	42	**63**	39	57	44	49	51	47	36
e	32	40	**62**	39	60	44	43	47	52	34
f	32	39	**66**	53	50	34	51	43	56	38
g	34	36	**67**	40	56	31	57	48	51	40
h	39	37	50	46	**62**	36	41	39	51	44
i	28	39	**58**	47	52	31	46	39	54	32
j	35	30	59	34	**61**	33	53	46	47	39

Table 4.6 - Single cluster self organization
Node Responses to 10 Patterns after 2000 cycles (×.01)

Pattern	Node									
	1	2	3	4	5	6	7	8	9	10
a	10	8	3	8	**88**	9	2	2	7	49
b	9	9	9	2	0	6	9	**89**	8	16
c	26	32	4	34	49	20	36	34	**89**	34
d	25	21	**88**	24	32	35	30	29	3	16
e	2	12	31	14	39	**89**	14	15	10	15
f	2	9	9	**88**	9	9	12	8	9	9
g	9	2	9	9	0	6	**88**	9	8	16
h	0	3	0	3	20	3	4	4	2	**89**
i	2	**89**	9	9	9	9	8	12	9	9
j	**89**	9	23	9	32	2	18	17	16	4

the absence of task-specific feedback. This simple demonstration indicates that a self-organizing mechanism is capable of providing a certain amount of discrimination among patterns in a fixed set. A second run, using 20 patterns yields the results presented in Table 4.7. In this case, each node in the cluster tends to develop a sensitivity to a small number of the 20 patterns, so that relatively high discrimination among the patterns is achieved. This mechanism does not optimally discriminate, however, as is clear from the fact that one node in the second simulation (node #6) does not give the maximum response for any of the patterns. In addition, the choice

Table 4.7 - Single cluster self organization
Node Responses to 20 Patterns after 2000 cycles (×.01)

Pattern	Node									
	1	2	3	4	5	6	7	8	9	10
a	42	8	28	14	46	17	6	10	24	**89**
b	36	30	26	8	53	14	18	**88**	18	13
c	13	22	1	25	20	1	20	9	**72**	19
d	38	27	**95**	33	59	88	53	56	6	33
e	4	36	**87**	42	62	75	6	23	9	33
f	0	13	20	**94**	13	14	47	9	24	11
g	15	6	35	38	34	19	**97**	33	30	23
h	11	19	14	26	28	10	8	20	17	**94**
i	0	**89**	14	20	27	10	21	21	15	6
j	**87**	2	19	5	54	8	67	36	46	30
k	5	**95**	20	12	13	14	7	22	24	11
l	**98**	9	34	32	11	19	17	34	44	26
m	25	19	13	25	78	10	8	10	**88**	2
n	13	28	28	**89**	6	19	20	9	36	20
o	26	34	14	32	6	6	**89**	33	24	6
p	17	8	27	14	**89**	16	6	24	69	8
q	28	24	26	9	26	14	**97**	7	32	16
r	4	**97**	11	3	23	9	28	10	44	20
s	4	28	20	2	**86**	12	6	23	32	32
t	4	3	11	**96**	21	9	7	19	42	20

of which patterns to group together for representation by a single node may not be the correct choice for the purposes of task specific processing at higher layers. We now turn to the problem of integrating this self-organizing mechanism with a supervised one.

4.3.2. 3-layer Classifier

The 3-layer network presented in Figure 4.10 will be used to classify input patterns into one of 10 classes. One approach to learning this task is to use backpropagation of errors to all layers. In the first experiment with this network, we present a set of 50 patterns to the network, five from each of 10 different classes. The classes are defined by arbitrarily assigning five patterns to each, without regard to

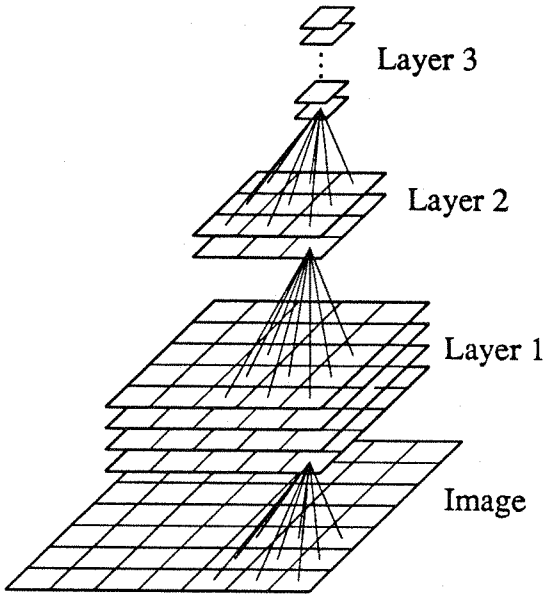
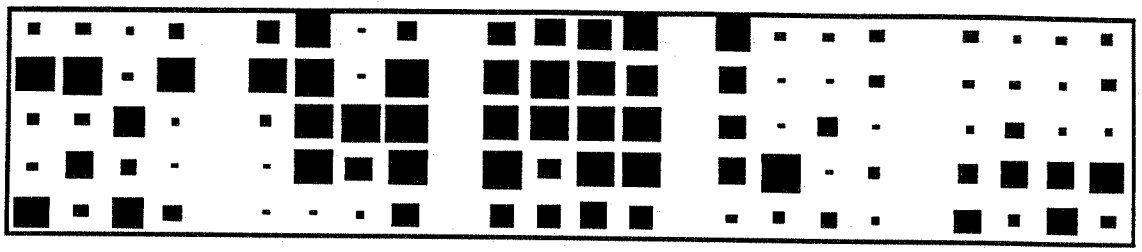


Figure 4.10

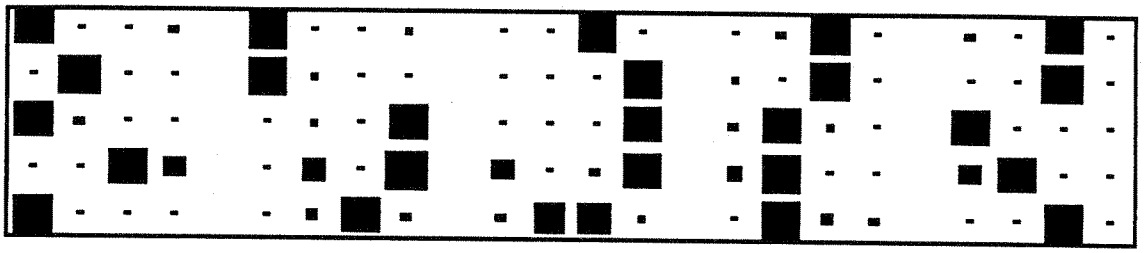
similarity of patterns. Since a self-organizing mechanism works by clustering patterns having similar features, it is most effective when the classification being learned by the network is based on similarities in the input patterns. In order to test the capabilities of our learning approach in unfavorable conditions, we use this arbitrary assignment of patterns to classes. The advantages of our approach can only increase for tasks in which members of a class have similar input patterns. Using the GDR to update weights, a maximum performance of 70% is obtained after 11000 cycles, with no further improvement up to 20000 cycles.

In order to obtain more effective weight modifications at the first layer, we run a second experiment on this network. The task is the same as the previous one, but now the GDR is used only at layers 2 and 3, while the self-organizing rule previously described is applied at layer 1. Performance now reaches 96% after 6500 cycles.

The advantages of using a self-organizing mechanism can be seen by comparing the development of feature detectors in the early stages of each of these two simulations. A typical response of the first layer nodes after 500 cycles of training is shown in Figure 4.11. Activation levels of each of the 4 nodes in each of the 5×5 clusters are represented by squares whose size is proportional to the magnitude of activation. Figure 4.11(a) and (b) correspond to the features developed by the GDR algorithm and the self-organizing algorithm respectively. The self-organizing mechanism has quickly developed in the first layer a representation of input patterns which provides an adequate base for the task driven feature development at higher



(a)



(b)

Figure 4.11

layers.

Although combining error-correction and self-organization in this straightforward way is effective in this case, there are potential problems with this approach. One problem is that the features developed at the first layer are not in any way affected by the demands of the recognition task. This means that there is no pressure at this layer to develop a critical feature if needed for discrimination of patterns by the higher layers.

A second problem is that the self-organizing component can undo the effects of the error-correction component, since the pressures driving the two mechanisms may be in conflict. In particular, the error could go to zero, but weights would continue to adjust at the first layer, possibly reintroducing errors into the network behavior.

To attack these problems, it is desirable to have components of both error-correction and self-organization in the first layer weight modification rule. One method is simply to compute the effects of each individual algorithm separately and adjust the weights according to the sum effect. Repeating the previous simulation using this combination of mechanisms yields 100% performance after 8500 cycles. This method addresses the first of the two problems, since weights are changed, at least partially, due to backpropagated error. However, the self-organizing component can still cause weight changes in the absence of error. In fact, in this simulation, the performance briefly dropped to 92% at 11500 cycles before returning to 100% for the remainder of the 20000 cycle simulation.

An alternative means of combining these mechanisms is to use the output weights of the layer 1 nodes to mediate the combination. As discussed in the previous chapter (see section 3.4.2), the output weight of a node is a measure of its usefulness. The GDR adjusts weights to reflect the correlation between features computed by pairs of nodes. If a given node computes a feature that has no correlation with the features being computed at the next higher layer, it will develop an output weight near zero. A significant output weight will develop if the node computes a feature that correlates highly with those of the next higher layer.

In backpropagation learning algorithms, a node receives weak training information if it has a low output weight, since the error signals are propagated back through the weights. A node that is useful, as measured by its output weight, will receive a substantial training signal. A node that is not useful to the higher layer computations will receive a weak training signal, due to its small output weight. This leads us to conclude that error correction is appropriate to nodes with high output weights, since they can make use of the network error information. Nodes with low output weights are not useful to the network, and will change very little due to error correction. A self-organizing mechanism is appropriate for these nodes.

Applying these two mechanisms on a node by node basis undermines the concept of cluster level cooperation by introducing two possibly inconsistent goals into each cluster. In order to maintain the coordinated development of all nodes within a cluster, the decision to apply error correction or self-organization must be

made at the cluster level. We use the following rule: Choose a target node using the TCR criterion. If the output weight for this node is less than some threshold, apply the self-organization rule to the cluster. Otherwise, apply the error correction rule to the cluster.

In Table 4.8, we present the results of applying various combinations of GDR and TCR to ten testcases of the 3-layer classifier network. These results are somewhat surprising, in that the best algorithm for obtaining 100% performance is the one that uses only self-organization at the lowest layer. This is an indication that the TCR rule is effective in extracting the regularities in the input patterns. Of the two rules which combine GDR and TCR at layer 1, the rule called MaxWgt performed better than the AveWgt rule. MaxWgt computes the output weight of each node as the maximum absolute value of all connection weights on the output of that node. AveWgt uses the average value of that same weight set. Although the average

Table 4.8 - Comparative learning performance on 3-layer Classifier task
# of cycles to reach 100% performance

Testcase	GDR	TCR	AveWgt	MaxWgt
A	13000	11000	10000	8000
B	77%*	7000	93%*	93%*
C	60%*	20000	6000	5000
D	77%*	14000	12000	6000
E	92%*	7000	15000	7000
F	96%*	9000	10000	5000
G	86%*	9000	96%*	9000
H	56%*	7000	13000	13000
I	96%*	6000	15000	9000
J	96%*	10000	9000	5000

* performance at cycle 20000

weight might at first seem to be the natural measure for output weights, it is less effective than maximum weight. One argument for the use of maximum weight is that a given node could represent a critical feature in the computation occurring at a particular node in the next higher layer. If all its other weights are low, this node will have a low average output weight. The use of maximum output weight allows such a node to be treated as useful.

4.4. Discussion and summary

Knowledge is represented in a connectionist network by the connections that mediate the interactions of the processing elements. Learning involves changing the strengths of connections so as to improve the overall behavior of the network. In a supervised learning task, the goal is to modify the connection weights in the network in such a way that the desired behavior is exhibited for any given stimulus. Finding the correct set of connection weights to perform any given task is a problem in searching the space of all possible weight configurations. Though direct search methods have been tried (see the review in [Anderson 1986]), it is clear that heuristic search is needed for all but the most trivial network.

Heuristics are used to reduce the size of the search space. Structural heuristics include the restricted connectivity of the network, as exemplified by our layered locally connected network. Gradient descent is a heuristic for finding an adequate weight configuration based on reducing the error in the network response. Nearly all approaches to learning in networks are based on reducing the response error. The

Widrow-Hoff rule and its multi-layer extension, the generalized delta rule, estimate the total network error using the current response error. Both algorithms lead to weight configurations that are locally optimal. Unfortunately, in multi-layer networks, locally optimal weight configurations may be far from globally optimal.

We have presented two learning methods that address three of the major weaknesses of the GDR. Error modification addresses the problem of locally minimal error configurations by reducing the occurrence of intra-cluster deadlocks. Error augmentation addresses the problems of weak training signals and node drop-out by introducing a bottom-up component to the weight adjustment rule.

Error modification is a heuristic that improves on the GDR method by enforcing local constraints concerned with desirable representations among groups of nodes. Instead of changing the weights in such a way that the error is maximally decreased for a given amount of weight change, the changes are made to decrease error while encouraging diversity of function within clusters of nodes. The experiments reported here demonstrate the effectiveness of this heuristic in efficiently searching for an adequate weight set.

The local minimum problem can be observed in tasks as (seemingly) simple as the XOR. In 25 testcases, each run for 1000 cycles, the GDR rule failed to solve the task 12 times. One error modification rule, based on reducing the effects of all errors coming into a cluster except the maximum, successfully solved all 25 testcases.

The 3-bit rotate task uses conjunctive connections to combine two sources of input in computing the network output. For this task, the GDR fails to converge to a correct network configuration within 2000 cycles in 13 of the 25 testcases. One error modification rule solves all 25 testcases within 450 cycles.

Using the 2-D shift network, a number of error modification variations are compared with the GDR and each other. In 10 testcases, the average network performance is 38% after 10000 cycles when the GDR is used to train the network. Using the best of the error modification rules, the average network performance is 90% after 10000 cycles.

Backpropagation of errors, used to implement the gradient descent search at all layers of a multi-layer network, works well for a two or three layer network. As the errors propagate back, however, error estimation noise increases, leading to weak training signals in lower layers of the network. Error augmentation methods address this problem by performing local searches through weight space based on the input patterns presented to the network. In order to make these local searches consistent with the requirements of the overall network task, output weights are used to estimate the usefulness of the feature represented by each node. This usefulness measure then guides the local search in finding appropriate representations.

A single cluster simulation demonstrates the effectiveness of a self-organization mechanism called the trace comparison rule (TCR). This mechanism uses the trace value of the node output as a reference value in determining how strongly the node is

activated, compared to other nodes in the cluster. This rule resulted in each of 10 nodes responding maximally to one of 10 patterns presented to the cluster. When 20 patterns are presented, three nodes each respond to three of the patterns, one node responds to one pattern, one node responds to no patterns, and five nodes respond to two patterns. This reasonably even spread of the pattern representation over the set of nodes is desirable for use by higher layer processing.

The 3-layer classifier network is used to demonstrate how this self-organizing mechanism can be combined with an error correction mechanism in the lowest layer to get efficient learning in deeper networks. Improved performance over the GDR rule was obtained both when the TCR rule was used alone in layer 1, and when GDR and TCR were combined using the maximum output weight. While the GDR led to 100% performance in only one of ten testcases, the TCR alone succeeded in solving all ten testcases. The combination rule failed to reach 100% on two testcases, but converged to a solution much faster than other rules, on the other eight testcases.

In the next chapter, we apply and compare the methods demonstrated here to the problem of transformation-invariant object recognition.

Chapter 5

Vision Network Simulation

A network structure for performing 2-D translation-invariant object recognition is presented in Chapter 2. The major characteristics of this network are its explicit representation of translation information, and its representation of object-centered features. In this chapter, we present a number of simulations used to study the learning and generalizing capabilities of the network. At the same time, these simulations demonstrate the capabilities of the learning algorithms described in the previous chapter.

5.1. The 2-D translation network

The 2-D translation network consists of a primary shape network and an auxiliary context network. Shape features of an object are represented in the pyramidal structured primary network. In the lower layers, simple features such as edges and lines are represented. Each of these features is retinotopic, since it corresponds to a specific position in the image. The shape representation is hierarchical in that each feature at a given layer is defined in terms of features represented in the layer below. In the upper layers of the primary network, object-centered shape features are represented.

Location information is extracted from low level shape information and represented in the context network. At one level of the context network, this

information is represented by one node per location. Since this representation is inadequate for efficient combination with shape information, two other layers represent coarse and fine location information.

To compute object-centered features, retinotopic features are paired with location features. The presence of a particular retinotopic feature, along with the activation of a particular location node, gives evidence for an object-centered feature. Each object-centered feature is defined as a disjunction of many such shape/location pairs. Thus, the course and fine context information is conjunctively combined with shape information at different layers of the primary network to compute the shape features in the upper layers. We now describe the network in greater detail.

5.1.1. Network description

The network is presented schematically in Figure 2.3. Layer 1^c has been omitted from this diagram under the assumption that context information would be supplied directly to layer 2^c. The context sub-network is more fully described in section 5.3.1, where the extraction of context information from the image is demonstrated.

Layer 0 of the pyramid is a 15×15 array of input units, representing pixel intensities of a synthetic image. The input patterns are presented as 5×5 pixel binary patterns within a 15×15 pixel image. The shape patterns used in these simulations are alphanumeric characters and pseudo-characters, composed of horizontal and vertical lines. A set of 50 shapes are defined, three of which are shown in the lowest

arrays of Figures 5.1(a), 5.1(b) and 5.4. Six other shapes are shown in Figure 5.2. Limiting patterns to the central 11×11 subarray, to eliminate border effects, allows 7 horizontal and 7 vertical translations of each object pattern. Layer 1 consists of a 13×13 array of processing element clusters, each cluster containing N_1 processing elements. In the fixed connection network, to be described, N_1 is 2, while in a later simulation we use 6 nodes per cluster in layer 1. Each cluster has 9 inputs from a 3×3 array of pixels in the image array. Each processing element has one weight associated with each cluster input, plus an additional *bias* weight. Elements compute their output values according to a weighted sum of inputs plus bias, modified by a logistic output function. Layer 2 consists of an 11×11 array of clusters each containing N_2 (=2-6) processing elements. Each element connects to 9 clusters in a 3×3 subarray of the Layer 1 array. Layer 3 consists of a 5×5 array of N_3 (=2) element clusters. The inputs to layer 3 elements come from a 3×3 subarray of clusters in Layer 2, but each pair of connected elements has 9 connections between them. Each of the nine is conjunctively modified by one of the outputs of a context node in layer 3^c. Layer 4 consists of a 3×3 array of N_4 (=2) element clusters, each connected to a 3×3 array of clusters in layer 3 and modified by context nodes in layer 4^c. Finally, layer 5 consists of a single N_5 element cluster, each element connecting to all clusters in layer 4. N_5 ranges from 6 to 50 in various simulations.

Context information is learned independently of object recognition, though the same input is used in both cases. The lowest two layers of the shape pyramid are

shared with the context network (see Figure 5.3 in section 5.3.1). These layers extract shape features from the image that are used both in recognizing the shape and in identifying the location of the object. Layer 1^c is a 9×9 array of 2 element clusters. Each node receives input from a 5×5 sub-array of layer 2 clusters. Layer 2^c is a 7×7 array of nodes, each receiving input from a 3×3 sub-array of layer 1^c clusters. This layer represents the location of an object in the image, using one node each to correspond to the 49 combinations of seven vertical and seven horizontal offsets with which shapes are presented in the image. This layer is trained through direct feedback from the environment, as is the top layer of the shape pyramid. Layer 3^c is a single cluster of nodes arranged in a 3×3 array. Each node is connected to a 5×5 sub array of layer 2^c nodes. Similarly, layer 4^c is a 3×3 array of nodes in a single cluster, each node connected to a 3×3 sub-array of layer 2^c nodes.

The outputs of these two highest level context layers are conjunctively connected to nodes in the shape pyramid. These layers together represent the location information in such a way that a 2-step transition from retina-centered to object-centered representations occurs in the shape pyramid. In particular, for each of the connections between layers 2 and 3 that were described above, nine connections actually exist, each conjunctively connected to one of the layer 3^c node outputs. Similarly, for each of the previously described connections between layers 3 and 4, there are nine connections, each a conjunctive connection involving an output from a layer 4^c node.

Due to the complexity of this network and the various characteristics of the network structure and learning algorithms to be demonstrated, we will describe various simulations involving separate pieces of the network.

5.1.2. Fixed weight network

First, we demonstrate the behavior of the entire network using a set of fixed (non-adaptive) weights. This serves as an existence proof that the network is capable of representing the required shape and location information. In addition, these weights define feature representations at intermediate layers of the network. These intermediate representations are used to define input patterns and desired outputs in the simulations of various sub-networks, described in later sections. This set of weights is not the only possible network configuration that satisfies the requirements of the task, but we use it to show an adequate configuration.

The nodes in layer 1 extract short vertical and horizontal lines from the image. Each node has a receptive field of 3×3 pixels, so it can detect a line of length three in some specific location in the image. In layer 2, each node represents a longer horizontal or vertical line at a particular location in the image. These nodes have inputs from a 3×3 array of layer 1 nodes yielding (due to overlapping receptive fields in layer 1) a 5×5 pixel receptive field in layer 2. In layer 3, each node again represents a long vertical or horizontal line, but a given node represents a line that may occur in a number of locations in the image. The precise interpretation of the node activation depends on the activity in layer 3^c which represents part of the

location information. That is, a layer 3 node represents a shape feature whose location in the image is represented in layer 3^c, rather than in the layer 3 node itself. Nodes in layer 4 represent long horizontal and vertical lines, whose image locations are represented in layers 3^c and 4^c. The layer 4 node itself represents the location of the line with respect to the object rather than the image. Thus, one node represents the uppermost of the horizontal lines of an object shape, another the leftmost vertical of the shape. The relation of this line to lines at layer 3 is mediated by the activation in layer 4^c.

Each node in layer 5 connects to all the nodes in layer 4. A layer 5 node represents one of the particular 5×5 shapes that may be presented in the image. Each layer 5 node connects positively to those object-centered features at layer 4 that are part of its definition, and negatively to all others. For instance, the letter *T* is represented by a node at layer 5 that connects positively to the layer 4 nodes representing the uppermost horizontal and the central vertical (see Figures 2.4 - 2.6 for further examples).

To simplify this demonstration, we supply location information directly to layer 2^c, bypassing two layers of connections in the context sub-network. (The description of the fixed context network is presented in section 5.3.1).

The fixed weight network has been simulated to demonstrate that it does indeed perform the desired task. Each of the nine shapes was presented in each of the 49 possible locations in the image. The desired response is that one output node

corresponding to the current input pattern be strongly active, while all other nodes be inactive. The result obtained is that all patterns are recognized in all locations. Two typical input patterns and the associated activations in all layers of the network are presented in Figure 5.1 (a) and (b). Activation of each node is represented in these figures by the size of the square associated with the node.

The elements of the shape pyramid appear on the right as six vertically arranged groups of dots representing elements in the six layers. The top row represents the 9 nodes of layer 5. The next three rows represent the 3×3 array of clusters of layer 4, with element 0 of each cluster in the left grouping, element 1 in the right grouping. The next five rows represent the 5×5 clusters of layer 3, each containing two elements. Similarly layer 2 is represented in the next 11 rows, layer 1 in the next 13 rows and layer 0 in the bottom 15 rows of the figure.

The elements of the context sub-network (except for layer 1^c) appear on the left as three groupings of nodes. The top group represents the 3×3 element cluster of layer 4^c. The next group represents the 3×3 element cluster of layer 3^c. The bottom group represents the 7×7 elements of layer 2^c.

5.2. Learning object-centered representations

This first learning simulation uses the top three layers of the shape pyramid, and the top layer of the context network to demonstrate that object recognition can be learned through the development of object-centered feature detectors and explicit representation of location information.

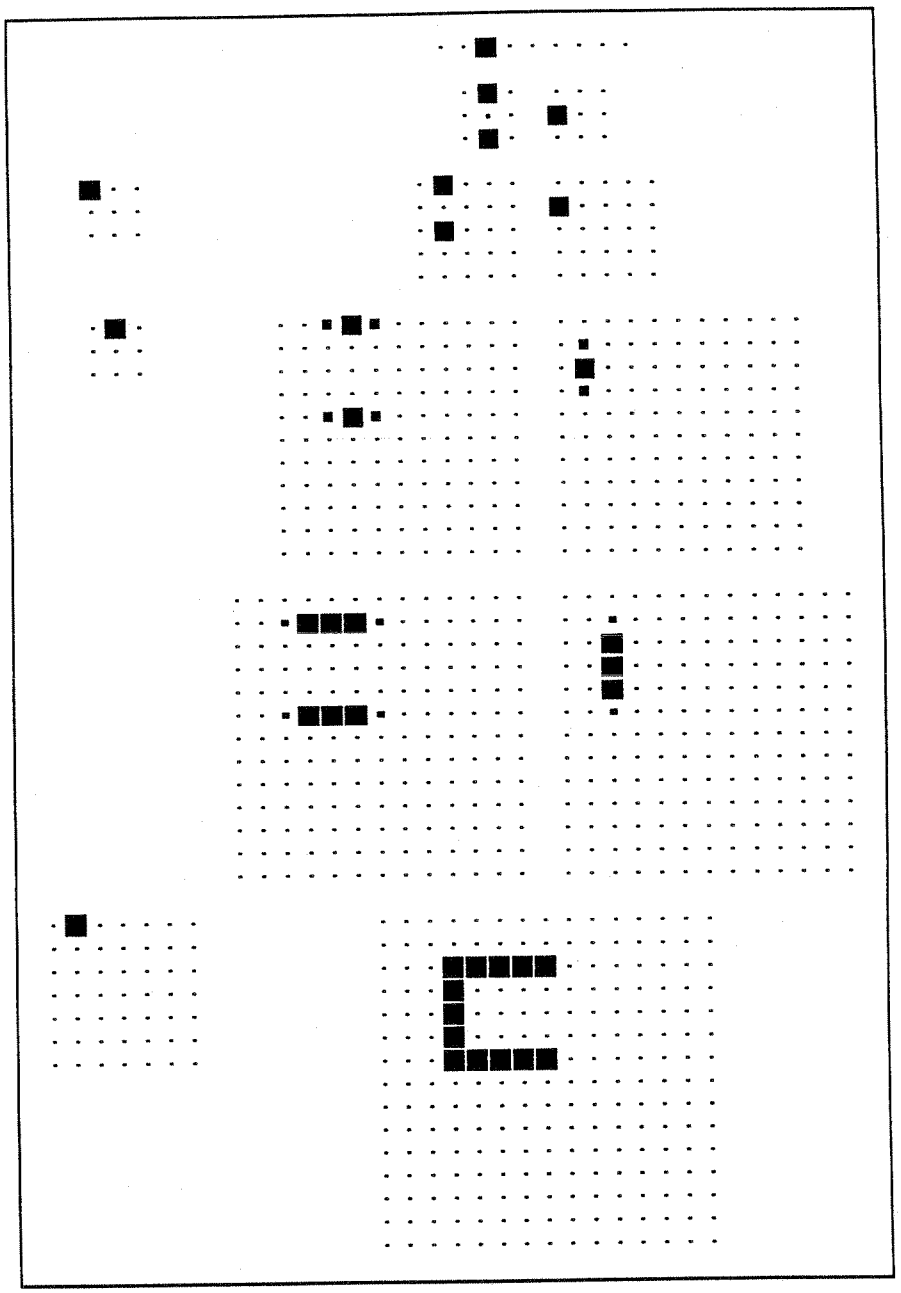


Figure 5.1 (a)

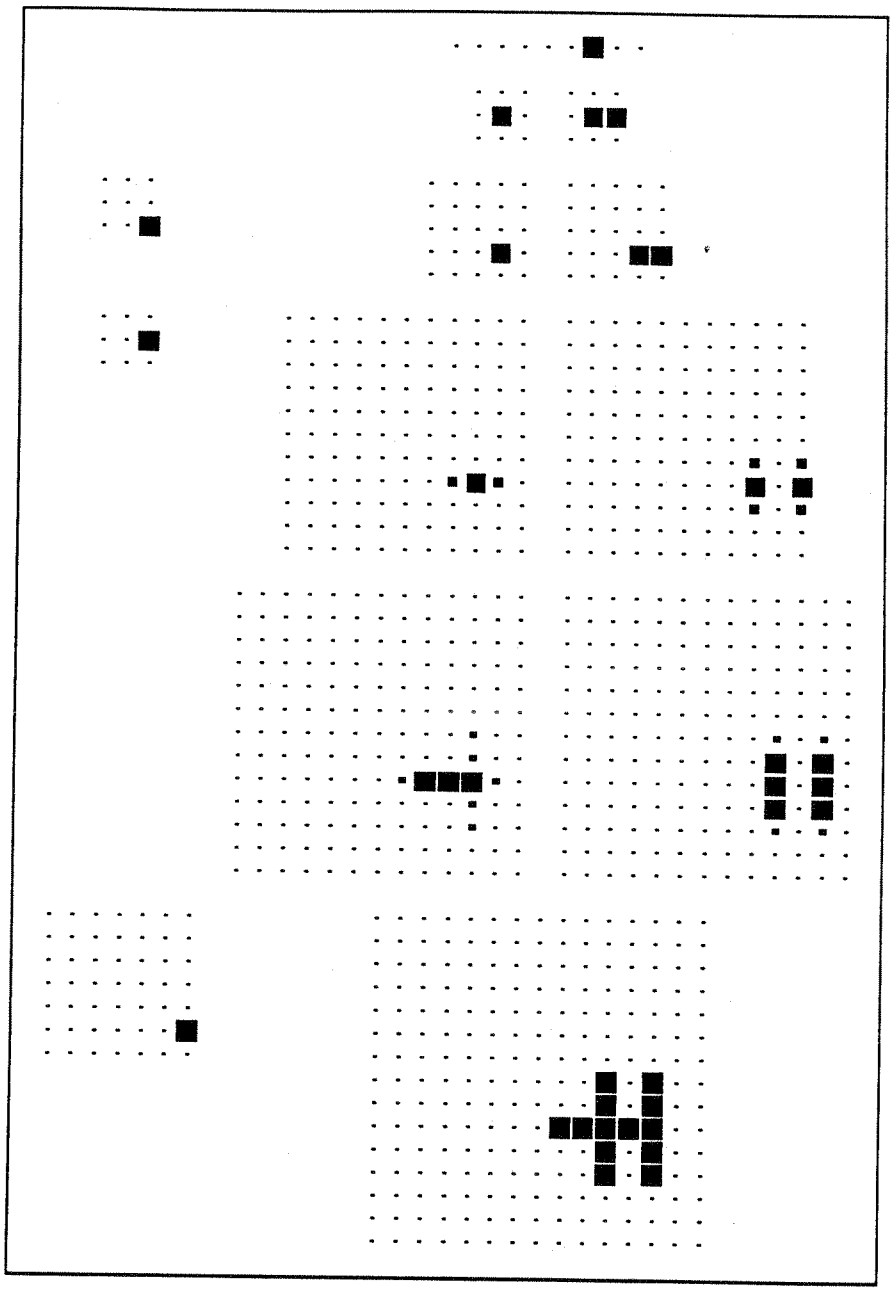


Figure 5.1 (b)

This central function of the network leads to a capacity to generalize that allows "recognition of familiar objects from novel viewpoints". In addition, good generalization leads to more efficient learning, since the system will already 'almost' recognize some objects. Without generalization, an adaptive vision system must be presented every object of interest at every possible transformation in order to recognize each object under any transformation.

The problem of making good generalizations is difficult for a machine learning system because it is underconstrained. The system is presented with a subset of instances of a number of concepts as training patterns. These instances belong to a much larger set of instances of each concept. Generalization of the system's representation of a concept is good to the extent that it includes many of the unknown (i.e., never previously presented) instances of the concept and does not include many instances of other concepts. With no information about these unknown instances, the best a learning system can do is to generalize according to syntactic similarity. Previous demonstrations of the generalization capabilities of adaptive networks have been limited to the use of large training sets and small test sets (e.g. [Hinton 1986]). It is the explicit representation of location information that allows this network to generalize along the 'right' dimension. This simulation demonstrates this ability to generalize knowledge of a given object across translations.

To simulate this 3-layer network, we provide input directly to layers 3 and 2^c as it would appear in the fixed network. One of six 'letter' patterns (see Figure 5.2) in

one of nine locations is presented in layer 3. A single activation of one of six nodes in layer 2^c represents the location information for the context sub-network. Using these six shape patterns and nine locations yields a set of 54 images to be presented to the network. These are split into a set of 27 training patterns and 27 test patterns.

Since the network response to unknown patterns is degraded compared with the response to known patterns, we use a somewhat weaker performance criterion in this section. A response is considered correct if the maximally active output node corresponds to the correct pattern class and has an output value greater than .5.

In the first experiment, learning of the 27 training patterns proceeds quickly, yielding 89% performance in 500 cycles using GDR. However, when the remaining 27 patterns are presented to the network as a test set, only 4 are correctly classified. This combination of relatively fast learning and low generalization indicates that the capacity of the network to store patterns is high compared to the number of patterns to be stored. This allows the network to perform well by representing each pattern individually, rather than as a set of shared features based on the regularities intrinsic to the pattern collection. The result is rote learning, which lacks generalization

Figure 5.2

capability. To limit the capacity of the network, six of the eighteen nodes of layer 4 are used.

Repeating the training experiment using GDR on the 27 training patterns results in 75% recognition performance after 5000 cycles. Error modification (Mod#4) improves this result to 96% performance after 5000 cycles. The reduced capacity of the network has increased the difficulty of the learning task, as expected. However, in the simulation using error modification, only two of the nine context nodes are utilized, in the sense that they fire for some patterns and not for others. Since the location information is implicit in the image, the primary network is capable of solving much of this task without the explicit representation of location supplied by the context network. This will not lead to the desired object centered features, however, which are generalizations over object locations. The results of applying the adapted network to the test patterns are again 4 of 27 correct.

In the next experiment we use the full set of 50 shape patterns and 9 locations for a total of 450 image patterns. In order to achieve good recognition performance on this task, the network must represent pattern features more efficiently than in the previous experiment. We use a training set of 150 patterns leaving 300 patterns as a test set. The set of 150 training patterns includes 3 presentations of each of the 50 fixed patterns in three different locations.

The increased number of patterns makes this task more difficult than the previous one. Using GDR, only 12% of the patterns are correctly recognized after

20000 cycles of training. Using error modification, performance on the training set reaches 100% after 12000 cycles (80 presentations per pattern). Table 5.1 presents the results of running a set of 10 testcases using GDR and error modification.

By observing the behavior of the network for testcase A, we can see that all nine context layer nodes are now contributing to the representation of location information. We would expect this from the strong generalization behavior.

Generalization allows unknown patterns to be correctly classified, as demonstrated above. In addition, generalization leads to efficient learning since some knowledge of learned patterns is transferred to related unknown patterns. There are a number of ways to demonstrate this transfer. Table 5.2 presents the number of cycles needed to reach 100% performance for various training set sizes, using testcase A. If no generalization takes place, we expect the time needed to train the network to increase as the size of the training set increases. Table 5.2 shows that the time

Table 5.1 - Learning performance on training and test patterns
% performance after 20000 cycles

Testcase	GDR		ErrMod	
	training	test	training	test
A	37	6	100	100
B	40	6	90	44
C	35	6	76	23
D	38	9	100	99
E	31	11	100	100
F	38	8	100	100
G	42	6	92	40
H	31	10	100	100
I	30	8	100	91
J	39	11	100	100

Table 5.2 - Learning performance of ErrMod Maximum performance and cycles needed to achieve it		
size	% performance	cycles(x1000)
50	99	20
100	96	20
150	100	12
200	100	14
450	100	10

needed to train the network on all 450 patterns is less than that needed to train on 100 patterns. This indicates that only a subset of the complete pattern set is needed for the network to develop an accurate model of the patterns, if good generalization occurs. Additional patterns beyond the required subset help, rather than hinder, the learning process.

The set of experiments described in this section demonstrates the key network capability that is desired for recognizing familiar objects from novel viewpoints.

5.3. Other sub-networks

The simulation just described is sufficient to demonstrate the most important characteristics of the 2-D Translation network. In this section, we present simulation results for some of the other subnetworks that perform functions necessary for realizing the representations used in the previously described simulation.

In the first of the three simulations to be described, the context sub-network is considered. We first describe the fixed weight configuration of the context network. We then demonstrate how the representation of location in layer 2^c is learned from

image inputs and direct training signals to that layer. The second simulation involves the lower layers of the network, where feature representations are purely retinotopic. Since they are far removed from the parts of the network receiving direct training information, these layers receive only weak estimates of errors as training signals. This simulation demonstrates how useful features can be developed in this part of the network using error augmentation techniques. The final simulation concerns the gradual transition of retina-based to object-based features. The two transition layers (3 and 4) of the primary network, and their corresponding context layers (3^c and 4^c) are adapted to demonstrate that a gradual mapping of retina-based to object-based features can be learned.

In a manner similar to the previously described simulation, these three simulations involve sub-networks of the 2-D Translation network. The layers of each sub-network are isolated from the rest of the network, by applying input patterns directly to the lowest layer and providing training signals directly to the highest layer of the sub-network. The input patterns and desired outputs are defined by the corresponding patterns in the fixed network.

5.3.1. Context

In the previous simulation, context (location) information was supplied to the network directly as an activation pattern in layer 2^c. From this pattern, layers 3^c and 4^c must learn appropriate representations of this information to mediate the mapping of retina-based to object-based features. Here we consider the problem of learning

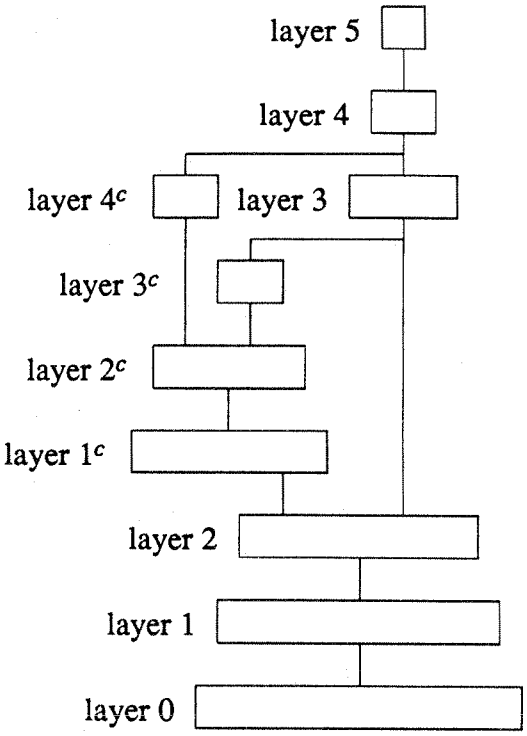


Figure 5.3

the representations in layer 2^c from information available in the shape network. In order to learn to locate an object in an image, the training information must include location information. This information is used as the training input to layer 2^c.

We begin this study by describing a fixed weight network for context, completing the description of the fixed network begun in section 5.1. A block diagram of the entire 2-D Translation network is presented in Figure 5.3. Nodes in layer 1^c detect the centers of the sets of vertical and horizontal lines represented in layer 2. Layer 2^c nodes represent a particular location of an object shape in the image, one node per location. Layer 3^c nodes represent location information at a fine

level of detail relative to the course level represented by layer 4^c. Thus, a shape in the central portion of the image that is slightly above and to the left of center would be represented as an active center node at layer 4^c and an active upper left node at layer 3^c. Figure 5.4 presents the activation values for all nodes involved in the representation of location. This figure is similar to Figure 5.1 except that the layers represented, from top to bottom, are layers 4^c, 3^c, 2^c, 1^c, 2, 1 and 0.

To demonstrate the learning of context information from shape features we simulate the network composed of layers 0, 1, 2, 1^c and 2^c of the 2-D Translation network. Patterns are presented as activations of the input nodes of layer 0. The weights of the fixed connection network are used for layers 1 and 2. The activations of layer 2, therefore, are those that would occur in the fixed connection network. As previously described (section 5.1), layer 1^c is a 9×9 array of 2 node clusters, each node connecting to a 5×5 sub-array of clusters in layer 2 of the shape network. The input patterns presented to layer 2 are those corresponding to 17 of the 50 shapes and 25 of the 49 locations used in the previous simulation. The desired output, at layer 2^c is to have a single active node corresponding to one of 49 locations of the shape in the image.

This two-layer learning task turns out to be fairly easy for the GDR algorithm. The network is able to reach 100% performance in 4500 cycles.

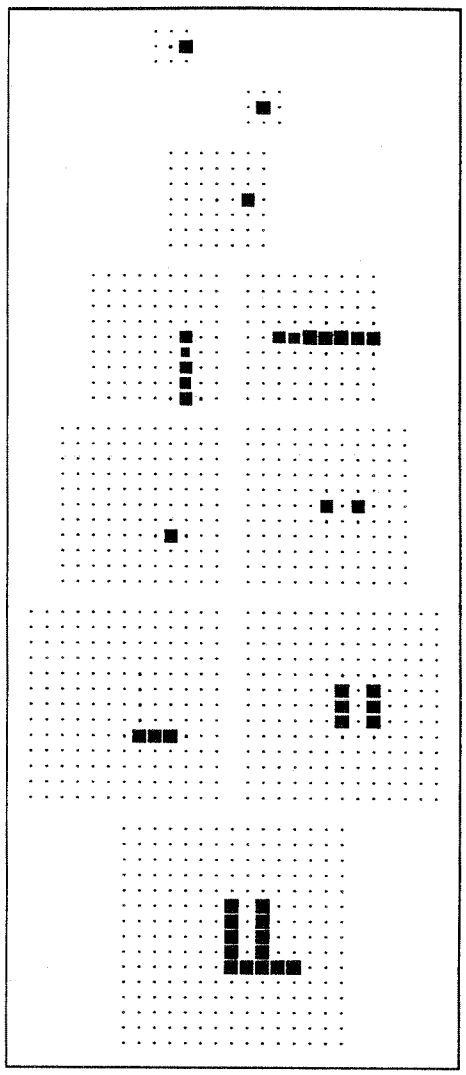


Figure 5.4

5.3.2. Retinotopic

The lower layers of the 2-D Translation network, those representing features at specific locations in the image, are 4 and 5 layers removed from direct training. This results in very weak training signals from the layers above. For this reason, we apply the methods of error augmentation to the task of learning appropriate low level representations.

For this simulation, we adapt the first four layers of the shape network. Input patterns are presented at layer 0. At layers 1 and 2, the MaxWgt error augmentation algorithm is used to adapt the weights (see section 4.3.2). At layers 3 and 4, GDR is used. Training is provided directly to layer 4, according to the patterns that would occur in the fixed network.

Using a set of 100 patterns, consisting of one of 47 shapes at one of 25 locations, the performance of this network reaches 61% after 20000 cycles. The significance of this simulation is in demonstrating that the low level features developed mostly through a bottom-up process, are sufficient to produce the features at higher layers that are required by the translation-invariant recognition task. On the other hand, using only the self-organizing component in the lower two layers yields a performance of only 11%. This indicates the need for some error correction component at the lower layers of the network.

5.3.3. Transition

In this section we demonstrate the ability of the network to learn the gradual mapping of retina-centered to object-centered features. For this purpose, we simulate layers 2, 3, 4, 2^c, 3^c and 4^c of the 2-D Translation network. Inputs are supplied by forcing the activations at layers 2 and 2^c to correspond to those obtained in the fixed connection network. Similarly, training is provided to layer 4 by directly supplying error signals that correspond to desired activity patterns as they would occur in the fixed connection network.

Learning in this sub-network is particularly difficult because it involves two context layers, each being prone to reaching node deadlocks, and involves one two-layer and one three-layer backpropagation path. Nodes that are three layers removed from direct training generally receive weak training signals.

In section 5.2 we observed the deficiencies of the GDR algorithm for learning in the upper context layer (4^c). An error modification rule was used at this layer to achieve a higher level of performance. Similarly for this network, error modification is used at both context layers to obtain sufficient differentiation of function among the nodes to allow learning to take place. Furthermore, the length of the backpropagation path suggests that error signals to layer 3^c in particular will be weak. For this reason, we apply error augmentation in the context layers along with error modification.

This simulation uses 100 input patterns consisting of one of 47 shapes and 9 locations presented to layer 2. After 20000 cycles, network performance reaches 80%, but does not improve during an additional 10000 cycles. Although the combination of new learning algorithms leads to only 80% performance on this task, the result is encouraging considering the difficulty of the task.

5.4. Discussion and summary

In this chapter we have demonstrated a network model of shape classification that is capable of translation-invariant recognition. In addition, this 2-D Translation network has the potential to "recognize familiar objects from novel viewpoints" by developing, through learning algorithms, representations of the shape and location information in such a way that generalization of shape information occurs across locations.

Although the scale of the problem we have been addressing is small relative to other computer vision models, the problem is difficult and the network is large relative to other connectionist learning work. While some work in this latter area has been concerned with finding useful applications for small adaptive networks [Sejnowski & Rosenberg 1986], we are interested in developing larger adaptive networks for application to a wider range of tasks. The disparity in scale between realistic computer vision tasks, and tasks that can currently be learned in connectionist networks is wide.

The 2-D translation network has not been designed as a minimal network for performing translation-invariant recognition. Rather, it is intended to demonstrate the general characteristics of a network that can perform the desired behavior. The result is an eight layer deep network, with a five layer computation path in the shape subnetwork, a four layer path for computing location, and a four layer path and a three layer path for computing shape from location. In addition, context layers 3^c and 4^c are composed of single clusters of equipotential nodes that are prone to developing deadlocks. This network and task are too complex for current learning algorithms.

On the other hand, this network is appropriate for the study of adaptive algorithms because its complexity is just beyond the limits of current learning algorithms. The series of simulations involving the various subnetworks that comprise the 2-D translation network demonstrate the adaptive and computational characteristics of these subnetworks. The results of these simulations indicate that the new learning algorithms presented here represent an improvement over previous algorithms. We can expect further work extending the current approach to allow the entire 2-D translation network to learn this recognition task.

The fixed connection simulation includes all layers but involves no adaptive connections. The weight on each node input is predefined in this case to yield appropriate behavior of the overall network. This simulation provides a demonstration that the overall structure of the network is adequate to represent a set

of features capable of solving the recognition task. In addition, the representations that are used at each layer of the fixed connection simulation have been useful as input or output patterns for the subsequent simulations where only some of the layers are considered.

The key simulation is that involving the top three layers of the shape network, along with the upper layer of the context network. In this simulation two important characteristics of the network are demonstrated. First, the nodes at layer 4 are able to learn features of the input shapes having an object-centered frame of reference. These features are then used by the nodes in layer 5 to perform the classification of each shape. Second, when the network is trained using only some of the shape/location combinations, it is subsequently able to correctly classify a large portion of the image patterns associated with the remaining shape/location combinations. It is this ability to generalize that allows a familiar shape to be recognized in a novel location.

The remaining three simulations demonstrate further requirements and capabilities of the overall 2-D Translation network in learning the classification task. The context simulation demonstrates that context information can be learned from information extracted from the image. Although a more sophisticated mechanism is needed to allow the network to focus on one object in a multi-layer scene, this model is sufficient for simpler single object scenes.

The retinotopic simulation demonstrates that a component of self organization, coupled with a mechanism that evaluates the usefulness of features to a supervised task, can be used to overcome the weakness of training signals in lower layers of the network. The use of output weights as a measure of usefulness addresses the related problem of drop-out nodes in a network that uses backpropagation of errors.

Finally, the transition simulation demonstrates that the context network can learn to share the representation of location information among multiple layers such that a gradual transition from retina-centered to object-centered features occurs. This gradual transition reduces the number of connections required to accomplish the mapping and fits more closely with the hierarchical locally connected structure of the pyramid.

Chapter 6

Conclusion

This thesis describes and examines a network model of shape recognition that learns to recognize shapes and their corresponding transformations. In addition, we have presented extensions to backpropagation learning methods that exhibit improved learning performance over previous methods. Here we summarize the details of the model, the results of the simulations and the contribution that this research makes in the fields of computer vision and machine learning. The chapter concludes with a discussion of future extensions to the current work.

6.1. Summary of the model

Our model of transformation-invariant shape recognition takes the form of a connectionist network that combines the hierarchical representation of shape described by Uhr [1972] with the conjunctive computation of invariants described by Hinton [1981] and Ballard [1984]. A particular example of the model is implemented in the 2-D Translation network, which recognizes various stick figure patterns under all translations within the image. The network consists of 675 processing elements and 10228 connections.

In the lower layers of the network, each node represents a feature in a specific location of the input image. These retinotopic features are mapped to object-centered features represented by nodes in the upper layers of the network. This mapping is

mediated by the value of the object location, which is represented in the context sub-network. The effect of this mapping is to invert the transformation, simplifying the recognition process.

In training the network, an object pattern is presented to the network as a 2-D image, and the translation of the object within the image is provided as a context pattern. Once the network output values are computed, a training pattern of desired values is supplied at the output. A modified backpropagation algorithm is used to adjust the connection weights in the network based on this training pattern.

The goal in training this network is to develop a representation of both shape and location that generalizes from one image pattern to another. By representing object-centered features in the upper layers of the network, and explicitly representing the location information, we have demonstrated that the system has the ability to correctly classify a shape that has been previously encountered, in an image location where the shape has never been experienced. To achieve this type of generalization was a key motivation in designing the network.

6.2. Summary of learning experiments

There has been much work in recent years on developing learning algorithms for multi-layer networks and applying these algorithms to specific network learning tasks. In Chapter 4 a number of simulations are presented which point out weaknesses in the generalized delta rule and demonstrate mechanisms which exhibit improved performance over the GDR. The weaknesses of the GDR that are

addressed in this work are:

(1) The tendency to find a weight configuration that represents a local minimum in the error function, rather than a global minimum,

(2) The weakness of the training signals at lower layers due to the successive estimation of errors from layer to layer, and

(3) The tendency for nodes representing features that are useless to the task to remain useless rather than developing different features.

To address these problems, the new algorithms use local interactions among clusters of nodes to coordinate the assignment of credit to each individual connection weight in the network.

When each element in a cluster of nodes has an equal potential for representing features, error modification techniques can be used to effect a coordinated sharing of the representational burden. Error modification involves a competition among nodes in a cluster for the error signal propagated from higher layers. This competition reduces the duplication of function that causes deadlocks in standard backpropagation methods. Different error modification mechanisms result from changing the basis of competition among nodes, and from the way errors are modified for the winners and losers of the competition.

In modifying the training errors received by the cluster, it is important to retain the essence of the training information. The error signal that has the most effect in the standard backpropagation method is that having the largest absolute magnitude.

One form of error modification is to simply retain that error as is, and set all others to zero. The simulations of the XOR network (section 4.2.1) demonstrate that this mechanism does indeed improve the learning behavior of the network over the GDR mechanism. Further improvements are obtained by attenuating the error signals for non-maximum nodes, or by setting their polarities opposite that of the maximum error. The simulation of the 3-bit Rotate network (section 4.2.2) demonstrates that this last error modification mechanism is particularly effective in the training of this conjunctively connected network.

The most effective form of error modification, demonstrated in Chapter 4 for the 2-D Shift network, is one in which the two nodes in a cluster receiving the extremes of the cluster error signals are updated. The maximal error is modified to $1-o$, to mimic a desired output of 1, while the minimal error is modified to $-o$, to mimic a desired output of 0. All other errors are modified similarly according to their signs, but attenuated by a factor of 10. For the 2-D Shift problem, this mechanism resulted in an average performance of 90%, compared with an average performance of 38% for the standard backpropagation rule (GDR). Error modification techniques reduce the likelihood of reaching a local error minimum due to deadlock by providing different training signals to the nodes in a cluster.

When the error signals to the lower layers of a network are weak, error augmentation learning techniques are useful. These techniques rely on a self-organizing component to the learning mechanism which develops features based on

regularities in the input patterns. The trace comparison rule (TCR) for self-organization leads to a representation in which each node in a cluster tends to represent a different feature from all other nodes. This representation simplifies the classification task at higher layers of the network, leading to faster learning of the task. The single cluster classifier simulation described in section 4.3.1 demonstrates the effectiveness of the TCR.

In order to make use of a self-organizing mechanism in learning a task in which specific output patterns are desired, some form of supervised learning must be included. We have used the output weights of each node to mediate the combination of the self-organization and error correction learning components. An output weight provides a measure of usefulness of the feature represented by a given node. A low output weight indicates a feature that is not being used in the discriminations taking place at higher layers. In addition, a node with a low output weight receives very little training through backpropagation. Such a node is a good candidate for self-organized learning. Nodes with higher output weights represent useful features and receive strong training signals.

The simulations of section 4.3.2 demonstrate the improved performance in a three layer network, when the backpropagated errors are augmented by 'errors' produced by a self-organizing mechanism. The standard backpropagation rule achieves an average performance of 85% in 13000 cycles on the classification task. The combined GDR/TCR mechanism leads to 100% performance in an average time

of 8000 cycles for eight testcases, and to 94% average performance in the remaining two testcases.

The simulations of Chapter 5 demonstrate the capabilities of the 2-D Translation network in learning to recognize translated objects. In addition, the learning algorithms introduced in Chapter 4 are shown to be of practical value in solving the learning task associated with this network. The main result of Chapter 5 is the demonstration that object-centered representations can be learned within the structure provided. This allows the network to correctly generalize to unknown patterns so that familiar shapes in unfamiliar locations can be recognized.

The remaining simulations in Chapter 5 demonstrate the learning capabilities of other components of the 2-D Translation network. These include the learning of context information from image input, the learning of a multi-level representation of the context information, and self-organization in the lower layers of the network.

6.3. Contributions

The main contributions of this work are the development of learning algorithms that combine backpropagation of error with cluster level interactions of processing elements, and the demonstration of a connectionist network that can learn to recognize objects under transformation.

6.3.1. Multi-layer learning

The use of gradient descent methods to reduce network error makes the process susceptible to reaching local error minima. Since gradient descent is an otherwise useful heuristic for searching the space of network weight configurations, it is worthwhile to attempt to overcome this major drawback. The technique of modifying the error signals derived by the gradient descent computation addresses a particular form of the local minimum problem, that caused by equipotential nodes taking on the same function. Error modification discourages multiple nodes within a cluster from taking on identical functions by enhancing the contrast among training signals presented to each node.

Backpropagation techniques provide training signals to all nodes in the network by estimating the errors for nodes at one layer based on the errors at the next layer. The accuracy of these estimates degrades as the nodes get farther from the top layer in the network. To compensate for this weak training at lower layers, error augmentation mechanisms introduce a bottom-up component to the weight update rule. A self-organizing mechanism develops node features that are based on the regularities in the input patterns. To the extent that these regularities reflect semantic distinctions important to the discrimination task being performed, the features developed by self-organization will be useful to higher level processing. As features develop that are useful to the task, the training signals to the corresponding nodes improve, until at some point backpropagated errors are sufficient for the training of

the network. The output weights of the nodes mediate this transition from self organization to error correction.

This work advances the field of machine learning by describing and demonstrating these two enhancements to the standard gradient descent backpropagation techniques. In addition, these cluster level interactions suggest that a higher level of design than individual nodes may be a fruitful approach to connectionist modeling. The traditional connectionist philosophy is that a large number of very simple, independently controlled processing elements can be made to interact in such a way that useful global behaviors result. The cluster provides a more complex computational unit that coordinates the activity of a group of nodes. Error modification and error augmentation provide cluster level mechanisms that represent an improvement in learning performance over mechanisms which treat each individual node separately.

6.3.2. Object recognition

The 2-D Translation network demonstrates an approach to viewpoint invariant object recognition in which the transformation information is represented explicitly. This information is used to invert the transformation producing object-centered features that simplify the recognition process. The development of object-centered features is responsible for the generalization capability of the network. The ability to generalize across different transformations of a given object has two desirable effects. First, it allows familiar objects to be recognized from novel viewpoints. Second, it

increases the efficiency of the learning process, since the learning of one pattern provides partial knowledge about other patterns as well. Subsequent learning of these other patterns is faster since they are already partially represented in the network.

6.3.3. Representation of knowledge

The method of conjunctively combining two sources of information has applications beyond the computation of translation invariant shape features. In addition to handling other 2-D and 3-D transformations, non-rigid transformations might also be represented in this way. For instance, the context network could be used to represent the regularities in the non-rigid deformations of characters written by different people. We have presented a means of combining two sources of information in a hierarchical fashion, which reduces the connectivity requirements and thus enhances the ability of the network to learn the task of interest.

6.4. Limitations

We have made numerous simplifications in designing this network model to allow basic issues to be studied. To adequately evaluate the contributions of this work, we discuss the ramifications of these simplifications.

The small size of the network simplifies its simulation on a serial computer. This restriction on size leads to the use of small images of simple objects. The question that must be addressed is: How does this 15 pixel diameter model relate to more realistic models of computer vision?

We presume that the types of operations that occur in our small networks are exactly what will be required in a larger system as well. The assumption from the start has been that the basic mechanisms of pattern matching and evidence accumulation using interactions among simple processors that communicate through weighted connections is the basis of computation in very large networks. In making this assumption, we must also address the time complexity of the computations we are describing.

In the hierarchical layered networks studied here, the time required to get a response after a stimulus has been presented is proportional to the number of layers in the network. Using a connectivity between layers in which the diameter of the retinal receptive field of each node doubles at each successive layer, the number of layers needed is proportional to the base 2 logarithm of the diameter of the image. Thus, the response time of the network increases as the log of the image diameter. This is very encouraging, but two other temporal characteristics must be considered.

First, the need for feedback from higher layers to lower ones requires a relaxation of the network whose time complexity is not well understood. In the worst case, a large complex network may not settle at all. In the next worst case, the settling time may be proportional to the number of nodes in the network. This must certainly be avoided. Any relaxation procedure that is used must have a time complexity that is proportional to the longest distance between nodes in the network. Then by insuring that this distance increases as the log of the image diameter, an

acceptable time complexity is achieved.

Second, the rate of learning as a function of network size is generally not well understood. The types of incremental learning algorithm that are used in these networks often require thousands of presentations of input stimuli before a particular task is learned. As image diameter increases, the number of possible input patterns increases exponentially. However, the hierarchical representation of these patterns reduces the combinatorics when pattern regularities exist. It is desirable for the learning algorithms to scale in this hierarchical manner as well. One of the goals of this thesis has been to find learning techniques that more efficiently solve the credit assignment problem for larger networks, by imposing greater local structure.

Another simplification is the lack of redundancy that has been built into the system. It is unrealistic to assume that a large network of the type studied here could be built in either neural tissue or from silicon chips that would have all units function without failure or error. What will certainly be required is a sufficient degree of redundancy to allow adequate network behavior to proceed given the expected failure rate of the technology from which the network is built. It is well known that some representations are inherently redundant, while others require more explicit replication of units. In any case, the number of units required for a minimal, non-redundant representation must be increased by the desired redundancy factor to obtain a useful system. This may require that cluster interactions enhance multiple nodes instead of single ones, but does not change the overall approach presented here.

In trying to relate massively parallel models to the structure and behavior of biological neural networks, an issue that must be addressed is how or how well do our processing elements model neurons? McCulloch and Pitts first abstracted neural behavior to the linear threshold element. The most general description of neural behavior is that a neuron integrates information coming from many sources through connections of varying efficacy. This description also matches the behavior of the LTE and many of its extensions. However, the details of how this integration of information takes place has been much simplified in our processing elements. The major simplifications are in the number of inputs (10^4–10^5 -vs- 10^1–10^2), the combining functions (non-linear -vs- linear) the effect of previous activations and input values (we use simple trace mechanisms), more global effects mediated by neural transmitters that may alter the computation, various temporal characteristics including refactory periods, frequency encoding of output, etc. Although the question of how much computational power is lost in these simplifications is still open, we can satisfy ourselves regarding the usefulness of our models in two ways. First, McCulloch and Pitts demonstrated that even the lowly LTE is sufficient as the only component type for a network that can perform any desired function. Thus, we presume that we are working with an element that can perform interesting computations when used in a network of similar components. Second, for the sake of machine intelligence, we do not require a close match to neural architecture, only a close match to intelligent behavior. In this case the usefulness of the processing element is borne out by the demonstrations of the network behavior.

The most difficult question of simplification that we will address here is that of the use of error correction learning. In error correction learning, a supervising agent in the environment provides relatively detailed information to the network concerning its response to the most recent stimulus. In our networks, this information indicates which output nodes should have fired, and by how much. This is a fairly sophisticated form of feedback which simplifies the task for the network, but may constrain the applicability of the approach when considering realistic problems. A weaker form of supervision is that provided in reinforcement learning situations. Here a supervising agent provides an indication to the network as to whether its response was appropriate or inappropriate to the given stimulus. Learning in this case requires that the system *guess* what might have been a more appropriate response whenever it is told that an inappropriate response was given. To the extent that such guessing can be assumed to eventually lead to the correct response being guessed, we can treat the error correction learning scheme as a finesse of certain issues in reinforcement learning. That is, the additional information provided by the error correction supervisor allows a faster learning rate, but does not provide qualitatively different information than what can be inferred eventually by a reinforcement learning system. This is an assertion that requires a more formal analysis than we present here.

It might be argued further that even a reinforcement learning mechanism is more than is required for many tasks. The self-organizing scheme described earlier, for instance, allows learning to occur in the absence of any supervising agent.

Furthermore, even with a supervising agent, a learning mechanism that uses only correlational type synapses rather than supervisor inputs can be used to solve at least some learning tasks, as demonstrated, for instance, by Rumelhart and Zipser [1985]. This type of solution obviates the need for propagating special error signals throughout the network. If such a mechanism should be demonstrated as sufficient for learning in the types of domains generally treated with reinforcement and error correction mechanisms, they would likely be preferred for their biological plausibility and simplicity of concept. On the other hand, information similar to that currently carried in error signals would likely need to be propagated by element activations, in which case the current approach would be useful in determining just what that information should be.

6.5. Future work

There are a number of ways in which the research described here can be extended. The most obvious of these is in bringing together the various subnetworks described in Chapter 5 to allow the entire 2-D Translation task to be learned at once. This task would require different, possibly time varying, learning coefficients to be used in different parts of the network, and may require further refinements in learning methods. Other learning experiments that could be performed on the 2-D Translation network or its subnetworks include learning in the presence of noisy and deformed images, learning in the presence of multi-object images, and development of representations that generalize across similar contexts.

Another way in which this work could be extended is in implementing networks that learn to recognize objects under various 2-D rotations and magnifications. A network capable of such recognition has a structure very similar to the one for 2-D translation. A 2-D viewpoint network can be created by combining the networks for the various transformations. The extension to 3-D transformations is straightforward in principle, though the increased size of the network would require powerful learning algorithms.

In addition to the general viewpoint invariance problem for rigid objects, we are interested in the applicability of this approach to problems involving non-rigid body transformations. Face recognition is an example of this type of problem. The representation of a particular face would be most useful to a recognition process if it were invariant to the non-rigid transformations associated with facial expression. A related problem, that is more tractable for current learning algorithms and simulation techniques, is that of character recognition. Characters produced in different fonts represent distinct transformations of the 'canonical' character. Sanocki [1986] explains psychological data on character recognition by proposing an explicit representation of font that is used to modulate the recognition process. This model agrees closely with the notion of conjunctively combining shape information with a description of the transformation. Similarly, characters produced by different writers can be treated as transformations of a canonical set of characters. To the extent that a given writer is systematic in the treatment of character features, this transformational representation will be effective.

The extension to non-rigid transformations is interesting for a number of reasons. First, it underlines the need for a learning mechanism, since the regular structure of rigid body transformations does not apply. Second, it emphasizes the general nature of this transformational approach to computation. That is, the use of conjunctive connections to implement transformations can be more generally viewed as a means of incorporating context information into a recognition process. Thus, the interpretation of features from each level of the network to the next can be modified according to current context. In the viewpoint networks, the context information relates to a rigid-body transformation of the object from some canonical spatial situation. In the character recognition networks, the context relates to the source of formation of the character, since different sources will produce different deformations. Another use for this structure would be in combining information from different sensory modes.

Finally, as a means of defining useful representations of image data, it is desirable to apply the methods discussed here to a sensory-motor task. By defining desired behaviors in terms of motor activity, the artificial nature of the output representation is eliminated. In addition, assumptions regarding the amount of information available to the trainer can be considered in a more realistic context.

References

Ackley, D. H., G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, vol. 9, pp. 147-169, 1985.

Ahuja, N. and S. Swamy, "Interleaved Pyramid Architectures for Bottom Up Image Analysis," *Proc. IJCPR*, vol. 6, pp. 388-390, 1982.

Amari, S-I., "Neural Theory of Association and Concept-Formation," *Biological Cybernetics*, vol. 26, pp. 175-185, 1977.

Amari, S-I., "Field Theory of Self-Organizing Neural Nets," *IEEE Trans. on Sys. Man and Cybernetics*, vol. SMC-13, September/October 1983.

Anderson, C. W., "Learning Problem Solving with Multilayer Connectionist Systems," PhD Dissertation, UMass - Amherst, September 1986.

Attneave, F., "Some Informational Aspects of Visual Perception," *Psychological Review*, vol. 61, 1954.

Augusteijn, M. F. and C. R. Dyer, "Model-based Shape from Contour and Point Patterns," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pp. 100-105, 1985.

Ballard, D. H., "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, vol. 13, pp. 111-122, April 1981.

Ballard, D. H., "Strip Trees: A Hierarchical Representation for Curves," *CACM*, vol. 24, pp. 310-321, May 1981.

Ballard, D. H., "Parameter Nets," *Artificial Intelligence*, vol. 22, pp. 235-267, 1984.

Barrow, H. G. and J. M. Tenenbaum, "Recovering Intrinsic Scene Characteristics from Images," in *Computer Vision Systems*, ed. A. R. Hanson and E. M. Riseman, pp. 3-26, Academic Press, New York, 1978.

Barrow, H. G. and J. M. Tenenbaum, "Computational Vision," *Proc. IEEE*, vol. 69, pp. 572-595, May 1981.

Barto, A. G. and P. Anandan, "Pattern-Recognizing Stochastic Learning Automata," *IEEE Trans. Sys. Man, and Cybernetics*, vol. SMC-15, pp. 360-375, May/June 1985.

Batcher, K. E., "Design of a Massively Parallel Processor," *IEEE Transactions on Computing*, vol. 29, pp. 836-840, 1980.

Block, H. D., N. J. Nilsson, and R. O. Duda, "Determination and detection of features in patterns," in *Computer and information sciences: Collected papers in learning, adaptation, and control in information systems*, ed. J. T. Tou and R. H. Wilcox (eds), pp. 75-110, Spartan Books, Washington, D. C., 1964.

Block, H. D., "A Review of "Perceptrons: An Introduction to Computational Geometry"," *Information and Control*, vol. 17, pp. 501-522, 1970.

Carpenter, G. A. and S. Grossberg, "Adaptive Resonance Theory: Stable self-organization of neural recognition codes in response to arbitrary lists of input patterns," *Proc. 8th Conf. of the Cognitive Science Society*, pp. 45-62, Amherst, MA, August 1986.

Chen, P. C. and T. Pavlidis, "Segmentation by Texture Using a Co-Occurrence Matrix and a Split-and-Merge Algorithm," *Computer Graphics and Image Processing*, vol. 10, pp. 172-182, 1979.

Cibulskis, J. M. and C. R. Dyer, "An Analysis of Node Linking in Overlapped Pyramids," *IEEE Trans. Systems, Man and Cybernetics*, vol. 14, pp. 424-436, 1984.

Cottrell, G. W., "Connectionist Parsing," *Proc. Seventh Annual Conf. of the Cognitive Science Society*, Irvine, CA, 1985.

Crowley, J. L., "A Multi-Resolution Representation for Shape," *Proc. ICVPR*, pp. 326-335, 1983.

Davis, L. S. and A. Rosenfeld, "Applications of Relaxation Labeling, 2: Spring-loaded Template Matching," *Proc. IJCPR*, vol. 3, pp. 591-597, 1976.

Davis, L. S., "Understanding Shape: Angles and Sides," *IEEE Trans. on Computers*, vol. C-26, pp. 236-242, March 1977.

Duff, M. J. B., "CLIP4: A Large Scale Integrated Circuit Array Parallel Processor," *Proc. IJCPR*, vol. 4, pp. 728-733, 1976.

Dyer, C. R., A. Rosenfeld, and H. Samet, "Region Representation: Boundary Codes from Quadtrees," *CACM*, vol. 23, pp. 171-179, March 1980.

Feldman, J. A., "A Distributed Information Processing Model of Visual Memory," Rochester Tech Report TR52, December 1979.

Feldman, J. A., "Dynamic Connections in Neural Networks," *Biological Cybernetics*, vol. 46, pp. 27-39, 1982.

Feldman, J. A. and D. H. Ballard, "Connectionist Models and Their Properties," *Cognitive Science*, vol. 6, pp. 205-254, 1982.

Feldman, J. A., "Four frames suffice: A provisional model of vision and space," *The Behavioral and Brain Sciences*, vol. 8, pp. 265-289, 1985.

Flanders, P. M., D. J. Hunt, S. F. Reddaway, and D. Parkinson, "Efficient High Speed Computing with the Distributed Array Processor," in *High Speed Computer and Algorithm Organization*, ed. D. J. Kuck, D. H. Lawrie, and A. H. Sameh, pp. 113-128, Academic Press, New York, 1977.

Freeman, H., "Computer processing of line drawing images," *Computer Surveys*, vol. 6, pp. 57-98, March 1974.

Fukushima, K., "Cognitron: A Self-organizing Multilayered Neural Network," *Biological Cybernetics*, vol. 20, pp. 121-136, 1975.

Fukushima, K., "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics*, vol. 36, pp. 193-202, 1980.

Grossberg, S., *Studies of Mind and Brain: Neural Principles of learning, perception, development, cognition and motor control*, Reidel Press, 1982.

Grossberg, S. and G. Stone, "Neural Dynamics of Word Recognition and Recall: Attentional Priming, Learning, and Resonance," *Psychological Review*, vol. 93, pp. 46-74, 1986.

Hanson, A. R. and E. M. Riseman, "Segmentation of Natural Scenes," in *Computer Vision Systems*, ed. A. R. Hanson and E. M. Riseman, pp. 129-163, Academic Press, Inc., New York, 1978.

Hebb, D. O., *The Organization of Behavior*, Wiley, New York, 1949.

Hinton, G. E., "A Parallel Computation that Assigns Canonical Object-Based Frames of Reference," *Proc. 7th IJCAI*, pp. 683-685, Vancouver, B.C., 1981.

Hinton, G. E., "Implementing Semantic Nets in Parallel Hardware," in *Parallel Models of Associative Memory*, ed. G. E. Hinton and J. A. Anderson (eds), Erlbaum Associates, Hillsdale, N. J., 1981.

Hinton, G. E., "Learning Distributed Representations of Concepts," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 1-12, Erlbaum Associates, Amherst, MA., August 1986.

Holland, J. H., "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," in *Machine Learning, Volume II*, ed. R. Michalski, J. G. Carbonell and T. M. Mitchell (eds.), Morgan Kaufmann, Los Altos, 1986.

Hong, T. H., K. A. Narayanan, S. Peleg, A. Rosenfeld, and T. Silberberg, "Image Smoothing and Segmentation by Multiresolution Pixel Linking: Further Experiments and Extensions," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-12, pp. 611-622, September/October 1982.

Horn, B. K. P., "Obtaining Shape from Shading Information," in *The Psychology of Computer Vision*, ed. P. H. Winston, McGraw-Hill, New York, 1975.

Hubel, D. H. and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J Physiol (London)*, vol. 160, pp. 106-154, 1962.

Ikeuchi, K., "Numerical Shape from Shading and Occluding Contours in a Single View," MIT AI Tech Report AI Memo 566, February 1980.

Ikeuchi, K., "Shape from Regular Patterns (an Example of Constraint Propagation in Vision)," *Proc. IJCPR*, vol. 4, pp. 1032-1039, 1980.

Kabrisky, M., O. Tallman, C. M. Day, and C. M. Radoy, "A Theory of Pattern Perception Based on Human Physiology," in *Contemporary Problems in Perception*, ed. A.T. Welford and L. Houssiadas, Taylor & Francis Ltd, London, 1970.

Kanade, T., "Recovery of the Three-Dimensional Shape of an Object from a Single View," *Artificial Intelligence*, vol. 17, pp. 409-460, 1981.

Kelly, M. D., "Edge Detection in Pictures by Computer Using Planning," in *Machine Intelligence*, ed. B. Meltzer and D. Michie, vol. 6, pp. 397-409, University Press, Edinburgh, 1971.

Kender, J. R., "Shape from Texture: An Aggregation Transform That Maps a Class of Textures into Surface Orientation," *Proc. 6th IJCAI*, pp. 475-480, Tokyo, 1979.

Kirkpatrick, S., C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, May 1983.

Kohonen, T., "Self-Organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, vol. 43, pp. 59-69, 1982.

LeCun, Y., "Learning process in an asymmetric threshold network," in *Disordered Systems and Biological Organization*, ed. Bienenstock, E.,F. Fogelman, G. Weisbuch (eds.), Springer, 1986.

Lettvin, J. Y., H. Maturana, W. S. McCulloch, and W. Pitts, "What the Frog's Eye Tells the Frog's Brain," *Proceedings of the IRE*, vol. 47, pp. 1940-1951, 1959.

Levine, M. D., "Region Analysis Using a Pyramid Data Structure," in *Structured Computer Vision*, ed. S. Tanimoto and A. Klinger, pp. 57-100, Academic Press, New York, 1980.

Malsburg, C. von der, "Self-Organization of Orientation Sensitive Cells in the Striate Cortex," *Kybernetik*, vol. 14, pp. 85-100, 1973.

Marr, D., "Representing Visual Information - a Computational Approach," in *Computer Vision Systems*, ed. A. R. Hanson and E. M. Riseman, pp. 61-80, Academic Press, New York, 1978.

Marr, D., *Vision,* Freeman, San Francisco, 1982.

Maxwell, T., C. L. Giles, Y. C. Lee, and H. H. Chen, "Nonlinear Dynamics of Artificial Neural Systems," *Conference on Neural Networks for Computing*, Snowbird, Utah, 1986.

McCarthy, J., "Recursive Functions of Symbolic Expressions and their Computation by Machine, Part I," *Comm. ACM*, vol. 7, April 1960.

McCarthy, J., "History of LISP," *SIGPLAN Notices*, vol. 13, 1978.

McClelland, J. L. and D. E. Rumelhart, "An interactive activation model of the effect of context in perception: Part I," *Psychological Review*, vol. 88, pp. 375-407, 1981.

McClelland, J. L. and J. L. Elman, "The TRACE Model of Speech Perception," *Cognitive Psychology*, vol. 18, pp. 1-86, 1986.

McCulloch, W. S. and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-137, 1943.

McCulloch, W. S., *Embodiments of Mind,* MIT Press, Cambridge, 1965.

Minsky, M. and S. Papert, *Perceptrons: An Introduction to Computational Geometry,* MIT Press, Cambridge, 1969.

Nass, M. M. and L. N. Cooper, "A Theory for the Development of Feature Detecting Cells in Visual Cortex," *Biological Cybernetics*, vol. 19, pp. 1-18, 1975.

Parker, D. B., "Second Order Backpropagation: An Optimal Adaptive Algorithm for any Adaptive Network," *Proc. IEEE 1st ICNN*, San Diego, June 1987.

Persoon, E. and K. S. Fu, "Shape discrimination using Fourier descriptors," *Proc. 2nd IJCPR*, pp. 126-130, August 1974.

Rosenblatt, F., "The perceptron: A perceiving and recognizing automaton," Rep. No. 85-460-1, Project PARA, Cornell Aeronautical Laboratory, Buffalo, NY, 1957.

Rosenblatt, F., *Principles of Neurodynamics*, Spartan Books, New York, 1962.

Rosenfeld, A. and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1976.

Rumelhart, D. E. and D. Zipser, "Feature Discovery by Competitive Learning," *Cognitive Science*, vol. 9, pp. 75-112, 1985.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing Volume 1*, ed. D. E. Rumelhart and J. L. McClelland (eds.), pp. 318-362, Bradford Books, Cambridge, MA., 1986.

Sabbah, D., "Computing with Connections in Visual Recognition of Origami Objects," *Cognitive Science*, vol. 9, pp. 25-50, 1985.

Samuel, A. L., "Some studies in machine learning using the game of checkers," *IBM J. Research and Development*, vol. 3, pp. 210-229, 1959.

Sandon, P. A., "A Pyramid Implementation using a Reconfigurable Array of Processors," *Proc. CAPAIDM*, pp. 112-118, Miami Beach, FL, November 1985.

Sanocki, T, "Visual Knowledge Underlying Letter Perception: A Font-Specific Approach," PhD Dissertation, University of Wisconsin - Madison, 1986.

Sejnowski, T. J. and C. R. Rosenberg, "NETtalk: A Parallel Network that Learns to Read Aloud," Johns Hopkins EECS Technical Report JHU/EECS-86/01, 1986.

Selfridge, O. G., "Pandemonium: a Paradigm for Learning," in *Proc. Symposium on Mechanization of Thought Processes*, ed. D. Blake & A. Uttley, H. M. Stationary Office, London, 1959.

Shannon, C. E., "Programming a Computer for Playing Chess," *Philosophical Magazine*, vol. 41, 1950.

Shastri, L. and J. A. Feldman, "Semantic Networks and Neural Nets," Rochester Tech Report #131, 1984.

Sutton, R. S. and A. G. Barto, "Toward a Modern Theory of Adaptive Networks: Expectation and Prediction," *Psychological Review*, vol. 88, pp. 135-170, 1981.

Sutton, R. S., "Two Problems with Backpropagation and other Steepest-Descent Learning Procedures for Networks," *Proceedings of the Eighth Annual Meeting of the Cognitive Science Society*, Amherst, MA, 1986.

Tanimoto, S. L. and T. Pavlidis, "A hierarchical data structure for picture processing," *Computer Graphics and Image Processing*, vol. 2, pp. 104-119, June 1975.

Tanimoto, S. L., "Regular Hierarchical Image and Processing Structures in Machine Vision," in *Computer Vision Systems*, ed. A. R. Hanson and E. M. Riseman, pp. 165-174, Academic Press, New York, 1978.

Tanimoto, S. L., "A Hierarchical Cellular Logic," Univ. Washington Tech Report #83-10-06, October 1983.

Tenenbaum, J. M. and H. G. Barrow, "IGS: A Paradigm for Integrating Image Segmentation and Interpretation," *Proc. IJCPR*, vol. 3, pp. 504-513, 1976.

Terzopoulos, D., "Multiresolution Computation of Visible-Surface Representations," PhD. Dissertation, MIT, January 1984.

Touretzky, D. S. and G. E. Hinton, "Symbols Among the Neurons: Details of a Connectionist Inference Architecture," *Proc. IJCAI-85*, pp. 238-243, Los Angeles, CA, August 1985.

Turing, A., "Computing Machinery and Intelligence," *Mind*, pp. 433-460, 1950.

Uhr, L. and C. Vossler, "A Pattern-Recognition Program that Generates, Evaluates and Adjusts its Own Operators," *1961 West. Joint Comput. Conf.*, pp. 555-569, 1961.

Uhr, L., "Layered "recognition cone" networks that preprocess, classify and describe," *IEEE Trans. Comput.*, vol. 21, pp. 758-768, 1972.

Uhr, L., "Recognition Cones and Some Test Results," in *Computer Vision Systems*, ed. A. R. Hanson and E. M. Riseman, pp. 363-377, Academic Press, New York, 1978.

Uhr, L. and R. Douglass, "A Parallel-Serial Recognition Cone System for Perception: Some Test Results," *Pattern Recognition*, vol. 11, pp. 29-39, 1979.

Uhr, L. and L. Schmitt, "The Several Steps from ICON to Symbol, Using Structured Cone/Pyramids," UW - Madison CS Tech Report #481, August 1982.

Ullman, S., "Visual Routines," MIT AI Memo No. 723, June 1983.

Van Essen, D. C. and J. H. R. Maunsell, "Hierarchical organization and functional streams in the visual cortex," *Trends in Neurosciences*, vol. 6, pp. 370-375, September 1983.

Waltz, D. L., "Generating Semantic Descriptions from Drawings of Scenes with Shadows," in *The Psychology of Computer Vision*, ed. P. H. Winston, McGraw-Hill, New York, 1975.

Widrow, B., "Generalization and information storage in networks of adaline "neurons"," in *Self-organizing systems*, ed. Yovits, M., G. Jacobi and G. Goldstein (Eds.), Spartan Books, 1962.

Widrow, G. and M. E. Hoff, "Adaptive Switching Circuits," IRE WESCON, Convention Record, Part 4, pp. 96-104, 1960.

Williams, R. J., "The Logic of Activation Functions," in *Parallel Distributed Processing*, ed. D. E. Rumelhart and J. L. McClelland (eds.), The MIT Press, Cambridge, 1986.

Witkin, A. P., "Recovering Surface Shape and Orientation from Texture," *Artificial Intelligence*, vol. 17, pp. 17-45, 1981.

Witkin, A. P. and J. M. Tenenbaum, "What is Perceptual Organization For?," *Proc. IJCAI-83*, pp. 1023-1026, Karlsruhe, West Germany, August 1983.

Witkin, A. P., "Scale-Space Filtering," *Proc. IJCAI-83*, pp. 1019-1022, Karlsruhe, West Germany, August 1983.