Technical Report 1086

Optimal Unsupervised Learning in Feedforward Neural Networks

Terence D. Sanger

MIT Artificial Intelligence Laboratory

REPORT DOCUMENTATION PAGE	**READ INSTRUCTIONS BEFORE COMPLETING FORM**
1. REPORT NUMBER AI-TR 1086 · 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) Optimal Unsupervised Learning in Feedforward Neural Networks	5. TYPE OF REPORT & PERIOD COVERED technical report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Terence D. Sanger	8. CONTRACT OR GRANT NUMBER(s) DACA76-85-C-0010 N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209	12. REPORT DATE January 1989
	13. NUMBER OF PAGES 129
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*) Office of Naval Research Information Systems Arlington, Virginia 22217	15. SECURITY CLASS. (*of this report*) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

neural networks	vision
learning	eigenvector analysis
connectionism	Karshunen-Loeve Transform

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

We investigate the properties of feedforward neural networks trained with Hebbian learning algorithms. A new unsupervised algorithm is proposed which produces statistically uncorrelated outputs. The algorithm causes the weights of the network to converge to the eigenvectors of the input correlation with largest eigen-

Optimal Unsupervised Learning in Feedforward Neural Networks

by

Terence D. Sanger
A.B. Applied Mathematics, Harvard University (1985)
S.M. Applied Mathematics, Harvard University (1986)

submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE,

Area VII

in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

at the

Massachusetts Institute of Technology

January, 1989

Signature of Author ____

Dept. Electrical Engineering and Computer Science, and HST
January, 1989

Certified by ____

Prof. Tomaso Poggio
Thesis Supervisor

Accepted by ____

Prof. Arthur C. Smith
Chairman, Committee on Graduate Students

Dedication

This thesis is dedicated to Abbie Wolfson, for her patience, inspiration, and support.

Optimal Unsupervised Learning in Feedforward Neural Networks

Terence D. Sanger

Submitted to the Department of Electrical Engineering and Computer Science on January 20, 1989, in partial fulfillment of the requirements for the degree of Master of Science.

Abstract

We investigate the properties of feedforward neural networks trained with Hebbian learning algorithms. A new unsupervised algorithm is proposed which produces statistically uncorrelated outputs. The algorithm causes the weights of the network to converge to the eigenvectors of the input correlation with largest eigenvalues. If a network trained in this way is used as input to a layer trained using the Widrow-Hoff (LMS) algorithm, the system implements an optimal Wiener filter. The algorithm is closely related to the technique of Self-supervised Backpropagation, as well as other algorithms for unsupervised learning. The algorithm can be generalized to nonlinear networks. Applications of the algorithm to texture processing, image coding, and stereo depth edge detection are given. It is shown that the algorithm can lead to the development of filters qualitatively similar to those found in primate visual cortex.

Thesis Supervisor: Prof. Tomaso Poggio

Acknowledgements

I would like to express my gratitude to the many people who provided inspiration and whose comments, criticisms, and suggestions have helped to increase my understanding of these results. In particular, I am indebted to my advisor Tomaso Poggio, as well as to Ralph Linsker, Steve Nowlan, Rich Sutton, Ellen Hildreth, Tom Breuel, Eero Simoncelli, Joel Wein, and Bror Saxberg.

This thesis describes research done at the MIT Artificial Intelligence Laboratory, and sponsored by a grant from the Office of Naval Research (ONR), Cognitive and Neural Sciences Division; by the Alfred P. Sloan Foundation; by the National Science Foundation; by the Artificial Intelligence Center of Hughes Aircraft Corporation (S1-801534-2); and by the NATO Scientific Affairs Division (0403/87). Support for the A. I. Laboratory's artificial intelligence research is provided by the Advanced Research Projects Agency of the Department of Defense under Army contract DACA76–85–C–0010, and in part by ONR contract N00014–85–K–0124. The author was supported during part of this research by a National Science Foundation Graduate fellowship, and later by a Medical Scientist Training Program grant.

Contents

Chapter 1

Introduction

There has been great interest in the study of neural networks, and this interest has led to the creation of many different structures which are all called "networks". In this thesis I will discuss algorithms for training "layered feedforward networks". A layered feedforward network is a network which has a distinct set of input units onto which values are clamped. These values are then passed through a set of weights to produce the inputs to the next layer of "hidden" or internal units. These units modify their input using a nonlinear function (usually sigmoid in shape) to produce their outputs. As many layers as desired of this form can be stacked, and the units of the final layer become the outputs of the network. The purpose of a learning algorithm is to adjust the weights between units (but not the nonlinearity) so that the output gives some desired function of the input. Note that the nonlinearity is essential, since several layers of purely linear units can always be compressed into an equivalent single layer (by matrix multiplication) and therefore add no further computational ability to the network.

1.1 Why use Networks?

Networks are difficult to train, and often have trouble performing even the simplest computations. So why have they sparked so much interest? There are four basic reasons for their current popularity:

1. They can learn to perform tasks for which computational algorithms may not exist.

2. They can adapt their behavior to a changing environment.

3. They may be a model for biological information processing.

4. They operate on distributed and fault-tolerant hardware.

We should emphasize at this point that these four properties are not always found in current models, and may be very difficult to achieve using neural networks at all. At the moment there is only the hope that networks may provide a useful structure for the design of algorithms which have perhaps some, but not all, of these properties.

The first property may be the most important for practical purposes. For many problems of current interest in computer science and artificial intelligence, the level of understanding of the problem and the means to solve it is quite poor. Self-training networks provide the possibility of solving a problem which we do not completely understand. For example, in computer vision, we might want to perform texture segmentation on an image, even though we do not have a clear specification of what we mean by "texture". We do, however, have the ability to specify how any algorithm should perform on particular examples, so if a network can be trained to discover an algorithm based upon examples, then it might be able to generalize to solve the problem for cases it has not yet seen. This is a form of "programmerless programming", since we do not even need to specify the problem in an exact way, yet a "program" to solve it can be found.

Thus all that one would need to train a network to discover an algorithm is an error measure which indicates whether the network has succeeded in finding the "correct" answer.

The use of networks to discover new algorithms depends crucially on their ability to generalize. Generalization refers to a trained network's ability to find a correct output when given an input on which it has not been trained. If successful, this is an indication that the network has somehow discovered the underlying structure of the problem. If a network does somehow find this structure, one might hope that subsequent analysis of the network structure would reveal the algorithm which it had discovered and thereby contribute to our understanding of the original problem. Unfortunately, generalization is an ill-posed problem, since a finite set of samples of a desired function is consistent with an infinite-dimensional class of possible functions. We therefore need to impose further constraints on the types of functions which the network can learn, in order for it to produce a useful mapping.

The second reason for using neural networks is their inherent adaptability. If their behavior was learned in the first place, then re-learning based on a changing environment may be relatively easy to implement. This is standard behavior for higher-level biological organisms, and mimicking this type of biological behavior is a primary goal for networks.

It is also hoped that successful neural networks may be able to guide neurophysiological research by providing models for the computational abilities of large interconnected systems of simple processing units. The more we understand the mathematics of such systems, the better we may be able to analyze data provided from electrophysiology studies of primate cortex.

Finally, it seems that networks may be relatively robust with respect to malfunction or loss of components. Unlike modern digital computers, for which any hardware malfunction is catastrophic, parts of a network can be destroyed with only a proportionate decrease in processing accuracy. This may have considerable implications for wafer-scale integrated components where the yield rate of 100% functional circuits is very low, and fault-tolerance is therefore required.

Networks with these properties may provide important advantages over other methods of solving specific problems. However, it is important to realize that current models do not perform well on real-world data, and that neural networks have so far been able to solve only the simplest of tasks. Research on networks is stimulated by the hope that in the future they will provide an important set of algorithms with properties different from those of more classic techniques.

1.2 Current Algorithms

Algorithms for training neural networks to compute input-output mappings are often divided into the categories of "supervised" and "unsupervised" learning. (For review, see [Hinton, 1987, Lippmann, 1987].) Supervised learning involves a knowledgeable teacher who for each example provided as input to the network, tells the network the correct (desired) output. In unsupervised learning no such teacher is available, and the network is left to discover useful computations on its own. Of course, implicit in the unsupervised learning algorithm is a set of mappings which it is desirable for the network to learn, and the properties which describe this set of mappings were determined by the programmer and may be quite specific (even if the programmer is unable to discover them himself in closed form). For our purposes, the difference between a supervised and an unsupervised learning algorithm lies in the fact that for a supervised algorithm the object of learning is to approximate a particular desired input-output mapping, whereas for unsupervised learning the object is to find a mapping which possesses certain specified properties. Clearly, there is a continuum of learning types between these two extremes, but this intuitive notion should suffice.

The true significance of the desired properties implicit in an unsupervised algorithm is that they allow networks to generalize. As mentioned above, generalization is an ill-posed problem if we are given only a finite number of samples of a function. However, if we combine these samples with a property which describes the class of functions which we will fit to the data, then we can restrict the possible computations done by the network. In this way, we are able to add enough additional constraints to the problem that the solutions found by the network may actually perform in a reasonable manner on data not in the training set.

Designing a training algorithm for a particular type of network therefore involves two major steps. The first step is to choose a set of desirable properties which the network should possess after training, and the second step is to find a learning algorithm which causes the network to attain these properties. Since we usually do not know beforehand whether a set of weights exists which accomplishes this, it can be useful to specify the desired properties in terms of a metric on the set of networks which we attempt to maximize. This formalizes the idea of a net which is "pretty good" for solving a task which perhaps might be better solved by some other structure. The network is not actually required to achieve the desired properties, but it can come close and still be useful. By considering networks in this way, the problem of finding a learning algorithm reduces to a constrained optimization problem. Since such problems tend to be very difficult and solutions tend to be problem-specific, we might expect that learning algorithms will have to be adapted specifically to the desired properties and the particular network structure involved.

We will refer to a desirable property of a network as an "optimality principle". Examples of such principles and some feedforward networks which employ them are given in table 1.2. The optimality principle which we will use to choose an algorithm is based on the idea of maximizing information which was first proposed by Linsker [Linsker, 1986a, Linsker, 1988b]. Unfortunately measuring

TABLE 1. Some Feedforward networks and their Optimality Principles

Network	*Optimality Principle*	*Algorithm*
Backprop	RMS function approximation error	Gradient descent
Encoder BP	RMS input approximation error	Gradient descent
Topological	Output topology matches input topology	Winner-take-all
Linsker	Infomax - maximize output information	Hebbian learning

the information in network outputs can be difficult without precise knowledge of the distribution on the input data, so we seek another measure which is related to information but which is easier to compute. If instead of maximizing information, we try to maximize our ability to reconstruct the input (with minimum mean-squared error) given the output of the network, we are able to obtain some useful results. Note that this is not equivalent to maximizing information except in some special cases. However, it contains the intuitive notion that the input data is being represented by the network in such a way that very little of it has been "lost".

1.2.1 Supervised Algorithms

In supervised learning, the network is presented with an example input and the desired output, and the weights between units are modified in such a way that it will be more likely to produce that output in the future in response to that input. Success in training is usually measured by the mean squared difference between the desired and actual outputs. This constitutes the optimality principle for many supervised algorithms, but it is important to note that there is still an infinite-dimensional space of functions which may minimize the mean-squared error. The constraints on these functions are implicit in the network structure, since a particular combination of linear layers and smooth nonlinear functions may constrain the computation to have certain smoothness properties. In this sense, the network is performing a smooth approximation to the sample points which it is given as training data.

One of the earliest supervised algorithms for a single-layer network is the Widrow-Hoff algorithm [Widrow and Hoff, 1960, Widrow *et al.*, 1976]. This algorithm can be proven to converge to a network in which the outputs are the optimal linear estimate of the desired output given the input. In its original form, this algorithm was suitable only for single-layer linear networks.

A variant of the Widrow-Hoff algorithm in which the outputs are forced to take the values -1,1 is the Perceptron Learning Rule[Rosenblatt, 1962]. This algorithm is used to train a single-layer network in which the output units are processed by a thresholding nonlinearity which forces them to assume one of two values. The purpose of the network is to classify inputs into groups based on decision regions of input space. The algorithm is a straightforward variation of the Widrow-Hoff algorithm, and can also be proven to converge whenever a solution exists [Rosenblatt, 1962]. When used in a multilayered network, however, the fact that the output of the first layer is boolean valued means that subsequent layers can do no more than compute boolean functions of the decision regions defined by the first layer [Minsky and Papert, 1969]. This means that all the "interesting" learning is occurring in the first layer, and the multi-layered architecture is not adding significant processing power.

For training multilayer feedforward networks, the most common algorithm is backpropagation, discovered independently in different forms by [Werbos, 1974, Le Cun, 1985, Parker, 1985, Rumelhart *et al.*, 1986b]. The backpropagation algorithm performs gradient descent in the space of the weights between units, along gradients of an error function defined by the mean-squared error in the outputs. It is possible to determine the direction of steepest descent for each weight independently using a clever recursive algorithm which propagates "error" terms from the output units backwards through the network. Backpropagation can be thought of as the "canonical" supervised learning procedure, since it uses the least information about the problem to minimize a standard supervised-type error. Like all gradient descent methods, it cannot be proven to converge to the optimal set of weights, and convergence can be arbitrarily slow due to shallow slopes on the energy surface.

Backpropagation has been applied to many different problems, and several modifications have been proposed to decrease the training time (for a partial review, see [Hinton, 1987]). However, training time still seems to be an exponential function of the number of units in the network. (The general learning problem has been shown to be NP-complete [Judd, 1987].) Also, there is no well-developed theory describing the properties of backpropagation-trained networks with respect to function approximation or generalization. Lapedes [1987, 1988] has hypothesized that networks approximate functions by using sums of a large number of "bumps". This technique was used explicitly to compute nonlinear mappings in [Saund, 1987].

If backpropagation finds a good solution for approximating some particular function, then there is no reason to believe that the solution makes any special use of the network structure. To solve a specific problem, it may often be possible to use an extremely non-intuitive set of weights, and there are no constraints on a gradient descent algorithm to impose a structure on the network. For this reason, a backpropagation trained network may be difficult to interpret. The experimenter may be unable to discover the "algorithm" which the network is using to solve a particular problem, besides simply graphing the output surface for individual weights. These output surfaces can often be helpful in understanding network behavior (see, for example, [Lapedes and Farber, 1987, Lapedes and Farber, 1988]), but for large networks with more than three layers the surfaces may not be self-explanatory.

There is one case in which backpropagation can be said to generalize in a meaningful sense. This case involves a network "bottleneck". A bottleneck layer is a layer within a feedforward network which has fewer units than the layers above and below. This means that in order to compute its mapping, the network must somehow compress the data representation in such a way that it fits through this layer and can still be used for computations by the layers above. When a bottleneck exists, empirical observation suggests that networks do indeed generalize properly. We will discuss this idea further in section 1.4.

1.2.2 Hebbian Unsupervised Algorithms

Unsupervised algorithms typically are used for single-layer networks which have only an input and output layer, with no hidden layers in-between. An early such algorithm was proposed by [Hebb, 1949] as a model of the change in synaptic weights between neurons in biological systems. Extensions to Hebb's original model have been given by [Grossberg, 1971, Grossberg, 1976, Kohonen, 1982, Oja, 1982, Linsker, 1986a, Kohonen, 1988], among many others. Hebbian algorithms modify the connection between two units in proportion to the cross-correlation of the units' activities. A detailed discussion of this type of algorithm will be given in chapter 2.

Unsupervised algorithms require some method to force different outputs to learn different things. One approach to this problem is to have "competition" between outputs for different functions. Competitive learning has been investigated by a number of authors, including [Von der Malsburg, 1973, Fukushima, 1975, Grossberg, 1976, Kohonen, 1982, Amari, 1983, Rumelhart and Zipser, 1985]. Although the basic Hebbian algorithms are understood (see, for instance, [Oja, 1982]), the effects of winner-take-all and other forms of competitive interaction have not been sufficiently analyzed.

1.3 Multi-layer Unsupervised Learning

In order to train a multilayer network more efficiently, it would be useful to have an algorithm for which the optimal weights for each layer were dependent only on the layers below. It would then be possible to train the network "bottom-up" by allowing the lowest layers to "mature" first, and then training the higher layers. Such an algorithm would require training time which was only a linear function of the number of layers, rather than exponential. These considerations lead us to search for unsupervised algorithms for which the optimality principle relates only to the weights at one layer, and is thus independent of the final outputs of the network and the other weights above.

Unsupervised learning may therefore give a direct advantage in the scaling of training time with network depth. In addition, an already trained network can take advantage of its previous training to learn a slightly modified task without retraining every level. This is in contrast to supervised algorithms such as back-propagation for which all weights will change as soon as any output errors are detected.

The structure of a multi-layer network trained by an unsupervised algorithm at each layer may be interpretable, depending on the optimality criterion used for training. This occurs because each layer is entirely dependent on the output of the previous layer, so whatever task it is trained to perform by the learning algorithm will be performed independently of the overall function which the network computes. Each layer will perform a function entirely local to that particular layer, and the function will be determined only by the training algorithm and the inputs which it receives from previous layers. Therefore, the trained network will implement a hierarchical processing system, in which each layer performs a certain transform of the output of the previous layer. The transform performed is specified by the learning algorithm itself, and since we have specified the algorithm, the function of each layer may be interpreted within this context.

1.4 Data Compression as a Fundamental Principle

Recently, it is becoming clear that the discovery of interesting hidden units and generalization in a multilayer network is related to the ability of the network to compress data [Baldi and Hornik, 1989]. For instance, if a three-layer network has N inputs and N outputs, but $M < N$ hidden units, then the M hidden units must develop an encoding of the input which is appropriate for computing the output. This problem has been called the "encoder" problem [Rumelhart *et al.*, 1986a]. If a supervised algorithm is used to train the network to approximate the input at the output units (the identity mapping), then the hidden units must represent as much of the information in the input as possible, in order for the last layer of weights to be able to reconstruct the input. If backpropagation finds an acceptable solution, then it must involve a good coding at the hidden units.

Since there are fewer hidden units than input units, a "good" coding depends upon the hidden units' ability to exploit redundancies in the input data. However, redundancies in the data probably represent what we would call "features", since they could be groups of inputs which tend to occur together. If a single hidden unit represents a large multi-dimensional component of the data, then it is clear that that unit has captured some "significant" parameter of the input. This implies that data compression caused by network bottlenecks may force hidden units to represent significant features of the data.

To understand how this makes generalization possible, we start by assuming that the response of a desired function f of multi-dimensional input actually depends on a relatively small number of parameters of that input. If the dimensionality of a bottleneck layer is less than is needed to represent the input completely, the outputs of the bottleneck layer must somehow choose a representation which conveys the "useful" information for the desired function f to the higher layers.

Presumably, this "useful" information represents some nonlinear subspace of the input space R^N, and hopefully this subspace represents the parameters useful for computing f. The bottleneck has therefore provided a constraint on the gradient descent which has forced it to train the network in a particular configuration. This constraint means that components of the data which are not useful for computing f are eliminated at the bottleneck layer. The network therefore generalizes in the sense that components of the data which are unimportant for computation of f are ignored. We can say that the data outside the test set is "projected" onto the subspace spanned by the bottleneck outputs, so that if these outputs really represent a set of parameters upon which the desired output depends, we would expect to obtain the proper output even for data not in the training set.

The data compression principle can serve as a link between many different types of algorithms. Linsker has proposed that a major function of Hebbian algorithms is to maximize the information present in the output layer [Linsker, 1988b]. Backpropagation through a bottleneck may also attempt to maximize the useful "information" at the bottleneck layer. One might expect that both algorithms would lead to similar results, and in fact it can be proven mathematically that the optimal information content is achieved by both types of algorithms for the single-unit case [Baldi and Hornik, 1989]. For the multi-unit case, we will give an example later which shows that in one case the unsupervised algorithm to be presented in this thesis achieves equivalent results to backpropagation through a bottleneck. Other authors have taken advantage of this property of backpropagation to design unsupervised learning algorithms by specifically training a three-layer network through a bottleneck (referred to as self-supervised backpropagation) [Ackley *et al.*, 1985, Ballard, 1987, Cottrell *et al.*, 1987]. The success of these different algorithms may therefore be based upon the underlying principle of data compression, and this principle may be very closely related to the properties of feature detection and generalization.

1.5 Our Optimality Principle

The optimality principle which we will choose to design the algorithm to be presented here is based upon the idea of finding an efficient coding for the data. In order to do this, the algorithm must be able to find dependencies (correlations) within the samples in the training set. These dependencies can then be exploited to reduce the dimensionality of the data. By so doing, it is hoped that components which represent noise or are not useful for computing a desired function may be eliminated. In certain cases, this principle will correspond to maximization of information, an idea proposed by Linsker as a basic organizing principle for biological information processing [Linsker, 1988b]. Since a compact coding represents the significant features of the data, by separating out these features we hope that higher layers in a multi-layer network will be able to compute useful functions. It is also hoped that the network will generalize, if indeed the features which are discovered relate to significant parameters in the environment.

1.6 Goals of this Thesis

This thesis is divided into three major parts:

1. Linear Hebbian Networks and the Generalized Hebbian Algorithm
2. Extensions to Nonlinear Networks
3. Examples of Uses for Unsupervised Networks

The first part presents previous results on Hebbian algorithms and generalizes them to the multiple-output case. This generalization provides an important new algorithm which allows a network to discover an optimal *set* of outputs. Others have conjectured that this is possible (see, for example, [Baldi and Hornik, 1989]) and our algorithm verifies this conjecture.

The second part applies the results of Part 1 to the nonlinear case, and shows that essentially the same algorithm can be used, although it can no longer be proven to converge.

The third part shows examples of the use of unsupervised learning in several domains. A linear network is used to process a textured image and segment it into different textured regions. It is shown how presentation of random visual input can lead to the development of "receptive fields" qualitatively similar to those of certain primate visual cortical cells. A real digitized image is coded at a bit rate of one-third of a bit per pixel. And we demonstrate how a simple stereo algorithm can be learned which is capable of detecting depth edges in a random-dot stereo image.

Part I

Linear Networks

Chapter 2

Hebbian Algorithms

Hebbian Learning Algorithms are based upon the idea suggested by Hebb in 1949, that an increase of synaptic strength between an input and an output neuron may be related to the firing rates of the input and output [Hebb, 1949]. As a result, synaptic strengths will increase fastest between pairs of neurons whose responses are correlated, and the resulting increase in synaptic strength will lead to a further increase in the correlation. Increasing the correlation in this manner may lead to a useful pattern of synaptic strengths over a population of neurons.

2.1 Learning Rules

2.1.1 Terminology

Measurements on the environment are represented as a zero-mean Gaussian random vector $x =< x_1, \ldots, x_N >^T$ which has correlation matrix $Q = R_{xx} = E[xx^T]$. ("E[z]" is the expected value of the random variable z.) The output of a single linear layer is the Gaussian random vector $y =< y_1, \ldots, y_M >^T$ which is computed from x by $y = Cx$ where $C = [c_{ij}]$ is an $M \times N$ matrix of "synaptic weights", and

c_{ij} is the weight from input j to output i. The correlation matrix of y is

$$R_{yy} = E[Cx(Cx)^T] = E[Cxx^TC^T] = CE[xx^T]C^T = CQC^T$$

2.1.2 Hebb Algorithm

The basic Hebbian algorithm in its usual form modifies weights by adding a small increment proportional to the product between input and output. This can be written as

$$c_{ij}(t+1) = c_{ij}(t) + \gamma x_j(t) y_i(t)$$

or, in matrix form

$$C(t+1) = C(t) + \gamma y(t) x^T(t) \tag{2.1}$$

where γ is a coefficient which determines the learning rate. At a particular time t_1, we have

$$C(t_1) = C(0) + \gamma \sum_{t=0}^{t_1-1} y(t) x^T(t)$$

and as t_1 becomes large, the sum approaches $t_1 E[yx^T]$, so the expression approaches $C(0) + t_1\gamma E[yx^T]$.

If we have supervised learning, so that desired values are supplied for y, then we assume that y is a deterministic function of x, so $y = f(x)$. If f is linear, we can write $y = Lx$, and

$$E[yx^T] = E[Lxx^T] = LE[xx^T] = LQ$$

so C approaches $t_1\gamma LQ$. If x is white noise, then $Q = I$, and C becomes proportional to L. This means that the weights will "learn" to compute the desired linear function of the input. (If y can take on only the values $\{-1, 1\}$, then this reduces to the perceptron learning algorithm [Rosenblatt, 1962].)

In contrast, for unsupervised learning, y is a time-varying function of x (since it depends on the changing weight matrix), so equation 2.1 is difficult to solve. If we assume that γ is sufficiently small that C changes slowly relative to x, then we can average equation 2.1 to get

$$\begin{aligned} C(t+1) &= C(t) + \gamma C(t) E[xx^T] \\ &= C(t) + \gamma C(t) Q \end{aligned} \tag{2.2}$$

Under certain conditions on γ (see [Ljung, 1977]), the convergence of C can be approximately described by the differential equation

$$\dot{C} = CQ \tag{2.3}$$

which describes the evolution of the weights for large t. Equation 2.3 may be solved to give:

$$C(t) = C(0)e^{Qt}. \tag{2.4}$$

Since Q is a symmetric positive-definite matrix, we can write it as $Q = T\Lambda T^T$ where T is the matrix whose columns are the orthonormal eigenvectors $\{e_j\}$, and Λ is the diagonal matrix of eigenvalues λ_j. Expanding the matrix exponential as a power series, and realizing that $T^T T = I$, allows us to write $e^{Qt} = Te^{\Lambda t}T^T$. Therefore,

$$\begin{aligned} C(t) &= C(0)T \begin{bmatrix} e^{\lambda_1 t} & & & \\ & e^{\lambda_2 t} & & \\ & & \ddots & \\ & & & e^{\lambda_N t} \end{bmatrix} T^T. \\ &= C(0)\sum_{j=1}^{N} e^{\lambda_j t} e_j e_j^T \\ &= \sum_{j=1}^{N} e^{\lambda_j t}(C(0)e_j)e_j^T \end{aligned} \tag{2.5}$$

Now, divide and multiply by $e^{\lambda_1 t}$, where λ_1 is the largest eigenvalue, to give:

$$C(t) = e^{\lambda_1 t}\sum_{j=1}^{N} e^{(\lambda_j - \lambda_1)t}(C(0)e_j)e_j^T$$

$$= e^{\lambda_1 t}\left(C(0)e_1e_1^T + \sum_{j=2}^{N} e^{(\lambda_j-\lambda_1)t}(C(0)e_j)e_j^T\right)$$

Clearly, this does not converge, due to the exponential factor. However, if we divide by a normalization factor in order to maintain the C so that the rows will have finite norm, then this normalization must be proportional to $e^{\lambda_1 t}$.

Since λ_1 is the largest eigenvalue, the exponents within the sum are all negative, and those terms will approach zero as $t \to \infty$. Therefore, with the normalization, we have

$$C \to C(0)e_1e_1^T$$

and each row c_i of C

$$c_i \to (c_i(0)\cdot e_1)e_1^T \tag{2.6}$$

which is proportional to e_1. (If $c_i(0)\cdot e_1 = 0$, this will not be true, but for now assume $c_i(0)\cdot e_1 \neq 0$.) Therefore all the rows of C will converge to the principal eigenvector e_1.

This convergence procedure can also be described as minimization of an energy function. If we define

$$E = -\sum_{n=1}^{N} E[y_n^2] = -\text{trace}[R_{yy}] = -\text{trace}[CQC^T] \tag{2.7}$$

then

$$\begin{aligned}
\frac{\partial E}{\partial c_{ij}} &= -\sum_{n=1}^{N}\frac{\partial}{\partial c_{ij}}\left(\sum_{k=1}^{N}\sum_{l=1}^{N} c_{nk}Q_{kl}c_{nl}\right)\\
&= -\left(\sum_{l\neq j} Q_{jl}c_{il} + \sum_{k\neq j} Q_{kj}c_{ik} + 2Q_{jj}c_{ij}\right)\\
&= -\left(\sum_{l=1}^{N} Q_{jl}c_{il} + \sum_{k=1}^{N} Q_{kj}c_{ik}\right)\\
&= -2\sum_{k=1}^{N} Q_{kj}c_{ik} \qquad (2.8)\\
&= -2[CQ]_{ij}. \qquad (2.9)
\end{aligned}$$

Equation 2.8 follows since Q is symmetric. Equation 2.9 shows that $\partial E/\partial c_{ij}$ is proportional to the ij^{th} element of the matrix CQ. To perform gradient descent, we need to set

$$\dot{c}_{ij} \propto -\frac{\partial E}{\partial c_{ij}} = 2[CQ]_{ij}$$

which can be written in matrix form as

$$\dot{C} \propto CQ.$$

This is exactly equation 2.3 which gives the approximate change in C after a long time, so we see that the Hebbian learning rule is performing gradient descent on the energy function specified by equation 2.7. Again, we must impose some restriction on the norm of the rows of C for the energy function to have a minimum at all. If we do so, then the procedure always converges for any initial value of the matrix C, since the energy surface has no local minima.

If we normalize so that the rows have norm 1, then this matrix has the interesting property that it maximizes the variance of each output (over all other matrices whose rows have norm 1). The output correlation is CQC^T, and it can be seen that every element of this matrix will be equal to $e_1^T Q e_1 = \lambda_1 e_1^T e_1 = \lambda_1$. For any other vector c, we write it in terms of the orthonormal eigenvectors of Q as $c = \sum \alpha_j e_j$. Then the variance of the output of that vector is

$$c^T Q c = \sum \lambda_j \alpha_j^2 = \lambda_1 \sum (\lambda_j/\lambda_1)\alpha_j^2 < \lambda_1$$

since if c is normalized, $\sum \alpha_j^2 = 1$, and $(\lambda_j/\lambda_1) < 1$ by definition.

The significance of this observation is that the Hebbian learning rule is attempting to maximize the variance of each output independently. The energy function is defined to be the negative of the sum of the output variances, so minimization of E will lead to maximization of the variance. Since there is only one best solution, all the outputs will come to have the same synaptic weights from the inputs. Therefore each output will compute the same function of the input, and the learning algorithm will choose the function which maximizes the total output variance.

Linsker points out that finding the first eigenvector is equivalent to "Principal Component Analysis" (PCA) [Linsker, 1988b]. The eigenvector of Q with largest eigenvalue is the "principal component" of the data, so-defined because it has the largest variance. Standard results in linear estimation theory show that if the scalar output of this filter is used to estimate the input vector, then it will minimize the mean-squared error in the estimate of the input. We consider this idea in further detail in chapter 4 below.

2.1.3 Oja Algorithm

Oja has elegantly shown that a slight modification of the Hebbian learning rule is capable of keeping the norm at 1 automatically, without the necessity for external control of the normalization. [Oja, 1982]. His synaptic modification rule is

$$c_{ij}(t+1) = \frac{c_{ij}(t) + \gamma y_i(t) x_j(t)}{\left(\sum_{j=1}^{N} [c_{ij}(t) + \gamma y_i(t) x_j(t)]^2\right)^{1/2}} \tag{2.10}$$

which can be written in matrix form as

$$\begin{aligned} C(t+1) &= \Gamma^{-1/2}(C(t) + \gamma y(t) x^T(t)) \\ \Gamma &= \mathrm{diag}[(C(t) + \gamma y(t) x^T(t))(C(t) + \gamma y(t) x^T(t))^T] \end{aligned} \tag{2.11}$$

where diag[] is an operation which sets off-diagonal matrix entries to zero. Oja proved that for γ small, equation 2.10 can be approximated by

$$c_{ij}(t+1) = c_{ij}(t) + \gamma y_i(t)[x_j(t) - y_i(t) c_{ij}(t)] \tag{2.12}$$

which has matrix form

$$C(t+1) = C(t) + \gamma y(t) x^T(t) - \gamma \mathrm{diag}[y y^T] C(t). \tag{2.13}$$

Writing $y(t) = C(t)x(t)$ gives

$$C(t+1) = C(t) + \gamma C(t)x(t)x^T(t) - \gamma \mathrm{diag}[C(t)x(t)x^T(t)C^T(t)]C(t).$$

If we assume, as in section 2.1.2, that γ is sufficiently small that C changes slowly compared to x, then we can take the average of both sides to give

$$\begin{aligned} C(t+1) &= C(t) + \gamma C(t)E[xx^T] - \gamma \mathrm{diag}[C(t)E[xx^T]C^T(t)]C(t) \\ &= C(t) + \gamma C(t)Q - \gamma \mathrm{diag}[C(t)QC^T(t)]C(t) \qquad (2.14) \end{aligned}$$

which, under appropriate conditions, can be approximated by a differential equation

$$\dot{C} = CQ - \mathrm{diag}[CQC^T]C \qquad (2.15)$$

Oja proves that for positive semidefinite matrices Q whose largest eigenvalue has multiplicity one, this differential equation will always converge to a matrix each of whose rows is the normalized principal eigenvector.

2.2 Other Hebbian Algorithms

Several authors have used variations of the basic Hebbian algorithm to train unsupervised networks, often in the context of development of visual information processing [Grossberg, 1976, Oja, 1982, Bienenstock *et al.*, 1982, Kohonen, 1982, Barrow, 1987, Cottrell *et al.*, 1987, Linsker, 1988b] (among many others). The differences between algorithms can often be reduced to differences in normalization (most methods to keep weights from growing indefinitely will produce the principal component of the data), or methods of lateral inhibition (which attempt to force different outputs to compute different functions). For example, Oja maintains the sum of the squares of the input weights near 1 (as do [Kohonen, 1982, Barrow, 1987, Kohonen, 1988]). Other authors maintain the sum of the input weights to any output at a constant value [Von der Malsburg, 1973, Linsker, 1986a]. Of course, this requires that all weights in the network be positive. Dynamic decay of weights is another possibility [Grossberg, 1976, Bienenstock *et al.*, 1982].

The success of these algorithms often depends upon the discovery of the principal component of the data, since this component may represent some significant and intuitive feature. (In the field of statistics, the principal component is often considered to be a "generating variable" for the data which has special significance.) Linsker points out that another reason for the significance of the principal component is that it preserves the maximum amount of information about the input which any scalar value can. He therefore claims that the Hebbian algorithms attempt to perform "maximum information preservation" or "infomax" [Linsker, 1988b]. (Of course, his algorithm and the others mentioned only achieve this in the single-output case.)

Given so many algorithms for finding the principal component, one is led to ask whether it would be possible to learn the other components of the data, thereby performing a more complete factor analysis. It is a well-known fact in linear algebra and signal processing that if we must linearly approximate an N-valued vector using $M < N$ linear functions with minimal mean-squared error, then we should choose those M functions to be a linear combination of the M eigenvectors with largest eigenvalue. If $M = 1$ we find that the optimal representation for the N-vector is its statistical principal component, which is found by the Hebbian algorithms. For $M > 1$ we need an algorithm which will find more of the eigenvectors.

The major problem encountered by all workers who attempt to use Hebbian algorithms is that there must be some mechanism for forcing different outputs to converge to different functions. Usually, it is not required that these functions be orthogonal, but simply that they be "different" (linearly independent, usually). One method is to use a winner-take-all scheme in which only the output with largest activation has its weights modified for any given presentation [Kohonen, 1982, Barrow, 1987, Carpenter and Grossberg, 1988]. Another method is to use lateral inhibition, in which the training of any particular output is weakened proportionate to the strength of other outputs [Grossberg, 1976, Kohonen, 1988]. Different outputs can also be generated by allowing random variation in initial parameters to cause a learning algorithm to "choose" one of several equally likely possibilities [Linsker, 1986b, Kammen and Yuille, 1988].

Although it does not contribute to the representation of information, many authors attempt to have topologically "nearby" outputs respond to similar components of the input. This is usually accomplished by having some form of facilitation between nearby outputs [Von der Malsburg, 1973, Kohonen, 1982, Linsker, 1986c, Kohonen, 1988]. The biological motivation for this modification is that neighboring cells in area VI of monkey visual cortex appear to respond to similar bar orientations [Hubel and Wiesel, 1974]. However, there is no theoretical reason why such an organization would be useful, and authors do not justify it. Kohonen [1982] perhaps goes furthest in generating a network whose sole purpose is to have output vectors converge to a representation of the output unit's physical point in space. He accomplishes this using a winner-take-all algorithm which ensures that different outputs converge to different vectors, but the nearest neighbors of the winning unit are also allowed to train. Of course, the choice of nearest neighbor topology for training determines the topology of the output map.

Chapter 3

The Generalized Hebbian Learning Algorithm

3.1 Optimality Principle

We choose an optimality principle based on the ability to linearly reconstruct the inputs to the network from the outputs with minimum mean-squared error. If the network is linear and the input distribution is Gaussian with Gaussian noise added, then this maximizes the information at the outputs as suggested by Linsker's "Infomax" principle [Linsker, 1988b]. For general input distributions, however, this will not maximize the information. However, this choice of optimality does allow us to construct a convenient algorithm for training single-layer networks.

Formally, for the case $M < N$, we wish to estimate an N-vector x given the value of the M-vector $y = Cx$ for an MxN matrix C (as always, x is zero mean). We will choose the estimate $\hat{x}$ which minimizes the expected mean squared error (MSE) $E[||x - \hat{x}||]$. It can be shown that the best estimate is given by

$$\hat{x} = R_{xy} R_{yy}^{-1} y$$

where R_{xy} is the cross-correlation matrix of the input and output. Define the error vector $\epsilon = x - \hat{x}$ which has correlation matrix

$$R_{\epsilon\epsilon} = Q - R_{xy}R_{yy}^{-1}R_{xy}^T$$

where $Q = E[xx^T]$ is the NxN correlation matrix of x. We can write

$$E[||x - \hat{x}||] = \text{trace}[R_{\epsilon\epsilon}]$$

Since $y = Cx$, we have

$$\begin{aligned} R_{xy} &= E[xy^T] = E[xx^TC^T] = QC^T \\ R_{yy} &= CQC^T \\ \hat{x} &= QC^T(CQC^T)^{-1}y \\ R_{\epsilon\epsilon} &= Q - QC^T(CQC^T)^{-1}CQ \end{aligned}$$

Now, we would like to pick C which allows us to get the smallest possible error using this procedure. Since C has rank M, and the rows of C have norm 1, it can be shown that the error is minimized if $CQC^T = \Lambda_M$, where Λ_M is the $M \times M$ diagonal matrix of the largest M eigenvalues of Q. If we write $Q = T\Lambda T^T$ where T is the matrix of orthonormal eigenvectors, then the error is minimized when C is equal to the first M rows of T (assuming the rows of T are ordered by decreasing eigenvalue), or any linear combination of them.

The Oja algorithm described above is capable of finding only the principal eigenvector. If we wish to train a network with multiple outputs according to our optimality principle, then we must be able to find the other eigenvectors as well. In this chapter we describe a new learning algorithm which generalizes the Oja algorithm. We prove that it forces different outputs to converge to uncorrelated random variables, and thus finds the first M eigenvectors in eigenvalue order. It therefore computes the Karhunen-Loève Transform (KLT) of the input, or the singular value decomposition (SVD) of the input correlation matrix Q. If the number of outputs is the same as the number of inputs, then the output will be a "whitened" version of the input. This algorithm therefore supplies the missing generalization to multi-output networks. Since it is capable of finding the eigenvectors of a large dimensional distribution without prior computation of the correlation matrix, it is also a useful tool for the analysis of real-world stochastic data. (Other algorithms with this ability have been proposed outside the field of neural networks, and a summary can be found in [Oja, 1983].)

3.2 Derivation

In section 2.1.2 we saw that for any initial weight vector, a Hebbian training algorithm would cause the weights to converge to the principal eigenvector of the input correlation matrix, so long as the norm of the vector is maintained at 1. In section 2.1.3 we saw that the Oja algorithm maintains the norm at 1 automatically while doing this. A careful look at equation 2.5 will show that if any row of the initial weight vector $C(0)$ is orthogonal to the principal eigenvector, then that row will converge to the eigenvector with next highest eigenvalue. (To show this, divide and multiply by $e^{\lambda_2 t}$.) If a row is orthogonal to both the first and second eigenvectors, then it will converge to the third, and so on. Therefore, if during training we artificially maintain the different weight vectors (rows of C) orthogonal to each other, then when one converges to the principal eigenvector, the next will be forced to converge to the next eigenvector, and so on. We see that each row will converge to a different eigenvector of the input correlation matrix. The output correlation will thus approach $CQC^T = \Lambda$, and we will have obtained a whitening filter.

We thus need to find a modification rule to maintain the rows of C orthogonal to each other, in addition to maintaining the rows normalized and maximizing their variance. In order to obtain such a rule, we can perform a Gram-Schmidt orthogonalization on C after each training iteration. The Gram-Schmidt process can be written as:

1. Pick an arbitrary unit-length c_0
2. For each i, set $c_i \leftarrow c_i - \sum_{k<i}(c_i \cdot c_k)c_k$
3. For each i, normalize c_i to length 1.

In matrix form, step (2) can be written

$$C \leftarrow C - \text{lower}[CC^T]C \tag{3.1}$$

where lower[] is the operator which sets all matrix values on or above the diagonal to zero.

We now combine the three modification rules:

(1) find the principal eigenvector

(2) normalize to length 1

(3) orthogonalize (3.1)

If this is done explicitly at every iteration step, then it can be proven that the matrix C will converge to the matrix of eigenvectors [Oja, 1983]. The algorithm which we present here is based upon a linearized combination of these three steps. Other non-network methods for performing the same computation have been described in [Oja and Karhunen, 1980, Karhunen and Oja, 1982, Kuusela and Oja, 1982, Oja, 1983, Karhunen, 1984, Oja and Karhunen, 1985, Karhunen, 1985].

We now propose the "Generalized Hebbian Algorithm" which has a weight update rule given by:

$$c_{ij}(t+1) = c_{ij}(t) + \gamma\left(y_i(t)x_j(t) - y_i(t)\sum_{k\leq i} c_{kj}(t)y_k(t)\right) \tag{3.2}$$

and can be written in matrix form as

$$C(t+1) = C(t) + \gamma\left(y(t)x^T(t) - \text{LT}[y(t)y^T(t)]C(t)\right) \tag{3.3}$$

where $x(t)$ is the sample input at time t, $y(t) = C(t)x(t)$, γ is a time-varying learning rate constant, and LT[] is an operator which sets all entries above the diagonal of its matrix argument to zero. We will prove later that this equation causes C to converge to the matrix of the first few eigenvectors of the correlation matrix of x. This is done without the necessity for explicitly computing the NxN correlation matrix Q, which can be difficult if N is large.

If we assume that C changes slowly relative to x and y, then we can take the expected value of both sides to get a matrix iteration equation

$$C \leftarrow C + \gamma_1 (CQ - \mathrm{LT}[CQC^T]C) \tag{3.4}$$

which describes an iterative method for finding the first few eigenvectors of any matrix Q.

3.3 Proof of Convergence

We must show that the algorithm

$$C(t+1) = C(t) + \gamma(t)\left(y(t)x^T(t) - \mathrm{LT}[y(t)y^T(t)]C(t)\right) \tag{3.5}$$

converges to the matrix T whose rows are the first M eigenvectors of $Q = E[xx^T]$ in descending eigenvalue order. C is an $M \times N$ matrix, $y(t) = C(t)x(t)$, $\mathrm{LT}[\cdot]$ sets all entries of its matrix argument which are above the diagonal to zero, x is a white bounded vector stationary stochastic process with autocorrelation matrix Q, and $\{e_k\}$ is the set of eigenvectors of Q indexed by k in order of descending eigenvalue $\{\lambda_k\}$.

We will write

$$C(t+1) = C(t) + \gamma(t)h(C(t), x(t)) \tag{3.6}$$

where $h(C(t), x(t)) \doteq y(t)x^T(t) - \mathrm{LT}[y(t)y^T(t)]C(t)$. We now apply theorem 1 of [Ljung, 1977] which states (in our notation):

If

(1) $\gamma(t)$ is a sequence of positive real numbers such that $\gamma(t) \to 0$, $\sum_t \gamma(t)^p < \infty$

for some p, and $\sum_t \gamma(t) = \infty$;

(2) $x(t)$ is bounded with probability 1;

(3) $h(C, x)$ is continuously differentiable in C and x and its derivative is bounded in time;

(4) $\bar{h}(C) \doteq \lim_{t\to\infty} E_x[h(C, x)]$ exists for each C;

(5) S is a locally asymptotically stable (Lyapunov sense) set for the differential equation

$$\dot{C} = \bar{h}(C)$$

with domain of attraction $D\{S\}$, and

(6) $C(t)$ enters some compact subset $A \subset D\{S\}$ infinitely often w.p.1;

Then with probability one,

$$\lim_{t\to\infty} C(t) \in S.$$

where $C(t)$ is given by equation 3.6. (A similar result is found in theorem 2.4.1 of Kushner and Clark [1978] .)

To satisfy the requirements of the theorem, we let $\gamma(t) = 1/t$ and, $\bar{h}(C) = \lim_{t\to\infty} E[h(C, x)] = CQ - \mathrm{LT}[CQC^T]C$. We now show that the domain of attraction $D\{S\}$ for the solution for which the rows of C are the first M orthonormal eigenvectors includes all matrices with bounded entries. We will "hard-limit" the entries of C so that their magnitudes remain below a certain threshold a, and thus within a compact region of R^{NM}. We will thereby show that C converges to the matrix of eigenvectors T for any choice of initial matrix $C(0)$.

We seek stable points of the differential equation

$$\dot{C} = CQ - LT[CQC^T]C \tag{3.7}$$

Fixed points exist for all C whose rows are permutations of any set of eigenvectors of Q. We will show that the domain of attraction of the solution given by $C = T$ whose rows are the first M eigenvectors in descending eigenvalue order, includes all matrices C with bounded entry magnitudes.

Oja showed that the first row of C will converge to the principal eigenvector with probability 1. (This is not the only fixed point, but it is the only asymptotically stable one.) We will use induction to show that if the first $i-1$ rows converge to the first $i-1$ eigenvectors, then the i^{th} row will converge to the i^{th} eigenvector. We assume that Q has N distinct strictly positive eigenvalues with corresponding orthonormal eigenvectors. (The case of repeated or zero eigenvalues is a straightforward generalization.)

The first row of C is described by the differential equation

$$\dot{c}_1^T = c_1^T Q - (c_1^T Q c_1) c_1^T$$

which is equivalent to equation 7 of [Oja, 1982]. Oja showed that this equation forces c_1 to converge with probability 1 to $\pm e_1$, the normalized principal eigenvector of Q.

At any time t, for $k < i$ write

$$c_k(t) = e_k + \epsilon_k(t) f_k(t)$$

where e_k is the k^{th} eigenvector (or its negative), f_k is a time-varying unit-length vector, and $\epsilon_k(t)$ is a scalar. We assume for the induction step that for $k < i$, $\epsilon_k(t) \to 0$ as $t \to \infty$. We must show that if this is true then $c_i(t) \to e_i$ as $t \to \infty$.

Each row of equation 3.7 can be written

$$\dot{c}_i = Qc_i - \sum_{k \leq i} (c_i^T Q c_k) c_k$$

Substituting $c_k = e_k + \epsilon_k f_k$ gives

$$\dot{c}_i = Qc_i - (c_i^T Q c_i) c_i - \sum_{k<i} (c_i^T Q e_k) e_k - O(\epsilon) + O(\epsilon^2) \tag{3.8}$$

where $Qe_k = \lambda_k e_k$, and ϵ indicates a term converging to zero at least as fast as the slowest row for $k < i$. Expanding c_i in terms of the entire orthonormal set of eigenvectors gives

$$c_i = \sum_{k=0}^{N} \alpha_k e_k$$

where $\alpha_k = c_i^T e_k$. Inserting this expression gives the following equalities:

$$\begin{aligned}
(c_i^T Q c_i) c_i &= c_i \sum_{l=0}^{N} \lambda_l \alpha_l^2 \\
&= \left(\sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \sum_{k=0}^{N} \alpha_k e_k \\
Q c_i &= \sum_{k=0}^{N} \lambda_k \alpha_k e_k \\
Q c_i - \sum_{k<i} \lambda_k (c_i^T e_k) e_k &= \sum_{k=0}^{N} \lambda_k \alpha_k e_k - \sum_{k<i} \lambda_k \alpha_k e_k \\
&= \sum_{k \geq i} \lambda_k \alpha_k e_k
\end{aligned}$$

We now assume t is large so we can ignore terms of order ϵ, and we write

$$\begin{aligned}
\dot{c}_i &= \sum_{k=0}^{N} \dot{\alpha}_k e_k \\
&= \sum_{k \geq i} \lambda_k \alpha_k e_k - \left(\sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \sum_{k=0}^{N} \alpha_k e_k \\
&= \sum_{k<i} \left(\sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \alpha_k e_k + \sum_{k \geq i} \left(\lambda_k - \sum_{l=0}^{N} \lambda_l \alpha_l^2 \right) \alpha_k e_k
\end{aligned}$$

If we multiply each side by e_k^T for each k, the orthonormality of the $\{e_k\}$ implies

$$\dot{\alpha}_k = \begin{cases} -\alpha_k \sum_{l=0}^{N} \lambda_l \alpha_l^2 & \text{if } k < i \\ \alpha_k (\lambda_k - \sum_{l=0}^{N} \lambda_l \alpha_l^2) & \text{if } k \geq i \end{cases} \tag{3.9}$$

We can use equation 3.9 to study the recursion relation on the α_k's. We examine three cases: $k < i$, $k > i$, and $k = i$.

Since Q is positive definite, all the eigenvalues λ_k are positive, so the term $\gamma \sum_{k=0}^{N} \lambda_k \alpha_k^2$ is always greater than zero. Therefore, for $k < i$ we have

$$\dot{\alpha}_k = -\eta \alpha_k$$

where η is strictly positive. This expression will converge to zero for any initial value of α_k.

When $k > i$, define $\theta_k = \alpha_k / \alpha_i$. (Assume that $\alpha_i \neq 0$, which is true with probability one for randomly chosen initial weights $C(0)$.) We have

$$\begin{aligned} \dot{\theta}_k &= (1/\alpha_i)(\dot{\alpha}_k - \theta_k \dot{\alpha}_i) \\ &= (1/\alpha_i)\left[\alpha_k(\lambda_k - \sum_{l=0}^{N} \lambda_l \alpha_l^2) - \theta_k \alpha_i(\lambda_i - \sum_{l=0}^{N} \lambda_l \alpha_l^2)\right] \\ &= \theta_k(\lambda_k - \lambda_i) \end{aligned}$$

Since the eigenvalues are numbered in decreasing order, λ_i is the largest eigenvalue in $\{\lambda_i, \cdots, \lambda_N\}$, $\lambda_i > \lambda_k$ for all $k > i$, and we see that $\theta_k \to 0$ for $k > i$.

Next, we examine the behavior of α_i (case $k = i$). This is described by

$$\begin{aligned} \dot{\alpha}_i &= \alpha_i(\lambda_i - \sum_{l=0}^{N} \lambda_l \alpha_l^2) \\ &= \alpha_i(\lambda_i - \lambda_i \alpha_i^2 - \sum_{l \neq i} \lambda_l \alpha_l^2) \end{aligned}$$

But we know that $\alpha_k \to 0$ for $k < i$. We therefore drop terms in α_k for $k < i$, which gives

$$\begin{aligned} \dot{\alpha}_i &= \alpha_i(\lambda_i - \lambda_i \alpha_i^2 - \sum_{l > i} \lambda_l \alpha_l^2) \\ &= \alpha_i(\lambda_i - \lambda_i \alpha_i^2 - \alpha_i^2 \sum_{l > i} \lambda_l \theta_l^2) \end{aligned}$$

But $\theta_k \to 0$ for $k > i$, so the last term above approaches zero, and we therefore drop it as well, giving

$$\dot{\alpha}_i = \lambda_i(\alpha_i - \alpha_i^3)$$

To show that this converges, note that

$$V = (\alpha_i^2 - 1)^2$$

is a Lyapunov function, since

$$\dot{V} = 4\lambda_i(\alpha_i^3 - \alpha_i)(\alpha_i - \alpha_i^3) < 0$$

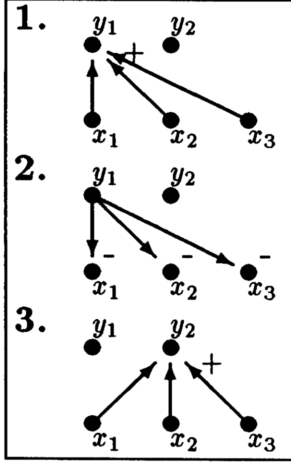
and V has a minimum at $\alpha_i = \pm 1$. Therefore, $\alpha_i \to \pm 1$ as $t \to \infty$. Since $\theta_k \to 0$ for $k > i$, we also have $\alpha_k \to 0$ for $k > i$. Thus for large t, the only significant α is α_i, so c_i will converge to $\pm e_i$, as desired. This completes the induction step and proves that the differential equation 3.7 converges to the matrix T of eigenvectors of Q. The domain of attraction of T includes all matrices with bounded weights.

Choose A to be the compact region in R^{NM} given by the set of matrices C with entry magnitudes $|c_{ij}| \leq a$ for a a fixed positive parameter. Clearly, $A \subset D\{S\}$. Choose a sufficiently large that $T \in A$. Choose the elements of $C(0)$ so that $C(0) \in A$. If $\|C(t)\| \doteq \max_z \|C(t)z\|/\|z\|$ remains bounded, then there exists a value of a such that $C(t) \in A$ infinitely often w.p.1. (To ensure that $\|C(t)\|$ is in fact bounded, we can "hard limit" its entries to be less than or equal to a. This does not affect convergence of the algorithm (w.p.1) so long as a is sufficiently large. Convergence of such constrained systems is discussed in chapter 5 of [Kushner and Clark, 1978].) Alternatively, it can be shown that for $\|C(t)\|$ sufficiently large, $\|C(t+1)\| < \|C(t)\|$ for any input $x(t)$, and $C(t)$ therefore remains within a (large) bounded region of R^{NM}.

We have now satisified all the conditions of Ljung's [1977] theorem. This proves that, under the assumptions given above, (3.5) will cause C to converge with probability one to the matrix T whose rows are the first M eigenvectors in descending eigenvalue order.

Note that theorem 2.3.1 of Kushner and Clark [1978] implies that the averaged form of the algorithm, given by

$$C(t+1) = C(t) + C(t)Q - \mathrm{LT}[C(t)QC^T(t)]C(t)$$



1. Max Var[y_1] $\Rightarrow$
First row converges to Principal Eigenvector.

2. $x - c_1 y_1$ removes this component from input.

3. Max Var[y_2] $\Rightarrow$
Second row converges to principal eigenvector of modified input, which is the *second* eigenvector.

Figure 3.1: Schematic illustration of the operation of the Generalized Hebbian Algorithm.

will also cause C to converge with probability 1 to the matrix of eigenvectors.

3.4 Network Interpretation

To help understand the operation of the algorithm, we rewrite (3.2) as

$$\Delta c_{ij}(t) = \gamma(t) y_i(t) \left(x_j(t) - \sum_{k<i} c_{kj}(t) y_k(t) \right) - \gamma(t) y_i^2(t) c_{ij}(t) \tag{3.10}$$

In this form, we see that the algorithm is equivalent to performing the Oja [1982] learning rule (2.10) using a modified version of the input given by

$$x'_j(t) = x_j(t) - \sum_{k<i} c_{kj}(t) y_k(t) \tag{3.11}$$

or, in matrix form

$$x'(t) = x(t) - \sum_{k<i} c_k(t) y_k(t)$$

where c_k is the k^{th} row of the matrix C. The modified input which trains output i is formed by subtracting the components $c_k(t)$ which contributed to the previous

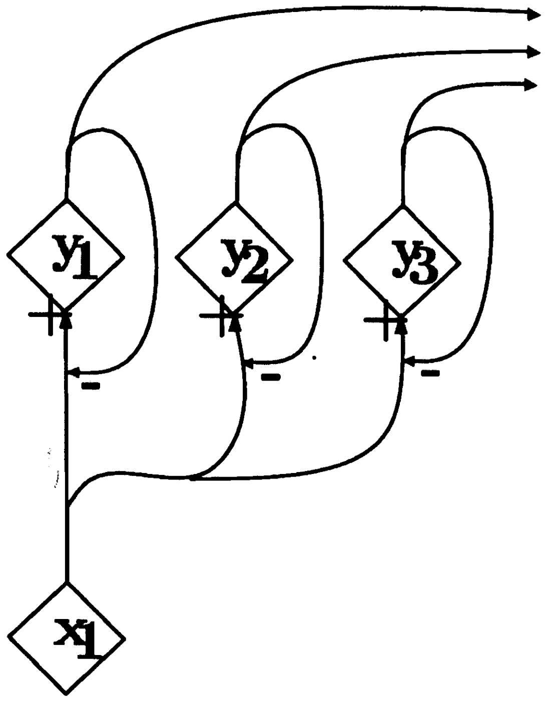


Figure 3.2: Network implementation of the Generalized Hebbian Algorithm.

outputs $y_k(t)$. If the first $i-1$ outputs respond to the first $i-1$ eigenvectors, then the i^{th} output "sees" an input from which those eigenvectors have been removed. The principal component of the modified input is now the i^{th} eigenvector of the original input. When the Oja algorithm is applied to the modified input, it causes the i^{th} output to learn the i^{th} eigenvector, as desired. (This method of finding successive eigenvectors is similar to a technique known as "Hotellings's Deflation" [Kreyszig, 1988].) This technique is known to be numerically poor, as errors tend to accumulate with the computation of successive eigenvectors. Nevertheless, it is useful for finding the first few eigenvectors of large-dimensional data, and it is often used in Statistics.

Equation 3.10 shows how we can implement the algorithm using only local operations. This ability is important for training neural networks using parallel hardware. We can compute the outputs in order for each training presentation, and subtract the components $c_k(t)y_k(t)$ progressively from the input as we go. This corresponds to "using up" some of the input "energy" as we train each of the outputs. (See figure 3.1.) A local synapse-like structure which can perform this operation is shown in figure 3.2. A given input x_1 is connected to several outputs. As each output y_i is activated, it inhibits the input unit x_1 by an amount proportional to the output activation. (The weights for forward propagation and reciprocal inhibition must be maintained equal and be modified together.) If the outputs learn in sequence, then each subsequent output sees an attenuated input. This leads to training according to (3.10) which is performed using only local "synaptic" operations. The fact that such a local implementation exists for this algorithm distinguishes it from other algorithms for computing the Karhunen-Loève transform and contributes to its importance for training neural networks.

3.5 Self-Supervised Back-Propagation

Several authors have experimented with the technique of Self-supervised Back-propagation (SSBP), also known as the "encoder" problem [Ballard, 1987, Cottrell *et al.*, 1987]. This algorithm seems to have optimal data coding properties similar to those of the algorithm presented here.

In linear SSBP, a three-layer network is trained to perform the identity mapping, yet the number of hidden units in the middle layer is set to be fewer than the number of inputs. The hidden units must therefore discover an efficient encoding of the input data. Since efficient coding is also the goal for the generalized Hebbian algorithm, we would expect both algorithms to produce similar results. Bourlard and Kamp [1987] have shown that a set of vectors which is a linear combination of the eigenvectors gives the optimal set of hidden units of such a network (according to our optimality principle). Baldi and Hornik [1988] proved that backpropagation will always converge to such a solution. There are many possible sets of weights with this property however, and SSBP will choose one which is based on the initial random choice of weights. In general, the outputs will not be uncorrelated, and the net will not be a "whitening filter". The Generalized Hebbian Algorithm (GHA) finds the unique set of weights which is both optimal for encoding and gives uncorrelated outputs. So in the linear case, SSBP and the GHA converge to almost equivalent but not identical solutions. We will now give a heuristic argument to show that the equations describing the two learning rules are similar.

Define a three-layer linear network of N input units x, M hidden units y, and N output units $\hat{x}$ which we want to approximate x. The weight matrices are W_1 and W_2, so that $y = W_1 x$ and $\hat{x} = W_2 y$. We train the network using backpropagation and error function $E[(x - \hat{x})^T(x - \hat{x})]$. Then, using the notation of Rumelhart *et. al.* (1986a) , we adapt the weights at each layer using $\Delta W_1 = \nu_1 \delta_1 x^T$ and $\Delta W_2 = \nu_2 \delta_2 y^T$ where $\delta_2 = x - \hat{x}$ and $\delta_1 = W_2^T \delta_2$. We then have:

$$\begin{aligned}
\Delta W_2 &= \nu_2(x - \hat{x})y^T \\
&= \nu_2(x - W_2 y)y^T \\
&= \nu_2(xy^T - W_2 yy^T) \\
\Delta W_2^T &= \nu_2(yx^T - yy^T W_2^T) \qquad (3.12)
\end{aligned}$$

This is similar to (3.3), except that we do not force the matrix yy^T to be lower triangular, and W_2^T is not the matrix used to generate y from x. The following discussion will show that W_2^T and W_1 are related so that it is reasonable to assume (3.12) is almost equivalent to (3.3).

Assume that $\nu_2 \ll \nu_1$, so that W_2 converges much slower than W_1. We then have:

$$\begin{aligned}
\Delta W_1 &= \nu_1 W_2^T \delta_2 x^T \\
&= \nu_1 W_2^T (x - \hat{x}) x^T
\end{aligned}$$

$$\begin{aligned}
&= \nu_1 W_2^T(x - W_2W_1x)x^T \\
&= \nu_1(W_2^Tx - W_2^TW_2W_1x)x^T \\
&= \nu_1(W_2^T - W_2^TW_2W_1)xx^T \\
E[\Delta W_1] &= \nu_1(W_2^T - W_2^TW_2W_1)Q
\end{aligned}$$

Since Q is positive definite, it can be shown that this equation will cause $W_2^TW_2W_1$ to approach W_2^T (for ν_1 decreasing to 0 as $1/t$). If we substitute $W_2^T \approx W_2^TW_2W_1$ into (3.12), we see that this is similar to (3.3) except for the lack of the LT[] operator and the inclusion of the positive definite scaling matrix $W_2^TW_2$.

Although the above discussion is not a rigorous proof, it gives us some insight into the reason for the close relationship between GHA and SSBP. The two algorithms are related both by the results they achieve and by their mechanisms of action. This means that our understanding of the convergence properties of GHA can be used to analyze certain backpropagation networks as well.

The major difference in function between GHA and SSBP is that GHA produces the first M eigenvectors themselves in eigenvalue order, while SSBP produces a linear combination of the first M eigenvectors. This difference can be important. Several authors have noted that SSBP tends to produce hidden units which have approximately equal variance [Cottrell *et al.*, 1987, Baldi and Hornik, 1989]. The variances do not descend by eigenvalue as they do for the KLT. The solution found is not unique, and although it spans the first few eigenvectors, the actual matrix which is learned cannot be predicted. In addition, the hidden units are not uncorrelated.

These factors seriously reduce the usefulness of SSBP-trained hidden units for data coding applications. Because the hidden units all have approximately equal variance, bits must be allocated evenly among them, and noise cannot be eliminated by removing the units with lowest variance. If the network is designed with too many hidden units (in the sense of Brailovsky [1983a,b,1985]) then the additional error introduced is spread evenly throughout the units and cannot be easily detected or removed by looking at the signal to noise ratio of the individual units. Cottrell et. al. [1987] point out that if channel errors affect certain units more than others, then it may be an advantage to distribute the information evenly so that high-variance channels are not corrupted excessively. If multiplicative noise is present in different amounts on different channels, then indeed this will be true. For additive noise, however, the KLT allows much easier reconstruction of the original "clean" signal, since the signal-to-noise ratio is maximized.

Ballard [1987] proposes using SSBP to train each layer of a multi-layer nonlinear network. The layers are interconnected such that the hidden units of one layer connect to both the inputs and outputs of the layer above, and this allows errors to be propagated back down through the network. He has not yet performed an extensive theoretical analysis of either SSBP or his coupling scheme, although the success of his method clearly depends upon the ability of SSBP to find good codings.

[Cottrell *et al.*, 1987] used SSBP to train one layer of a linear network on gray-scale image data. Since the hidden layer determines a good coding of the input (in the linear case, it is the optimal coding [Bourlard and Kamp, 1988, Baldi and Hornik, 1989]), they used this coding to perform image compression at rates up to .625 bits per pixel. The hidden units can be considered to represent features of the input data, and these features are extracted, quantized, and then recombined to produce estimates of the original image. The authors mention that the network is performing the "Principal Components Transform" (SVD) of the input data, and in so doing discovers the same solution as is often used in standard image coding applications.

Comparison of equation 3.12 (page 49) with equation 3 of [Oja, 1982] makes it clear that for the single hidden unit case, the linear SSBP algorithm *is exactly equivalent* to the Oja algorithm. For multiple hidden units, [Baldi and Hornik, 1989] predict that some Hebbian algorithm probably exists which achieves the same results as SSBP. The Generalized Hebbian Algorithm is, in fact, the algorithm they predict. It determines a unique solution which is (with probability one) independent of the initial choice of weights.

3.6 Rate of Convergence

Assume that the first output converges to the first eigenvector in a time t_1. Note that t_1 will be approximately independent of the number of inputs, since the weight from each input adapts according to

$$\Delta c_j = \gamma(yx_j - c_j y^2)$$

and the only effect of increasing the number of inputs is through y. The dependence on the number of outputs should be approximately linear (Mt_1), since the second output cannot mature properly until after the first one has. It may be worse, however, since the effect of subtracting eigenvectors from the input is to decrease the variance of the input and this will slow learning for the outputs which mature later. This effect is compensated for by the fact that the second output tends to "track" the first while it is still evolving. Empirically, it seems that the two effects combine to produce training time which is approximately linear in the number of outputs.

3.7 Noise

If uncorrelated noise with energy σ is added to the input vector, we can write the new input correlation as $Q + \sigma I$ where σI is the diagonal noise correlation matrix. The eigenvectors of $Q + \sigma I$ are the same as the eigenvectors of Q, since if $CQC^T = \Lambda$, then $C(Q+\sigma I)C^T = \Lambda + \sigma CC^T = \Lambda + \sigma I$ which is diagonal. We thus have $(Q + \sigma I)C^T = C^T(\Lambda + \sigma I)$ so the new eigenvalues are given by $\lambda_i + \sigma$, while the eigenvectors are the same and their ordering by eigenvalue remains the same.

The Generalized Hebbian algorithm will thus cause the network to achieve exactly the same weights, even though the outputs will be linearly affected by the noise. The signal to noise ratio for output i is λ_i/σ. The higher the correlation between inputs, the larger the initial eigenvalues will be, and thus the effect of adding noise of power σ to the outputs will be relatively less. In this sense, the network will attempt to reject the noise by maximizing the signal to noise ratio (since the noise variance at the outputs is constant, maximizing the output variance corresponds to maximizing the signal variance).

An important point which is often overlooked in this situation is that, contrary to the noise-free situation, the addition of more outputs in the presence of noise can actually *decrease* the accuracy of computations based on the ouputs [Brailovsky, 1985, Brailovsky, 1987]. In order to avoid this problem and select only significant outputs, we must either know the noise variance in advance, or we must rely on subsequent processing to eliminate unreliable outputs (if, for instance, they did not correlate with any useful information). Linsker points out that the eigenvectors do not generate the maximal information which can be obtained when noise is present, and that in fact some redundancy in the output data can help to eliminate noise and improve the information content [Linsker, 1988b, Linsker, 1988a].

Chapter 4

The Singular Value Decomposition

After a network has been trained according to the procedure described in chapter 3, the outputs of the network will be uncorrelated with each other. This is why we describe the network as computing a "whitening" filter - it has found a linear filter which transforms the environmental colored noise into its principal uncorrelated components. We can also describe the network as computing the Singular Value Decomposition (SVD) of the input correlation. This decomposition is important for whitening, data compression, and the discovery of significant features. It is a well-known and thoroughly analyzed technique used in the fields of signal processing and statistics (where it is called Factor Analysis or Principal Components Analysis (PCA)). In many texts, the term "Karhunen-Loeve transform" (KLT) is used to refer to the output of the whitening filter specified by the SVD.

4.1 Matched Filters

The singular value decomposition has an interesting interpretation when the input is generated by a fixed vector in random noise. It turns out that the first principal component will be a matched filter for the signal. To see this, let the noise vector be n with correlation $\sigma^2 I$. Then we have

$$\begin{aligned} x(t) &= g + n(t) \\ Q &= gg^T + E[nn^T] = gg^T + \sigma^2 I \\ Qg &= gg^T g + \sigma^2 g = g(\|g\| + \sigma^2) \end{aligned}$$

so clearly g is an eigenvector of Q with eigenvalue $\|g\| + \sigma^2$. If e is any other eigenvector orthogonal to g, then $g^T e = 0$ and $Qe = gg^T e + \sigma^2 e = \sigma^2 e$. So all other eigenvectors have eigenvalue $\sigma^2 < \|g\| + \sigma^2$. Therefore, g is the principal component of the data, and the Hebbian learning algorithm will find a matched filter for this vector.

In the slightly more complicated case of multiple random signals summed together, each eigenvector will converge to a matched filter for one of the input patterns.

Formally, let the input be

$$x = \sum_{i=1}^{M} g_i \nu_i$$

where ν is a vector of independent zero-mean scalar Gaussian random variables with correlation R_ν (which is diagonal), and $\{g_i\}$ is a set of fixed orthonormal vectors. Then, since the ν_i are independent, we have

$$\begin{aligned} Q &= \sum_{i=1}^{M} g_i g_i^T E[\nu_i^2] \\ &= \sum_{i=1}^{M} g_i g_i^T \sigma_i^2 \qquad (4.1) \\ & \qquad (4.2) \end{aligned}$$

where σ_i is the variance of ν_i. Multiplying through by g_n gives

$$Qg_n = \sum_{i=1}^{M} g_i g_i^T g_n \sigma_i^2$$

and since the g_i are assumed to be orthonormal, we have

$$Qg_n = g_n \|g_n\|^2 \sigma_n^2 = g_n \sigma_n^2$$

This shows that g_n is an eigenvector of Q with eigenvalue σ_n^2. Therefore, the extended Hebbian algorithm will converge to a matrix of M different "matched filters" for the different fixed patterns, and the vectors will be ordered by their average power $\|g_n\|^2 \sigma_n^2$. For this very simple case, the algorithm allows us to separate out the different components of the input signal, without having to know what these components are in advance.

To see this another way, write $G =< g_1, \ldots, g_M >$ whose columns are the orthonormal vectors g_n. Then $Q = GR_\nu G^T$, whose eigenvector matrix is simply given by G^T. Therefore, the generalized Hebbian algorithm gives $C \rightarrow G^T$, so the output of the trained network is $y = CG\nu = G^T G\nu = \nu$, and we have recovered the original generating vector ν.

4.2 Linear Estimation

Often, when interpreting the real world, a system will need to solve the inverse problem of estimating some parameter of the world from a complex and inaccurate series of measurements. The parameter could be something as simple as the average number of true bits in a data stream, or something as complicated as the probability that a cat is present in a visual image. We model the measurements as the operation of a multivalued (nonlinear) probabilistic function upon the true value of the desired parameter. The fact that the function is probabilistic means that it will not always give the same output for the same values of the input. The linear estimation problem is to find a linear filter which takes the measurement as input and produces an estimate of the parameter which minimizes some cost function.

4.2.1 Finding the Optimal Estimator

Standard results in linear algebra give the minimal mean-squared error estimator by:

$$\hat{\eta} = R_{\eta x} Q^{-1} x \tag{4.3}$$

Stone [Stone, 1986] has shown that a single level of the familiar backpropagation rule [Rumelhart *et al.*, 1986a] will converge to precisely this filter. This is equivalent to the Widrow-Hoff or LMS algorithm which has been extensively analyzed [Widrow and Hoff, 1960, Widrow *et al.*, 1976, Widrow and Walach, 1984]. The learning in this case is supervised, since we must know the true value of η during training so that the system can correct its errors. The learning rule is given by

$$p_{ij}(t+1) = p_{ij}(t) + \gamma[\eta_i(t) - \hat{\eta}_i(t)]x_j(t)$$

where $P = [p_{ij}]$ is the weight matrix, η is the actual parameter value, and $\hat{\eta} = Px$. This equation has matrix form

$$P(t+1) = P(t) + \gamma[\eta(t) - P(t)x(t)]x^T(t) \tag{4.4}$$

referred to as the "Delta rule" or the "Widrow-Hoff rule". If γ is small, this can be averaged to give

$$P(t+1) = P(t) + \gamma[R_{\eta x} - P(t)Q]$$

whose convergence can be approximately described by the differential equation [Ljung, 1977]

$$\dot{P} = -PQ + R_{\eta x}$$

This equation can be shown to converge to $R_{\eta x}Q^{-1}x$, since

$$V = \text{trace}[(P - R_{\eta x}Q^{-1})(P - R_{\eta x}Q^{-1})^T]$$

is a Lyapunov function if Q is positive definite and invertible.

Note that the Delta rule is an extension to continuous variables of the Perceptron learning rule [Rosenblatt, 1962]. If we restrict η' to the values 0 and 1, and let Π be an operator such that $\Pi x = \text{thresh}[Px]$ (thresh is a threshold function operating on each element of Px so that if an element is negative it is set to 0, and if an element is positive it is set to 1), then the equation

$$P(t+1) = P(t) + \gamma[\eta'(t) - \Pi(t)x(t)]x^T(t)$$

is exactly the Perceptron learning rule.

Note that the Delta rule, since it is a supervised algorithm, is able to reject "noise" in its input. In fact, it finds the linear optimal estimator (minimal mean-squared error) of the intended data, independent of whether or not noise has been added.

4.2.2 Using Whitened Input

If we use the generalized Hebbian algorithm to learn a whitening filter C, then we can apply the Delta learning rule to the output $y = Cx$. We then obtain a filter

$$P = R_{\eta y}\Lambda_y^{-1}$$

where Λ_y is the (diagonal) autocorrelation matrix of the whitened output y. Substituting $\Lambda_y = CQC^T$ gives

$$P = R_{\eta y}CQ^{-1}C^T = R_{\eta x}Q^{-1}C^T.$$

The optimal estimate is thus

$$\hat{\eta} = Py = PCx = R_{\eta x}Q^{-1}$$

which is exactly the optimal estimate of η based on measurement of x directly, if $C^TC = I$. However, C is an MxN matrix, so C^TC has rank $M < N$. This means that we have lost some of the data. The SVD has preserved as much as possible, but we do not know whether the network as a whole is performing as well as it could. An advantage to using a two-layer net in this way is that training the top layer using LMS may be faster, since $M < N$ so there are fewer outputs, and these outputs are uncorrelated.

4.2.3 Choosing the number of Outputs

The estimation error for approximating the original input is related to the sum of the lowest $N - M$ eigenvalues whose eigenvectors are not represented at the hidden layer. Therefore, we can choose the width of the hidden layer by computing the variance of the outputs and discarding those whose variance falls below a certain threshold. In fact, if there is noise present at the inputs, then estimation is *improved* by discarding outputs whose variance is less than the noise variance [Brailovsky, 1985]. Noise effectively sets an absolute limit on the approximation ability of the network. The fact that the outputs are ordered by decreasing eigenvalue (variance) is very convenient when we are trying to compute just enough outputs for a particular task, for we can allow the first few outputs to converge, and stop training new ones as soon as the variance falls below a chosen threshold. In this way, we can build up the network to achieve any desired level of accuracy, without having to train the full-sized network initially. Orthogonality of the outputs means that training a new output will not require retraining of the previous ones. Note that this is not true for Self-supervised Backpropagation and other unsupervised algorithms.

4.2.4 Other Fields

The use of whitening filters is common in several fields outside neural networks. For signal processing, the Karhunen-Loeve expansion is a standard data compression technique. It is commonly used in image processing to compress the data in blocks of an input image [Lim, 1988]. Each square block is coded according to a small number of linear basis vectors, and the coefficients are quantized for transmission or storage. Since the KLT is usually assumed to order the eigenvectors by decreasing eigenvalue, optimal quantization and transmission can be achieved by assigning bits to each coefficient in proportion to the log of its eigenvalue (variance). In theory, the optimal coding is achieved if the KLT is computed independently for each image that is to be transmitted. In practice, it has been found that most images contain very similar eigenvectors, and that these do not need to be recomputed each time. In particular, it can be shown that the Discrete Cosine Transform (DCT) of the image is equal to the KLT under very reasonable assumptions about local correlations [Ahmed *et al.*, 1974].

In other signal processing applications, the KLT is used for either data coding or noise reduction. In the field of Statistics, the KLT is equivalent to the technique of Principal Component Analysis (PCA) [Watanabe, 1965]. PCA can be used either to rotate a set of data coordinates into an orthogonal system, or it can be used to select the optimal set of nonlinear basis functions to predict a desired value (Factor Analysis). The use of PCA assumes that the world can be described by a set of independent generating parameters, and that these parameters will be useful for prediction if they can be discovered. Components of the data with high variance represent "real" statistics of the data, and can be related to values which we might want to predict.

Part II

Nonlinear Networks

Chapter 5

Optimality in the Nonlinear Case

Most types of layered neural network involve some form of nonlinear processing between each level. The nonlinear processing allows the network to compute nonlinear functions, although there has not yet been a clear exposition of precisely which functions can and cannot be represented as multilayer networks. In addition, there is no foolproof method for training nonlinear networks to accomplish any particular task. The familiar backpropagation algorithm [Rumelhart *et al.*, 1986a] is not guaranteed to converge for nonlinear networks, and no other general algorithm has been proposed. Learning algorithms abound for special cases, but very few of these have been proven to converge even for a limited range of inputs.

5.1 Nonlinear Optimality

In general, training an unsupervised network which contains nonlinear functions is very difficult. Because of the large (infinite-dimensional) space of possible functions, it is important to have detailed knowledge of which functions are likely to be useful in order to design an efficient network algorithm.

The network structure we consider is a linear layer represented by a matrix C

(which is perhaps an interior layer of a larger network) followed by node nonlinearities $s(y_i)$, followed by another linear layer (perhaps followed by more layers). We assume that the nonlinearities $s(\cdot)$ are fixed, and that the only parameters susceptible to training are the linear weights C.

If z is the M-vector of outputs after the nonlinearity, then we can write each component $z_i = s(y_i) = s(c_i x)$ where c_i is the i^{th} row of the matrix C, and $y_i = c_i x$ is the i^{th} output of the network before the nonlinearity. Note that the level contours of each function z_i are determined entirely by the vector c_i, and that the effect of $s(\cdot)$ is limited to modifying the output value. Intuitively, we thus expect that if y_i encodes a useful parameter of the input x, then z_i will encode the same parameter, although scaled by the nonlinearity $s(\cdot)$.

This can be formalized, and if we choose our optimality principle to again be minimum mean-squared approximation of the original input x given the output z, the best solution remains when the rows of C are a linear combination of the first M eigenvectors of the input correlation Q [Bourlard and Kamp, 1988]. This means that we want a nonlinear algorithm which, like the linear one, finds the eigenvectors of the input correlation. However, orthogonality may be different, since the nonlinearity will affect it.

5.1.1 Monotonic Node Functions

Let $z_i = s(y_i)$ be the output, where $s(\cdot)$ is an odd and monotonic function. This type of function includes the "sigmoid" functions often used in the literature. Note that its Taylor series has a constant term of zero, so that if y is zero-mean, z will be as well.

In section 2.1.2 we noted that the Hebbian algorithm is maximizing the variance of each output. For the generalized Hebbian algorithm, the variance is being maximized subject to the constraint of orthonormality of the weight vectors (hence statistical independence of the outputs). We now show that for x described by a Gaussian distribution, (1) maximizing the variance of y_i is equivalent to maximizing the variance of z_i, and (2) orthogonality of y_i implies the orthogonality of z_i.

First, we show that maximizing the variance of any y_i is equivalent to maximizing the variance of z_i. (For the moment, we will drop the indices i to simplify the expressions.)

The variance of z is given by

$$E[z^2] = E[s^2(y)]$$

For a Gaussian function, multiplying the variance σ by a factor r is equivalent to multiplying the random variable y by r and using the original distribution. (y is Gaussian because it is a linear function of x, which is Gaussian.) Therefore, if we change the variance of y from σ to $r\sigma$, we will now have

$$E[z^2] = E[s^2(ry)]$$

and since $s(\cdot)$ is monotonic, it is clear that for $r > 1$, the variance of z will increase.

To see that orthogonality of the scalars y_i implies orthogonality of z_i, we need only realize that since y_i is Gaussian, orthogonality implies independence. Thus the z_i are independent, and are therefore uncorrelated (provided they are zero-mean).

5.1.2 Rectification Nonlinearity

In the simulations, one nonlinearity $s(\cdot)$ which we use is a rectification nonlinearity, given by

$$s(y_i) = \begin{cases} y_i & \text{if } y_i \geq 0 \\ 0 & \text{if } y_i < 0 \end{cases}$$

Note that at most one of $\{s(y_i), s(-y_i)\}$ is nonzero at any time, so these two values are uncorrelated. Therefore, if we maximize the variance of y (before the nonlinearity) while maintaining the elements of z (after the nonlinearity) uncorrelated, we need $2M$ outputs to represent the data from an M-vector y, where the outputs are given by

$$z_i = \begin{cases} \sigma(c_i x) = \sigma(y_i) & 0 < i < M \\ \sigma(-c_i x) = \sigma(-y_i) & M \leq i < 2M \end{cases}$$

Note that $2M$ may be greater than the number of inputs N, so that the "hidden layer" z can have more elements than the input.

The outputs z_i will not be zero-mean, and this will present problems for training subsequent layers. In order to make them zero-mean, we can use a standard iterative algorithm (see [Oja, 1983]) to approximate the mean, and then subtract this value from the outputs.

5.2 Semi-Supervised Learning

To compute a particular desired function at the network outputs, a supervised algorithm such as Widrow-Hoff (LMS) for a single layer, or Backpropagation for multi-layer, could be used "on top" of the unsupervised network. For the case of Widrow-Hoff, the top layer is linear. For Backpropagation, the fact that the unsupervised net causes reduction in dimensionality, whitening of the input, and elimination of noise components, may allow faster and more accurate convergence of the supervised network (which is on top) to the desired function.

If a problem can be solved using a combination of unsupervised and supervised techniques to train the network, such a combination will have important advantages in terms of learning speed. Since the Generalized Hebbian Algorithm causes training time to scale linearly with the size of the network, while Backpropagation causes it to scale exponentially, we gain time by making the backpropagation-trained part of the network as small as possible. One possible construction technique for a network would be to insert a bottleneck layer at some point, and then to train everything before the bottleneck using an unsupervised algorithm, while training everything after the bottleneck using a supervised algorithm. While this may not produce optimal results in all cases, in some instances it may perform adequately while seriously reducing training time.

Chapter 6

The Nonlinear Generalized Hebbian Algorithm

In this chapter we show how a simple modification of the linear Generalized Hebbian Algorithm (GHA) can be used to train a single-layer nonlinear network. Such a network is defined to be a linear layer followed by a nonlinearity at each output. The object of this unsupervised learning algorithm is to find the optimal set of outputs for linear approximation of the original input with minimal mean-squared error.

6.1 Algorithm

In chapter 2 we derived the Generalized Hebbian Algorithm for the linear case which resulted in a weight update rule given by equation 3.2:

$$c_{ij}(t+1) = c_{ij}(t) + \gamma \left(y_i(t)x_j(t) - y_i(t) \sum_{k \le i} c_{kj}(t) y_k(t) \right)$$

or, in matrix notation, by equation 3.3

$$C(t+1) = C(t) + \gamma \left(y(t)x^T(t) - LT[y(t)y^T(t)]C(t) \right)$$

In the linear case, we compute y from x by

$$y(x) = Cx$$

In the nonlinear case, we have

$$z_i(x) = s(c_i^T x)$$

where c_i is the i^{th} row of C. To extend the Generalized Hebbian Algorithm to the nonlinear case, we apply the *same* weight update rule 3.2 to the values of z_i generated by the nonlinear network. In other words, the update rule remains the same, although the feedforward propagation now involves nonlinearities:

$$C(t+1) = C(t) + \gamma \left(z(t)x^T(t) - LT[z(t)z^T(t)]C(t) \right) \tag{6.1}$$

Note that equation 3.4

$$C \leftarrow C + \gamma_1(CQ - \mathrm{LT}[CQC^T]C)$$

is no longer a valid description of the procedure, and that CQC^T is no longer the output autocorrelation. In general, it will be difficult to predict the output autocorrelation, and this must actually be measured. However, for $s(\cdot)$ monotonic and odd and x Gaussian, we know that increasing the variance of y_i increases the variance of z_i, and orthogonality of y implies orthogonality of z. Therefore, in this case the variance of the outputs after the nonlinearity will have the same ordering and orthogonality that the variance before the nonlinearity had.

6.2 Explanation of the Algorithm

Although we have not proven that the nonlinear Generalized Hebbian Algorithm (GHA) converges, we can give a heuristic analysis of its stable points. First, assume that $C(t)$ changes slowly relative to $x(t)$ (γ is small) so that we can take the expected value over time of both sides of equation 6.1 to obtain:

$$C(t+1) = C(t) + E[z(t)x^T(t)] - \mathrm{LT}[E[z(t)z^T(t)]]C(t)$$

(For the correlations to make sense, we should estimate $E[z]$ and subtract it from the output to make it zero mean.) If $x(t)$ is a time-independent Gaussian process with correlation Q, then $E[z(t)x^T(t)] = kCQ$ for some proportionality constant k (see, for instance, [Hunter and Korenberg, 1986]). If the algorithm converges, we expect $C(t+1) \approx C(t)$, so we seek solutions of

$$kCQ - \mathrm{LT}[E[z(t)z^T(t)]]C(t) = 0$$

If we look at the first row of C at steady-state, it satisfies

$$kQc_1 = E[z_1^2]c_1$$

so it is proportional to some eigenvector of Q. (In practice, it usually converges to the eigenvector of largest eigenvalue.) Similarly, the second row c_2 satisfies

$$kQc_2 = E[z_2^2]c_2 + E[z_1z_2]c_1$$

If z_1 and z_2 are uncorrelated, then $E[z_1z_2] = 0$, and if a solution exists, it is proportional to some eigenvector of Q (possibly proportional to, but not the same as c_1). If $E[z_1z_2] \neq 0$, then a possible solution is $c_2 \propto c_1$, and others may exist as well. We can continue this type of analysis for the other rows of C.

For a rectification nonlinearity, we have $E[z_1z_2] = 0$ if either $c_2 = -c_1$ or $c_2^TQc_1 = 0$ (where the input is Gaussian and the output has been modified to have mean zero). We thus expect that C may converge to a matrix each of whose rows is proportional to an eigenvector of Q, and for which no three rows are proportional to the same eigenvector. In practice, the first two rows usually converge to the eigenvector of largest eigenvalue and its negative, and all subsequent rows occur in pairs ordered by eigenvalue. The algorithm is therefore finding the outputs which (for Gaussian inputs) maximize the output variance yet remain uncorrelated. Although these are only heuristic arguments, they help to explain the observed behavior of the algorithm when a rectification nonlinearity is present.

6.3 Multi-layer Networks

For linear networks, having more than one layer is unnecessary, since a multi-layer linear network can always be compressed into a single layer. In chapter 4 we suggested that a two layer linear network can be useful if the first layer is trained according to an unsupervised learning rule, while the second is trained using a supervised rule. However, after the training is complete, the network could still have been compressed into only a single layer.

For nonlinear networks, however, multiple layers confer a great increase in potential. The range of functions which can be computed with a multi-layer network is significantly greater than that which can be computed with a single layer. Although there may well be useful functions which can be well approximated with only a single layer, many more functions can be approximated if we can add layers arbitrarily. For data compression, it is possible that higher layers may be able to find more efficient encodings than lower layers.

Since the nonlinear Generalized Hebbian Algorithm is an unsupervised algorithm, the training of each layer is independent of the training of the layers above. The layers below affect training only to the extent that they change the input distribution. Therefore, a multi-layer network can be trained "bottom-up" in the sense that the layers closest to the input are allowed to mature, and then the layers above can mature. As each layer converges to a stable set of weights, the output distribution will remain fixed, and the layer above will be able to use that distribution to change its set of weights. Once the next layer has converged, its output distribution will be fixed, and training can propagate upward through the network until the entire net is trained. Note that there is no reason not to train all layers simultaneously, except that each layer will not converge to its final set of weights until all the layers below have converged.

We see that the training time will be at worst equal to the sum of the training time for each layer, and perhaps better if higher layers "track" the development of the ones below during training. In section 3.6 we saw that the rate of convergence for any layer is approximately a linear function of the number of outputs of that layer. The discussion also holds true for the nonlinear case, since convergence of any output unit requires all previous outputs to have converged so that the effective (subtracted) input is stable. Therefore, the training time for the entire network is approximately proportional to the number of units in the network. This is considerably better than the exponential time dependence which has been observed for the backpropagation algorithm (see section 1.2.1).

Part III

Examples

Chapter 7

Learning Curves

In this chapter we will show curves for the rate of learning of networks trained with the Generalized Hebbian Algorithm (GHA). The networks are given input which is a set of jointly Gaussian variables with a randomly chosen correlation matrix. Convergence of any particular layer is shown in terms of the diagonalization of the output correlation matrix. Note that the Generalized Hebbian Algorithm must not only diagonalize the output, but must also keep the basis vectors orthogonal and normalized. Therefore, diagonalization of the output does not completely determine the energy function for convergence, and the function which we define will not change monotonically with time (for the true energy function, gradient descent would force the energy to monotonically decrease).

7.1 Linear Networks

Figure 7.1 shows the diagonalization and output errors for a linear network being trained to auto-associate. The network has five input units, three hidden units, and five output units, and it is trained "semi-supervised". The first layer is trained using the Generalized Hebbian Algorithm (GHA), and the second layer is trained in a supervised manner using the Widrow-Hoff algorithm, so that the

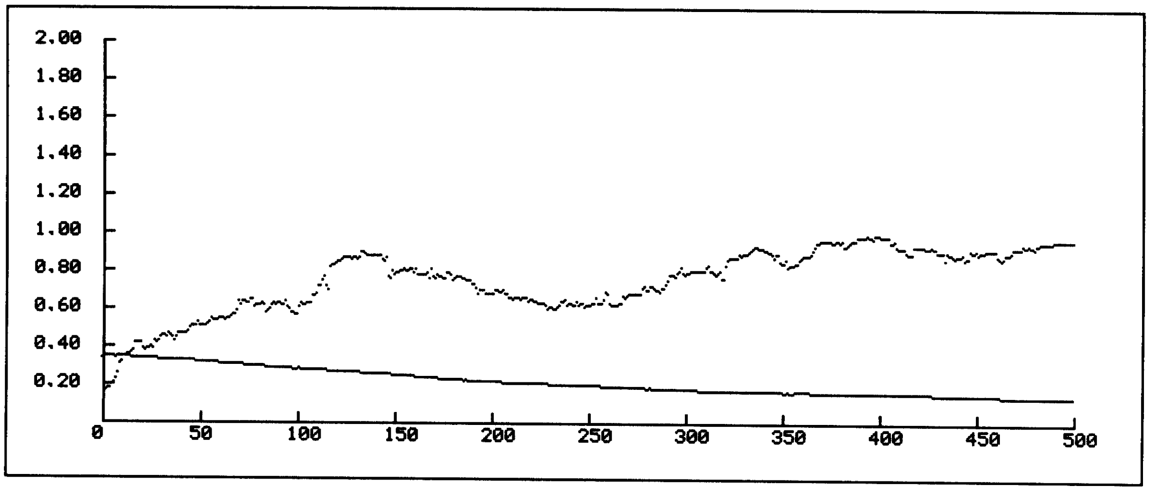


Figure 7.1: Learning curve for a two-layer linear network showing both progressive diagonalization of the autocorrelation at the hidden layer, as well as progressive decrease in approximation error at the output (see text). Vertical axis is both the diagonalization and the mean-squared output error.

outputs attempt to approximate the inputs.

Figure 7.1 shows the diagonalization of the hidden layer correlation increasing with time, while the mean-squared output error decreases. (The printed values are computed at each time using a running average of past values.) The diagonalization is computed as

$$D = \frac{2\text{trace}[CQC^T] - \text{trace}[CQC^TCQC^T]}{\text{trace}[CQC^TCQC^T]}$$

which compares the diagonal entries to the off-diagonal entries in the output correlation matrix, and normalizes by the sum of the squares of all the entries. The diagonalization approaches 1, indicating that the off-diagonal elements of CQC^T are small compared to the diagonal elements. The output error does not reach zero, however, since it is impossible to code five inputs with only three hidden units without a certain amount of error (the minimum error is determined by the smallest two eigenvalues).

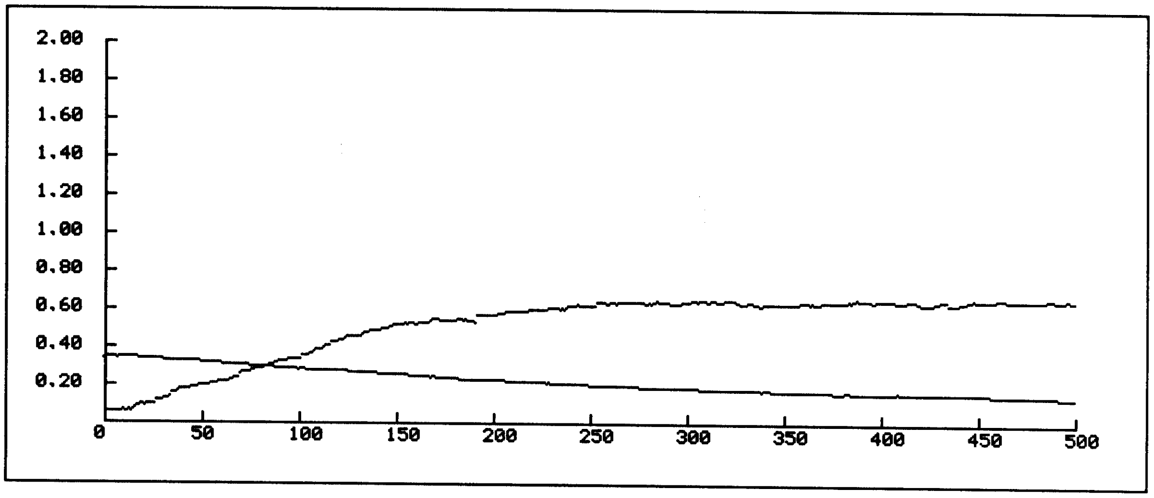


Figure 7.2: Learning curves for the two-layer network of figure 7.1 when trained using Self-supervised Backpropagation.

Figure 7.2 shows the effect of training the same network of figure 7.1 using self-supervised backpropagation (SSBP). The sequence of inputs, the initial random weights, and the learning rates were identical, and both the output error and hidden-layer diagonalization are shown. The output error is almost identical, as we would expect. The diagonalization is not quite as good, indicating that the hidden layer units are not completely uncorrelated. As mentioned previously, GHA and SSBP are equivalent for linear networks, except for the fact that SSBP does not find the actual eigenvectors, but instead finds some linear combination of them. Note that we cannot compare the speed of convergence between the two different algorithms, since the speed can be increased arbitrarily by increasing the rate constant for learning (until instability results). We can, however, compare the final value of the mean squared output error once each algorithm has had sufficient time to converge.

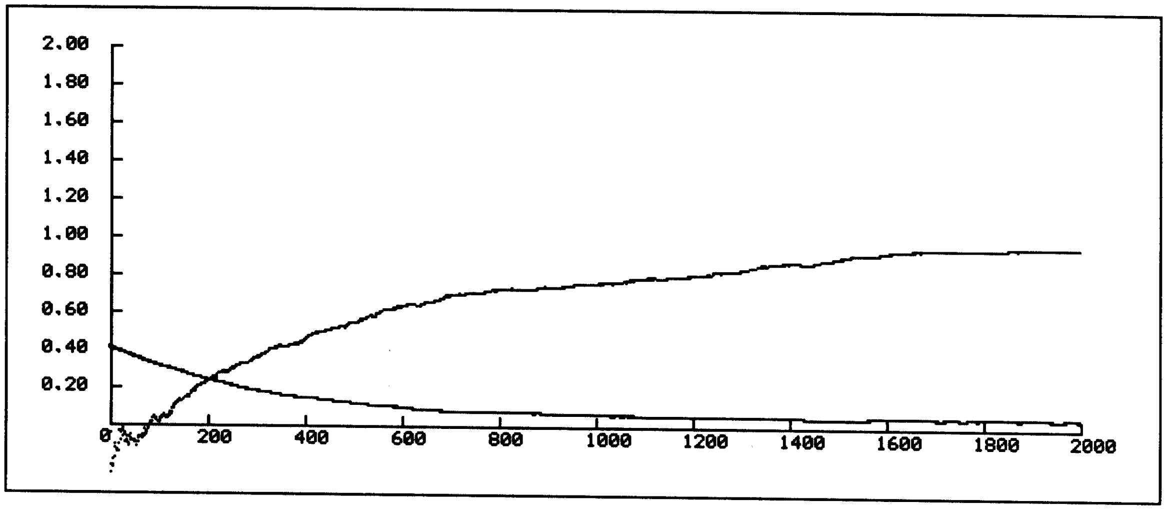


Figure 7.3: Learning curves for a two-layer nonlinear network.

7.2 Nonlinear Networks

For the nonlinear examples, we use a sigmoid nonlinearity given by

$$s(x) = \frac{2}{1+e^{-x}} - 1$$

and the diagonalization is computed using

$$D = \frac{2\mathrm{trace}E[yy^T] - \mathrm{trace}E[yy^Tyy^T]}{\mathrm{trace}E[yy^Tyy^T]} \tag{7.1}$$

The expected value $E[yy^T]$ was approximated using past values of the matrix yy^T. Note that one cannot use CQC^T in the nonlinear case, since this expression does not describe the true correlation matrix at the hidden units (after the nonlinearity).

Figure 7.3 shows the diagonalization and output errors curves for a two-layer nonlinear network trained using semi-supervised learning. There are five inputs, three hidden units, and five outputs. There is a sigmoid nonlinearity at the hidden units, but not at the outputs. The first layer is trained using the nonlinear

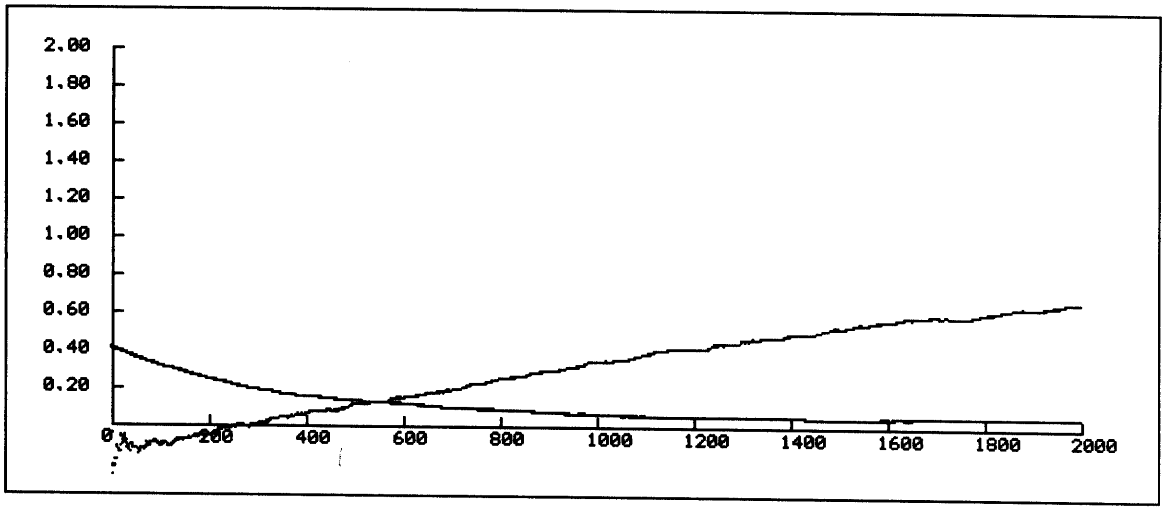


Figure 7.4: Learning curves for the two-layer nonlinear network of figure 7.3 trained using Self-supervised Backpropagation.

GHA, while the second layer is trained using the Widrow-Hoff algorithm. The diagonalization is graphed according to equation 7.1, and approaches the limit of 1. The output error achieves its minimum value by step 1000, even though the diagonalization continues to improve slightly.

Figure 7.4 shows the same network as figure 7.3 trained using SSBP. The data, initial weights, and learning rates were the same for both networks. The output errors decreased at almost exactly the same rate. The diagonalization of the hidden units proceeded much more slowly, however, and appears to be almost linear with time as opposed to GHA which gave a more exponential change in diagonalization with time. Note that for the auto-association problem, we would expect semi-supervised learning and backpropagation to achieve practically identical results in terms of output error, since SSBP and GHA are equivalent. However, for backpropagation in which the desired output is not the same as the input, we would expect semi-supervised learning to be considerably worse. This is due to the fact that the only supervised layer is the last one, so the hidden layers

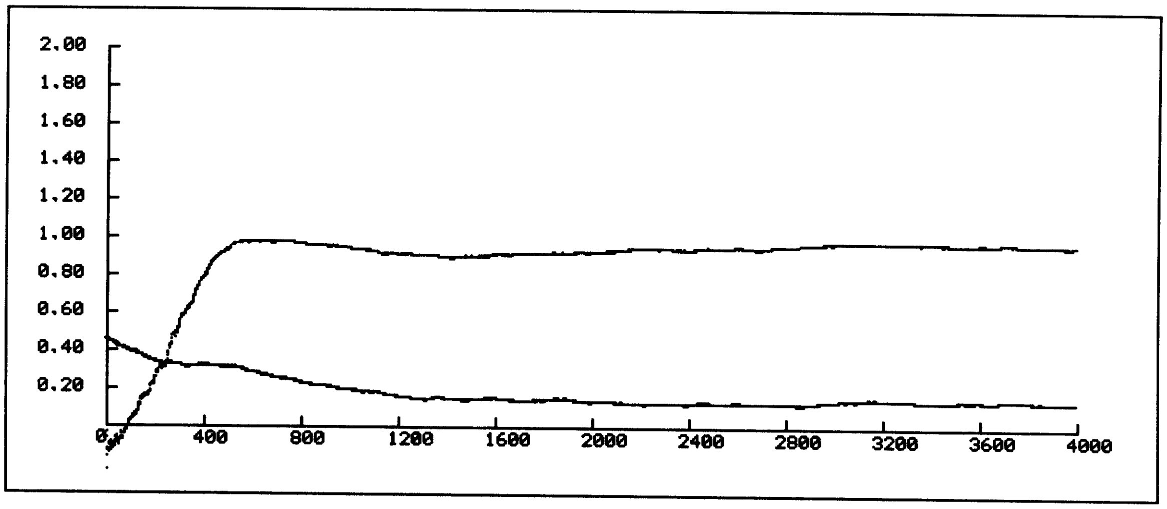


Figure 7.5: Learning curves for a three-layer nonlinear network. Diagonalization is shown only for the first layer.

cannot optimize themselves to perform the particular desired task. However, since the Generalized Hebbian Algorithm was designed to optimize the reconstruction of the input, the two algorithms are equivalent in this particular case.

Figure 7.5 shows the learning curves for a nonlinear network with five inputs, two layers of three hidden units, and five outputs. Each hidden layer has a sigmoid nonlinearity. The two hidden layers are trained using the GHA, while the output layer is trained using Widrow-Hoff to approximate the inputs. Figure 7.5 shows the diagonalization for the first layer of hidden units, as well as the mean squared output error. The time between convergence of the first layer (step 500) and the minimum output error (step 1300) represents the time needed for the second and third layers to converge.

For comparison, figure 7.6 shows the network of figure 7.5 trained using Backpropagation. The data, initial weights, and rates were the same. The network achieves the same final value of mean squared approximation error. (Although it does not converge until step 2200, we should not use this as a speed comparison

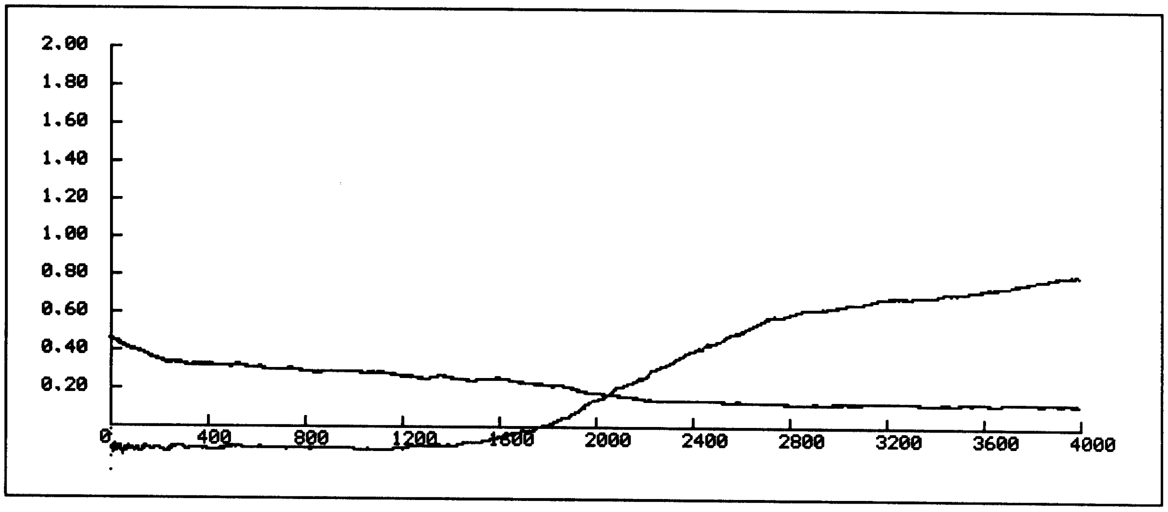


Figure 7.6: Learning curves for the three-layer nonlinear network of figure 7.5 when trained using Self-Supervised Backpropagation.

since it depends on the rate constants chosen for the two algorithms.) However, note that the first layer diagonalization does not occur until significantly later. In fact, this diagonalization occurs while the output error is approaching its minimum value. This reflects the fact that for backpropagation, the top layers train first, and then the errors propagate to the lower layers. Conversely, for semi-supervised learning, the lower layers train first, and the top layers converge afterwards. Therefore, in figure 7.5, diagonalization of the first layer is the first thing to occur, wherease in figure 7.6 the first layer is not diagonalized until the second layer has had time to converge. At this point, both hidden layers have converged and the output error achieves its minimum value almost immediately.

Chapter 8

Texture Segmentation

One field in which neural networks are potentially useful is the field of image processing. There are many unsolved problems in image processing, and the capability of networks to discover useful algorithms on their own suggests many important applications. Of course, the current state of neural network research limits the real-world applications to very simple types of processing. The significance of unsupervised algorithms for image processing is the ability to find "features" of the image which can be used for coding or to locate distinct regions. Decomposition of an image into features is an important component of both machine and biological vision.

For our purposes, we define texture segmentation as the discovery of a set of features which vary over the image in such a way that regions of different "texture" are maximally separated. Unfortunately, this definition suffers from a lack of knowledge of what we mean by "texture". Texture has been defined in the computer science literature in many different ways. Here, we assume that a region of uniform texture can be described by a spatially stationary stochastic process which is determined by second-order (or higher) statistics. The stationarity is what defines a texture region. Clearly, the statistics will not remain stationary over the entire image, but we require them to remain stationary over what we thereby define as a uniformly textured region. The only example we present is an artificially simplified texture consisting of horizontal and vertical lines. We must emphasize that this is not a general model of texture, and that these results should be considered a preliminary demonstration of the algorithm and not a workable method for real texture segmentation.

Spatial stochastic processes described entirely by second-order statistics may be important for humans, since we can often discriminate textures differing in second-order, but not third-order, statistics [Julesz, 1971] (although there are counter-examples). Second-order stationary regions are described completely by the mean luminance and the two-dimensional autocorrelation function. These statistics can be generated by passing a white noise image through a two-dimensional linear filter. We will therefore model such texture regions as filtered white noise. Different regions of uniform texture have second-order statistics generated by different linear filters applied to white noise. The linear least squares estimator (LLSE) for distinguishing two such regions is formed by comparing the output amplitude of two filters each of which is matched to one of the texture regions. This is the optimal linear estimator, and if the textures are truly second order, then it is the nonlinear optimal estimator as well.

In the following discussion, we will show examples of a single-layer linear network which has been trained to solve a simple texture discrimination problem. The two textures differ in only their second order statistics (they have the same mean luminance, and the same higher order statistics). Therefore, the linear estimators determined by the network could theoretically be the optimal estimators, if the network is trained properly.

8.1 Feature Discovery

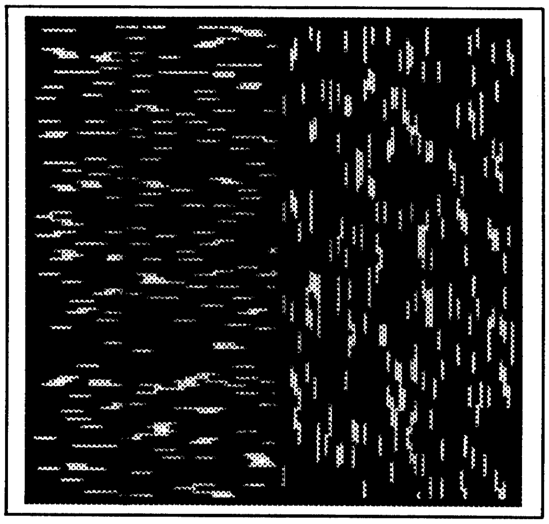


Figure 8.1: 128x128 (8 bit) image consisting of two different textured regions.

Figure 8.1 consists of two regions of texture which differ only in their second-order statistics. (This is a very artificial example of texture, and we use it only to demonstrate simply the behavior of the algorithm.) This image was used as input to train a network in the following way. Small 8 by 8 blocks of the image were extracted, and the 64 grey values were used as inputs to the network. Blocks were chosen beginning at the upper left and proceeding across and down over the image, and the entire image was scanned ten times for a total of 2500 input examples. Before training the network, each 8 by 8 input block was multiplied by a spatially Gaussian mask with variance 2. The purpose of the mask was to force the input to have greatest power near the center of the blocks, so the network would not attempt to represent the edges (which have a great deal of power, but not as much useful data). The network was linear and had 64 inputs and 8 outputs. After training, we display the vectors of weights to the outputs as 8 by 8 masks, as in figure 8.2.

The output order proceeds from left to right and top to bottom. Note that due to the Gaussian mask which was applied to the input, each output has learned a

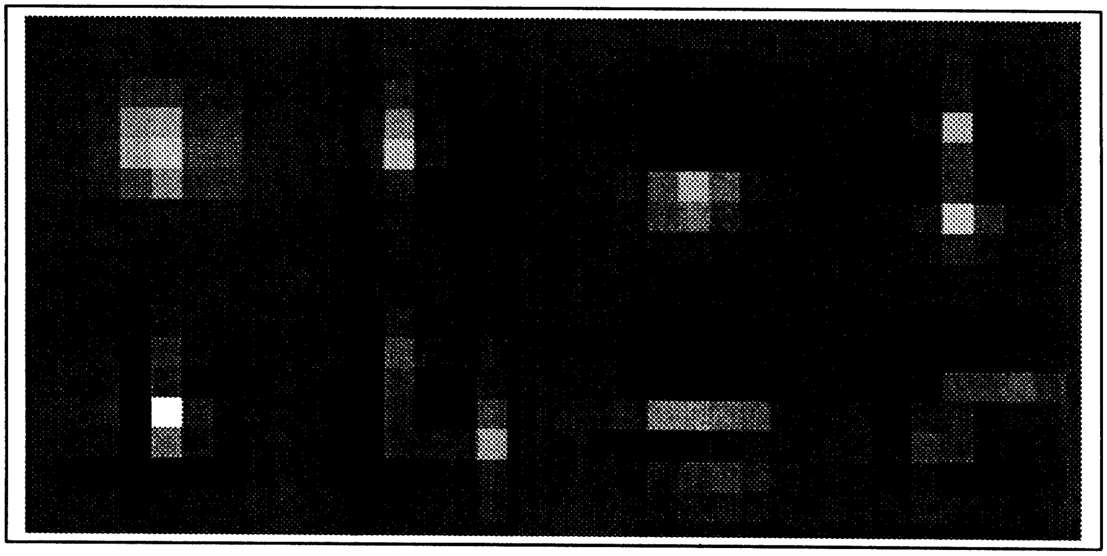


Figure 8.2: The first eight masks learned by a network trained on 8 by 8 patches of figure 8.1. Masks are ordered left to right and top to bottom.

pattern with a roughly Gaussian amplitude response. The first mask represents the average local image intensity, since it is essentially a lowpass spatial filter. The second mask can be considered to be an "edge-detector" for vertical edges. Likewise, the third mask is an "edge-detector" for horizontal edges. The fourth and fifth masks are "bar-detectors" for vertical and horizontal bars, respectively. The sixth, seventh, and eighth masks perhaps recognize corners or ends of lines.

We may now ask why these particular masks evolved. The answer lies in the fact that the training algorithm attempts to maximize the output variance while maintaining the outputs orthogonal. These eight masks can be shown to be approximately orthogonal (by computing the output cross-correlation). For example, horizontal and vertical "edge-detectors" are orthogonal both with respect to the input distribution (horizontal and vertical edges never occur together), and with respect to the uniform distribution (multiplying the two masks together and summing yields zero). The fact that the first mask is a lowpass filter indicates that the low-frequency variations in intensity have very high variance over the image.

The second through fifth masks will have higher variance in one half than the other. They maximize their variance by maximizing it with respect to a particular texture, since the only mask with higher variance over the entire image is the lowpass filter. A vertical "edge-detector" will have very high variance in the right half of the image, but lower variance in the left half (and conversely for a horizontal "edge-detector"). Since it has high variance only for half the image, its variance is not as great as the lowpass filter. A "bar-detector" also has high variance in one half or the other, although not as much as the edge detectors. However, bar detectors can be orthogonal to edge detectors, so the next two masks are bar detectors. The sixth, seventh, and eighth masks will respond to both halves of the image, but their variance in each half is not particularly great, so these filters appear only in the later outputs. (If the input image had consisted of short lines at different orientations, or a different set of patterns, we might imagine that masks would develop to match these patterns, and in fact this is what happens, although we do not present these results here.)

8.2 Amplitude Response

Figure 8.3 shows the result of convolving the input image of figure 8.1 with the masks learned by the network (figure 8.2). Since each of the learned masks has an approximately Gaussian amplitude distribution, the convolved images are somewhat blurred. However, it is clear that the first mask is a lowpass filter, while the edge- and bar-detectors respond to one half of the image or the other. What is meant by "respond" is that the output variance is higher for one textured region than for the other. To see this, we take the absolute value of the convolution outputs of figure 8.3 and then lowpass filter the result by convolving with

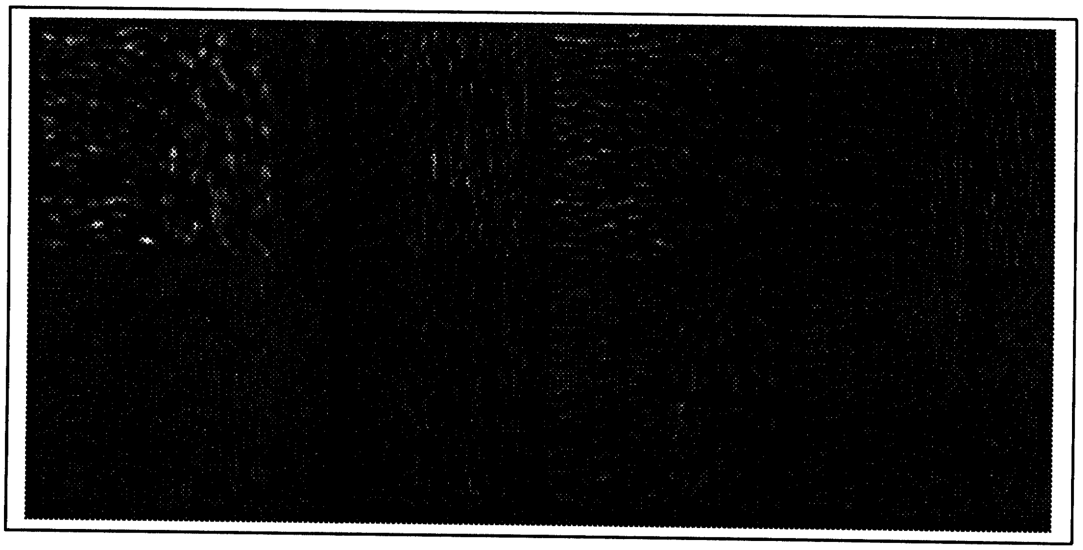


Figure 8.3: Result of convolving the input image 8.1 with each of the masks of figure 8.2.

a Gaussian mask of variance 3 pixels. This procedure gives an approximation to the local variance, and the result is shown in figure 8.4.

It can clearly be seen that masks two and three have significantly higher variance on one half of the image or the other. The other filters show less of a difference. But the first two filters can be used in this way to separate the original input into textured regions. Note that the network has *not* discovered "matched" filters for the texture regions which give high positive amplitude in the presence of a particular texture. Since the algorithm will maximize variance, the network has learned a set of masks which give high output *variance* in the presence of a particular texture. To use this output to perform texture segmentation, we need to have some method for measuring the local variance. Taking the absolute value of the convolution product and filtering is one way to accomplish this.

Note that the network will not necessarily find a set of masks such that each mask responds optimally to a particular texture. Rather, it will find masks such that the *entire* set can be used to discriminate different textures. This allows a

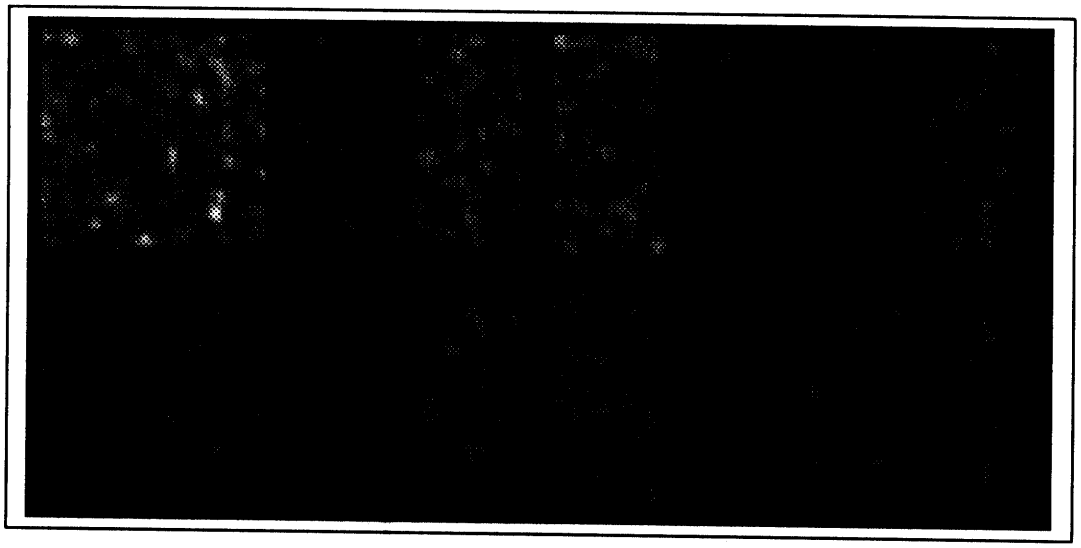


Figure 8.4: Approximation to the local variance of 8.3.

much larger number of textures to be discriminated, since they could theoretically be recognized as a combinatorial pattern of variance on the output units.

8.3 Biological Relevance

There is extensive evidence that early stages in primate cortical image processing involve filters which are similar to edge and bar detectors (see, for example [Hubel and Wiesel, 1974, Foster *et al.*, 1985]). This processing is performed by the (approximately spatially) linear simple cells. There is also increasing evidence that the nonlinear complex cells can be modelled as "envelope detectors" or "amplitude demodulators" of the input signal [Cavanagh, 1984, Baker and Cynader, 1986, Pollen *et al.*, 1988]. One way to implement such a model is by taking the absolute value of the output of a linear filter and lowpass spatial filtering the result. This is the procedure that we used to isolate texture regions for figure 8.4. This method has been suggested as a model for certain types of complex cells, as well

[Pollen and Ronner, 1983, Cavanagh, 1984, Spitzer and Hochstein, 1985, Gaska *et al.*, 1987]. Therefore, we might expect that one of the functions of cortical complex cells might be texture discrimination, and that these cells might perform this task using a method similar to the one we used to interpret the output of the artificial neural network. In fact, there is some evidence that complex cells can perform texture discrimination [Nothdurft and Li, 1985].

A mechanism similar to that proposed here has been simulated using Gabor functions to model the first-layer linear masks [Turner, 1986]. The absolute value function can be performed by adding the (rectified) outputs of simple cells with 180-degree differences in phase response [Pollen and Ronner, 1983, Turner, 1986]. Summation over many such simple cell outputs would then produce a local estimate of the output variance of the first layer. Gabor filters were also used in [Daugman, 1988a] to perform texture segmentation, although the local variance or response amplitude was not explicitly computed. Similarly, Voorhees (1987,1988) used $\nabla^2 G$ filtering to discriminate texture regions, while Bergen and Adelson (1988) used a center-surround with a rectification nonlinearity.

Chapter 9

Cortical Receptive Fields

There have been several attempts to have a neural network learn to produce visual receptive fields similar to those found in primates [Bienenstock *et al.*, 1982, Linsker, 1986a, Barrow, 1987, Kammen and Yuille, 1988, for example]. In this chapter we present yet another attempt. An important difference between the method presented here and previous attempts is that we can show why the particular receptive fields developed and that these receptive fields have a significance which is independent of the learning algorithm which produced them.

9.1 Methods

We assume that the world passing in front of the retina of a developing animal can be modelled (to second-order statistics) as white noise. We also assume that the retina performs linear processing which can be modelled as bandpass spatial-frequency filtering. The inputs to the network will come from only a single type of bandpass retinal filter, and these inputs will be limited to a small circular region of the input image by having a spatially Gaussian weighting applied as an input mask.

With these assumptions, the input to the network is generated by starting with a white noise image with mean zero, lowpass filtering using a Gaussian filter, and then applying a Gaussian windowing function to the result. The network has 4096 inputs arranged as a 64x64 array. For the following simulations the filtering Gaussian has a standard deviation of 3 pixels and the windowing Gaussian has a standard deviation of 8 pixels.

The simulated inputs have been applied to two different types of network. The first is a linear network with 4096 inputs and 16 outputs. We would expect the 16 outputs to converge to approximate the first 16 eigenvectors of the input distribution. The second type of network is also a single layer, but a rectification nonlinearity is applied so that negative outputs are set to zero. This is intended as a simulation of the fact that in a biological system there is no ability to represent a negative response using the output spike frequency from a cell. The effect on the masks which are learned by the network is that the negative of a filter is orthogonal to the filter, since if one is positive, the other must be zero. Therefore, the network can learn both a filter and its negative (which is not possible in the linear case, since these two filters are strongly (anti-) correlated).

9.2 Results

Figure 9.1 shows the first sixteen receptive fields which are learned by a single-layer linear network with 4096 inputs and 16 outputs. (White is positive and black is negative values.) 1500 sample inputs were used. The first mask is an "on-center" cell which has a strong positive center and a circularly symmetric inhibitory surround. It is therefore an isotropic bandpass spatial frequency filter, of the type often found in retina, LGN, and cortex of the cat and monkey [Maffei and Fiorentini, 1973]. The second and third filters are "edge-detectors" and the fourth, fifth, and sixth are oriented "bar-detectors", as often are found in the cortex of cat and monkey [Hubel and Wiesel, 1962, Hubel and Wiesel, 1974, Foster

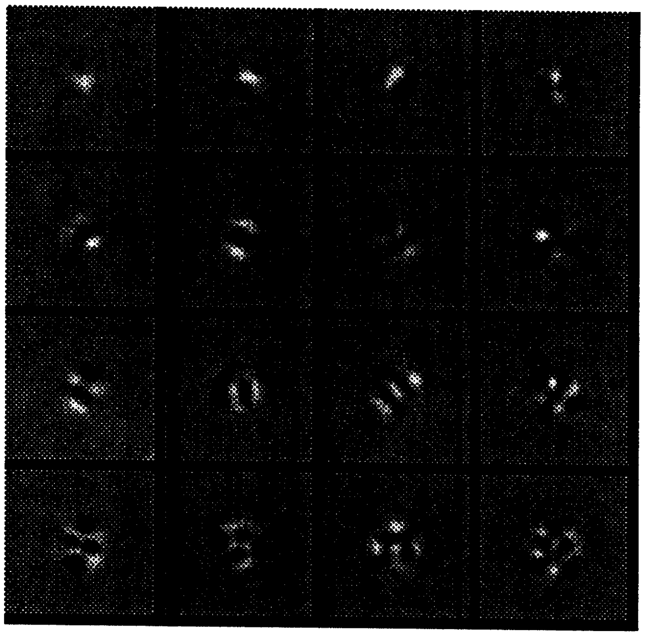


Figure 9.1: First 16 receptive field masks learned by a linear network with random input. Ordering is left to right and top to bottom. Brightness has been normalized so that the maximum for each mask appears white. The actual magnitudes are determined by the requirement that the sum of the squares of all the pixel values for each mask must be 1.

et al., 1985, for example]. Beyond these filters, we find higher-order filters with different spatial patterns. No cells with such receptive fields have ever been found in primate cortex.

Figure 9.2 shows cross-sections through the major axes of the first, third, and sixth masks of figure 9.1. The first cross-section is qualitatively similar to the receptive field shapes of retinal ganglion cells shown in [Enroth-Cugell and Robson, 1966]. The second and third cross-sections are similar to the receptive field shapes of cortical simple cells shown in [Andrews and Pollen, 1979].

Figure 9.3 shows the receptive fields which develop when a rectification non-linearity is present during learning. In this case, a field and its negative will be orthogonal, and we would expect both to be present after learning. The first two

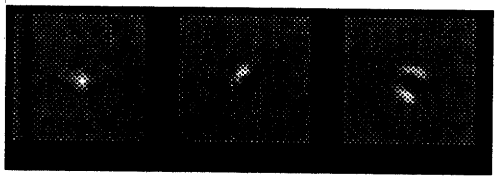
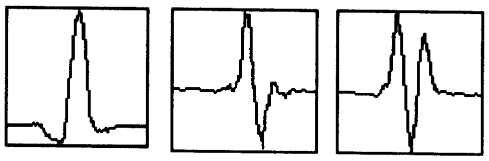
a.

b.


Figure 9.2: (a) First, third, and sixth receptive fields from fig. 9. (b) Cross-sections of (a) through major axes.

filters are off- and on-center circularly symmetric bandpass filters. The third and fourth are edge detectors both tuned to horizontally oriented edges of opposite polarity. The following filters often do not come in pairs, but represent a number of different oriented edge and bar detectors. These filters have differing orientations as well as differing spatial phase (as found for cortical cells in [Pollen and Ronner, 1981]). Since more of the outputs are taken up due to the extra filters possible with the rectification nonlinearity, we see fewer of the "higher-order" filters that we found in figure 9.1. However, if more outputs were available, these filters would develop in this case, as well.

9.3 Discussion

These results are not meant to imply that the visual system develops through an adaptation mechanism similar to the algorithm presented here. We have no evidence for such a claim. Rather, we claim only that our results imply that there exist simple algorithms through which the observed receptive field shapes can develop.

Since the Generalized Hebbian Algorithm finds the eigenvectors of the input

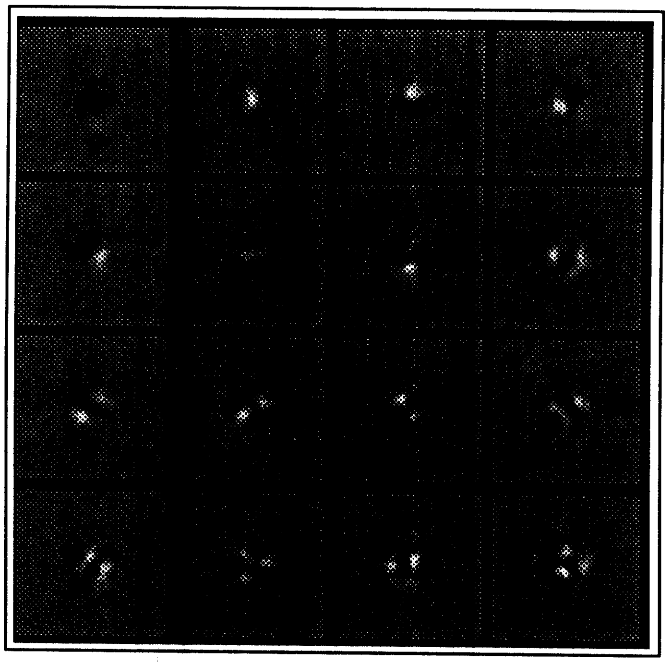


Figure 9.3: Nonlinear receptive fields ordered from left-to-right and top-to-bottom.

distribution, the masks which it learns must be the eigenvectors of the data we presented. Since these masks are similar to the actual observed receptive fields, it follows that the receptive fields may actually be performing an eigenvector decomposition of the input and therefore have the optimality properties of the KLT (if we assume that the statistics of typical visual input can be modelled as bandpass filtered white noise).

This interpretation of the role of the visual cortex unifies several other disparate views of its computations. Some authors believe that the visual cortex recognizes simple features in the environment which represent significant parts of the image such as edges or blobs [Hubel and Wiesel, 1962, Marr, 1982, for instance]. Other authors consider the early stages of the visual system to be performing a localized spatial-frequency decomposition of the image, or a sort of "local Fourier transform" [Shapley and Lennie, 1985, for review]. However, these two seemingly different viewpoints are both equivalent to the observation that simple cells represent the eigenvectors of the input distribution. It is well known that the eigenvectors of any stationary distribution are given by complex exponentials, which is the same basis as that of the Fourier transform (see [Kazakos, 1983, Yuille *et al.*, 1988] for a detailed discussion). Therefore, the Generalized Hebbian Algorithm will learn a set of filters which can perform a spatial-frequency analysis of the input image. Also, if the input image has statistically significant features such as edges, maximization of variance will produce "edge-detectors". In figure 9.1 we see that spatial-frequency detectors and edge detectors can have very similar shapes.

The significance of the filters learned by the Generalized Hebbian Algorithm is thus greater than either spatial frequency analysis or feature detection. Both types of decomposition of the image occur simultaneously due to the maximization of output variance. The algorithm chooses its representation of the input based upon the true statistics of the input. For real scenes, these statistics include features as well as spatial frequency components.

9.4 Other Work

We will compare this model of cortical development with two others which have recently been proposed. There are many examples in the literature, [Von der Malsburg, 1973, Bienenstock *et al.*, 1982, Yuille *et al.*, 1988, among others] but the main ideas presented here will be explained in the context of these two.

9.4.1 Linsker

Linsker has proposed a multi-layer linear network model of cortical development [Linsker, 1986a, Linsker, 1986b, Linsker, 1986c]. The input is random white noise, and he shows that the network converges to cells which in early layers resemble center-surround cells, while in later layers resemble edge- and bar-detectors. Since Linsker's network is capable only of finding the principal eigenvector, he uses multiple layers to enable the principal eigenvector to be different at each layer. For the earlier layers, the principal eigenvector is the center-surround, as we found in the simulation using the Generalized Hebbian Algorithm. At later layers, for certain values of his parameters, oriented edge- and bar-detectors become the principal eigenvectors (perhaps due to symmetry-breaking [Kammen and Yuille, 1988]). Since he can only derive the principal eigenvector, Linsker requires multiple layers in order to obtain different types of cells. Note that the layers are all linear, and could therefore after training be represented with only a single layer.

As a model of cortical synaptic development, there is not sufficient evidence to choose between the model that Linsker proposes and that presented here. Linsker's method is less intuitive from a mathematical standpoint, since it does not derive all the eigenvectors at the same time, and since it is not clear that anything beyond the first few are capable of being found by his network. However, such facts mean nothing to biology, and this does not rule out his model as an explanation for the actual development of cortical synapses.

One apparent difference in implementation between the method we present and Linsker's is that he starts with a white noise image, while we provide a carefully chosen image which is lowpass filtered and then windowed. However, if we look at Linsker's first layer, he chooses parameters such that all the synaptic weights are +1, and he constrains these synapses to have a distribution which falls off spatially as a Gaussian function. Therefore, the second layer of Linsker's network presents the third layer with exactly the same input which we use here. This is why the third layer of his network develops center-surround cells, which we found to represent the principal eigenvector of this particular distribution. Note that, if we assume Gaussian windowing but wish to derive the lowpass filter, a lowpass filter is the principal eigenvector of any distribution for which the mean is nonzero.

Since there is no such thing as a negative luminance, whatever signal is sent from the retina (or within it) must have positive mean, and therefore the first layer which adapts to that information will respond to the principal eigenvector which is a lowpass filter.

9.4.2 Barrow

Barrow [1987] has developed a very similar model to ours. He also uses a Gaussian windowing function, but with a bandpass spatial filter preceding it (which he claims is a model of retinal and LGN processing). He uses a natural image, however, rather than white noise, and scans over it much as we did for the texture segmentation problem. The learning algorithm is equivalent to the Oja algorithm [Oja, 1982] but with a winner-take-all competitive rule to allow different outputs. This generates results equivalent to the Generalized Hebbian Algorithm, although the ordering of the eigenvectors is random, and there may be linear combinations. Barrow's results show very clear edge- and bar-detectors, whose receptive field profiles are remarkably similar to those found in cortical cells [Marcelja, 1980].

Barrow's work is equivalent to the model of cortical development presented here. However, he does not explain *why* that particular set of cells should develop. The fact that his results are similar to those produced by the GHA, which is known to converge to the KLT, provides an answer. By realizing that algorithms such as Barrow's or GHA accomplish an approximate eigenvector decomposition, we can understand the true significance of these cells.

9.4.3 Spatial-Frequency Channels

Primate visual systems are divided into many different "channels" for different ranges of spatial frequency [Shapley and Lennie, 1985, for review]. None of the models discussed here is capable of learning to represent such channels. The spatial frequency response of the learned filters is entirely determined by the shape of the bandpass filter used to generate the input images (for Linsker, this is the width of the region of first-layer inputs, and for Barrow it is the shape of the retinal and LGN filters). This occurs because once we filter the input with a bandpass filter, most of the variance of the input now occurs in that particular spatial frequency band.

To model actual cortical processing we would have to assume that different bandpass filters were "hard-wired" at some early point in the visual system. Note that filters tuned to different ranges of spatial frequency are orthogonal. However, the Generalized Hebbian Algorithm does not discover filters with different frequency response until very late in eigenvalue order, and it requires many iterations to produce poor-quality results. One method to allow different ranges of spatial frequency to develop is to change the bandpass filter characteristics with time, and train different outputs at different times. For instance, if we started with a lowpass filter input and trained one set of filters, then we could increase to a bandpass filter at the input and train a different set of filters which would now have a different frequency response. If we desired the new filters to be orthogonal to the old ones, we could easily "subtract off" the responses of the old filters from the input before training the new filters, precisely as GHA normally does. The only difference would be that the weights to the old filters would not be modified.

There is some biological evidence that a change in spatial frequency response occurs during human development. Wilson [1988] provides a review of some of the literature. He also derives a theoretical model to explain the increase in infant visual acuity, which is related to a change in foveal cone density and outer segment size. This kind of change is exactly what would be required to provide a changing spatial-frequency distribution to a cortical learning network. For each range of spatial-frequency inputs, on- and off-center cells would develop first, and then orientation cells would develop (Wilson cites results showing that orientation sensitivity does indeed develop slowly over a period of about ten months). As the cone spacing and size change, successively higher ranges of spatial frequency sensitivity would develop.

We must emphasize here that although these results suggest a mechanism by which GHA could learn multiple spatial frequencies, *there is no evidence that such a learning algorithm is actually used.* The results described here are pure conjecture, and demonstrate only that it is not impossible that a similar mechanism might be functioning in early human development. If this is true, the Generalized Hebbian Algorithm must be a gross simplification of the biochemical events which lead to early visual development, and it must be construed as the vaguest possible generalization of an extremely complex process which researchers in the neurosciences are only beginning to understand.

9.5 Higher Layers

If we allow a nonlinearity in the network, then higher layers can be allowed to adapt to the filtered white noise which we used to derive the linear layer of cortical processing. We can model the nonlinearity at the output of simple cells by a simple rectification, and use the rectified outputs as input to another layer. We have not performed this simulation. We can, however, predict that since the outputs are always positive, the mean is nonzero, and therefore the first eigenvector is a constant (a lowpass filter). This eigenvector will therefore produce an estimate of local variance in exactly the same way that we produced figure 8.4. Other eigenvectors might be similar to oriented edge- and bar-detectors, in which case they would produce estimates of local *changes* in variance in different directions. It becomes difficult to conjecture possible functions for higher layers, but we see that this form of analysis may be a useful technique for understanding hierarchical processing of visual information.

Chapter 10

Image Coding

Although any given image contains an enormous amount of information, the data is not uncorrelated. The aim of image coding is to take advantage of the correlations in the image to find a representation which is considerably smaller yet which sacrifices a relatively small approximation error. One way to code an image is to break it up into small blocks and then to approximate each block in some manner. Typically, a set of filters will be chosen such that linear combinations of these filters can be used to approximate different image blocks. The mean-squared difference between the original and the reconstructed image is used as the error measure. If the image is to be transmitted, then only the coefficients of each filter need to be transmitted, since presumably both the transmitter and receiver can agree on a set of filters beforehand.

The data compression is achieved by choosing fewer filters than necessary to exactly represent any image block. The transmitted coefficients are quantized into a small number of bits for transmission. Compression is usually measured in units of bits per pixel, which is the total number of bits transmitted per image divided by the total number of pixels in the image. The error is usually measured as the

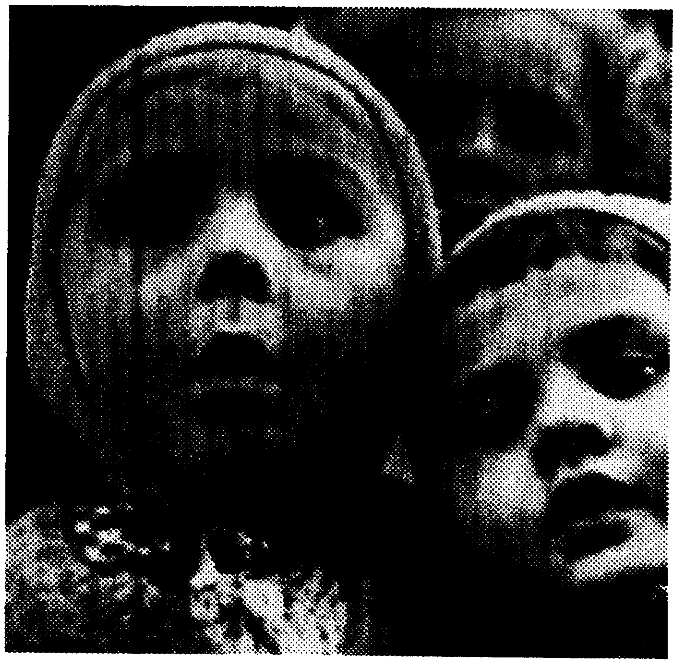


Figure 10.1: 256x256 pixel (8 bit) test image for coding. (A. Eisenstadt, "The Dragon is Slain")

signal to noise ratio

$$SNR = \frac{E[f(n_1,n_2)^2]}{E[(f(n_1,n_2) - \hat{f}(n_1,n_2))^2]}$$

where f is the original image data, and $\hat{f}$ is the reconstruction from the transmitted coefficients.

10.1 Results

Figure 10.1 shows an original image taken from part of an Eisenstadt photograph and digitized to form a 256x256 image with 256 greylevels. We use a single-layer linear neural network with 64 inputs and 8 outputs. 8x8 blocks of the image are used as training samples, with the image being scanned from left to right and top to bottom. The sample blocks do not overlap, and the image

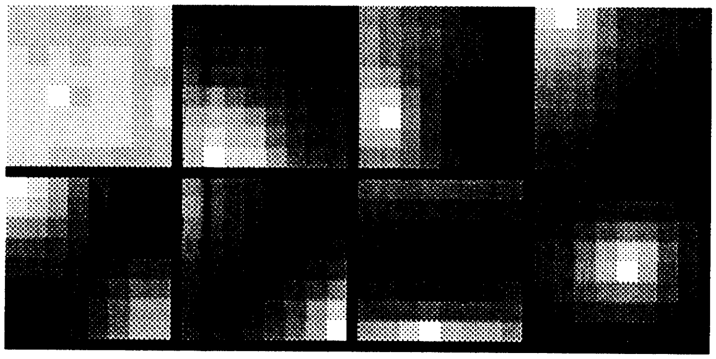


Figure 10.2: 8x8 masks learned by a network trained on fig. 10.1.

is scanned twice to allow time for the network to converge. The weights which the network learns are represented as 8x8 masks shown in figure 10.2. Each mask shows the set of weights associated with a single output. White indicates positive weights, black indicates negative, and grey indicates zero. In our notation, the masks are the rows c_i of the 8x64 weight matrix C after it has converged.

To code the image, each 8x8 block of the image is multiplied by each of the eight masks to generate eight coefficients for coding. The coefficients for the block starting at position n, m in the image I are thus given by

$$\nu_i^{n,m} = \sum_{p=1}^{8} \sum_{q=1}^{8} c_{i,p+8q} I_{n+p,m+q}$$

The coefficients $\nu_i^{n,m}$ are then uniformly quantized with a number of bits approximately proportional to the log of the variance of that coefficient over the image. This results in the first two masks being assigned five bits each, the third mask three bits, and the remaining masks two bits each. Therefore, each 8x8 block of pixels is coded using a total of 23 bits, so the data rate is 0.36 bits per pixel. To reconstruct the image from the quantized coefficients $\hat{\nu}_i^{n,m}$, each block of the image is re-formed by adding together all the masks weighted by their quantized

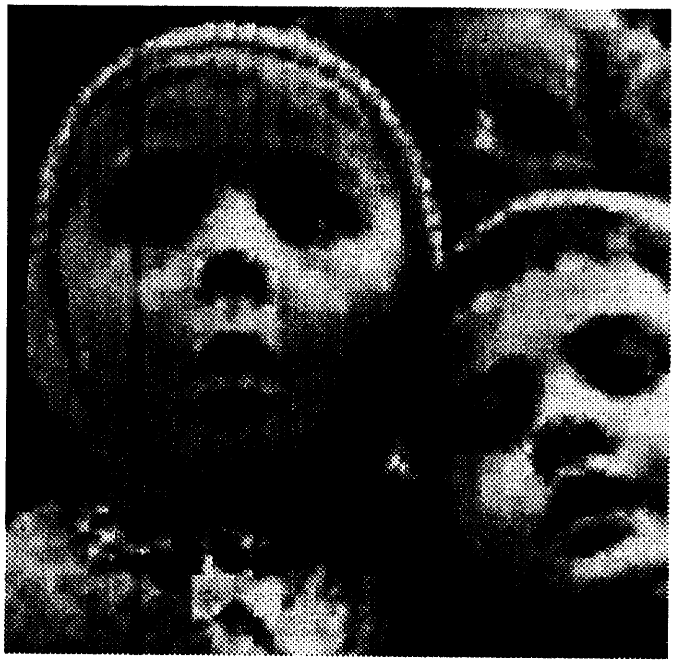


Figure 10.3: Image of fig. 10.1 coded at .36 bits per pixel.

coefficients:

$$\hat{I}_{n+p,m+q} = \sum_{i=1}^{8} c_{i,p+8q} \hat{v}_i^{n,m}$$

The resulting image is shown in figure 10.3. We calculate the normalized mean square error as the ratio of the error variance to the data variance

$$NMSE = \frac{E[(I_{n,m} - \hat{I}_{n,m})^2]}{E[I_{n,m}^2]}$$

which is 0.043 for this image. Note that no post-processing was performed to remove the blocking effect, although it can be significantly reduced by smoothing.

The network has learned a linear coding for the input data which approximates the optimal KLT. The masks are the "eigenvectors" of the input image, and they represent most of the variance in the 8x8 blocks which were used for training. Because the network outputs have high variance, they convey much of the input information, and we only need to use a few outputs (eight, in this case) to estimate the input data. This is the meaning of data coding; we have reduced 64 eight-bit pixels (512 bits total) to eight coefficients quantized with from two to five bits (23 bits total). The network chose masks which allow only 23 bits to represent most of the information in the original 512 bits.

We might now ask whether this same set of masks would be useful on a different image. Figure 10.4 is an image of a dog, and figure 10.5 shows the image after it has been reconstructed from quantized coefficients derived from the set of masks in figure 10.2. Note that the network was never trained on the image of figure 10.4. In this case, the output of the first two masks was quantized using seven pixels each, the third with five pixels, the fourth with four pixels, and the remaining coefficients with three pixels each. This gives a total of 35 bits, or a bit rate of 0.55 bits per pixel. The NMSE is 0.023 here which is actually lower than the error for the image of figure 1, due to the increased number of bits used for coding. The fact that the same set of masks can be used to code two different pictures is an example of "generalization" of the network. Although the images are different, their statistics may be similar enough that their respective KLTs are similar. A network trained on either image will compute a set of masks which will be useful for the other. This generalization property is a direct consequence of the statistical similarity of the two images.

Filters similar to those given in figure 10.2 have been used for image coding by many authors. The Discrete Cosine Transform masks are qualitatively similar (see [Lim, 1988] for a description), and Daugman [1988] has performed image coding using two-dimensional Gabor filters which are similar to the masks which our network learned.

Cottrell et. al. (1987) used self-supervised backpropagation to perform image coding, with bit rates as low as 0.625 bits per pixel. They used a network with 64 inputs, eight hidden units, and 64 outputs. Their training samples were, as here, 8x8 blocks of the image, and the network was trained to approximate the input

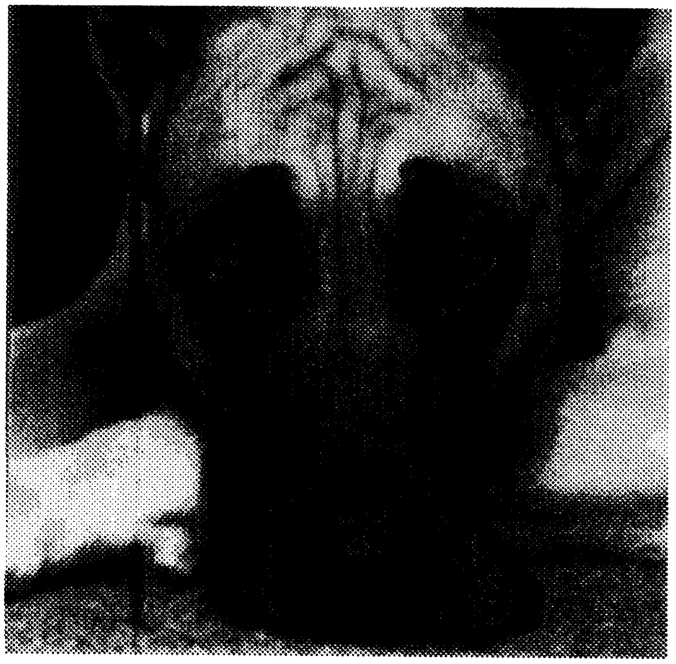


Figure 10.4: 256x256 pixel (8 bit) test image for coding.

data at the output units. As mentioned above, such a network will not actually find the KLT, but can be proven to converge to outputs which represent linear combinations of the KLT vectors [Baldi and Hornik, 1989]. Since the hidden units will all have approximately equal variance [Cottrell *et al.*, 1987, Baldi and Hornik, 1989], it is not possible to quantize them with different numbers of bits, as it was for the KLT. This fact significantly reduces the maximum bit rate which can be achieved. It should also be noted that Cottrell *et. al.* (1987) trained their network for 150,000 iterations, while the network which learned the masks of figure 10.2 was trained for 2048 iterations.

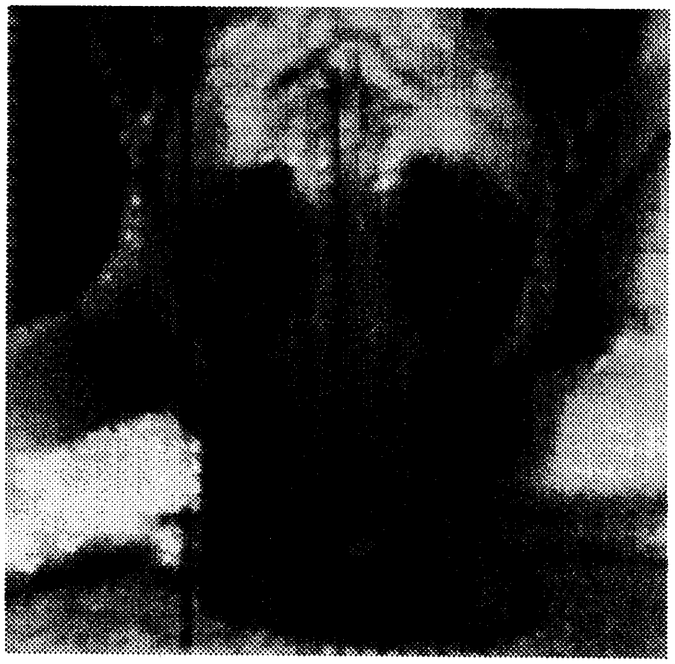


Figure 10.5: Image of fig. 10.4 coded at .55 bits per pixel using the same masks as in fig. 10.2.

10.2 Coding and the KLT

The coding technique we are using is equivalent to the block Karhunen-Loeve transform, a standard procedure in image coding [Lim, 1988]. The KLT is provably optimal for block coding in that it produces the minimum value of mean squared error. It has been found that for large classes of images, the eigenvectors determined by the KLT are not significantly different, and therefore the masks determined from one image can often be used to code a different image (with similar statistics) with acceptable quality.

The KLT for certain classes of image is similar to the Discrete Cosine Transform (DCT) which is a popular image coding technique. Since the DCT masks are the same for all images (unlike the KLT) and fast computation algorithms exist, the DCT is used more often than KLT-based algorithms for image coding.

Nevertheless, we have shown that an unsupervised learning algorithm can be used to solve an image coding problem with accuracy close to that of currently used algorithms. This fact provides strong justification for the study of unsupervised algorithms such as the Generalized Hebbian Algorithm, and shows that neural networks are capable of performing important practical tasks.

Chapter 11

Learning Stereopsis

In this chapter we present an example of how the Generalized Hebbian Algorithm can be used to perform a simple type of analysis of a pair of stereo images. The images we will use have been artificially chosen so that the algorithm performs well and the results are easy to understand. We do not mean this to be a generally useful stereo algorithm, but rather a preliminary demonstration of some of the abilities of the algorithm.

The analysis of stereo images usually involves matching points in the left and right images and computing the difference in position of a given point. This difference is called the "disparity". Many techniques exist to compute the disparity (for a partial review, see [Poggio and Poggio, 1984, Sanger, 1988c].) As will be shown below, the Generalized Hebbian Algorithm will train a properly constructed network to perform stereopsis based upon local bandlimited cross-correlation of the left and right images. Since cross-correlation is a linear operation, a single-layer linear network which used it would not be able to find a feature such as a depth edge, or a change in disparity. It could only give outputs proportional to the disparity itself. We therefore will use a two-layer nonlinear network to demonstrate that additional power can be gained from the nonlinearity. We will see that such a network will be able to respond to a depth edge in the artificial

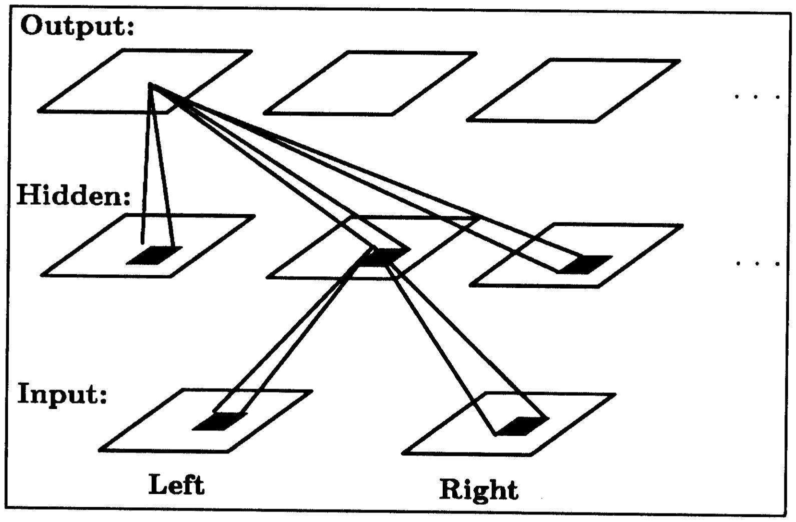


Figure 11.1: Structure of the stereo network. See text for a description.

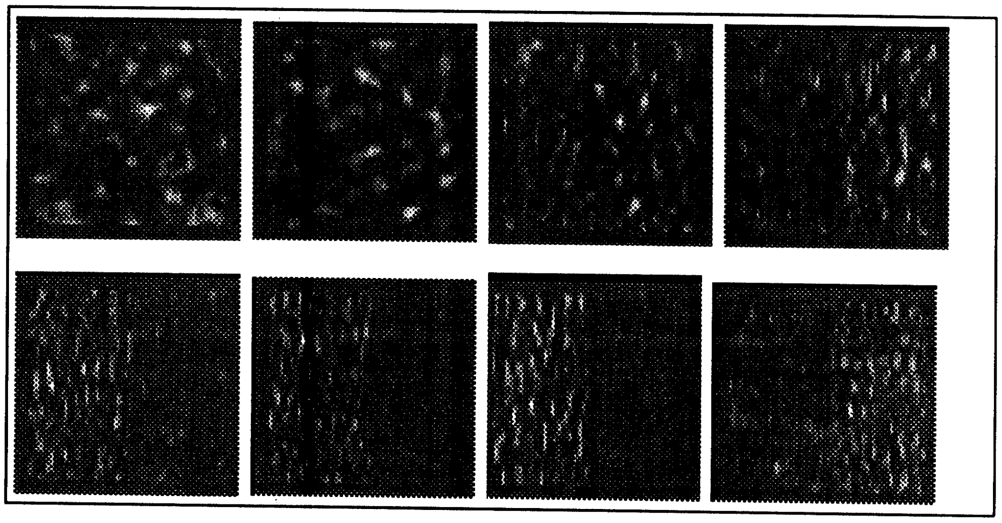


Figure 11.2: Hidden layer response for a two-layer nonlinear network trained on stereo images. The left half of the input random dot image has a 2 pixel disparity, and the right half has zero disparity.

test images we present.

11.1 Methods

We now show how the nonlinear Generalized Hebbian Algorithm can be used to train a two-layer network which can detect disparity edges. The network has 128 inputs, 8 types of unit in the hidden layer with a rectification nonlinearity, and 4 types of output unit. We think of this structure as representing a "hypercolumn" processing unit, so that for a 128x128 retina, there is a set of 8 hidden units over each point. Each of these hidden units responds to an 8x8 region centered at its position in both the left and right images. (See figure 11.1.) A type of unit is determined by its particular receptive field, so all hidden units of the same type will have identical shaped but partially overlapping receptive fields in the two images. Since there is one hidden unit of each type at each spatial location, there are 128x128=16384 hidden units of each type, and a total of 8x16384=131072 hidden units altogether. The output of each hidden unit is rectified. The output layer has four types of units, and again we assume that there is one of each type centered at every retinal position, giving a total of 128x128x4=65536 output units. Each unit can respond to a 16x16 region in *each* of the hidden unit planes, so there are a total of 8x16x16=2048 inputs to each output unit. Note that the output unit inputs span several adjacent "hypercolumns" in the hidden unit layer, and that adjacent output units of the same type have the same receptive fields and respond to overlapping regions in each hidden unit plane.

Since all units of a given type at a given layer have the same receptive field, only one is trained, and this mask is copied into the others. To compute the output at the hidden layer for a particular type of unit, its mask is convolved with the input image and the result is rectified, since convolution applies the mask at every spatial position. Similar processing is performed to compute the output

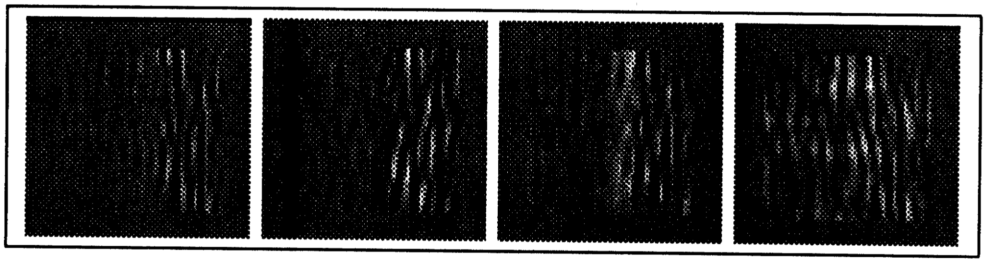


Figure 11.3: Output layer response for a two-layer nonlinear network trained on stereo images.

layer, although without the nonlinearity.

To train the network, a random-dot stereo pair was generated in which the left half had a disparity of two pixels, and the right half had zero disparity. The image was convolved with a vertically-oriented elliptical Gaussian mask to remove high-frequency vertical components. Corresponding 8x8 blocks of the left and right images (64 pixels from each image) were multiplied by a Gaussian window function and presented as input to the network, which was allowed to learn the first layer according to the unsupervised algorithm. After 4000 iterations, the first layer had converged to a set of 8 pairs of masks. Each of these 8 pairs represents the input to one type hidden unit. These masks are vertically-oriented "edge-detectors" (bandpass filters) with differing spatial frequency and phase sensitivity. Pairs of masks for the left and right images always have approximately the same shape, although they may have differing phase sensitivity.

These masks were convolved with the images (the left mask was convolved with the left image, and the right mask with the right image, and the two results were summed and rectified) to produce a pattern of activity at the hidden layer. Figure 11.2 shows this activity, and we can see that the last four masks are disparity-sensitive since they respond preferentially to either the 2 pixel disparity or the zero disparity.

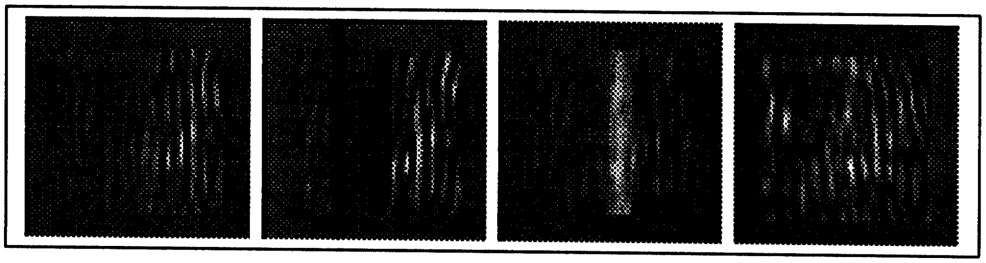


Figure 11.4: Output layer response averaged over ten different stereogram inputs with the same disparity.

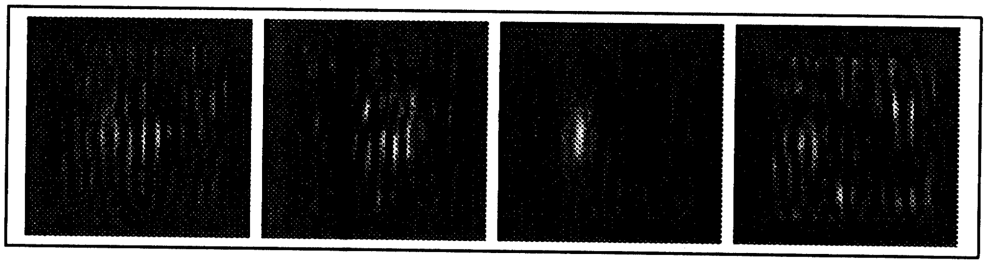


Figure 11.5: Output layer response averaged over ten stereograms with a central 2 pixel disparity square and zero disparity surround.

Since we were interested in disparity information, we trained the second layer using only the last four hidden unit types. The second layer had 1024 (=4x16x16) inputs organized as a 16x16 receptive field in each of the last four hidden unit "planes". The outputs had no nonlinearity. Training was performed by scanning over the hidden unit activity pattern (successive examples overlapped by 8 pixels) and 6000 iterations were used to produce the second-layer weights. The masks that were learned were then convolved with the hidden unit activity pattern to produce an output unit activity pattern, shown in figure 11.3.

The third output is sensitive to a change in disparity (a depth edge). If we generate ten different random-dot stereograms and average the output results, we see that the other outputs are also sensitive (on average) to disparity changes, but not as much as the third (figure 11.4). The averaged results of presenting 10 stereograms with a central 2 pixel disparity square against a zero disparity background are shown in figure 11.5. Note that the ability to detect disparity edges requires the rectification nonlinearity at the hidden layer, and that no purely linear function has this property.

11.2 How it did Stereo

The algorithm for stereo which the network has discovered can be understood in terms of simple linear operations on image pairs. Since the image is anisotropically filtered noise, the eigenvectors learned by the first layer will be oriented filters which respond in the direction of highest variance. They are therefore vertically oriented bandpass filters which respond optimally to the horizontal components in the input. The left and right masks for any hidden unit tend to converge to the same filter (perhaps with a phase shift). The response of the hidden unit is found by adding the responses of all the inputs, which means that the left and right mask outputs are added.

Adding the left and right mask outputs is equivalent to optically "interfering" the two convolution results. A rectification linearity is equivalent to squaring and taking the square root, and we know that squaring the sum of the left and right filters will yield terms which contain their cross correlation, in addition to the sum of the energies. We have thus performed a bandlimited cross-correlation, which is one method for computing stereo disparity. This procedure has effectively transformed the relative phase difference of bandlimited components of the left and right images into an amplitude change, and this amplitude is evident as the response of the rectified hidden units. (See [Sanger, 1988c] for a discussion of the use of bandlimited cross-correlation for stereopsis. See also [Daugman, 1988a] for a very closely related technique.) The fact that some of the hidden units have left and right masks with differing relative phase means that these units will respond to non-zero disparity better than to zero disparity. In other words, the "phase offset" determines the region of disparity sensitivity.

The amplitude information from the original image is also present in the hidden units, and this explains the "blotchy" appearance in figure 11.2. The hidden layer responses are a combination of the disparity and the intensity information. During training, hidden units maximize their variance by responding to either intensity information or disparity information or both. It is for this reason that we see some units which respond preferentially to one type of data or the other. Note that adjacent hidden units of one type are *not* uncorrelated, and that non-adjacent hidden units of differing types are not necessarily uncorrelated. Hidden units of different types at the same spatial position are uncorrelated, however, since the algorithm trained them this way.

After the first layer converged, the second layer was trained. The second layer was linear, so the masks which are learned will correspond to the eigenvectors of the spatial correlation function present at the hidden layer. Since only the last four hidden unit types were used, there was always a strong edge present in the center of the input, due to the disparity edge in the original image. Therefore, a horizontally-oriented edge detector at the second layer would be capable of responding well to this edge. Other second-level masks might respond better to the higher-frequency patterns generated from the image intensity information.

T. Poggio has pointed out that such a system could also be used for computing motion information. Disparity is equivalent to motion along a horizontal line (under epipolar assumptions on the cameras), so any system capable of measuring disparity should be able to measure motion of points between two frames in a sequence. We have not yet tried this, but it should yield interesting results.

11.3 Significance

The method for detecting disparities which was found by the first layer (through the hidden units) is a very simple algorithm equivalent to bandlimited cross-correlation (for a partial review of related algorithms, see [Sanger, 1988c]). The ability of the second layer to detect disparity edges is a straightforward hierarchical extension of this algorithm. The fact that a network could learn to perform this task does not reflect upon the innate computational power of the network. It does show that simple solutions to apparently hard problems can often be found using unsupervised algorithms. This case presents an important example of the type of information available from unsupervised algorithms. The network structure is not sufficient to produce outputs which compute the actual disparity or rate of change of disparity. There will always be incorporated pattern information as well, and the values produced will not be linearly related to the actual disparities. Nevertheless, the network has found a representation for the data which makes much of the important disparity information explicit. The construction of subsequent processing stages now seems considerably more straightforward than it might without this representation.

We should also note that the information at the hidden layer is present as spatial patterns of variance, rather than intensity. This conversion of disparity to variance is a typical result for the Generalized Hebbian Algorithm. The algorithm is attempting to maximize the variance, so it will do this by discovering filters which measure high-variance components of the input. In order to convert variance into intensity, we must rectify and spatial filter, which is what the second layer does. Note that without the rectification, it is not possible to detect disparity edges, since no linear function of the input could have this response. This therefore provides an example of the increase in computational power that can be gained from even a very simple type of nonlinearity in a two-layer network.

The biological relevance of this network structure is not clear. There is some evidence that inputs from the two eyes are summed during processing, but in general, most cells in the visual cortex of primates are not binocular, but rather respond preferentially to one eye or the other [Poggio and Poggio, 1984]. Probably the only conclusion we can draw is that there exist very simple algorithms which are capable of performing early stereo analysis.

Chapter 12

Conclusion

12.1 Contents of the Thesis

In this thesis, we proposed a new algorithm for unsupervised learning in multilayer feedforward neural networks, and applied the algorithm to

1. texture segmentation,
2. cortical development models,
3. image coding, and
4. stereopsis.

The algorithm finds the eigenvectors of the input correlation matrix, and thus is equivalent to performing a Karhunen-Loève expansion of the input. The eigenvectors are found using only samples from the input distribution, without the need to explicitly compute its correlation matrix. This gives the algorithm the ability to find the first few eigenvectors of large-dimensional inputs, a task which may be impractical using other techniques. The outputs are uncorrelated, and are ordered by decreasing variance. Training additional outputs does not require retraining previously learned weights. These are useful properties for a whitening filter, or any filter for which the signal to noise ratio must be maximized.

12.2 Other Work

There is much other work in the field of neural networks which relates to what we have shown here, and this work has been discussed in previous chapters. The Generalized Hebbian Algorithm is not the only way to derive the Karhunen-Loeve transform, and it may not be the most efficient. It represents a method by which a neural network can be made to perform this task. The Generalized Hebbian Algorithm specifies how the problem of finding the KLT can be broken down into local computations which can be performed by simple "neuron-like" units. However, we should point out that the existing algorithm known as Self-supervised Backpropagation is extremely similar and almost mathematically equivalent. The major difference between GHA and SSBP is that GHA finds the eigenvectors *in order* while SSBP finds a linear combination of the eigenvectors such that the outputs all have approximately equal variance. For many applications, this fact will not make any difference in selection of an algorithm, while for certain applications one or the other may be more useful.

Linsker [1988] and others predicted that some form of Hebbian learning algorithm must exist which can derive more than the principal eigenvector. The Generalized Hebbian Algorithm is just such an algorithm. It may not be the only such algorithm, since a winner-take-all strategy seems to give almost equivalent results in some cases [Barrow, 1987].

12.3 Problems

We have defined optimality in terms of the ability to linearly reconstruct the input vector given the outputs of the network. However, this criterion is not equivalent to maximizing information [Linsker, 1988b], and is not necessarily meaningful in many cases. The fact that the algorithm is unsupervised means that a layer of a large network trained using the Generalized Hebbian Algorithm is not necessarily optimized for whatever task the entire network is performing. In any particular case, a supervised algorithm such as Backpropagation may be able to perform better. A further problem is that the algorithm is numerically poor in the sense that errors in computation of eigenvectors cause increased errors in the computation of subsequent eigenvectors. If there is a large number of outputs, this means that the later ones may have weights which differ greatly from the desired eigenvectors.

The Generalized Hebbian Algorithm has been proven to converge only for linear networks. It is the author's belief that the algorithm is not guaranteed to converge in the nonlinear case, due to local minima in the energy surface. It has been impossible, so far, to actually derive the energy function which the algorithm is following, so a detailed analysis is lacking. The effects of nonzero means and different nonlinearities are not well understood. These are important problems with the analysis presented here, and should be solved if we are to have a rigorous understanding of the behavior of this algorithm. Empirically, it seems that the algorithm performs well for both linear and nonlinear networks. Theoretically, the nonlinear case is much more difficult to analyze, and many important questions remain.

12.4 Understanding Neural Networks

The relationship between the algorithm proposed here and the Singular Value Decomposition, Principal Components Transform, and Karhunen-Loeve Transform has been explained. The fact that these transforms are well studied in other domains gives a rigorous mathematical background from which we can extract a clear understanding of the function of unsupervised neural networks. This background, may also help us to understand the function of other algorithms (both supervised and unsupervised) for training networks.

12.5 Future Work

It is clear that much work remains to be done to determine the effect of nonlinearities on the learning algorithm. It is important to gain an understanding of their effect on the computational power of the network. Perhaps other nonlinearities might yield different methods of solving problems. The rectification nonlinearity which was demonstrated above is very simple, yet it was able to yield some useful behavior when applied to a stereo algorithm.

The theoretical relationship between the Generalized Hebbian Algorithm and the maximization of output information should be studied. We know that for linear networks and Gaussian inputs with certain types of noise, maximizing the output variance also maximizes the information. But in the nonlinear, non-Gaussian case, we do not understand whether these two ideas are related.

There are also many potential applications of unsupervised networks which have not been attempted. Tomaso Poggio has suggested that it would be interesting to have a network learn three-dimensional filters which are sensitive in time, as well as two spatial dimensions. They could converge to the eigenvectors of a spatio-temporal distribution, and would therefore function as motion detectors, or creation/annihilation detectors [Fahle and Poggio, 1981]. A nonlinear network might be able to find velocity-sensitive or directionally selective units with similar responses to those found in biological visual systems.

There are other potential applications outside the field of computer vision. Principal Components Analysis is an important technique for analyzing any statistical data, and we would therefore expect that neural nets which find the components would have applicability to many different problems. This thesis has presented just a few results and applications of a neural network training algorithm which should provide an important method for the analysis of real-world data.

Bibliography

Ackley D. H., Hinton G. E., Sejnowski T. J., 1985, A learning algorithm for boltzmann machines, *Cognitive Science*, 9:147–169.

Ahmed N., Natarjan T., Rao K. R., 1974, Discrete cosine transform, *IEEE Trans. Comput.*, C-23:90–93.

Amari S., 1983, Field theory of self-organizing neural nets, *IEEE SMC*, 13:741–748.

Andrews B. W., Pollen D. A., 1979, Relationship between spatial frequency selectivity and receptive field profile of simple cells, *J. Physiol.*, 287:163–176.

Baker C. L., Cynader M. S., 1986, Spatial receptive-field properties of direction-selective neurons in cat striate cortex, *J. Neurophysiology*, 55(6):1136–1152.

Baldi P., Hornik K., 1989, Neural networks and principal component analysis: Learning from examples without local minima, *Neural Networks*, 2(1):53–58.

Ballard D. H., 1987, Modular learning in neural networks, In *Proc. Sixth National Conference on AI (AAAI-87)*, volume 1, pages 279–284, Seattle, Wash.

Barrow H. G., 1987, Learning receptive fields, In *Proc. IEEE 1st Ann. Conference on Neural Networks*, volume 4, pages 115–121, San Diego, CA.

Bergen J. R., Adelson E. H., 1988, Early vision and texture perception, *Nature*, 333:363–364.

Bienenstock E. L., Cooper L. N., Munro P. W., 1982, Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex, *J. Neuroscience*, 2(1):32–48.

Bourlard H., Kamp Y., 1988, Auto-association by multilayer perceptrons and singular value decomposition, *Biological Cybernetics*, 59:291–294.

Brailovsky V., 1983a, On the problem of function approximation by sample set processing for an incompletely determined model, *Ann. NY Acad. Sci.*, 410:137–147.

Brailovsky V., 1983b, On the problem of function system selection for function approximation based on the use of a sample set with defects, *Ann. NY Acad. Sci.*, 410:149–161.

Brailovsky V. L., 1985, On an incompletely determined model for function approximation by experimental data, *Annals NY Acad. Sci.*, 452:316–333.

Brailovsky V. L., 1987, A predictive probabilistic estimate for selecting subsets of regressor variables, *Annals NY Acad. Sci.*, 491:233–244.

Carpenter G. A., Grossberg S., 1988, The ART of adaptive pattern recognition by a self-organizing neural network, *Computer*, 21:77–88.

Cavanagh P., 1984, Image transforms in the visual system, In Dodwell P. C., Caelli T. M., ed.s, *Figural Synthesis*, Lawrence Erlbaum Assoc., Hillsdale, NJ.

Cottrell G. W., Munro P., Zipser D., 1987, Learning internal representations from gray-scale images: An example of extensional programming, In *Proc. 9th Ann. Conf. of the Cognitive Science Society*, pages 461–473, Seattle, WA.

Daugman J. G., 1988a, Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression, *IEEE Trans. Acoustics, Speech, and Signal Processing*, 36(7):1169–1179.

Daugman J. G., 1988b, Pattern and motion vision without laplacian zero crossings, *J. Opt. Soc. Am. A*, 5(7):1142–1148.

Enroth-Cugell C., Robson J. B., 1966, The contrast sensitivity of retinal ganglion cells of the cat, *J. Physiol.*, 187:517–552.

Fahle M., Poggio T., 1981, Visual hyperacuity: Spatiotemporal interpolation in human vision, *Proc. R. Soc. Lond. B*, 213:451–477.

Foster K. H., Gaska J. P., Nagler M., Pollen D. A., 1985, Spatial and temporal frequency selectivity of neurones in visual cortical areas V1 and V2 of the macaque monkey, *J. Physiol.*, 365:331–363.

Fukushima K., 1975, Cognitron: A self-organizing multilayered neural network, *Biological Cybernetics*, 20:121–136.

Gaska J. P., Pollen D. A., Cavanagh P., 1987, Diversity of complex cell responses to even- and odd-symmetric luminance profiles in the visual cortex of the cat, *Exp. Brain Res.*, 68:249–259.

Grossberg S., 1971, Pavlovian pattern learning by nonlinear neural networks, *Proc. Natl. Acad. Sci.*, 68:828–831.

Grossberg S., 1976, On the development of feature detectors in the visual cortex with applications to learning and reaction-diffusion systems, *Biological Cybernetics*, 21:145–159.

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

Hinton G. E., 1987, Connectionist learning procedures, CMU Tech. Report CS-87-115.

Hubel D. H., Wiesel T. N., 1962, Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex, *J. Physiol.*, 160:106–154.

Hubel D., Wiesel T., 1974, Orientation columns in monkey striate cortex, *J. Comp. Neuro.*, 158:267–291.

Hunter I. W., Korenberg M. J., 1986, The identification of nonlinear biological systems: Wiener and Hammerstein cascade models, *Biological Cybernetics*, 55:135–144.

Judd J. S., 1987, Complexity of connectionist learning with various node functions, COINS Tech. Report 87-60.

Julesz B., 1971, *Foundations of Cyclopean Perception*, U. Chicago P., Chicago.

Kammen D. M., Yuille A. L., 1988, Spontaneous symmetry-breaking energy functions and the emergence of orientation selective cortical cells, *Biological Cybernetics*, 59:23–31.

Karhunen J., Oja E., 1982, New methods for stochastic approximation of truncated Karhunen-Loève expansions, In *Proc. 6th Int. Conf. on Pattern Recognition*, pages 550–553.

Karhunen J., 1984, *Recursive Estimation of Eigenvectors of Correlation Type Matrices for Signal Processing Applications*, PhD thesis, Helsinki Univ. Tech., Espoo, Finland.

Karhunen J., 1985, Simple gradient type algorithms for data-adaptive eigenvector estimation, Technical Report TKK-F-A584, Helsinki Univ. Tech.

Kazakos D., 1983, Optimal constrained representation and filtering of signals, *Signal Processing*, 5:347–353.

Kohonen T., 1982, Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, 43:59–69.

Kohonen T., 1988, The "neural" phonetic typewriter, *Computer*, pages 11–22.

Kreyszig E., 1988, *Advanced Engineering Mathematics*, Wiley, NY.

Kushner H. J., Clark D. S., 1978, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, Springer-Verlag, New York.

Kuusela M., Oja E., 1982, The averaged learning subspace method for spectral pattern recognition, In *Proc. 6th Int. Conf. on Pattern Recognition*, pages 134–137.

Lapedes A., Farber R., 1987, Nonlinear signal processing using neural networks, Los Alamos National Laboratory LA-UR-87-2662, Submitted to Proc. IEEE.

Lapedes A., Farber R., 1988, How neural nets work, In Anderson D. Z., ed., *Neural Information Processing Systems*, pages 442–456, Am. Inst. Physics, NY, Proceedings of the Denver, 1987 Conference.

Le Cun Y., 1985, A learning scheme for asymmetric threshold networks, In *Proceedings of Cognitiva*, volume 85, pages 599–604, Paris, France.

Lim J. S., 1988, *Two-Dimensional Signal and Image Processing*, Prentice Hall, in press.

Linsker R., 1986a, From basic network principles to neural architecture, *Proc. Natl. Acad. Sci. USA*, 83:7508–7512.

Linsker R., 1986b, From basic network principles to neural architecture, *Proc. Natl. Acad. Sci. USA*, 83:8390–8394.

Linsker R., 1986c, From basic network principles to neural architecture, *Proc. Natl. Acad. Sci. USA*, 83:8779–8783.

Linsker R., 1988a, Implementing the principle of maximum information preservation: Local algorithms for biological and synthetic networks, abstract submitted to the IEEE conf. on Neural Information Processing Systems, Denver CO.

Linsker R., 1988b, Self-organization in a perceptual network, *Computer*, 21(3):105–117.

Lippmann R. P., 1987, An introduction to computing with neural nets, *IEEE ASSP Magazine*, pages 4–22.

Ljung L., 1977, Analysis of recursive stochastic algorithms, *IEEE Trans. Automatic Control*, AC-22(4):551–575.

Maffei L., Fiorentini A., 1973, The visual cortex as a spatial frequency analyzer, *Vision Research*, 13:1255–1267.

Marcelja S., 1980, Mathematical description of the responses of simple cortical cells, *J. Opt. Soc. Am.*, 70:1297–1300.

Marr D., 1982, *Vision*, W.H. Freeman and Co., San Francisco.

Minsky M. L., Papert S., 1969, *Perceptrons*, MIT Press, Cambridge, MA.

Nothdurft H. C., Li C. Y., 1985, Texture discrimination: Representation of orientation and luminance differences in cells of the cat striate cortex, *Vision Research*, 25(1):99–113.

Oja E., Karhunen J., 1980, Recursive construction of Karhunen-Loève expansions for pattern recognition purposes, In *Proc. 5th Int. Conf. on Pattern Recognition*, pages 1215–1218.

Oja E., Karhunen J., 1985, On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix, *J. Math. Analysis and Appl.*, 106:69–84.

Oja E., 1982, A simplified neuron model as a principal component analyzer, *J. Math. Biology*, 15:267–273.

Oja E., 1983, *Subspace Methods of Pattern Recognition*, Research Studies Press, UK.

Parker D. B., 1985, Learning-logic, Tech. Report TR-47, Sloan School of Management, MIT.

Poggio G. F., Poggio T., 1984, The analysis of stereopsis, *Ann. Rev. Neurosci.*, 7:379–412.

Pollen D. A., Ronner S. F., 1981, Phase relationships between adjacent simple cells in the visual cortex, *Science*, 212:1409–1411.

Pollen D. A., Ronner S. F., 1983, Visual cortical neurons as localized spatial frequency filters, *IEEE Trans. Systems, Man, and Cybernetics*, 13:907–916.

Pollen D. A., Gaska J. P., Jacobson L. D., 1988, Responses of simple and complex cells to compound sine-wave gratings, *Vision Research*, 28(1):25–39.

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

Rumelhart D. E., Zipser D., 1985, Competitive learning, *Cognitive Science*, 9:75–112.

Rumelhart D. E., Hinton G. E., Williams R. J., 1986a, Learning internal representations by error propagation, In *Parallel Distributed Processing*, chapter 8, pages 318–362, MIT Press, Cambridge, MA.

Rumelhart D. E., Hinton G. E., Williams R. J., 1986b, Learning representations by back-propagating errors, *Nature*, 323(9):533–536.

Sanger T. D., 1988a, Optimal unsupervised learning, *Neural Networks*, 1(S1):127, Proc. 1st Ann. INNS meeting, Boston, MA.

Sanger T. D., 1988b, Optimal unsupervised learning in a single-layer linear feedforward neural network, submitted to *Neural Networks*.

Sanger T. D., 1988c, Stereo disparity computation using gabor filters, *Biological Cybernetics*, 59:405–418.

Sanger T. D., 1989, An optimality principle for unsupervised learning, In *Advances in Neural Network Information Processing Systems*, Morgan Kaufmann, Proc. IEEE Neural Information Processing Systems conf., Denver CO.

Saund E., 1987, Dimensionality-reduction using connectionist networks, MIT AI Memo 941.

Shapley R., Lennie P., 1985, Spatial frequency analysis in the visual system, *Ann. Rev. Neurosci.*, 8:547–583.

Spitzer H., Hochstein D., 1985, Simple and complex-cell response dependencies on stimulation parameters, *J. Neurophysiology*, 53:1244–1265.

Stone G. O., 1986, An analysis of the delta rule and the learning of statistical associations, In Rumelhart D. E., Hinton G. E., Williams R. J., ed.s, *Parallel Distributed Processing*, chapter 11, pages 444–459, MIT Press, Cambridge, MA.

Turner M. R., 1986, Texture discrimination by Gabor functions, *Biological Cybernetics*, 55:71–82.

Von der Malsburg C., 1973, Self-organization of orientation sensitive cells in striate cortex, *Kybernetik*, 54:85–100.

Voorhees H., Poggio T., 1988, Computing texture boundaries from images, *Nature*, 333:364–367.

Voorhees H., 1978, Finding texture boundaries in images, Technical Report 968, MIT AI Lab.

Watanabe S., 1965, Karhunen-Loeve expansion and factor analysis: Theoretical remarks and applications, *Transactions of the 4th Prague Conference on Information Theory*, pages 635–660.

Werbos P. J., 1974, Beyond regression: New tools for prediction and analysis in the behavioral sciences, Ph.D. Thesis, Harvard Univ.

Widrow B., Hoff M. E., 1960, Adaptive switching circuits, In *IRE WESCON Conv. Record, Part 4*, pages 96–104.

Widrow B., Walach E., 1984, On the statistical efficiency of the LMS algorithm with nonstationary inputs, *IEEE Trans. Info. Theory*, IT-30(2):211–221.

Widrow B., McCool J. M., Larimore M. G., Johnson C. R., 1976, Stationary and nonstationary learning characteristics of the LMS adaptive filter, *Proc. IEEE*, 64(8):1151–1162.

Wilson H. R., 1988, Development of spatiotemporal mechanisms in infant vision, *Vision Research*, 28(5):611–628.

Yuille A. L., Kammen D. M., Cohen D., 1988, Quadrature and the development of orientation selective cortical cells by Hebb rules, Submitted to *Biological Cybernetics*.