Negative Feedback as an Organising Principle for Artificial Neural Networks

By

Colin Fyfe

A Thesis submitted in fulfilment
of the requirements for the degree
of Doctor of Philosophy

University of Strathclyde, Glasgow
Department of Computer Science March, 1995

Acknowledgements: I would like to gratefully acknowledge the help and support of a great number of people.

First, I must record the help I received from all (past and present) members of the IKBS group of the Computing Science Department at the University of Strathclyde. All have shown a tolerance of my stumbling steps that has made my time with the group most enjoyable. I must especially thank Craig Renfrew who helped me enormously in the initial stages of this research and pointed my interest originally towards unsupervised learning.

Secondly, I must acknowledge the help of the group of Dr. E. Rolls in the Experimental Psychology Department of the University of Oxford. My knowledge of "wet" neural networks increased hugely during my brief stay at Oxford. I must especially express my gratitude to Dr. Roland Baddeley who quickened my interest in statistical aspects of Artificial Neural Networks.

I have an enormous debt to Professor Douglas McGregor for many aspects of the last 3 years: Douglas provides both a test-bed and a sounding board for half-formed ideas. His incisive comments are always valuable and his good humour legendery. His humility and honesty provides a model for all new researchers and is a living expression of "nullius in verba".

Finally, I must thank my wife, Mary Teresa, for supporting my efforts during this time; without that support, this thesis would never have been completed.

Abstract

We investigate the properties of an unsupervised neural network which uses simple Hebbian learning and negative feedback of activation in order to self-organise. The negative feedback circumvents the well-known difficulty of positive feedback in Hebbian learning systems which causes the networks' weights to increase without bound. We show, both analytically and experimentally, that not only do the weights of networks with this architecture converge, they do so to values which give the networks important information processing properties: linear versions of the model are shown to perform a Principal Component Analysis of the input data while a non-linear version is shown to be capable of Exploratory Projection Pursuit.

While there is no claim that the networks described herein represent the complexity found in biological networks, we believe that the networks investigated are not incompatible with known neurobiology. However, the main thrust of the thesis is a mathematical analysis of the emergent properties of the network; such analysis is backed by empirical evidence at all times.

Contents

Chapter 1

Introduction

The purpose of this thesis is the identification and analysis of means of extraction of information from high dimensional data sets. We will later define "information" in a Shannon-rigorous way but, for now, it is sufficient to take "information" to mean any increase in our (human) knowledge.

The central idea of the thesis is that negative feedback of activation can be used to control the development of very simple self-organising artificial neural networks which nevertheless have powerful information-extraction properties.

We will use simple Hebbian learning in our networks. There is a well known difficulty with this type of learning which is that the network weights will tend to increase without bound unless we perform some type of specific remedial action. However, we will show that the negative feedback of activation not only causes the weights to converge, it also causes it to converge to directions which have powerful information processing properties.

Since we insist that the retention of simplicity is a major design criterion, we cannot claim that such networks will be accurate models of biological information processors. Yet we will base our models on biologically-plausible premises wherever possible.

The aim of creating intelligent machines is not new: one of the first attempts to mimic human capabilities was the study of cybernetics.

1.1 The Influence of Cybernetics

We begin by relating this thesis to the topic of cybernetics, a term which has become unfashionable in recent years. The word was coined by the Austrian mathematician and engineer Norbert Weiner (Wiener, 1948) who used it in defining a science of communication and control in both animals and machines.

The defining feature of cybernetics is feedback which was thought to be the principal organising principle of all complex systems: hence, it came to be used as a basic methodology of systems theory and management science. The study has often become tinged with an anti-reductionist slant because of its emphasis on the emergent properties of complex systems and it is perhaps this which has contributed most of all to its recent neglect.

The application of cybernetic principles as a paradigm of neural network development will be used in this thesis:

- Its theme is that negative feedback of activation may be used as an organising principle of neural network development.
- The neural networks' development will be environment driven.
- The networks will use unsupervised learning to self-organise.
- Several properties of the networks discussed will appear as emergent properties which were not predicted *a priori*.
- The interaction of parts of the networks, simple though they are, will be important in the final properties of the networks.
- At any one time we will be able to identify the state of the system and can, if we wish, inspect all component states of the system at that time.
- and, most important of all, the networks will be quintessentially involved in information processing.

1.2 Artificial Neural Networks

Artificial Neural Networks, on the other hand, are very much a currently-active research topic. The history of its rise (1950s ,60s), sudden decline (1969) and gradual re-emergence (1980s) has been detailed elsewhere (e.g. (Hertz et al., 1992)) and gives a fascinating insight into the Sociology of Science but will not be repeated here.

This thesis deals with a class of nets which self-organise using unsupervised Hebbian learning. We will use (mostly) 2 layer nets in which one layer is designated the input layer. They will be feedforward nets in the sense that they will be strongly directional, though we will use a feedback transfer of activation to stabilise the growth of the network. We will concentrate on a static analysis of the networks and rarely consider the dynamics of how the network converges to a particular state. The nets will initially be linear in Chapters 2 to 5 and subsequently non-linear. We will base our nets on biologically possible models but will sacrifice biological plausibility where necessary in order to investigate statistical properties of the network.

1.2.1 Principles for Self-organisation in Neural Networks

We will treat the need for organisation in information processors as axiomatic: a network with random weights is unlikely to have important information processing properties. We will create algorithms which cause the weights to become organised in such a way that the network develops information processing properties such as the transmission of maximal information in noise-free environments.

Discussion of the process of organisation in neural nets must begin with a statement of what it is that is to be organised. For example, we may organise the actual structure of the network by adding new nodes as necessary. This is the methodology used in Cascade Correlation (see e.g. (Fahlman and Lebiere, 1991; Shultz and Schmidt, 1991)), in Adaptive Logic Nets (e.g. (Armstrong et al., 1991; Dwelly, 1990)) and in similar methods based on other types of nets such as Kohonen nets (see below) (e.g. (Fritzke, 1991; Fritzke, 1993b)) or Principal Component nets (e.g. (Rubner and Tavan, 1989; Rubner and Schulten, 1990)). An alternative is to prune network links which seem to become redundant during learning (e.g. (Frean, 1990; McClelland

et al., 1986)). Occasionally this is taken to the extreme case where sets of nodes (layers) are brought into play at one time - this is usually disguised as having some layers reacting passively, if at all, while other layers are learning (see e.g. (Linsker, 1986a)). The view taken in this thesis is that the introduction and pruning of nodes can be subsumed in a process which simply organises the weights through which neurons pass their activations to one another: a new neuron is one which is joined to all other nodes (and the external environment) by means of connections with weights zero until such time that the neuron begins learning (at which time the weights will become non-zero). Therefore we will be interested in that form of organisation of artificial neural networks which during the learning phase changes the weights between neurons. This section is devoted to a discussion of the principles on which such organisation takes place.

Probably the most frequently used type of Artificial Neural Network is that which uses backpropagation; since the principles on which the development of the network weights is founded is the same as that on which the perceptron's weights are adjusted, such a network is often called a Multi-Layered Perceptron. Such networks require a teacher as well as examples from the environment of the mapping to be learned; they are thus grouped under the genre "supervised learning". They organise on the basis that the network's weights should be adjusted to make the network's output more like the teacher's output than it was prior to the adjustment. This reflects the principle that there exists a teacher who has expert knowledge of the environment and will use that knowledge to guide the development of the network. Such a principle would be interpreted in educational circles as advocating a didactic methodology. A special case of the didactic principle is that known as "reinforcement learning" (e.g. (Thrun, 1992; Barto et al., 1991; Barto et al., 1989)) in which the teacher merely tells the pupil that it is right or wrong. This has been shown to be particularly effective in control technology.

In contrast to a didactic methodology, one might propose an exploratory principle. While educationists might baulk at the description of unsupervised learning as an instance of exploratory methods, we will group methods which do not include a teacher as unsupervised learning. An alternative name might be environment-driven

learning.

Since there is no teacher, such nets must use information in the training examples on which to base any changes to the current weights i.e. on which to base their learning. However, as we shall see, this does not in itself constitute an "organising principle": we will show in the next chapter that the simplest of all network learning methods will cause the weights to grow without bound unless some organising principle is invoked to control their growth. Some examples of organising principles in the development of unsupervised neural networks are found in

1. Attractor Neural Networks (Hopfield Networks)(Hopfield, 1982; Amit, 1989) in which, in general, all nodes are thought of as belonging to a single layer. An input pattern is represented by a set of activations (typically 1s or -1s) across the set of nodes. The organising principle used for this net is that nodes which fire together for a particular pattern have the weights between them reinforced. The net effect of this organising principle is that if part of a pattern or a noisy version of the pattern is presented to the trained network, activation will be passed back and forth across the network before finally settling to an attractor state - hopefully that corresponding to that which has been learned. Note that we are making a distinction here between the organising principle - reinforcing weights between concurrently active nodes (one-shot learning) - and the aim of the process - pattern completion.

2. ART (Adaptive Resonance Theory)(e.g. (Carpenter and Grossberg, 1987b; Carpenter and Grossberg, 1987a)) Networks which have the aim of creating a network which retains its ability to learn from new data while not losing its memory of previously learned data. This is Grossberg's stability-plasticity dilemma. Here the organising principle is the "resonance" of a new input with those currently learned; resonance takes place if the new input is sufficiently like previously learned inputs. If resonance with a node's previous learning takes place, the node adjusts its learned weights to more closely match the new input while if resonance does not take place, a new node must be created.

3. (Kohonen) Feature Maps(e.g. (Martinetz, 1993; Fritzke, 1991; Fritzke, 1993a))

which aim to provide a low dimensional representation of the input data which remains topographically true to the major features of the input data. In this type of network, the organising principle is adjustment of weights so that neighbouring neurons respond similarly to any input. Now we can make a 3-way distinction between the aim, the organising principle and the implementation since the implementation may be done through lateral connections (Willshaw and von der Malsburg, 1976) or simple updating of a winner's neighbours (Kohonen, 1984).

We will use negative feedback of activation as the organising principle.

1.3 About this Thesis

Three aspects of the thesis must be made clear from the start:

1. Throughout this thesis, we will repeatedly return to the biological plausibility as a reason for investigating certain networks. This is not to be taken as implying that our networks have reached the level of sophistication of biology nor that we consider the thesis to be an exploration of carbon-based neural networks. We do however believe that, since our motivation lies in emulating biological neural networks' properties, we should not gainsay nature's technologies without due cause. Therefore the networks found herein are more aptly described as not being biologically implausible rather than the more positive assertion that they are biologically plausible.

2. We have stated that this is not a biological investigation. It is in fact a mathematical/statistical investigation; the structures and properties investigated in this thesis have an abstract existence independent of any implementation details. In almost all cases, the simulations on which we report are extremely simple simulations in which our aim is to illustrate a point rather than to demonstrate an implementation.

3. Where possible, we have related the emergent properties of our network to the methods of traditional statistics. We make no claims about the network's

efficiency with respect to these methods; indeed, in the case of Principal Component Analysing networks, there is a widespread acknowledgement that traditional statistical methods are more efficient than neural network methods. Our purpose is to investigate the properties of the negative feedback network *per se*; we will not compare the implementation of these properties with the standard methods.

Chapters 2,3,4 and 5 deal with Networks which perform Principal Component Analysis: Chapter 2 gives a short discussion of Hebbian learning and Principal Component Analysis, and a brief review of currently-popular PCA networks; the importance of Anti-Hebbian learning and negative feedback are discussed for the first time. Chapter 3 introduces the negative feedback network and shows its equivalence to current Principal Subspace networks; the PCA properties of the network are analytically developed and an algorithm which finds the actual Principal Components is devised; empirical results are given to complement the analysis; finally the feedforward weights are dissociated from the feedback weights to give a more biologically plausible network. Chapter 4 extends the basic network by allowing the negative feedback to influence those neurons giving the feedback; it is shown that this alone is not sufficient to cause convergence to the actual Principal Components but ways of causing this convergence are investigated. The importance of asymmetry in causing convergence to the Principal Components is highlighted. Chapter 5 gives 2 variations of the network based on different biologically-plausible features of either the input data or the means of transmission of the activation; for the first time a non-linearity is introduced and its effects analysed and then used in a particular application.

Chapters 6 deals with networks which introduce non-linearity into the networks of the previous chapters to create networks which perform exploratory data analysis. A general outline of the network is given along with specific examples of its use; different projection indices are used on various sets of input data and the effect of using different indices simultaneously is investigated. We conclude by giving examples of the networks which are discussed therein being used in biologically necessary operations viz. in vision processing.

Chapter 7 provides a summary of the thesis and suggests directions for future research.

It will be noted that there is a marked disparity between the space devoted to linear networks and that devoted to non-linear networks; we feel that this is essential in the light of our current lack of understanding of artificial neural networks' properties. There are many properties of many of our most popularly used networks which are beyond current analysis and the emphasis on linearity in the current thesis has, we feel, been justified by the insights gained through this emphasis. This is not to deny that the insights gained from our investigations of the linear networks may be useful in our investigations of the non-linear networks.

Chapter 2

Principal Components and ANNs

In this chapter, we define the Information Theory background to Principal Component Analysis(PCA) and give a brief survey of the major most-popular Artificial Neural Networks(ANNs) which perform a PCA. We will not, in this chapter, provide proofs of convergence of the various nets discussed since such proofs are very similar to those we use in Chapter 3 to prove convergence of our new network. We begin by outlining the simplest possible ANNs and review a very simple unsupervised learning rule.

2.1 Hebbian Learning

The aim of unsupervised learning is to present a neural net with raw data and allow the net to make its own representation of the data - hopefully retaining all information which we humans find important. Unsupervised learning in neural nets is generally realised by using a form of Hebbian learning which is based on a proposal by Donald Hebb (Hebb, 1949) who wrote:

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Neural nets which use Hebbian learning are characterised by making the activation of a unit depend on the sum of the weighted activations which feed into the unit. They use a learning rule for these weights which depends on the strength of the simultaneous

activation of the sending and receiving neuron. With respect to the network depicted in Figure 1, these conditions are usually written as

$$y_i = \sum_j w_{ij}x_j \tag{1}$$

$$\text{and } \Delta w_{ij} = \alpha x_j y_i \tag{2}$$

the latter being the learning mechanism. Here y_i is the output from neuron i, x_j is the j^{th} input, and w_{ij} is the weight from x_j to y_i. α is known as the learning rate and is usually a small scalar which may change with time. Note that the learning mechanism says that if x_j and y_i fire simultaneously, then the weight of the connection between them will be strengthened in proportion to their strengths of firing. However, we will not, unlike Kosko (Kosko, 1991), rename the Hebb Learning rule when an activation function is used. i.e. when

$$y_i = g(\sum_j w_{ij}x_j) \tag{3}$$

$$\text{and } \Delta w_{ij} = \alpha x_j y_i \tag{4}$$

for some function, $g()$, we will still call this Hebb learning.

Substituting Equation (1) into Equation (2), we can write the Hebb learning rule as

$$\begin{aligned} \Delta w_{ij} &= \alpha x_i \sum_k w_{kj}x_k \\ &= \alpha \sum_k w_{kj}x_k x_i \end{aligned} \tag{5}$$

$$\text{which is equivalent to } \frac{d}{dt}\,\mathbf{W}(t) \propto \mathbf{C}\mathbf{W}(t) \tag{6}$$

where C_{ij} is the correlation coefficient calculated over all input patterns between the i^{th} and j^{th} terms of the inputs and $\mathbf{W}(t)$ is the matrix of weights at time t. In moving from the stochastic equation (5) to the averaged differential equation (6), we must place certain constraints on the process particularly on the learning rate α which we will discuss in more detail later.

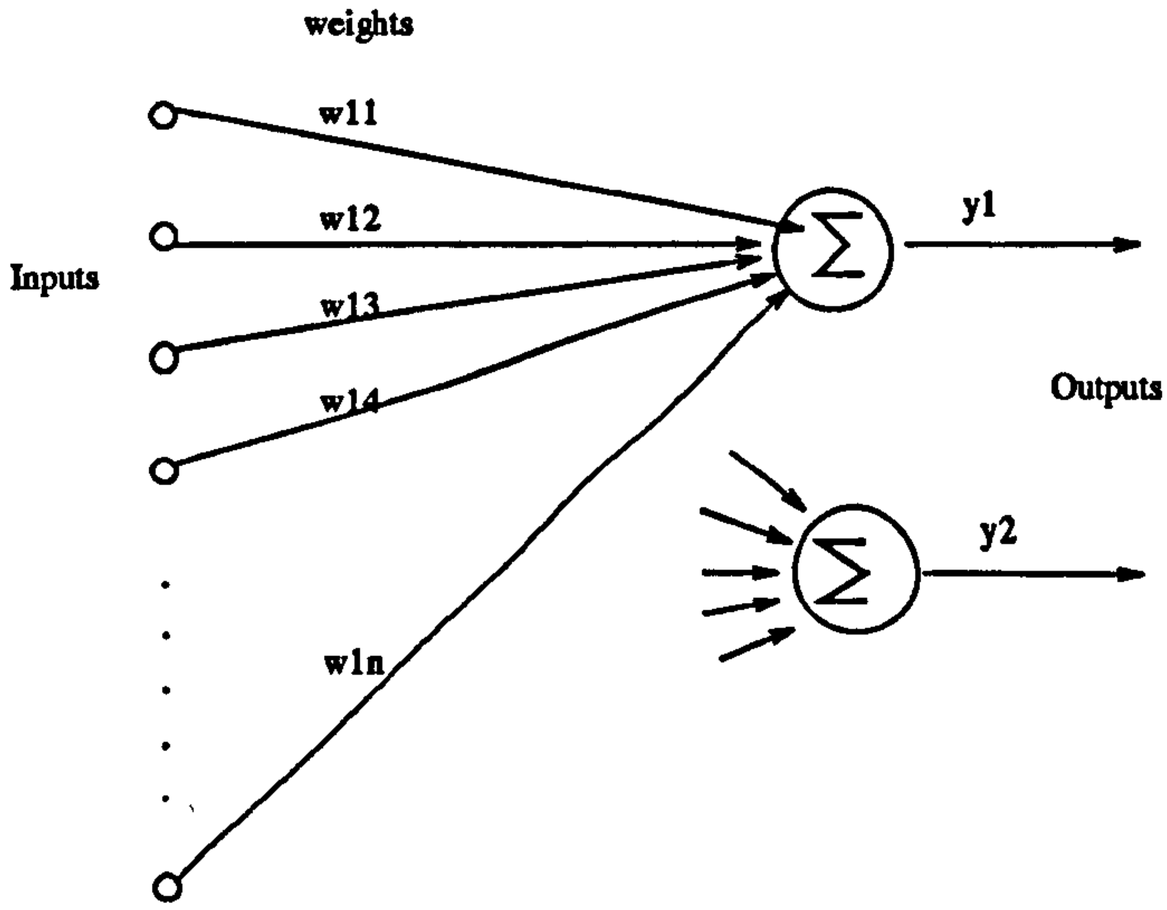


Figure 1: A Simple Neural Net
The inputs are the x_js and the outputs are the y_is . The strength of the connections between x's and y's are the w's. The learning rule changes the strengths of the w's till the net can be said to have learned a mapping.

The advantage of this formulation is that it emphasises the fact that the resulting weights depend on the second order statistical properties of the input data. A review of the importance of this aspect of the Hebb learning rule is given in Section 2.3.

Because of these statistics-based properties, Hebb learning has found applications in a number of early associative-type memories e.g. Steinbuch's Learning Matrix (Steinbuch, 1961), Anderson's linear associative memory (Anderson, 1968), Kohonen's Adaptive Associative Memory (Kohonen, 1974) and the Willshaw Model (Willshaw et al., 1969).

However, a major difficulty with this learning rule is that unless there is some limit on the growth of the weights, the weights tend to grow without bound: we have a positive feedback loop - a large weight will produce a large value of y (Equation 1) which will produce a large increase in the weight (Equation 2). It is instructive to follow e.g.(Hertz et al., 1992), in examining the Hebb rule's stability:

Recall first that a matrix **A** has an eigenvector **x** with a corresponding eigenvalue λ if

$$\mathbf{Ax} = \lambda\mathbf{x}$$

In other words, multiplying the vector x or any of its multiples by A is equivalent to multiplying the whole vector by a scalar λ. Thus the direction of x is unchanged - only its magnitude is affected.

Consider a one output-neuron network and assume that the Hebb learning process does cause convergence to a stable direction, w*; then if w_k is the weight vector linking x_k to y,

$$0 = \langle \Delta w_i^* \rangle = \langle y x_i \rangle = \langle \sum_j w_j x_j x_i \rangle = \sum_j R_{ij} w_j^*$$

where the angled brackets indicate the expected value taken over the distribution and R is the correlation matrix of the distribution. Now this happens for all i , so $\mathbf{Rw} = 0$. Now the correlation matrix, R, is a symmetric, positive semi-definite matrix and so all its eigenvalues are non-negative. But the above formulation shows that $\mathbf{w}^*$ must have eigenvalue 0. Now consider a small disturbance, ϵ, in the weights in a direction with a non-zero (i.e. positive) eigenvalue. Then

$$\langle \Delta w* \rangle = R(\mathbf{w}^* + \epsilon) = R\epsilon > 0$$

i.e. the weights will grow in any direction with non-zero eigenvalue (and such directions must exist). Thus there exists a fixed point at W=0 but this is an unstable fixed point. In fact, it is well known that in time, the weight direction of nets which use simple Hebbian learning tend to be dominated by the direction corresponding to the largest eigenvalue.

We will later discuss one of the major ways of limiting this growth of weights while using Hebbian learning and review its important side effects. However, we begin with short reviews of 2 subjects which will be important to the thesis: Information Theory and Principal Component Analysis.

2.2 Quantification of Information

Shannon (Shannon, 1948) devised a measure of the information content of an event in terms of the probability of the event happening. He wished to parameterise the intuitive concept that the occurrance of an unlikely event tells you more than that of

a likely event. He defined the information in an event i, to be $-\log p_i$ where p_i is the probability that the event labelled i occurs.

Using this, we define the entropy (or uncertainty or information content) of a set of N events to be

$$H = -\sum_{i=1}^{N} p_i \log p_i$$

That is, the entropy is the information we would expect to get from one event happening where this expectation is taken over the ensemble of possible outcomes.

For a pair of random variables X and Y, if $p(i,j)$ is the joint probability of X taking on the i^{th} value and Y taking on the j^{th} value, we define the entropy of the joint distribution as:

$$H(x,y) = -\sum_{i,j} p(i,j) \log p(i,j)$$

Similarly, we can define the conditional entropy (or equivocation or remaining uncertainty in x if we are given y) as:

$$H(x|y) = -\sum_{i,j} p(i,j) \log p(i|j)$$

Shannon also showed that if x is a transmitted signal and y is the received signal, then the information which receiving y gives about x is

$$I(x;y) = H(x) - H(x|y) \quad (7)$$

$$\text{or } I(x;y) = H(y) - H(y|x) \quad (8)$$

$$\text{or } I(x;y) = H(x) + H(y) - H(x,y) \quad (9)$$

Because of the symmetry of the above equations, this term is known as the mutual information between x and y.

The channel capacity is defined to be the maximum value over all possible values of x and y of this mutual information.

The basic facts in which we will take an interest are:

- Because the occurance of an unlikely event has more information than that of a likely event, it has a higher information content.

- Hence, a data set with high variance is liable to contain more information than one with small variance.

- A channel of maximum capacity is defined by 100% mutual information i.e. $I(x;y) = H(x)$

2.3 Principal Component Analysis

Inputs to a neural net generally exhibit high dimensionality i.e. the N input lines can each be viewed as 1 dimension so that each pattern will be represented as a coordinate in N dimensional space.

A major problem in analysing data of high dimensionality is identifying patterns which exist across dimensional boundaries. Such patterns may become visible when a change of basis of the space is made, however an *a priori* decision as to which basis will reveal most patterns requires fore-knowledge of the unknown patterns.

A potential solution to this impasse is found in Principal Component Analysis which aims to find that orthogonal basis which maximises the data's variance for a given dimensionality of basis. The usual tactic is to find that direction which accounts for most of the data's variance - this becomes the first basis vector. One then finds that direction which accounts for most of the remaining variance - this is the second basis vector and so on. If one then projects data onto the Principal Component directions, we perform a dimensionality reduction which will be accompanied by the retention of as much variance in the data as possible.

In general, it can be shown (Jolliffe, 1986) that the k^{th} basis vector from this process is the same as the k^{th} eigenvector of the co-variance matrix [1], C where

$$c_{ij} = \langle (x_i - \langle x \rangle)(x_j - \langle x \rangle) \rangle$$

where the angled brackets indicate an ensemble average i.e. the average over all possible sequences of values x_j from any arbitrary starting position of the sequence.

[1] where the eigenvectors are enumerated in normal form i.e. the eigenvector corresponding to the largest eigenvalue is first, that corresponding to the second largest is second etc.

For zero-mean data, the covariance matrix is equivalent to a simple correlation matrix.

Now, if we have a set of weights which are the eigenvectors of the input data's covariance matrix,C, then these weights will transmit the largest values to the outputs when an item of input data is in the direction of the largest correlations which corresponds to those eigenvectors with the largest eigenvalues. Thus, if we can create a situation in an Artificial Neural Network where one set of weights (into a particular output neuron) converges to the first eigenvector (corresponding to the largest eigenvalue), the next set of weights converges to the second eigenvector and so on, we will be in a position to maximally recreate at the outputs the directions with the largest variance in the input data.

Note that representing data as coordinates using the basis found by a PCA means that the data will have greatest variance along the first principal component, the next greatest variance along the second, and so on. While it is strictly only true to say that information and variance may be equated in Gaussian distributions, it is a good rule-of-thumb that a direction with more variance contains more information than one with less variance. PCA is the process of projecting the n-dimensional input data onto that m-dimensional subspace (where m<<n) which is spanned by those vectors which contain most variance. Thus PCA provides a means of compressing the data whilst retaining as much information within the data as possible. It can be shown that if a set of input data has a covariance matrix whose eigenvalues are $\{\lambda_1, \lambda_2, ..., \lambda_n\}$ and if we represent the data in coordinates on a basis spanned by the first m eigenvectors, the loss of information due to the compression (i.e. due to projecting the data onto the lower dimensioned subspace) is

$$E = \sum_{i=m+1}^{n} \lambda_i \tag{10}$$

2.3.1 Calculation of Principal Components

Since most users of Principal Components will use a standard statistical package (and hence not care about the method of calculating the PCs) this section provides only a short summary of the major methods of calculating Principal Components

; more detailed discussion of methods can be found in many standard statistical texts. For computer scientists, the invaluable "Numerical Recipes in C" (Press et al., 1988)provides an excellent introduction. We discuss 3 methods

1. The Power Method
2. The QL (or similar QR) method
3. The method of Singular Value Decomposition

The discussion below owes much to that found in Joliffe (Jolliffe, 1986).

The Power Method

The power method, though not in general use now, is included as it is the most intuitively obvious method and was the first to be identified. We describe the simplest form. Let us wish to find the largest eigenvalue and corresponding eigenvector of a p*p matrix **T**. We choose an intitial p-vector, $\mathbf{u}_0$ and then form the sequence

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{T}\mathbf{u}_0 \\
\mathbf{u}_2 &= \mathbf{T}\mathbf{u}_1 = \mathbf{T}^2\mathbf{u}_0 \\
\mathbf{u}_3 &= \mathbf{T}\mathbf{u}_2 = \mathbf{T}^3\mathbf{u}_0 \\
\mathbf{u}_r &= \mathbf{T}\mathbf{u}_{r-1} = \mathbf{T}^r\mathbf{u}_0
\end{aligned}$$

If the eigenvectors of **T** are $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \ldots \mathbf{a}_p$ with corresponding eigenvalues λ_j, then for any vector $\mathbf{u}_0$,

$$\mathbf{u}_0 = \sum_{j=1}^{p} w_j \mathbf{a}_j \tag{11}$$

for scalars w_j. Then, for our sequence above, we have

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{T}\mathbf{u}_0 \\
&= \sum_{j=1}^{p} w_j \mathbf{T}\mathbf{a}_j \\
&= \sum_{j=1}^{p} w_j \lambda_j \mathbf{a}_j
\end{aligned}$$

Similarly, we have

$$\begin{aligned}\frac{\mathbf{u}_r}{w_1\lambda_1^r} &= \frac{\sum_{j=1}^{p} w_j\lambda_j^r\mathbf{a}_j}{w_1\lambda_1^r} \\ &= (\mathbf{a}_1 + \frac{w_2}{w_1}(\frac{\lambda_2}{\lambda_1})^r\mathbf{a}_2 + \ldots + \frac{w_p}{w_1}(\frac{\lambda_p}{\lambda_1})^r\mathbf{a}_p) \\ &\rightarrow \mathbf{a}_1 \text{ as } r \rightarrow \infty\end{aligned}$$

provided $\lambda_1 > \lambda_i, i \neq 1$. Speed of convergence of the algorithm depends both on the initial choice of $\mathbf{u}_0$ and also on the relative magnitude of the first eigenvalue to the second.

Various refinements of the method can be shown to enable convergence to subsequent eigenvectors.

The QL (or QR) Algorithms

The method of the QL algorithm depends on the fact that any (non-singular) matrix $\mathbf{T}$ can be decomposed as $\mathbf{T} = \mathbf{QL}$ where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{L}$ is a lower triangular matrix. Again an iterative procedure is used: let $\mathbf{T}_1 = \mathbf{T}$; then use $\mathbf{T}_1 = \mathbf{Q}_1\mathbf{L}_1$ to enable calculation of $\mathbf{Q}_1$ and $\mathbf{L}_1$ and then calculate $\mathbf{T}_2$ using $\mathbf{T}_2 = \mathbf{L}_1\mathbf{Q}_1$. This is the first step in an iterative procedure which can be shown to cause convergence of $\mathbf{T}_r$ to a diagonal matrix comprising the eigenvalues of $\mathbf{T}$ in order.

The QR algorithm is very similar except that it uses the fact that any (non-singular) matrix $\mathbf{T}$ can be decomposed as $\mathbf{T} = \mathbf{QR}$ where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an *upper* triangular matrix.

Singular Value Decomposition

SVD is the method which is most often used by modern statisticians in the calculation of Principal Components due to its proven efficiency. This relies on the fact that any n*p matrix $\mathbf{T}$ can be written as $\mathbf{T} = \mathbf{ULA}^T$ where

$\mathbf{U}$ is a n*r matrix such that $\mathbf{U}^T\mathbf{U} = \mathbf{I}_r$

$\mathbf{A}$ is a p*r matrix such that $\mathbf{A}^T\mathbf{A} = \mathbf{I}_r$

$\mathbf{L}$ is a r*r diagonal matrix and r is the rank of $\mathbf{X}$.

Since this is defined for a non-square matrix, it may be used for situations in which the number of observations is lower than the number of variables (i.e. where the rank of the sample covariance matrix is reduced). The actual method of calculation of the singular values is very similar to the iterative methods of the QR / QL section above.

Artificial Neural Networks and PCA

Artificial Neural Networks and PCA come together in 2 ways:

1. There are some networks which use Principal Components as an aid to learning e.g. (Huang and Huang, 1993)

2. Some networks have been explicitly designed to calculate Principal Components

It is the latter with which this thesis will deal.

2.4 Weight Decay in Hebbian Learning

As noted in Section 2.1, if there are no constraints placed on the growth of weights under Hebbian learning, there is a tendancy for the weights to grow without bounds. It is possible to renormalise weights after each learning epoch, however this adds an additional operation to the network's processing.

Another possibility is to allow the weights to grow until each reaches some limit (Linsker, 1986b), e.g. have an upper limit of w^+ and a lower limit of w^- and clip the weights when they reach either of these limits. Clearly a major disadvantage of this is that if all weights end up at one or other of these limits[2] the amount of information which can be retained in the weights is very limited.

A third possibility is to prune weights which do not seem to have importance for the network's operation. However, this is an operation which must be performed using non-local knowledge - typically which weights are of much smaller magnitude than their peers.

[2]This will certainly happen if simple Hebbian learning is used

Hence, interest has grown in the use of decay terms embedded in the learning rule itself (e.g. (McClelland et al., 1986),Chapter 17). Ideally such a rule should ensure that no single weight should grow too large while keeping the total weights on connections into a particular output neuron fairly constant. One of the simplest forms of weight decay was developed as early as 1968 by Grossberg(Grossberg, 1968) and was of the form:

$$\frac{dw_{ij}}{dt} = \alpha y_j x_i - w_{ij} \tag{12}$$

It is clear that the weights will be stable (when $\frac{dw_{ij}}{dt} = 0$) at the points where $w_{ij} = \alpha < y_j x_i >$ where the angled brackets indicate an ensemble average. Using a similar type of argument to that employed for simple Hebbian learning, we see that at convergence we must have $\alpha \mathbf{C}\mathbf{w} = \mathbf{w}$. Thus $\mathbf{w}$ would have to be an eigenvector of the correlation matrix of the input data with corresponding eigenvalue $\frac{1}{\alpha}$. We shall be interested in a somewhat more general result.

Grossberg went on to develop more sophisticated learning equations which use weight decay e.g. for his instar coding, (Grossberg, 1988a) he has used

$$\frac{dw_{ij}}{dt} = \alpha\{y_j - w_{ij}\}x_i \tag{13}$$

where the decay term is gated by the input term x_i and for outstar coding

$$\frac{dw_{ij}}{dt} = \alpha\{x_i - w_{ij}\}y_j \tag{14}$$

where the decay term is gated by the output term y_j. These, while still falling some way short of the decay in which we will be interested, show that researchers of this time were beginning to think of both differentially weighted decay terms and allowing the rate of decay to depend on the statistics of the data presented to the network.

2.4.1 Principal Components and Weight Decay

Miller and MacKay (K.Miller and MacKay, 1992) have provided a definitive study of the results of a decay term on Hebbian learning. They suggest an initial distinction between Multiplicative Constraints and Subtractive Constraints.

They define Multiplicative Constraints as those satisfying

$$\frac{d}{dt}\,\mathbf{w}(t) = \mathbf{C}\mathbf{w}(t) - \gamma(\mathbf{w})\mathbf{w}(t)$$

where the decay in the weights is governed by the product of a function of the weights, $\gamma(\mathbf{w})$, and the weights, w(t), themselves. The decay term can be viewed as a feedback term which limits the rate of growth of each weight in proportion to the size of the weight itself while the first term defines the Hebbian learning itself.

Subtractive Constraints are satisfied by equations of the form

$$\frac{d}{dt}\mathbf{w}(t) = \mathbf{C}\mathbf{w}(t) - \epsilon(\mathbf{w})\mathbf{n}$$

where the decay in the weights is governed by the product of a function of the weights ,$\epsilon(\mathbf{w})$, and a constant vector,$\mathbf{n}$, (which is often $\{1, 1, ..1\}^T$).

They prove that

- Hebb rules whose decay is governed by Multiplicative Constraints will, in cases typical of Hebb learning, ensure that the weights will converge to a stable point
- This stable point is a multiple of the principal eigenvector of the covariance matrix of the input data
- Hebb rules governed by Subtractive Constraints will tend to lead to saturation of the weights at their extreme permissible values[3]
- Under Subtractive Constraints, there is actually a fixed point within the permitted hypercube of values but this is unstable and is only of interest in anti-Hebbian learning(see below).
- If specific limits (w^+ and w^-) do not exist, weights under Subtractive Constraints will tend to increase without bound.

In summary then, Subtractive Constraints offer little that cannot be had from simple clipping of the weights at preset upper and lower bounds. Multiplicative

[3]Such values may be partially determined by the eigenvalues of the covariance matrix but are not, in general, multiples of the eigenvectors.

Constraints, however, seem to give us not just weights which are conveniently small, but also weights which are potentially useful since

$$y_i = \sum_j w_{ij} x_j = \mathbf{w_i}.\mathbf{x}$$

where $\mathbf{w_i}$ is the vector of weights into neuron y_i and $\mathbf{x}$ is the vector of inputs. But,

$$\mathbf{w_i}.\mathbf{x} = |\mathbf{w_i}||\mathbf{x}| \cos\theta$$

where $|\mathbf{d}|$ is the length of $\mathbf{d}$ and θ is the angle between the 2 vectors.

This is maximised when the angle between the vectors is 0. Thus, if $\mathbf{w_1}$ is the weight into the first neuron which converges to the first Principal Component, the first neuron will maximally transmit information along the direction of greatest correlation, the second along the next largest, etc. In Section 2.3, we noted that these directions were those of greatest variance which from Section 2.2, we are equating with those of maximal information transfer through the system.

Given that there are statistical packages which find Principal Components, we should ask why it is necessary to reinvent the wheel using Artificial Neural Networks. There are 2 major advantages to PCA using ANNs:

1. Traditional statistical packages require us to have available prior to the calculation, a batch of examples from the distribution being investigated. While it is possible to run the ANN models with this method - "batch mode" - ANNs are capable of performing PCA in real-time i.e. as information from the environment becomes available we use it for learning in the network. We are, however, really calculating the Principal Components of a sample, but since these estimators can be shown to be unbiased and to have variance which tends to zero as the number of samples increases, we are justified in equating the sample PCA with the PCA of the distribution. The adaptive/recursive methodology used in ANNs is particularly important if storage constraints are important.

2. Strictly, PCA is only defined for stationary distributions. However, in realistic situations, it is often the case that we are interested in compressing data from distributions which are a function of time; in this situation, the sample PCA

outlined above is the solution in that it tracks the moving statistics of the distribution and provides as close to PCA as possible in the circumstances.

However, most proofs of PCA ANNs convergences require the learning rate to converge to 0 in time and, in practice, it is the case that convergence is often more accurate when the learning rate tends to decrease in time. This would preclude an ANN following a distribution's statistics, an example of the well-known trade-off between tracking capability and accuracy of convergence.

We now look at several ANN models which use weight decay with the aim of capturing Principal Components. We will make no attempt to be exhaustive since that would in itself require a thesis; we do however attempt to give representative samples of current network types.

2.5 Early Models

There were a number of ANN models developed in the 1980s which used Hebbian learning. We will investigate 2 for comparative purposes:

1. Linsker's Model

2. Oja's Single Neuron Model

2.5.1 The InfoMax Principle in Linsker's Model

Linsker(Linsker, 1986b) has developed a Hebb learning ANN model which attempts to realise the InfoMax principle - the neural net created should transfer the maximum amount of information possible between inputs and outputs subject to constraints needed to inhibit unlimited growth. Linsker notes that this criterion is equivalent to performing a Principal Component Analysis on the cell's inputs.

Although Linsker's model is a multi-layered model, it does not use a supervised learning mechanism; he proposes that the information which reaches each layer should be processed in a way which maximally preserves the information. That this does not, as might be expected, lead to an identity mapping, is actually due to the effect

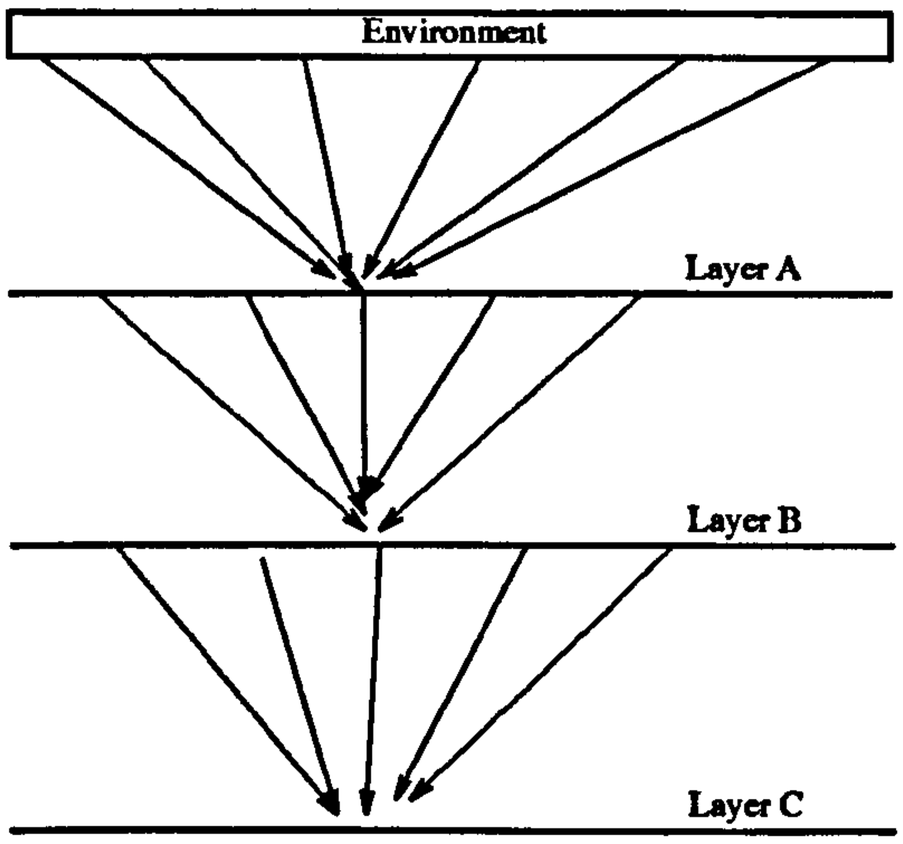


Figure 2: Linsker's model

of noise. Each neuron "responds to features that are statistically and information-theoretically most significant"((Linsker, 1988), page 116). He equates the process with a Principal Component Analysis.

Linsker's network is shown in Figure 2. Each layer comprises a 2-dimensional array of neurons. Each neuron in layers from the second onwards receives input from several hundred neurons in the previous layer and sums these inputs in the usual fashion. The region of the previous layer which sends input to a neuron is called the receptive field of the neuron and the density of distribution of inputs from a particular region of the previous layer is defined by a Gaussian distribution. At the final layer, lateral connections within the layer are allowed.

The Hebb-type learning rule is

$$\Delta w_{ij} = a(x_i - \langle x \rangle)(y_j - \langle y \rangle) + b$$

where a and b are constants.

In response to the problem of unlimited growth of the network weights, Linsker uses a hard limit to the weight-building process i.e. the weights are not allowed to exceed w^+ nor decrease beyond w^- where $w^- = -w^+$.

Miller and MacKay (K.Miller and MacKay, 1992) have observed that Linsker's model is based on Subtractive Constraints, i.e.

$$\Delta w_{ij} = a x_i y_j - a\langle x \rangle y_j - a\langle y \rangle (x_i - \langle x \rangle)$$

Both y_j and $< y >$ are functions of w, but in neither case are we multiplying these by w itself. Therefore, as noted earlier, the weights will not tend to a multiple of the principal eigenvector but will saturate at the bounds (w_{ij}^{+} *or* w_{ij}^{-}) of their permissible values.

Because the effects of the major eigenvectors will still be felt, there will not be a situation where a weight will tend to w^{-} in a direction where the principal eigenvector has a positive correlation with the other weights. However, the directions of the weight matrix will, in general, bear little resemblence to any eigenvector of the correlation matrix. The model will not, in general, enable maximal information transfer through the system.

2.5.2 Oja's One Neuron Model

Oja (Oja, 1982) proposed a model which extracts the largest principal component from the input data. He suggested a single output neuron which sums the inputs in the usual fashion

$$y = \sum_{i=1}^{m} w_i x_i$$

His variation on the Hebb rule, though, is

$$\Delta w_i = \alpha(x_i y - y^2 w_i)$$

Note that this is a rule defined by Multiplicative Constraints ($y^2 = \gamma(w)$) and so will converge to the principal eigenvector of the input covariance matrix. The weight decay term has the simultaneous effect of making $\sum w_i^2$ tend towards 1 i.e. the weights are normalised.

However, this rule will find only the first eigenvector (that direction corresponding to the largest eigenvalue) of the data. It is not sufficient to simply throw clusters of neurons at the data since all will find the same (first) Principal Component; in order to find other PCs, there must be some interaction between the neurons. Other rules which find other principal components have been identified by subsequent research, an example of which is shown in the next Section.

2.6 Recent PCA Models

We will consider 3 of the most popular PCA models. It is of interest to begin with the development of Oja's models over recent years.

2.6.1 Oja's Subspace Algorithm

The One Neuron network reviewed in the last section is capable of finding only the first Principal Component. While it is possible to use this network iteratively by creating a new neuron and allowing it to learn on the data provided by the residuals left by subtracting out previous Principal Components, this involves several extra stages of processing for each new neuron.

Therefore Oja's(Oja, 1989) Subspace Algorithm provided a major step forward. The network has N output neurons each of which learns using a Hebb type rule with weight decay. Note however that it does not guarantee to find the actual directions of the Principal Components; the weights *do* however converge to an orthonormal basis of the Principal Component Space. We will call the space spanned by this basis the Principal Subspace. The learning rule is

$$\Delta w_{ij} = \alpha(x_i y_j - y_j \sum_k w_{ik} y_k) \tag{15}$$

which has been shown to force the weights to converge to a basis of the Principal Subspace[4].

One advantage of this model compared with some other networks (e.g. (Sanger, 1990)) is that it is completely homogeneous i.e. the operations carried out at each neuron are identical.

The major disadvantage of this algorithm is that it finds only the Principal Subspace of the eigenvectors not the actual eigenvectors themselves.

---

[4]In this case $\gamma(w_{ij}) = y_j^2$. However, the additional weight decay constraints from the other outputs $y_j \sum_{j \neq k} w_{ik} y_k$ force decay in the directions of other eigenvectors. Therefore the total of the decay parameters only forces weight convergence to the subspace

2.6.2 Oja's Weighted Subspace Algorithm

The final stage is the creation of algorithms which find the actual Principal Components of the input data. In 1992, Oja *et al* recognised the importance of introducing asymmetry into the weight decay process in order to force weights to converge to the Principal Components. The algorithm is defined by the equations

$$y_j = \sum_{i=1}^{n} w_{ij} x_i$$

where a Hebb-type rule with weight decay modifies the weights according to

$$\Delta w_{ij} = \eta y_j (x_i - \theta_j \sum_{k=1}^{N} y_k w_{kj})$$

Ensuring that $\theta_1 < \theta_2 < \theta_3 < ...$ allows the neuron whose weight decays proportional to θ_1 (i.e. whose weight decays least quickly) to learn the principal values of the correlation in the input data. That is, this neuron will respond maximally to directions parallel to the principal eigenvector, i.e. to patterns closest to the main correlations within the data. The neuron whose weight decays proportional to θ_2 cannot compete with the first but it is in a better position than all of the others and so can learn the next largest chunk of the correlation, and so on.

It can be shown that the weight vectors will converge to the principal eigenvectors in the order of their eigenvalues. The algorithm clearly satisfies Miller and Mackay's definition of Multiplicative Constraints with $\gamma(w_i) = \theta_i \sum_k y_k w_{ki} x_i$.

2.6.3 Sanger's Generalized Hebbian Algorithm

Sanger (Sanger, 1990) has developed a different algorithm (which he calls the "Generalized Hebbian Algorithm") which also finds the actual Principal Components. He also introduces asymmetry in the decay term of his learning rule:

$$\Delta w_{ij} = \alpha(x_i y_j - y_j \sum_{k=1}^{j} w_{ik} y_k) \tag{16}$$

Note that the crucial difference between this rule and Oja's Subspace Algorithm is that the decay term for the weights into the j^{th} neuron is a weighted sum of the first

j neurons' activations. Sanger's algorithm can be viewed as a repeated application of Oja's One Neuron Algorithm by writing it as

$$\Delta w_{ij} = \alpha([x_i y_j - y_j \sum_{k=1}^{j-1} w_{ik} y_k] - y_j^2 w_{ij}) \tag{17}$$

We see that the central term comprises the residuals after the first j-1 Principal Components have been found, and therefore the rule is performing the equivalent of One Neuron learning on subsequent residual spaces. However, note that the asymmetry which is necessary to ensure convergence to the actual Principal Components, is bought at the expense of requiring the j^{th} neuron to 'know' that it is the j^{th} neuron by subtracting only j terms in its decay. It is Sanger's contention that all true PCA rules are based on some measure of deflation such as shown in this rule.

2.7 Principal Components and Anti-Hebbian Learning

All the ANNs we have so far met have been feedforward networks - activation has been propagated only in one direction. However, many real biological networks are characterised by a plethora of recurrent connections. This has led to increasing interest in networks which, while still strongly directional, allow activation to be transmitted in more than one direction i.e. either laterally or in the reverse direction from the usual flow of activation. One interesting idea is to associate this change in direction of motion of activation with a minor modification to the usual Hebbian learning rule called Anti-Hebbian learning (a comprehensive analysis of Anti-Hebbian learning is given in (Palmieri et al., 1993)).

If inputs to a neural net are correlated, then each contains information about the other. In information theoretical terms, there is redundancy in the inputs ($I(x;y) > 0$).

Anti-Hebbian learning is designed to decorrelate input values. The intuitive idea behind the process is that more information can be passed through a network when the nodes of the network are all dealing with different data. The less correlated the

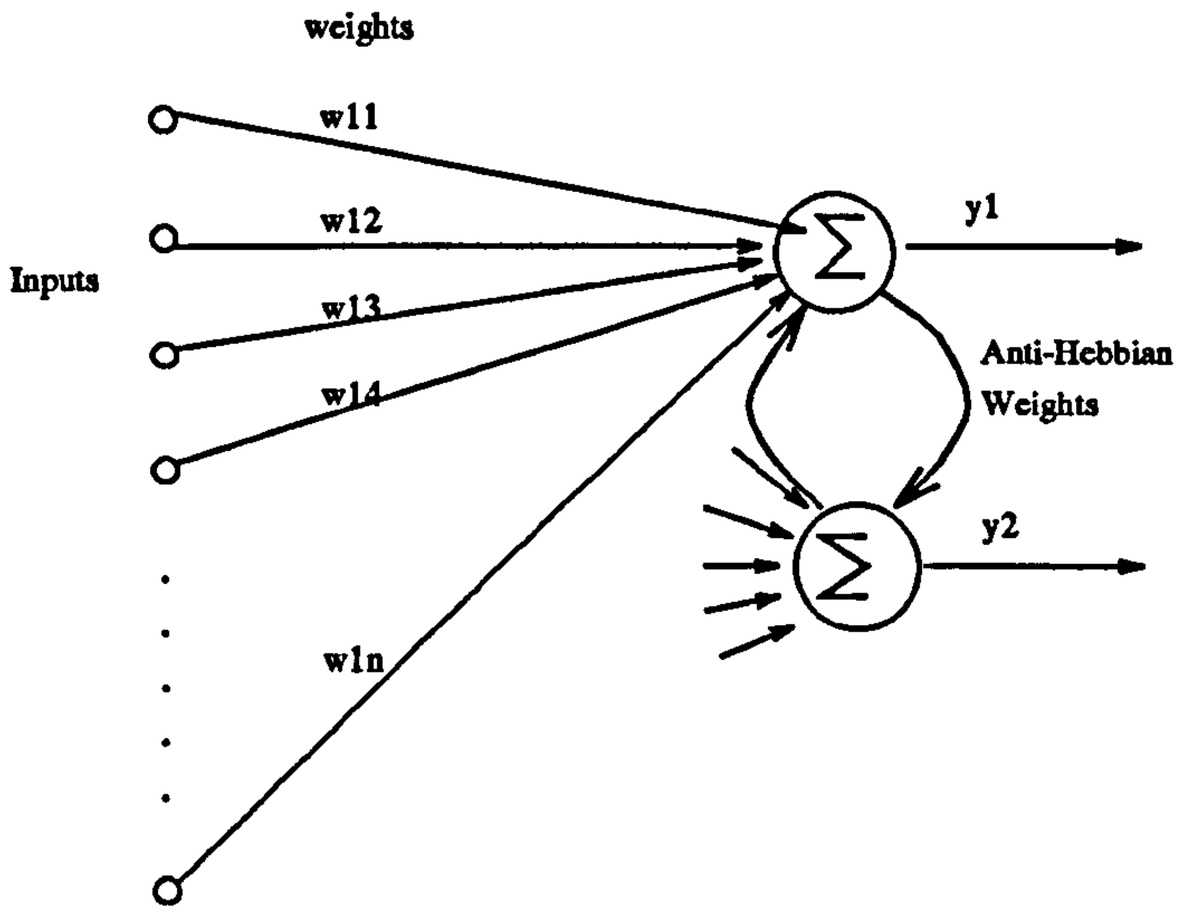


Figure 3: Anti-Hebbian Weights
Negative decorrelating weights between neurons in the same layer are learned using an "anti-Hebbian" learning rule

neurons' responses, the less redundancy is in the data transfer. The aim of producing decorrelated responses, however, in order to maximise information transfer must be modified if the outputs are subject to noise: intuitively, the more noise in the network the more correlation is necessary to optimise information transfer((Plumbley, 1991)). If 2 neurons respond to the same signal, there is a measure of correlation between them and this is used to affect their responses to future similar data. Anti-Hebbian learning is sometimes known as lateral inhibition as this type of learning is generally used between members of the same layer and not between members of different layers. The basic model is defined by

$$\Delta w_{ij} = -\alpha \langle y_i y_j \rangle$$

Therefore, if initially y_i and y_j are highly correlated then the weights between them will grow to a large negative value and each will tend to turn the other off.

It is clear that there is no need for weight decay terms or limits on anti-Hebbian weights as they are automatically self-limiting, provided decorrelation can be attained.

$$(\langle y_i . y_j \rangle \to 0) \Longrightarrow (\Delta w_{ij} \to 0) \tag{18}$$

i.e. weight change stops when the outputs are decorrelated. Success in decorrelating the outputs results in weights being stabilised.

It has been shown (Rubner and Tavan, 1989) that not only does anti-Hebbian learning force convergence in the particular case of a deflationary algorithm but that the lateral connections do indeed vanish.

The method is valid for all deflationary networks.

Several authors have developed Principal Component models using a mixture of one of the above PCA methods (often Oja's One Neuron Rule) and Anti-Hebbian weights between the output neurons e.g. (Brause, 1993b; Rubner and Schulten, 1990; Brause, 1993a; Palmieri, 1993; White, 1993).

We first note a similarity between the aims of PCA and anti-Hebbian learning: the aim of anti-Hebbian learning is to decorrelate neurons. If a set of neurons performs a Principal Component Analysis, their weights form an orthogonal basis of the space of principal eigenvectors. Thus, both methods perform a decorrelation of the neurons' responses.

Further, in information theoretic terms, decorrelation ensures that the maximal amount of information possible for a particular number of output neurons is transferred through the system. We will consider only noise-free information-transfer since if there is some noise in the system, some duplication of information may be beneficial to optimal information transfer.

2.7.1 The Interneuron Model

Plumbley (Plumbley, 1991) has developed a model of Hebb learning which is based on the minimisation of information loss throughout the system. However, for Gaussian signals there is no difference between this principle and InfoMax.

Since there are no known biological examples of neurons which both excite and inhibit other neurons of the same type (Dale's Law), Plumbley postulates a layer of interneurons which act as decorrelating neurons for the output neurons.

He develops these interneurons in 2 ways, suggesting that he is giving 2 different views of the same network. However, we will see that these interneurons have different capabilities depending on which network is used.

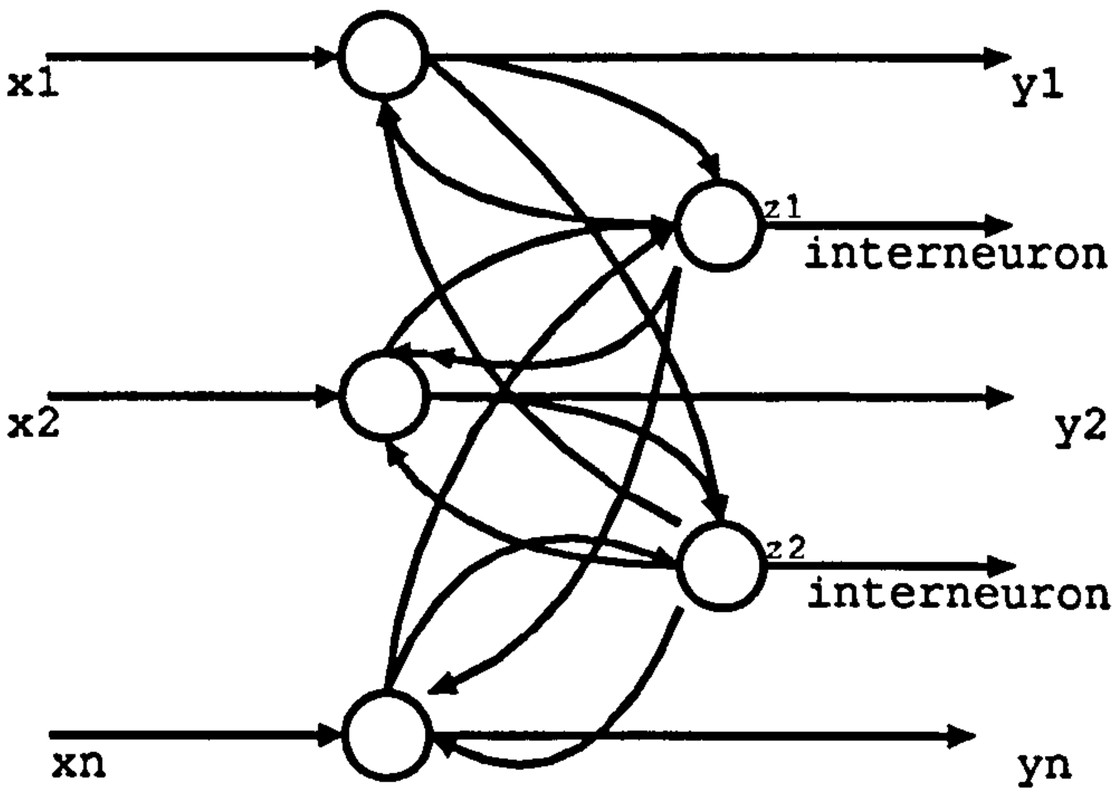


Figure 4: The Interneuron Model

In both networks the interneurons are developed as anti-Hebbian neurons with the additional property of trying to optimise information transfer within limited power constraints. Plumbley notes that the best information transfer rate will be found when the outputs are decorrelated; however,he also tries to equalise the variance of the outputs to ensure that they are then carrying equal information.

Figure 4 shows the form of the first model.

The dynamics of the network are described by

$$\mathbf{z} = \mathbf{V}^T\mathbf{y}$$

where z_j is the activation of the interneuron

y_i is the output from the network

and V_{ij} is the weight joining the i^{th} ouput neuron to the j^{th} interneuron.

This makes the output response

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z}$$

Plumbley concentrates on the information preserving properties of the forward transformation between inputs and outputs and shows

$$\mathbf{y} = (\mathbf{I} + \mathbf{V}\mathbf{V}^T)^{-1}\mathbf{x}$$

Plumbley uses a weight decay mechanism in his learning:

$$\Delta v_{ij} = \eta(y_i z_j - \lambda v_{ij})$$

This is equivalent to a learning rule in the limit of

$$\frac{d}{dt}\,\mathbf{v}(t) = (\mathbf{C} - \gamma\mathbf{I})\mathbf{v}$$

A solution to this equation is

$$\mathbf{v}(t) = A\exp(\mathbf{C} - \gamma\mathbf{I})t$$

Therefore, the weights will increase without limit in directions where the eigenvalue of the correlation matrix exceeds γ . Thus the weights will never tend to a multiple of the principle eigenvector and no selectivity in information transfer will be achieved. Note that there are fixed points on the eigenvectors but these are not stable.

The crucial difference between this model and Oja's model is that in Oja's model the decay term is a function of the weights times the weights. In this model, the decay term is not strong enough to force the required convergence.

Equally, the anti-Hebbian learning rule does not force convergence to a set of decorrelated outputs.

$$\Delta v_{ij} = \eta(y_i z_j - \lambda v_{ij})$$

does not mean that

$$(\Delta v_{ij} = 0) \Longrightarrow (\langle y_i z_j \rangle = 0).$$

However, in taking "another view of the skew-symmetric network", Plumley uses the interneurons as the outputs to the network.

In this model, we have forward excitations U and backward excitations V where

$$\mathbf{z} = \mathbf{U}^T\mathbf{y}$$

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z}$$

i.e.

$$\mathbf{z} = \mathbf{U^T}(\mathbf{I} + \mathbf{V}\mathbf{U}^T)^{-1}\mathbf{x}$$

where the weight update is done using the same update rule $\Delta v_{ij} = \eta(y_i z_j - \lambda v_{ij})$

Since the output is from the interneurons we are interested in the forward transform from the x values to the z values.

$$\begin{aligned} y_i &= x_i - \sum_k u_{ki} z_k \\ \text{Now, } \Delta u_{ij} &= \eta(y_i z_j - \lambda u_{ij}) \\ &= \eta((x_i - \sum_k u_{ki} z_k) z_j - \lambda u_{ij}) \end{aligned}$$

Plumbley states that the last term is the weight decay term. In fact, as can be seen from the above equations, the second term is the important weight decay term, being a form of Multiplicative Constraint. There is an implicit weight decay built into the recurrent architecture - a fact which we will use in the next Chapter.

However, if we consider the network as a transformation from the x values to the y values we do not find the same implicit weight decay term.

$$\begin{aligned} z_i &= \sum_j u_{ij} y_j \\ &= \sum_j u_{ij}(x_j - \sum_k u_{kj} z_k) \\ &= \sum_j u_{ij} x_j - \sum_k z_k (\sum_j u_{ij} u_{kj}) \end{aligned}$$

And so,

$$\begin{aligned} \Delta u_{ij} &= \eta(y_i z_j - \lambda u_{ij}) \\ &= \eta(y_i(\sum_j u_{ij} x_j - \sum_k z_k (\sum_j u_{ij} u_{kj})) - \lambda u_{ij}) \end{aligned}$$

Using this form, it is hard to recognise the learning rule as a Hebb rule, let alone a decaying Hebb rule of a particular type.

However, as we shall see in the next chapter, the negative feedback in Plumbley's first network is an extremely valuable tool.

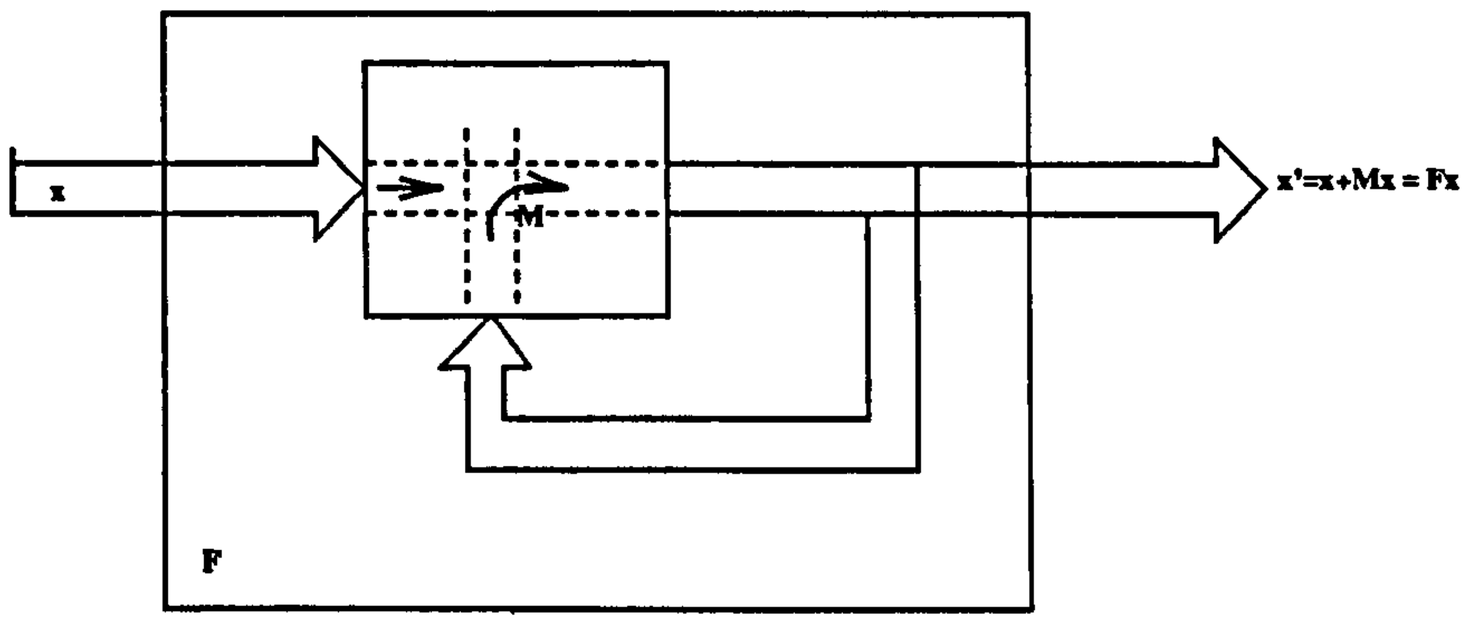


Figure 5: The System Model of the Novelty Filter

2.8 Negative Feedback in Neural Networks

Plumbley's model may be viewed as a negative feedback ANN. In this section, we consider other ANNs which have used negative feedback. We review separately those models where the dynamics of the network settling to an attractor state have been important to the value of the state reached and those models which have considered only the transfer of activation as a single event.

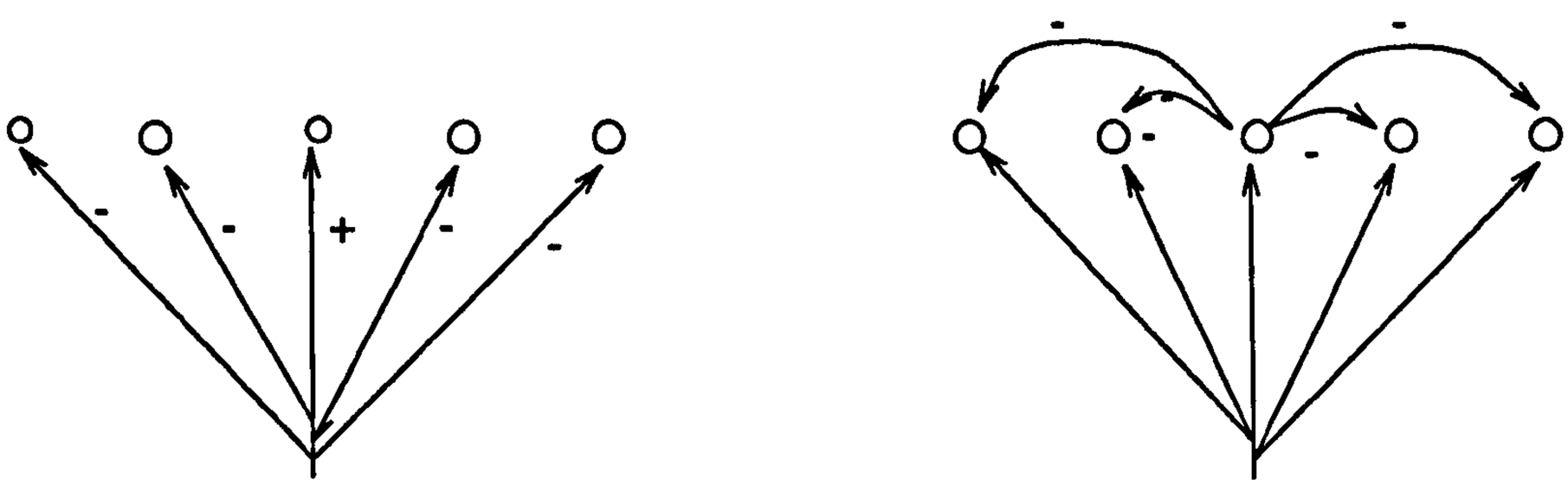
2.8.1 Static Models

The role of negative feedback in static models has most often been as the mechanism for competition (see e.g.(Carpenter, 1989; Kohonen, 1984) for summaries) often based on biological models of activation transfer e.g.(von der Malsburg, 1973) and sometimes based on psychological models e.g. (Cohen et al., 1988; Grossberg and Schmajuk, 1989; Grossberg, 1984)

An interesting early model was proposed by Kohonen (Kohonen, 1984) who uses negative feedback in a number of models, the most famous of which (at least of the simple models) is the so-called "novelty filter" (see Figure 5). Here we have an input vector $\mathbf{x}$ which generates feedback gain by the vector of weights, M. Each element of M is adapted using anti-Hebbian learning:

$$\frac{dm_{ij}}{dt} = -\alpha x_i' x_j' \tag{19}$$

$$\text{where } \mathbf{x}' = \mathbf{x} + \mathbf{M}\mathbf{x}' \tag{20}$$



(a) Feeedforward Inhibition (b) Feedback Inhibition

Figure 6: Two models of inhibition both of which yield on-centre, off-surround networks:(a) Feedforward inhibition (b) feedback inhibition

$$= \quad (\mathbf{I} - \mathbf{M})^{-1}\mathbf{x} = \mathbf{F}\mathbf{x} \tag{21}$$

"It is tentatively assumed $(I - M)^{-1}$ always exists." Kohonen shows that, under fairly general conditions on the sequence of x and the initial conditions of the matrix M, the values of F always converge to a projection matrix under which the output $\mathbf{x}'$ approaches zero although F need not converge to the zero matrix i.e. F converges to a mapping whose kernel ((Lipschutz, 1968), page 125) is the subspace spanned by the vectors x. Thus any new input vector $\mathbf{x_1}$ will cause an output which is solely a function of the novel features in $\mathbf{x_1}$.

2.8.2 Dynamic Models

The negative feedback of activation has most often been used in those models of Artificial Neural Networks which are based on a dynamic settling of activation. These are generally called Hopfield nets(Hertz et al., 1992) after John Hopfield (Hopfield, 1982) who performed an early analysis of their properties though earlier work on their properties was performed by other researchers e.g. following Grossberg (Grossberg, 1988b), we note (see Figure 6) that there are 2 types of on-center off-surround networks possible using inhibition. It is possible to generate

- Feedforward inhibition: the activation transfer rule is

$$\frac{dy_i}{dt} = -Ay_i + (B - y_i)x_i - y_i \sum_{k \neq i} x_k \tag{22}$$

A,B constants and x_i is the input to the i^{th} neuron. Grossberg points out that, if the activation is allowed to settle, this model has a stationary point ($\frac{dy_i}{dt} = 0$) when

$$y_i = \frac{x_i}{\sum_k x_k} * \frac{B \sum_k x_k}{A + \sum_k x_k} \tag{23}$$

Possibly of most interest is its self normalisation property, in that the total activity

$$\sum_k y_k = \frac{B \sum_k x_k}{A + \sum_k x_k} \tag{24}$$

is a constant.

- Feedback inhibition: we use here Grossberg's term though we will in future make a distinction between feedback inhibition between layers (as in Plumbley's network) and lateral inhibition between neurons in the same layer. Here Grossberg discusses the activation passing equation

$$\frac{dy_i}{dt} = -Ay_i + (B - y_i)[x_i + f(y_i)] - y_i[J_i + \sum_{k \neq i} f(y_k)] \tag{25}$$

where $J_i = \sum_{k \neq i} x_k$. The most interesting properties from this model develop when the activation function, f(), is a sigmoid which has the property that it forms a winner take-all network which suppresses noise, and quantises the total activity. Again these properties arise from an analysis of the dynamic properties of the negative feedback acting on the network activations.

For the remainder of this thesis we will be interested in negative feedback of activation in static models. We will use Plumbley's model with a simplified learning rule and investigate its emergent properties.

Chapter 3

The Interneuron Network

3.1 Introduction

In this chapter[1] we investigate more closely a network based on Plumbley's network (Chapter 2). We will, in fact, develop an extremely simple and effective Principal Component network which needs no weight decay in its learning rule: because of the negative feedback of activation, we can use simple Hebbian learning which will not cause instability in the weight growth process and which moreover causes the weights to converge to the Principal Components of the input data.

3.1.1 The Interneuron Network

For convenience we show again Plumbley's network in Figure 7. We recall from the previous chapter that Plumbley uses Hebbian learning with weight decay in his network. We retain the activation-transfer rules of his network but use no weight decay in the learning term.

We will show that the decay mechanism is unnecessary - that the architecture of the network alone is sufficient to guarantee convergence to the relevant principal subspace.

[1]Some of this work has already appeared in (Fyfe, 1993c; Fyfe, 1993d).

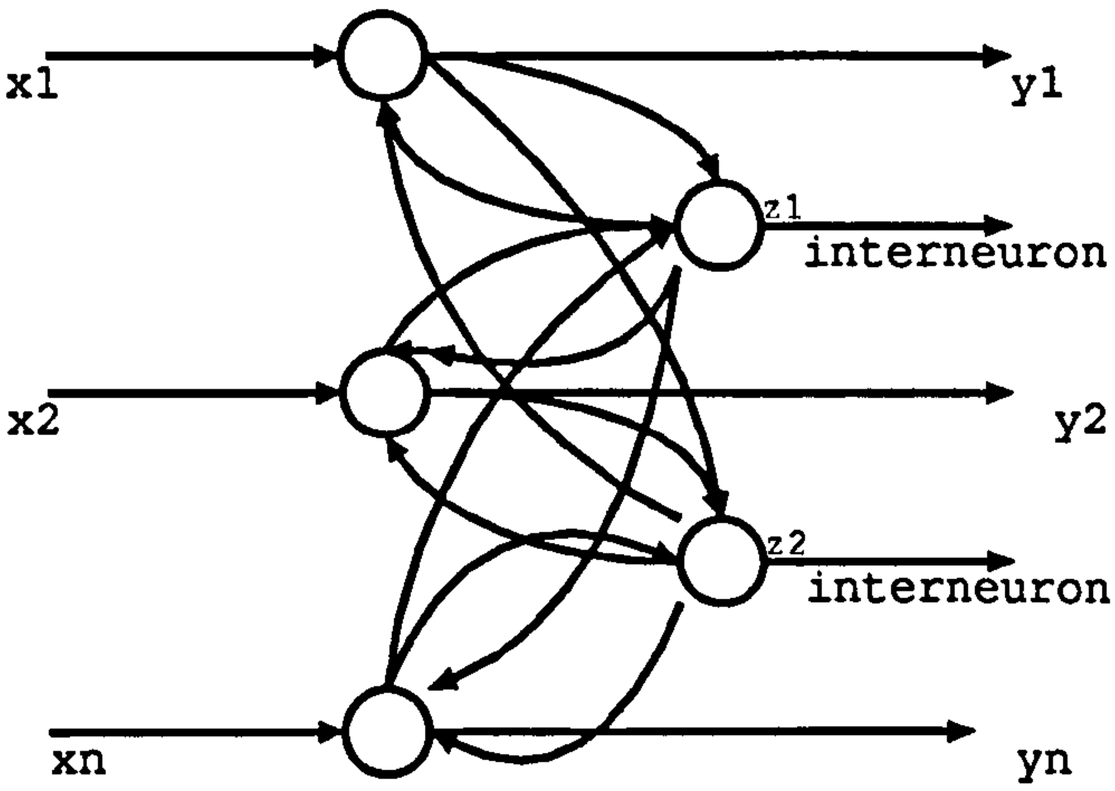


Figure 7: The Interneuron Model

The rules governing the organisation of the network are

$$\mathbf{y} = \mathbf{x} - \mathbf{W}\mathbf{z}$$

$$\mathbf{z} = \mathbf{W}^T\mathbf{y}$$

$$\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{z}^T$$

where $\mathbf{x}$ is the vector of inputs, $\mathbf{z}$ is the vector of activations at the interneurons and W is the weights joining the two layers of neurons. We use $\mathbf{y}$ to represent the activation at the summing neurons when the interneurons' activations have been returned.

There is no explicit weight decay, normalisation or clipping of weights in the model. The subtraction of the weighted sum of the interneuron values acts like anti-Hebbian learning. We will consider the network as a transformation from inputs $\mathbf{x}$ to interneuron outputs z; by considering the effects of these rules on individual neurons, we can quickly show that the resultant network is equivalent to Oja's Subspace Algorithm. We have

$$\begin{aligned} y_i &= x_i - \sum_k w_{ki} z_k \\ z_i &= \sum_j w_{ij} y_j \end{aligned}$$

Therefore,

$$\begin{aligned}\Delta w_{ij} &= \eta y_i z_j \\ &= \eta(x_i - \sum_k u_{ki} z_k) z_j \\ &= \eta(x_i z_j - z_j \sum u_{ki} z_k) \qquad (26)\end{aligned}$$

This last formulation of the learning rule (26) is exactly the learning rule for the Subspace Algorithm(Oja, 1989), Equation (15). A more formal analysis is given in Section 3.2

In order to compare this network with Oja's Subspace Algorithm, simulations were carried out on similar data[2]to that which Oja *et al*(Oja et al., 1992a) used to compare the Subspace and Weighted Subspace Algorithms. The results shown in Table 1 are from a network with 5 inputs each of zero mean random Gaussians, where x_1's variance is largest, x_2's variance is next largest, and so on.

Therefore, the largest eigenvalue of the input data's covariance matrix comes from the first input, x_1, the second largest comes from x_2 and so on. The advantage of using such data is that it is easy to identify the principal eigenvectors (and hence the principal subspace). There are 3 interneurons in the network and it can be seen that the 3-dimensional subspace corresponding to the first 3 principal components has been identified by the weights. There is very little of each vector outside the principal subspace i.e. in directions 4 and 5. The left matrix represents the results from the interneuron network, the right shows Oja's results.

The lower (W^TW) section shows that the weights form an orthonormal basis of the space and the upper (W) section shows that this space is almost entirely defined by the first 3 eigenvectors. The interneuron network also maintains the advantages of homogeneity and locality of computation (indeed, it is difficult to imagine a computationally simpler model).

Note that while we report, in general, on simulations run on this very special type of input data, all the networks developed in this thesis (other than those specifically identified in Chapter 5) perform excellently on all types of data.

[2]I did not have the value of the variances Oja used and therefore used variances of 5,4,3,2,1

W			W		
Interneuron 1	Interneuron 2	Interneuron 3	Output 1	Output 2	Output 3
0.249	**0.789**	**0.561**	**0.207**	**-0.830**	**0.517**
0.967	**-0.234**	**-0.100**	**-0.122**	**0.503**	**0.856**
-0.052	**-0.568**	**0.821**	**0.970**	**0.241**	-0.003
0.001	0.002	0.016	-0.001	0.001	0.001
-0.001	0.009	0.005	0.000	0.000	-0.001
W^TW			W^TW		
1.001	0.000	0.000	**1.000**	0.000	0.000
0.000	**1.000**	0.000	0.000	**1.000**	0.000
0.000	0.000	**1.000**	0.000	0.000	**1.000**

Table 1: Results from the simulated network and the reported results from Oja *et al.* The left matrix represents the results from the interneuron network, the right from Oja's Subspace Algorithm. Thus the first column represents the weights from the input neurons to the first interneuron (or alternatively the first row represents the weights from the first input neuron to the interneurons). Note that the weights are very small outside the principal subspace and that the weights form an orthonormal basis of this space. Weights above 0.1 are shown in bold font.

3.1.2 Algorithm for PCA

While the above networks may be adequate for biological information processors, a more precise engineering requirement is that of finding the actual Principal Components.

Recall that Oja *et al*(Oja et al., 1992a) amended the Subspace Algorithm by proposing the following modification to the learning rule

$$\Delta w_{ij} = \eta y_j (x_i - \theta_j \sum_{k=1}^{N} y_k w_{kj})$$

Ensuring that $\theta_1 < \theta_2 < \theta_3 < ...$ allows the neuron whose weight decays proportional to θ_1 (i.e. whose weight decays least quickly) to capture the principal component of the variance. The second captures the next largest component, and so on. The crucial point is the introduction of asymmetry into the learning algorithm.

This algorithm is local and homogeneous in that each neuron knows only its own value of θ_i. Analysis of the interneuron learning rule shows that, to simply insert a parameter, θ_i, would require computation at the level of the synapse. Whilst this may

be biologically feasible and algorithmically simple to implement, a different algorithm is developed here which uses the fact that the proposed network already incorporates subtraction of values.

The algorithm is: the system is created with 1 interneuron; this interneuron finds the first principal component using the above learning rule. It then loses its plasticity i.e. its weights will not subsequently change. We then create a second interneuron. Since the first neuron has found and subtracted the first principal component, the second neuron will find the largest remaining principal component. It too now loses its plasticity. Then the third interneuron is created etc.. Therefore, we have introduced our asymmetry in the time dimension; note that whereas to do so with e.g. Oja's Single Neuron Network would have required the introduction of an extra mechanism - that of subtracting the projection of the data onto the subspace already found - we do not require this here as the network automatically finds and subtracts this subspace.

To compare the results with Oja's Weighted Subspace Algorithm, we repeated the above experiment with the algorithm. Oja's simulation was carried out for 40000 iterations. The interneuron simulation allowed each interneuron to learn in 13000 iterations. The first interneuron learned during the first 13000 iterations, the second learned during the next 13000 and the third learned during the last 13000 iterations. The results are shown in Table 2; the left set is from the interneuron network, the right from Oja(1992).

Clearly both methods find the Principal eigenvectors. We note that the interneuron results have the advantage of equally weighting each eigenvector.

The algorithm retains the advantages of homogeneity and locality of computation. A more analytical proof of the convergence algorithm is developed in the next section.

3.2 An Analytical Investigation of Convergence

This section provides an analytical investigation of the algorithm which causes the interneuron weights to converge to the principal components of the input data's covariance matrix.

W			W		
1.000	-0.036	-0.008	**1.054**	-0.002	-0.002
0.036	**0.999**	-0.018	0.002	**1.000**	0.001
0.010	0.018	**1.000**	0.003	-0.002	**0.954**
-0.002	-0.002	0.016	-0.001	0.001	-0.002
0.010	0.003	0.010	0.001	-0.001	0.000
W^TW			W^TW		
1.001	0.000	0.000	**1.111**	0.000	0.000
0.000	**1.000**	0.000	0.000	**1.000**	0.000
0.000	0.000	**1.000**	0.000	0.000	**0.909**

Table 2: Results from the interneuron network (left) and from Oja (right).
Both methods find the principal eigenvectors of the input data covariance matrix. The interneuron algorithm has the advantage that the each vector is equally weighted.

The proof of the algorithm follows closely the methods developed by Oja and Karhunen (e.g. (Oja and Karhunen, 1985)) over the last decade; it is in 3 parts each of which refers to the interneuron learning rules:

$$\mathbf{y} = \mathbf{x} - \mathbf{W}\mathbf{z}$$

$$\mathbf{z} = \mathbf{W}^T\mathbf{y} = \mathbf{W}^T\mathbf{x}$$

$$\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{z}$$

In the first section we show that the weights of a single interneuron will converge to an eigenvector of the co-variance matrix; in the second, we show that these weights in fact converge to the principal eigenvector; in the third, we show that the algorithm ensures that the i^{th} interneuron's weights converge to the i^{th} eigenvector.

Theorem 1 *The weights,W, of a single interneuron with the above learning rules converges to an eigenvector of the input data co-variance matrix.*

Let w_i be the weight of the connection between y_i and z.

If the weights of a single interneuron converges to a limit, the expected weight change over a sufficiently long time will tend to zero. Given some assumptions[3],

[3]Which will be discussed later

particularly regarding the learning rate η and the nature of the distribution of $\mathbf{x}$, and using $\langle x \rangle$ to indicate the expected value of x with respect to the distribution from which it is drawn,

$$\begin{aligned}
\langle \Delta w_i \rangle = 0 &\iff \langle \eta y_i z \rangle = 0 \\
&\iff \langle y_i z \rangle = 0 \\
&\iff \langle (x_i - w_i z) z \rangle = 0 \\
&\iff \langle (x_i - w_i \sum_k w_k x_k) \sum_l w_l x_l \rangle = 0 \\
&\iff \langle \sum_l w_l x_l x_i - w_i \sum_{kl} w_k x_k x_l w_l \rangle = 0 \qquad (27) \\
&\iff \sum_l w_l C_{li} - w_i \sum_{kl} w_k C_{kl} w_l = 0 \qquad (28)
\end{aligned}$$

where C_{ij} is that element of the co-variance matrix showing the co-variance between the i^{th} and j^{th} elements of the input data $\mathbf{x}$. If the weights of the interneuron are to converge, then the above must be true for all values of w_i. Therefore the above may be written in matrix notation as

$$\begin{aligned}
\langle \Delta \mathbf{w} \rangle = \mathbf{0} &\iff \mathbf{Cw} - (\mathbf{w}^T \mathbf{Cw})\mathbf{w} = \mathbf{0} \\
&\iff \mathbf{Cw} = (\mathbf{w}^T \mathbf{Cw})\mathbf{w}
\end{aligned}$$

Now it is a standard result that the co-variance matrix C is positive-semidefinite; and hence

$$\mathbf{w}^T \mathbf{Cw} = \lambda \geq 0$$

where λ is a non-negative real number. Hence ,

$$\mathbf{Cw} = \lambda \mathbf{w}$$

Therefore, $\mathbf{w}$ converges to an eigenvector of C.

Theorem 2 *The weights, W, of a single interneuron with the above learning rules converges to the eigenvector with the largest eigenvalue of the input data co-variance matrix.*

Proof

The proof is by contradiction.

Assume that $\mathbf{w}$ converges to an eigenvector $\mathbf{c^*}$ of C with corresponding eigenvalue λ^*. Then, we will show that if there exists an eigenvector $\mathbf{c^1}$ of C with corresponding eigenvalue $\lambda^1 > \lambda^*$ a small perturbation in the direction of $\mathbf{c^1}$ will cause $\mathbf{w}$ to be unstable i.e. convergence will not take place.

Let $\mathbf{w}$ have converged to a direction close to $\mathbf{c^*}$ but to have a component ϵ in the direction of $\mathbf{c^1}$. Then,

$$\begin{aligned}
\langle \Delta \mathbf{w} \rangle &= \mathbf{C}\mathbf{w} - (\mathbf{w}^T\mathbf{C}\mathbf{w})\mathbf{w} \\
&= \mathbf{C}(\mathbf{c^*} + \epsilon) - ((\mathbf{c^{*T}} + \epsilon^T)\mathbf{C}(\mathbf{c^*} + \epsilon))(\mathbf{c^*} + \epsilon) \\
&= \mathbf{C}\mathbf{c^*} + \mathbf{C}\epsilon - (\mathbf{c^{*T}}\mathbf{C}\mathbf{c^*})\mathbf{c^*} - (\mathbf{c^{*T}}\mathbf{C}\mathbf{c^*})\epsilon - (\mathbf{c^{*T}}\mathbf{C}\epsilon)\mathbf{c^*} - (\epsilon^T\mathbf{C}\mathbf{c^*})\mathbf{c^*} + \mathbf{O}(\epsilon^2) \\
&= \lambda^*\mathbf{c^*} + \lambda^1\epsilon - \lambda^*\mathbf{c^*} - \lambda^*\epsilon - \mathbf{c^{*T}}\mathbf{C}(\epsilon\mathbf{c^*}) - (\mathbf{C}\epsilon)^T\mathbf{c^*}\mathbf{c^*} + O(\epsilon^2) \\
&= \lambda^1\epsilon - \lambda^*\epsilon - (\lambda^1\epsilon^T\mathbf{c^*})\mathbf{c^*} + O(\epsilon^2) \\
&= \lambda^1\epsilon - \lambda^*\epsilon + O(\epsilon^2)
\end{aligned}$$

where we have used the facts that $C^T = C$ and that its eigenvectors are mutually orthogonal.

So, ignoring terms of $O(\epsilon^2)$, if $\lambda^1 > \lambda^*$, a perturbation in the direction of $\mathbf{c^1}$ will always be unstable. Therefore, c^* is the principal eigenvector corresponding to the largest eigenvalue of the co-variance matrix.

Theorem 3 *If interneuron i is installed in the network at time t_i, where $t_1 < t_2 < t_3 < ...$, and if the weights into the first $i-1$ interneurons have already converged to the first $i-1$ eigenvectors, the weights of the i^{th} interneuron will converge approximately to the i^{th} eigenvector of the input data's covariance matrix, where such eigenvectors are ordered such that the eigenvalue of vector 1 is the largest, that of vector 2 is next largest and so on.*

Proof

Let interneurons 1,...,M-1 be already connected to the network. We assume that their weights have already converged to the subspace of the first M-1 eigenvectors, and

show that the weights of interneuron M (where M> 1) will converge approximately[4] to the M^{th} eigenvector of the co-variance matrix C.

In this proof, let $\mathbf{W_p}$ be the weight vector associated with the p^{th} interneuron. Then,

$$\begin{aligned}
\langle \Delta \mathbf{W_M} \rangle / \eta &= \langle \mathbf{y} z_M \rangle \\
&= \langle (\mathbf{x} - \mathbf{W}\mathbf{z}) z_M \rangle \\
&= \langle (\mathbf{x} - \sum_{k=1}^{M} z_k \mathbf{W_k}) z_M \rangle \\
&= \langle ((\mathbf{x} - \sum_{k=1}^{M-1} z_k \mathbf{W_k} - z_M \mathbf{W_M}) z_M \rangle \\
&= \langle ((\mathbf{x} - \sum_{k=1}^{M-1} (\mathbf{W_k}^T \mathbf{x}) \mathbf{W_k}) - z_M \mathbf{W_M}) z_M \rangle \\
&= \langle (\mathbf{x}_{M-1}^{\perp} - z_M \mathbf{W_M}) z_M \rangle
\end{aligned}$$

where $x_{M-1}^{\perp}$ is the projection of $\mathbf{x}$ onto the subspace of possible values orthogonal to the first M-1 eigenvectors.

Consider the application of this equivalence to the i^{th} component of W_M, i.e. W_{Mi}, the weight on the connection between y_i and z_M. Then, denoting the i^{th} component of $\mathbf{x}_{M-1}^{\perp}$ by p_i,

$$\begin{aligned}
\langle \Delta W_{Mi} \rangle = 0 &\iff \langle (p_i - z_M W_{Mi}) z_M \rangle = 0 \\
&\iff \langle (p_i - W_{Mi} \sum_k W_{Mk} x_k) \sum_l W_{Ml} x_l \rangle = 0 \\
&\iff \langle \sum_l W_{Ml} x_l p_i - W_{Mi} \sum_{kl} W_{Mk} x_k x_l W_{Ml} \rangle = 0 \qquad (29)
\end{aligned}$$

We note the similarity between this equation and Equation (27) in Theorem 1. For values of x_l within the subspace $\mathbf{x}_{M-1}^{\perp}$, the first term of Equation (29) acts exactly like $p_l p_i$ and so the remainder of Theorems 1 and 2 hold for values of $\mathbf{x}$ restricted to this subspace. For values of $\mathbf{x}$ outwith this subspace, the first term is 0 (x_l is in the subspace whose basis is the first M-1 eigenvectors, p_i is in the orthogonal projection

[4]Approximately,since the proof really requires an infinite convergence time for each weight vector. For a stationary source,x, the finite time intervals used are close to perfect but we can only claim 'approximately' here

of this space) and the second term causes the weights to decrease to zero (recall that $\mathbf{w}^T\mathbf{C}\mathbf{w} = \lambda$ is a scalar).

Therefore we can apply Theorems 1 and 2 to this subspace to show that the M^{th} interneuron weights will converge to the eigenvector corresponding to the largest eigenvalue of this subspace. This eigenvector has eigenvalue smaller than those of the M-1 eigenvectors already allocated to weight vectors $W_1, ...W_{M-1}$ but is larger than any other. Hence this eigenvalue is the Mth largest eigenvalue of the covariance matrix of the original input vector, x.

Therefore, if the result is true for M-1 interneurons, it is true for M interneurons. We know (Theorem 1) that it is true for 1 interneuron. Therefore, the algorithm will force the weights to converge as required.

The Assumptions in the Proofs of Convergence

The proof given above is based on a proof developed by Oja and Karhunen (Oja and Karhunen, 1985) and by Oja *et al* (Oja et al., 1992a; Oja et al., 1992b) for their feedforward networks. The major difficulty with the proof is the step from the stochastic equations (27) which are used in an empirical algorithm to the ordinary differential equations (28) which are solvable as seen above.

Denoting by C_k the covariance matrix of the input data after k presentations of input vectors from the distribution, the proof given in (Oja and Karhunen, 1985) makes 4 critical assumptions:

1. Each C_k is almost surely bounded and symmetric and the C_k are mutually statistically independent with $\langle C_k \rangle = C$ for all k.

2. The eigenvalues of C have unit multiplicity

3. $\eta_k \geq 0, \sum \eta_k^2 < \infty, \sum \eta_k = \infty$

4. Each C_k has a probability density which is bounded away from zero uniformly in k in some neighbourhood of C in R^{n*m}

The first constraint is easiest to satisfy since by taking k large enough we can sample the distribution sufficiently often so that the condition is almost surely satisfied.

The second one cannot be guaranteed for every distribution, however not satisfying it will only result in (a pair of) neurons converging to the subspace which is spanned by the eigenvectors with equal eigenvalues.

The third one is the difficult one to satisfy in any particular stochastic realisation of the algorithm: we are constraining the learning rate in a way which will not be practicable to sustain in any actual simulation - not only must the learning rate converge to zero (which is easy to manage) but it must do so sufficiently slowly that $\sum_j \eta_j$ is infinite. This leads to a long simulation! In practice, it has been found that slow annealing of the learning rate will, under a wide range of annealing schedules, cause the weights to converge to the principal components.

Another way to regard the problem is to say that we have not proved convergence; we have only proved that if the weights converge, they do so in a specific direction. We know also that if the weights reach this direction, they will be stable there but we have not proved that, in any single simulation, they must reach this direction. The proof that they would so converge with probability one uses the fact that each point in the neighbourhood of the attractor is sampled infinitely often.

3.2.1 Alternative Derivations

Derivation from Constrained Variance Maximisation

Attempts have been made to derive the above algorithm from the criterion that we wish to maximise the function $J(\mathbf{W}) = \sum_{i=1}^{M} \langle (\mathbf{w_i}.\mathbf{x})^2 | \mathbf{w_i} \rangle$ which is equal to the variance of the z-values (for zero-mean data).Thus the underlying concept is the maximisation of the information available at the interneurons. In order to keep solutions finite we add the constraint that the weights $\mathbf{w_i}$ must be orthonormal. We use Lagrange multipliers to include this constraint in the function to give

$$J(\mathbf{W}) = \frac{1}{2}\sum_{i=1}^{M} \langle (\mathbf{w_i}.\mathbf{x})^2 | \mathbf{w_i} \rangle + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M} \lambda_{ij}(\mathbf{w_i}.\mathbf{w_j} - \delta_{ij}) \tag{30}$$

where $\delta_{ij} = 1$ if $i = j, \delta_{ij} = 0$ if $i \neq j$. In matrix terms, we may write this as

$$\mathbf{J(W)} = \mathbf{1}^T \frac{1}{2} \langle \mathbf{W x x}^T \mathbf{W} | \mathbf{W} \rangle + \frac{1}{2} tr[\mathbf{\Lambda}(\mathbf{W}^T\mathbf{W} - \mathbf{I}] \tag{31}$$

where tr[] denotes the trace of the matrix, 1 is the vector of 1s, Λ is the matrix whose elements are λ_{ij}, and I is the identity matrix.

Taking derivatives of 30 with respect to the weight vector, $\mathbf{w}_i$, we get

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}_i} = \langle \mathbf{x}\mathbf{x}^T\mathbf{w}_i|\mathbf{w}_i\rangle + \sum_{j=1}^{M}\lambda_{ij}\mathbf{w}_j \tag{32}$$

Since at an optimum, the derivatives must vanish for all i i.e.

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \langle \mathbf{x}\mathbf{x}^T\mathbf{W}|\mathbf{W}\rangle + \mathbf{W}\Lambda = 0 \tag{33}$$

Differentiation of $J(\mathbf{W})$ with respect to the Lagrange multipliers and again finding the point where the derivatives vanish gives

$$\mathbf{W}^T\mathbf{W} = I \tag{34}$$

Premultiplying (33) by $\mathbf{W}^T$ and substituting 34 gives

$$\Lambda = -\mathbf{W}^T < \mathbf{x}\mathbf{x}^T\mathbf{W}|\mathbf{W} > \tag{35}$$

Using this value of Λ in (33) gives

$$\begin{aligned}\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} &= [I - \mathbf{W}\mathbf{W}^T]\langle \mathbf{x}\mathbf{x}^T\mathbf{W}|\mathbf{W}\rangle \\ &= [I - \mathbf{W}\mathbf{W}^T]\mathbf{C}\mathbf{W} \\ &= \mathbf{C}\mathbf{W} - \mathbf{W}\mathbf{W}^T\mathbf{C}\mathbf{W}\end{aligned} \tag{36}$$

As we have seen above, this equation completely defines the learning of the interneuron network. Therefore the interneuron network may be thought of as maximising the value of the function $\sum_{i=1}^{M}\langle(\mathbf{w_i}.\mathbf{x})^2|\mathbf{w_i}\rangle$ under the stated orthonormality constraints.

However this derivation, too, is not secure because we have used the converged value of Λ during the convergence process. Indeed, Baldi and Hornik (Baldi and Hornik, 1988) have shown that this algorithm is not derivable from such a gradient-descent procedure. The effect of the approximation is discussed in more detail in a slightly more general setting in Chapter 6.

Derivation from Error Minimisation

Attempts have also been made to derive the algorithm by minimising the error, e, at y after the interneuron activation is returned.

We wish to minimise

$$J(\mathbf{W}) = \frac{1}{2}\mathbf{1}^T\langle \mathbf{e}^2|\mathbf{W}\rangle = \frac{1}{2}\mathbf{1}^T\langle (\mathbf{x} - \mathbf{W}\mathbf{W}^T\mathbf{x})^2|\mathbf{W}\rangle \tag{37}$$

where **1** is the vector of 1s.

Consider the j^{th} component of the reconstruction error, e_j.

$$e_j = x_j - \sum_{i=1}^{M} w_{ij}\mathbf{w_i}.\mathbf{x} \tag{38}$$

where, as before, $\mathbf{w_i}$ is the vector of weights into the i^{th} interneuron. Then we wish to find stationary point(s) of the derivative of $J(\mathbf{W})$ i.e. where

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}_m} = \sum_{j=1}^{M} e_j \frac{\partial e_j}{\partial \mathbf{w}_m} = 0 \tag{39}$$

Now,

$$\frac{\partial e_j}{\partial \mathbf{w}_m} = -w_{mj}\mathbf{x} - (\mathbf{w_m}.\mathbf{x})[0, 0, ..., 1, 0, ..0]^T \tag{40}$$

where the last vector has a 1 in only the j^{th} position. Then,

$$\begin{aligned} \frac{\partial J(\mathbf{W})}{\partial \mathbf{w}_m} &= -\sum_{j=1}^{M}(x_j - \sum_{i=1}^{M} w_{ij}\mathbf{w_i}.\mathbf{x}).\{w_{mj}\mathbf{x} + \mathbf{w_m}.\mathbf{x}[0, 0, .., 1, 0.., 0]^T)\} \\ &= -(\mathbf{x} - \mathbf{W}^T\mathbf{W}\mathbf{x})\mathbf{w}_m.\mathbf{x} - (\mathbf{x} - \mathbf{W}^T.\mathbf{W}\mathbf{x})(\mathbf{w}_m.x)\mathbf{1}^T \end{aligned} \tag{41}$$

This can be used in the usual way in the gradient descent algorithm

$$\Delta\mathbf{W} \propto -\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

to give a learning rule

$$\Delta\mathbf{W} = \mathbf{x}^T\mathbf{x}(\mathbf{I} - \mathbf{W}^T\mathbf{W})\mathbf{W} + (\mathbf{x} - \mathbf{W}^T\mathbf{W}\mathbf{x})(\mathbf{W}\mathbf{x})^T \tag{42}$$

Now while this last equation is not quite the algorithm we wished, Xu (Xu, 1993) has shown that "on the average", the scalar product of our algorithm and the above learning rule is positive. Thus "on the average", the interneuron network can be thought of as minimising the residuals at **y**.

Derivation from Statistical Mechanics

Recently, several authors have investigated PCA-type neural networks from statistical mechanics considerations e.g. (Prugel-Bennett and Shapiro, 1993; Biehl and Mietzner, 1993; Biehl, 1993; Shapiro and Prugel-Bennett, 1992) in an attempt to find an absolutely secure derivation of convergence. We will not consider such investigations in detail but merely note that such derivations always rely on arguments using infinity - in their case, infinitely large networks. Such derivations then may be analytically sound but the step of equating them with actual implementations must remain suspect.

In summary, then, we have analytically derived the PCA properties of the network but have had to rely on arguments which use approximations at some point. This leaves the possibility that some particular network - operating in the real world under constraints of finiteness (of time, magnitude etc.) - will not converge from a specific set of initial conditions while being trained on a particular set of data. In practice, this does not seem to be a problem - we have yet to find a case where a network did not converge for any data-set.

3.3 Network Properties

In this section, we investigate empirically some of the emergent properties of the interneuron network. We view these properties as emergent properties as we do not believe that they could be expected *a priori* to exist , i.e. without a detailed investigation of the network.

3.3.1 Plasticity and Continuity

The results reported in the last section were based on a model which suggested that only a new interneuron could learn. The underlying assumptions are

- an interneuron can only learn during a special period of its existence
- only one interneuron can learn at any instant in time

Disjoint	Learner	Model	Contin.	Learner	Model
W			W		
1.000	-0.036	-0.008	**1.000**	-0.015	-0.014
0.036	**0.999**	-0.018	0.014	**0.999**	0.045
0.010	0.018	**1.000**	-0.015	-0.046	**0.999**
-0.002	-0.002	0.016	-0.015	0.006	-0.021
0.010	0.003	0.010	0.003	-0.007	0.010
W^TW			W^TW		
1.001	0.000	0.000	**1.000**	0.000	0.000
0.000	**1.000**	0.000	0.000	**1.000**	0.000
0.000	0.000	**1.000**	0.000	0.000	**1.000**

Table 3: Results from the interneuron network in which each interneuron stopped learning as a new one was created(left) and from the network in which each interneuron continued to learn (right).

These are clearly not good properties for biological learners to have; we do not wish to have new learning remove the hard-won gains already achieved from previous learning; but equally, we do not wish to have to specify in advance how much time each neuron will have to learn. Further, in setting a specific time period during which learning will take place, we are providing the system with a form of meta information.

To test the effects of allowing interneurons to continue to learn even after other new interneurons were created, two more simulations were carried out. In the first, the interneurons lost their plasticity gradually and there was an overlap in the times when two or more interneurons were learning; in the second, interneurons kept their plasticity throughout.

Thus, in this last model, the first interneuron learns from its creation till the end of the simulation, the second interneuron learns from its creation at iteration 13000 till the end of the simulation and the last interneuron learns from iteration 26000 till the end of simulation.

Only the results of the last model are reported, as the conclusions are identical: we do not have to postulate that interneuron weights lose their plasticity. The left matrix of Table 3 repeats the results from the interneuron model described in the previous section; the results from interneurons which continue learning are shown on the right. The table shows that the interneurons can retain their plasticity without

there being major loss of precision in finding the actual principal components.

We suggest that this model then represents a more plausible model of the form of learning which takes place in biological learners and further, that in most cases of unsupervised learning, the Continuing Learning Model is to be preferred.

3.3.2 Speed of Learning and Information Content

One of the most interesting aspects of the proposed model is its reaction to statistical data which have inherently differing amounts of information. One might hope that a model would react to data which has more information more quickly than it does to data with less. This, in fact, happens.

It is well known (e.g.(Cover and Thomas, 1991), page 225) that for the Shannon information content (Shannon, 1948) of a Gaussian with variance σ^2

$$h(\gamma) \propto \log \sigma$$

which is a mathematical formulation of the fact that there is more information in random variables with large variance than in random variables with small variance. It would seem plausible to argue that an organism which can quickly identify data-sources with large information content would have an advantage over an organism which does not have this ability. This is, in fact, an emergent property of the model.

Therefore, in the current set of experiments, there is more information in x_1 than in x_2 etc. i.e. $h(x_1) > h(x_2) > h(x_3) > h(x_4) > h(x_5)$. We therefore hope that x_1 will be learned quickest, etc.

Figure 8 shows the length of time which individual interneurons take to converge to the appropriate solution. The first solid line on the graph shows how long the first interneuron took to converge to (1,0,0,0,0), the second to (0,1,0,0,0) and the third to (0,0,1,0,0).

Additional experiments to ensure that this rate was not merely a function of the order of the interneuron's learning confirm that data with larger variances is learned more quickly.

Clearly, interneuron 1 is the fastest learner; it learns the component/direction with the largest information content. Interneuron 2 makes a bad start; actually,

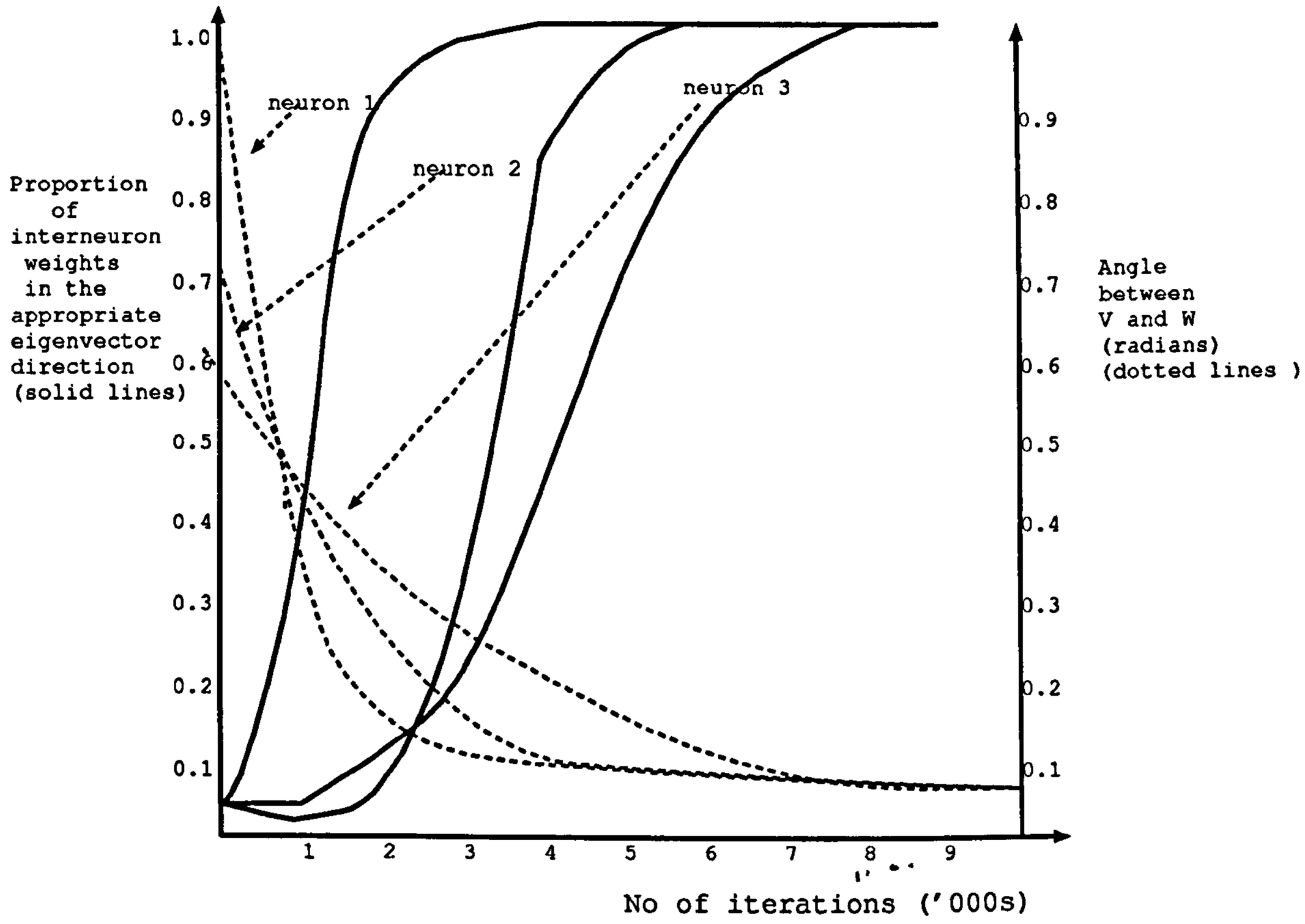


Figure 8: Interneuron convergence times.
Solid lines show the time for each interneuron to converge to the appropriate eigenvector of the covariance matrix of the input data: the first interneuron converges most quickly.
Dotted lines show the speed with which individual interneuron's feedforward and feedback weights in the VW model converge to the same direction: those of the first interneuron converge most quickly.

because of the initial conditions (which were randomly generated), interneuron 2 attempts initially to make a play for the first eigenvector direction before giving up the unequal struggle against interneuron 1. It then converges quickly to the appropriate eigenvector. That this is not a necessary property of PCA networks is shown in (Diamantaras, 1992), Figure 2.7 where the network takes longest to converge to the first eigenvector.

3.4 The VW Model

The results of the last section have one major drawback when considered as a model of biological systems: the weights of the connections from the interneuron,z, to the summing neuron,y, are assumed to be identical to those from the summing neuron,y, to the interneuron,z. This is biologically implausible. We now propose a model where these weights are initially different.

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z} \tag{43}$$

$$\mathbf{z} = \mathbf{W}\mathbf{y} = \mathbf{W}\mathbf{x} \tag{44}$$

$$\Delta\mathbf{W} = \alpha_w \mathbf{y}\mathbf{z}^T \tag{45}$$

$$\Delta\mathbf{V}^T = \alpha_v \mathbf{y}\mathbf{z}^T \tag{46}$$

where the initial values of both $\mathbf{V}^T$ and $\mathbf{W}$ are small random numbers not correlated in any way with each other.

Note that both learning rules for $\mathbf{W}$ and $\mathbf{V}$ are identical up to the learning rate and use only simple Hebbian learning.

The convention we will use here is that w_{ji} is the weight of the connection from y_j to z_i; similarly, v_{ij} is the weight of the connection from z_i to y_j. Unless specifically stated otherwise, we shall be interested in the vectors to and from the interneurons. Therefore we take the vectors $\mathbf{v_i}$ to be the weight vector into the i^{th} interneuron, i.e. to be the vector of form $\{v_{ik}\}$ for all k; similarly we take the vector $\mathbf{w_i}$ to be the vector of weights from the i^{th} interneuron i.e. to be the vector $\{w_{ki}\}$ for all k; we note here that $\mathbf{v_i}$ corresponds to a column of the matrix V of weights while $\mathbf{w_i}$ is a

row of W. Both are vectors of length n where n is the number of summing neurons. We consider the effect of these rules on a network with a single interneuron,z.

Lemma 1

*If the weights,***w***, of a single interneuron with the above learning rules converge to an eigenvector of the input data co-variance matrix, then the weights* **w** *and the weights* **v** *converge to the same eigenvector.*

Proof

Let w_i be the weight of the connections from y_i to z and and v_i be that from z to y_i.

If the weights of a single interneuron converges to a limit, the expected weight change over a sufficiently long time will tend to zero. Given the usual approximations, particularly regarding the learning rate η, and using $\langle x \rangle$ to indicate the average value of x over the time period,

$$\begin{aligned}
\langle \Delta w_i \rangle = 0 &\iff \langle \eta y_i z \rangle = 0 \\
&\iff \langle y_i z \rangle = 0 \\
&\iff \langle (x_i - v_i z) z \rangle = 0 \\
&\iff \langle (x_i - v_i \sum_k w_k x_k) \sum_l w_l x_l \rangle = 0 \\
&\iff \langle \sum_l w_l x_l x_i - v_i \sum_{k,l} w_k x_k x_l w_l \rangle = 0 \\
&\iff \sum_l w_l C_{li} - v_i \sum_{k,l} w_k C_{kl} w_l = 0
\end{aligned}$$

where C_{ij} is that element of the co-variance matrix of the input data **x** showing the co-variance between the i^{th} and j^{th} elements. We note that the same criterion may be deduced from $\langle \Delta v_i \rangle = 0$. If the weights of the interneuron are to converge, then the above must be true for all values of w_i. Therefore it may be written in matrix notation as

$$\begin{aligned}
\langle \Delta \mathbf{w} \rangle = \mathbf{0} &\iff \mathbf{Cw} - (\mathbf{w}^T \mathbf{Cw})\mathbf{v} = \mathbf{0} \\
&\iff \mathbf{Cw} = (\mathbf{w}^T \mathbf{Cw})\mathbf{v}
\end{aligned}$$

Now it is a standard result that the co-variance matrix C is positive-semidefinite; and

Inter	neuron	model	VW	Model				
W			W			V		
1.000	-0.036	-0.008	**0.985**	-0.041	-0.003	**1.013**	-0.017	-0.024
0.036	**0.999**	-0.018	-0.019	**1.033**	0.031	-0.027	**0.965**	0.032
0.010	0.018	**1.000**	0.022	-0.032	**1.028**	0.020	-0.017	**0.969**
-0.002	-0.002	0.016	-0.024	-0.041	0.038	-0.007	-0.034	0.037
0.010	0.003	0.010	0.098	-0.007	-0.011	0.010	0.000	0.002

Table 4: Results from the interneuron network (left) with symmetric weights,W. and for the V and W vectors from the VW Model(see text)

hence

$$\mathbf{w}^T\mathbf{C}\mathbf{w} = \gamma \geq 0 \tag{47}$$

where γ is a non-negative real number. Hence ,

$$\mathbf{C}\mathbf{w} = \gamma\mathbf{v} \tag{48}$$

Therefore, if $\mathbf{w}$ converges to an eigenvector of C (see below), then $\mathbf{C}\mathbf{w} = \lambda\mathbf{w}$ for some real number,λ, and so $\mathbf{v} = \alpha\mathbf{w}$, where α is a scalar; that is, $\mathbf{v}$ and $\mathbf{w}$ converge to the same eigenvector. Therefore, it is possible to apply the further analysis developed for the WW network and hence show that the i^{th} interneuron converges to the i^{th} eigenvector of the covariance matrix.

Experimental results, shown in Table 4 confirm this. It can be seen that both $\mathbf{v}$ and $\mathbf{w}$ converge to the same eigenvector, although the results are slightly less clear cut that in the previous algorithm. However, given the simplicity of this biologically inspired model, the results are extremely clear: any entity which used such a method would be able to extract the greatest amount of information from its environment with a minimal amount of interneurons using a very simple learning rule.

However there remains the possibility that the weight $\mathbf{w}$ will not converge to an eigenvector. Therefore, the next theorem is necessary.

Theorem 4 *If the weights,*$\mathbf{w}$*, of a single interneuron with the above learning rules converge, then the weights* $\mathbf{w}$ *and the weights* $\mathbf{v}$ *converge to the same eigenvector of the input data's covariance matrix.*

From the lemma, we know that the weights converge as stated if they converge to an eigenvector. Therefore now, we must prove that if w converges, it does so to an eigenvector of C. We use a contradiction argument.

Assume that there is a solution of

$$\mathbf{Cw} = \gamma \mathbf{v} \tag{49}$$

where $\mathbf{w}$ is not an eigenvector nor the degenerate solution,$\mathbf{w} = \mathbf{0}$.

Let the eigenvectors of $\mathbf{C}$ be $\mathbf{c_1}, \mathbf{c_2}, ..., \mathbf{c_n}$. Then

$$\mathbf{w} = \sum_{i=1}^{n} w_i \mathbf{c_i}$$

Since $\mathbf{w} \neq 0$, there exists a direction $\mathbf{c_b}$, such that $w_b \neq 0$. Since $\mathbf{w}$ is not an eigenvector, there exists 1 other direction $\mathbf{c_a}$ with a non-zero component w_a.

Then

$$\mathbf{w} = w_a \mathbf{c_a} + w_b \mathbf{c_b} + \sum_{i \neq a,b} w_i \mathbf{c_i}$$

where $1 \leq a, b \leq n, a \neq b$, and

$$\mathbf{v} = v_a \mathbf{c_a} + v_b \mathbf{c_b} + \sum_{i \neq a,b} v_i \mathbf{c_i}$$

and from Equation 49,

$$\lambda_b w_b = \gamma v_b$$

$$\lambda_a w_a = \gamma v_a$$

Consider a disturbance of magnitude $\epsilon > 0$ in the direction of $\mathbf{c_a}$ i.e. a disturbance of $\epsilon_\mathbf{a}$. Then if $\mathbf{w}$ is a stable point of convergence of the weights, the expected change in the weights over time is zero. Therefore,

$$\begin{aligned}
\langle \Delta \mathbf{w} \rangle = 0 \iff & \mathbf{Cw} - (\mathbf{w}^T \mathbf{Cw})\mathbf{v} = 0 \\
\iff & \mathbf{C}(w_a \mathbf{c_a} + w_b \mathbf{c_b} + \epsilon_\mathbf{a} + \sum_{i \neq a,b} w_i \mathbf{c_i}) - \gamma'(v_a \mathbf{c_a} + v_b \mathbf{c_b} + \sum_{i \neq a,b} v_i \mathbf{c_i}) = 0 \\
\iff & \lambda_a w_a \mathbf{c_a} + \lambda_b w_b \mathbf{c_b} + \lambda_a \epsilon_\mathbf{a} + \sum_{i \neq a,b} \lambda_i w_i \mathbf{c_i} \\
& -\gamma' v_a \mathbf{c_a} - \gamma' v_b \mathbf{c_b} - \gamma' \sum_{i \neq a,b} v_i \mathbf{c_i} = 0
\end{aligned}$$

where $\gamma' = (\mathbf{w} + \epsilon_\mathbf{a})^T \mathbf{C}(\mathbf{w} + \epsilon_\mathbf{a}) \geq 0$ since $\mathbf{C}$ is a positive semi-definite matrix.

Now, considering the components of the transformation in the direction of $\mathbf{c_b}$,

$$\lambda_b w_b - \gamma' v_b = 0$$

Then, $\gamma v_b - \gamma' v_b = 0$

Therefore, $\gamma = \gamma'$ since $v_b \neq 0$. Now, considering the components of the transformation in the direction of $\mathbf{c_a}$,

$$\lambda_a w_a + \lambda_a \epsilon - \gamma' v_a = 0$$

$$\gamma v_a - \gamma v_a + \lambda_a \epsilon = 0$$

$$\lambda_a \epsilon = 0$$

which is a contradiction. Hence there does not exist a non-zero, non-eigenvector solution to equation (49).

Theorem 5 *At equilibrium, the weights* $\mathbf{v}$ *and* $\mathbf{w}$ *converge to the same eigenvector,* $\mathbf{c_a}$ *with*

$$\mathbf{v} = \frac{1}{|\mathbf{w}|^2}\mathbf{w} \tag{50}$$

Proof

At equilibrium,

$$\mathbf{Cw} = (\mathbf{w}^T\mathbf{Cw})\mathbf{v} = \gamma\mathbf{v}$$

and, by theorem 4, $\mathbf{w}$ is an eigenvector of $\mathbf{C}$, $\mathbf{c_a}$. Therefore,

$$\mathbf{Cw} = \lambda_a \mathbf{w}$$

where λ_a is the eigenvalue corresponding to eigenvector $\mathbf{c_a}$.

Therefore, $\lambda_a \mathbf{w} = \gamma \mathbf{v}$

Therefore, $\mathbf{v} = \frac{\lambda_a}{\gamma}\mathbf{w} = \frac{\lambda_a}{\mathbf{w^T Cw}}\mathbf{w}$

Now, $\mathbf{w^T C w}$ is a scalar; hence,

$$\mathbf{w^T C w} = |\mathbf{w^T C w}| = |\mathbf{w}||\mathbf{C w}| = |\mathbf{w}||\lambda_a \mathbf{w}| = \lambda_a |\mathbf{w}|^2$$

Therefore,

$$\mathbf{v} = \frac{1}{|\mathbf{w}|^2}\mathbf{w}$$

Note: The theorems in this section imply that the further analysis of this interneuron network is identical to that performed previously for the WW network. In other words, an interneuron network with asymmetric weights, $\mathbf{w_i}$ and $\mathbf{v_i}$, will calculate the principal components if the interneurons are created in the network as in the previous section.

3.4.1 Properties of the VW Network

The motivation for the introduction of the VW model is that it removes a constraint from the network builder: in the WW model, the weights into and out of each interneuron must be the same and so must be known in a meta-sense i.e. outwith the learning space. One feature of symmetry still remaining in the network is the equivalence of the learning rates in the V and W weights.

Experimental results show that, when $\mathbf{v}$ and $\mathbf{w}$ learn with different rates, the angle between $\mathbf{v}$ and $\mathbf{w}$ converges as quickly as before but the weight, $\mathbf{v}$ or $\mathbf{w}$, with the larger learning rate acquires a larger length than the other. Indeed the result of the last theorem still applies.

While most of the emergent properties of the symmetric (WW) network still are found with the VW network, there is one property which this network does not have: the interneurons cannot retain their plasticity when new interneurons are created.

There always remains a slight angle between $\mathbf{v}$ and $\mathbf{w}$ (dotted lines in Figure 8); even although this can be made arbitrarily small, it is sufficient to destabilise the interneuron weights. It is not possible for the weights $\mathbf{v}$ and $\mathbf{w}$ to be both exactly orthogonal to any new interneuron's weights; therefore the new interneuron will destabilise the weights of existing interneurons. The interaction between $\mathbf{v}$ and

w will further move the weights away from the eigenvector and so the weights will be rotated in the principal subspace. This is also an empirical finding.

Therefore, for VW interneurons, each interneuron's weights must be allowed to converge to the eigenvector but must then loose their plasticity. This is algorithmically easy to implement but the need to take this action has led to a search for other algorithms.

3.5 Relation to Other Models and Biology

The basic interneuron network has been discovered independently by other researchers in the past - though the link between them and the network described above is not often obvious.

The first reference to an interneuron-type of network appears to be William's Symmetric Error Correction (SEC) Network (Williams, 1985) where the residuals at y were used in a symmetric manner to change the network weights. The SEC network may be easily shown to be equivalent to the network described in this Chapter.

A second reference to an interneuron-type network was given in (Levin, 1990). Levin introduces a network very similar to Plumbley's network and investigates its noise resistant properties. He develops a rule for finding the optimal converged properties and, in passing, shows that it can be implemented using simple Hebbian learning. His derivation is a simplified version of that given in Section 3.2.1 and must equally be described as approximative.

A third strand has been the adaption of simple Elman nets((Hinton and Shallice, 1991; Kehagias, 1991; Elman, 1991; Elman, 1992; Bates and Elman, 1992)) which have a feedforward architecture but with a feedback from the central hidden layer to a "context layer". Typically, the Elman nets use an error-descent method to learn, however Dennis and Wiles (Dennis and Wiles, 1993; Dennis et al., 1992) have modified the network so that the feedback connection uses Hebbian learning. However, the Hebbian part of the network uses weight decay to stop uncontrolled weight growth and the other parts of the network continue to use back propagation of errors to learn.

More recently, Xu (Xu, 1993) has rediscovered the interneuron network and has

given a very strong analysis of its properties. While he begins by considering the dynamic properties of a multi-layer network (all post-input layers use negative feedback of activation), it is clear from his discussion that the single layer model which he investigates in detail is identical to the network outlined above.

An interesting feature is Xu's empirical investigation into using a sigmoid activation function at the interneurons; he reveals results which show that the network is performing a PCA and suggests that this feature enabled the network to be more robust i.e. resistant to outliers, a finding in agreement with other researchers (e.g. (Karhunen and Joutsensalo, 1993a; Oja et al., 1991; Oja and Karhunen, 1993)). We will return to non-linearity in Chapter 6 where we will base our findings in contemporary statistical practice in a way which permits an analytical investigation of non-linearity.

In keeping with our overall aim, we would like to link our networks with those of biology. The overall aim for an early processing network has been described as the minimisation of redundancy so that the further network can be developed as a "suspicious coincidence" (Barlow, 1989) detector. The decorrelation of inputs formed by projection onto the Principal Components clearly achieves this. The network most like that described above was devised by Ambrose-Ingerson *et al* (Ambrose-Ingerson et al., 1990) in which a network which uses negative feedback between layers attempts to simulate the transfer of olefactory information in the paleocortex. The sole difference between that network and the interneuron network is that the network uses a competitive activation transfer arrangement; the authors conjecture that a form of PCA is taking place.

Murphy and Sillito (Murphy and Sillito, 1987) have shown that LGN neurons seem to be inhibited by the V1 cells (in the visual cortex) which they excite. Pece (Pece, 1992) has developed a model based on negative feedback which simulates the reduction in redundancy in an information-transferring network.

As an interesting aside, we note that Robinson (Robinson, 1987) has shown that negative feedback cannot be used to control the visuomotor system in a continuously operating closed-loop system with a finite delay term. He shows that the negative feedback in the system can be made stable if the system is refractory: each eye saccade

is followed by a short period when it will not respond to another change in target position (due to the sampling rate having a finite frequency). Because of this, we can think of such a system as running on open-loop dynamics for much of the time which is equivalent to having discrete time intervals in which activation is passed forward and back. It is results like this which underlie our conviction that the interneuron network is based on cybernetic principles.

A network very like the network which we have investigated, has been developed in (Jonker, 1992) in which inhibition is specifically used in an Artificial Neural Network to model the cerebellum. The network appears identical to that displayed in Figure 4 but is considered as a dynamic model where the activation is allowed to pass round the network till settling takes place. However, since Jonker makes "the biologically plausible assumption that the characteristic time-scales in the evolution of interactions are much larger than the time-scales involved in the neuronal dynamics",((Jonker, 1992) ,page 87)it is not surprising that the emergent properties of the network are very similar to those which are developed in the next section from a static network.

Chapter 4

Peer-Inhibitory Interneurons

4.1 Parallel Learning Networks

Three factors make the interneuron network especially exciting as a PCA network:

simplicity - there are no logistic or hyperbolic functions to be calculated; there is no additional computation within the learning rule; there is no sequential passing back of errors or decay terms.

homogeneity - every interneuron is performing exactly the same calculation as its neighbours; every summing neuron is performing exactly the same calculation as its neighbours.

locality of information - each synapse uses only the information which it receives from its own connections; similarly with the summing neurons which calculate the y values

However, the phased creation of neurons described in the last chapter does not utilise the inherent potential of this network for parallel information processing. We now develop learning algorithms which do this while retaining as much as possible of the other features.

Thus, in this chapter[1], we create the entire network at one instant in time and train all weights simultaneously. Recall that when we do this with the first interneuron

[1]Some of this work has already appeared in (Fyfe, 1993f; Fyfe, 1993b).

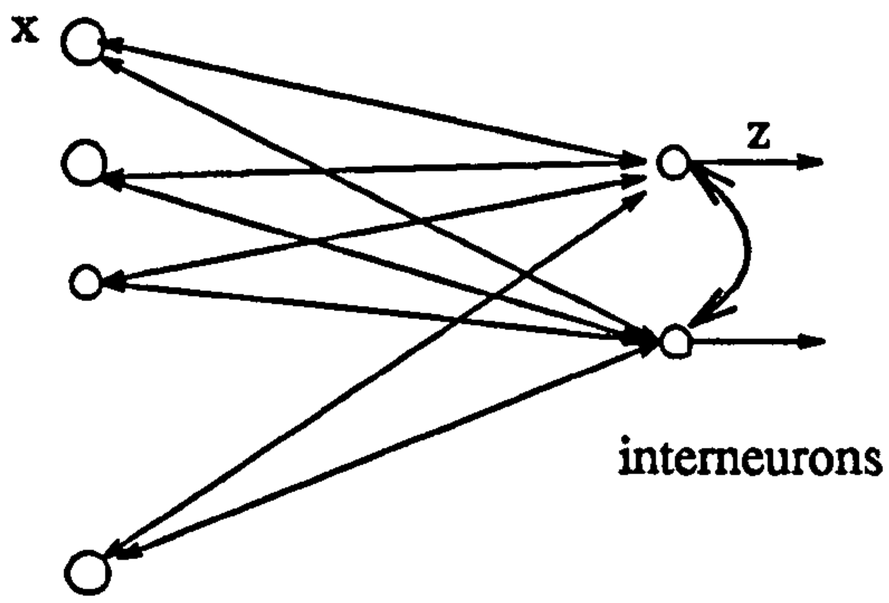


Figure 9: The inputs, x, are passed forward to the interneurons via the W weights as before but before the interneuron's activations are passed back, the activation from each interneuron is passed to the others as inhibition

network, we find the principal subspace but not the principal components themselves.

We amend the basic network by allowing the inhibitory effect of each interneuron to act on the other interneurons as well as the summing neurons(see Figure 9). Two methods will be used with this amended network in order to create the necessary asymmetry: in the first, we will allow the network weights to be upgraded at different rates; in the second, we will use different activation functions to force convergence to the Principal Components.

The first type of network will be characterised by

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z} \tag{51}$$

$$\mathbf{z}' = \mathbf{W}\mathbf{y} = \mathbf{W}\mathbf{x} \tag{52}$$

$$\mathbf{z} = \mathbf{z}' - \mathbf{U}\mathbf{z}' \tag{53}$$

$$\Delta\mathbf{W} = \eta_w \mathbf{y}\mathbf{z}^T \tag{54}$$

$$\Delta\mathbf{V} = \eta_v \mathbf{y}\mathbf{z}^T \tag{55}$$

$$\Delta\mathbf{U} = \gamma \mathbf{z}\mathbf{z}^T \tag{56}$$

where $\mathbf{z}'$ is the initial activation of the interneuron before receiving the lateral inhibition from other interneurons and $\mathbf{U}$ is the matrix of weights between the interneurons. As before, the initial input vector $\mathbf{x}$ is fedforward through the W weights to the interneurons. Now the interneurons feed their activation (as inhibition) to the other

interneurons through the U weights before the interneurons' activations are fed back as inhibition (through the V weights) to the summing neurons.

We do not however allow self-connections from interneurons to themselves.

We note that we have now a 3-phase operation:

1. The activation is fed forward from the summing neurons to the interneurons
2. The interneurons feed their activation to their peers and recalculate their activations
3. The activation is fed back to the summing neurons from the interneurons

While this is more computationally complex than before, we only require $O(m^2)$ additional calculations, where m is the number of interneurons. Further all learning processes continue to use simple Hebbian learning.

We will introduce a matrix G(x)=(I-U)W(x),[2] which represents the forward function from **x** to **z**. G is an integral part of the mathematical model which we will use for understanding the network but it makes no overt contribution to the development of the network in the real, stochastic world. The actual learning in the network i.e. the weight updates, is accomplished by updating the actual weights U,V and W although we will discuss $\frac{dG}{dt}$ as though it were being performed in the same sense that e.g. $\frac{dW}{dt}$ is performed.

We can prove (an obvious special case of Theorem 7) that the learning rules detailed above are equivalent to

$$\frac{dV^T}{dt} = \frac{dW}{dt} = (I-U)WC - (I-U)WCW^T(I-U)^TV^T \quad (57)$$

$$\frac{dU}{dt} = (I-U)WCW^T(I-U)^T \quad (58)$$

$$\frac{dG}{dt} = (I-U)\frac{dW}{dt} - \frac{dU}{dt}W \quad (59)$$

where G is the forward function relating **x** and **z** and C is the covariance matrix of the input data.

[2] I being the identity matrix

We will show, as with other models with lateral inhibition, that U = 0 is a stable stationary point of the system

$$\begin{aligned}
\text{Now, } G &= (I-U)W \\
\text{and so } \frac{dG}{dt} &= (I-U)\frac{dW}{dt} - \frac{dU}{dt}W \\
&= (I-U)\{(I-U)WC - (I-U)WCW^T(I-U)^TV^T\} \\
&\quad -(I-U)WCW^T(I-U)^TW \\
&= (I-U)\{GC - GCG^TV^T\} - GCG^TW \\
&\rightarrow GC - GCG^TV^T - GCG^TW \\
&\quad \text{as U} \rightarrow 0
\end{aligned}$$

Now G $\rightarrow W$ as U $\rightarrow$ 0 and so $\frac{dG}{dt} \rightarrow WC - 2WCW^TW$ using the fact that $V^T = W$.

It can be seen that the necessary asymmetry between the Hebbian learning term and the weight decay term has not been achieved; however the important point to note is that part of the weight decay term comes from the $\frac{dU}{dt}$ term which we can manipulate independently of the $\frac{dW}{dt}$ term in order to create the necessary asymmetry.

4.2 Analysis of Differential Learning Rates

$$\mathbf{y} = \mathbf{x} - \mathbf{Vz} \tag{60}$$

$$\mathbf{z}' = \mathbf{Wy} = \mathbf{Wx} \tag{61}$$

$$\mathbf{z} = \mathbf{z}' - \mathbf{Uz}' \tag{62}$$

$$\mathbf{\Delta W} = \eta_W \mathbf{z}\mathbf{y}^T \tag{63}$$

$$\mathbf{\Delta V^T} = \eta_V \mathbf{z}\mathbf{y}^T \tag{64}$$

$$\mathbf{\Delta U} = \Gamma \mathbf{z}\mathbf{z}^T \tag{65}$$

Let us review our naming conventions: the convention we will use is that w_{ij} is the weight of the connection from summing neuron y_j to interneuron z_i; similarly, v_{ji} is the weight of the connection from z_i to y_j; u_{ij} is the weight of the connection from z_j to z_i. Unless specifically stated otherwise, we shall be interested in the vectors to and from

the interneurons. Therefore we take the vectors $\mathbf{v_i}$ to be the weight vector into the i^{th} interneuron, i.e. to be the vector of form $\{v_{ki}\}$ for all k; similarly we take the vector $\mathbf{w_i}$ to be the vector of weights from the i^{th} interneuron i.e. to be the vector $\{w_{ik}\}$ for all k. Both are vectors of length n where n is the number of summing neurons. Note that the learning rates of U values are different for different interneurons as we wish to force the first interneuron to learn the first principal component, the second the next and so on. Thus we have a diagonal matrix , $\mathbf{\Gamma}$.

Since we ensure that there are no self-connections, the main diagonal of U is composed of zeros. Also note that Γ is the matrix $\text{diag}\{\gamma_1, \gamma_2, ..., \gamma_m\}$ where m is the number of interneurons and γ_i is the learning rate for the U weights of the i^{th} interneuron such that $\gamma_1 < \gamma_2 < ... < \gamma_m$. We allow all learning rates to decrement to zero as time tends to infinity.

As introduced in the previous Section, G(x)=(I-U)W(x) is the forward function from $\mathbf{x}$ to $\mathbf{z}$. We will assume that, if $\gamma_i(t)$ is the value of γ_i during time interval t, $\lim_{t\to 0} \frac{\gamma_i(t)}{\eta_w(t)}$ exists and is positive. This assumption will be discussed in Section 4.2.1.

Theorem 6 *$\mathbf{v_i}$ converges if and only if $\mathbf{w_i}$ converges, where $\mathbf{v_i}$ is the weight vector from the i^{th} interneuron and $\mathbf{w_i}$ is the weight vector into the i^{th} interneuron. Further,*

$$\mathbf{v_i} = a\mathbf{w_i} + \mathbf{p}$$

where $a = \lim_{t\to 0} \frac{\eta_V(t)}{\eta_W(t)}$,
$\eta_V(t)$ is the value of η_V during time interval t
and $\mathbf{p}$ *is a vector depending on the initial conditions of $\mathbf{v_i}$ and $\mathbf{w_i}$.*

Proof

At time B, we have

$$w_{ji}(B) = w_{ji}(B-1) + \eta_{ji}(B)y_j(B)z_i(B)$$

If we start from time 0, we can equate the continuous time point T with the sum of the discrete intervals η_{ji}. i.e.

$$T = \sum_{p=0}^{B} \eta_{ji}(p)$$

Thus, we are breaking up continuous time into discrete time steps η_{ji} . Now the convergence, if it exists must be taking place simultaneously over all weights. Therefore, we must ensure that $\eta_{ji} = \eta$ for all values of i, j. In order to have no limit on continuous time, we must have

$$\sum_{p=0}^{\infty} \eta(p) = \infty$$

If we further assume that $\eta(p) > 0$ for all p, then we have

$$\begin{aligned} \Delta w_{ij} &= \eta y_j z_i \\ &= \eta(x_j - \sum_s v_{js} z_s) z_i \\ &= \eta(x_j - \sum_s v_{js}(\sum_l w_{sl} x_l - \sum_p u_{sp} \sum_l w_{pl} x_l))(\sum_t w_{it} x_t - \sum_q u_{iq} \sum_r w_{qr} x_r) \end{aligned}$$

Or, in matrix terms,

$$\begin{aligned} \frac{W(B) - W(B-1)}{\eta(B)} &= (I - U(B))W(B)\mathbf{x}(\mathbf{B})\mathbf{x}(\mathbf{B})^T \\ &\quad -(I - U(B))W(B)\mathbf{x}(\mathbf{B})\mathbf{x}(\mathbf{B})^T W(B)^T (I - U(B))^T V(B)^T \end{aligned} \tag{66}$$

If we also assume that

$$lim_{p \to \infty} \eta(p) = 0$$

then the sequence of $w_{ji}(T)$ asymptotically approaches a continuous-time function and the left-hand side of Equation (66) approaches its derivative. Then we can replace Equation (66) with the corresponding averaged differential equation

$$\frac{dW}{dt} = (I - U)WC - (I - U)WCW^T(I - U)^T V^T \tag{67}$$

where C is the covariance matrix of the stationary distribution producing the x_k values Now, under the same assumptions as in the previous chapter about the rate η it can be shown that the solution of the stochastic algorithm approaches the solution of the differential equation (67) with probability 1.

Now consider **v**'s learning.

$$v_{ij}(B) = v_{ij}(B-1) + \eta v(B) y_j(B) z_i(B)$$

Therefore,

$$\frac{v_{ij}(B)-v_{ij}(B-1)}{\eta(B)} = \frac{\eta v(B)}{\eta(B)}\{\sum_l w_{li}(B)x_l(B)x_j(B)(1-\sum_q u_{iq}) - \sum_k v_{kj}(B) * (1-\sum_q u_{iq})\sum_{p,l} w_{pk}(B)x_p(B)x_l(B)w_{li}(B)(1-\sum_p u_{sp})\}$$

Given the same assumtions as before and making the additional assumption that

$\lim_{p\to 0}\frac{\eta v(p)}{\eta(p)} = a > 0$ i.e. the limit exists and is positive

we have the corresponding differential equation,

$$\frac{dV^T}{dt} = a((I-U)WC - (I-U)WCW^T(I-U)^TV^T) \tag{68}$$

Therefore,

$$\frac{dV^T}{dt} = a\frac{dW}{dt}$$

Therefore, W converges to a solution (where $\frac{dW}{dt}=0$) if and only if V converges to a solution.

Now, $\frac{dW}{dt} = (I-U)WC - (I-U)WCW^T(I-U)^TV^T = f(W,V)$

Let $F(W,V) = \int_0^\infty f(W,V)dt$

Then, $V^T = aF(W,V) + aK$ and $W = F(W,V) + K$

where K is a function of the initial values of V and W.

Thus, $\mathbf{v_i} = \mathbf{w_i} + \mathbf{p}$, where $\mathbf{p}$ is a vector depending only on the initial values of the system.

Thus if $\mathbf{v_i}$ and $\mathbf{w_i}$ converge, they do so simultaneously and close to the same vector.

Note 1 For the remainder of this section we will assume that a=1. i.e. the learning rates for V and W are equal.

Note 2 We note that the vectors $\mathbf{v_i}$ and $\mathbf{w_i}$ may be made arbitrarily close by limiting the original size of vectors $\mathbf{v_i(0)}$ and $\mathbf{w_i(0)}$. i.e. $\mathbf{p}$ may be made arbitrarily small by appropriate initial choice of $\mathbf{v}$ and $\mathbf{w}$. Hence we are able to assume that $\mathbf{v_i} = \lambda \approx \mathbf{w_i}$.

Theorem 7 *The learning rules detailed above are equivalent to*

$$\frac{dV^T}{dt} = \frac{dW}{dt} = (I-U)WC - (I-U)WCW^T(I-U)^TV^T \quad (69)$$

$$\frac{dU}{dt} = A(I-U)WCW^T(I-U)^T \quad (70)$$

$$\frac{dG}{dt} = (I-U)\frac{dW}{dt} - \frac{dU}{dt}W \quad (71)$$

where G is the forward function relating $\mathbf{x}$ and $\mathbf{z}$
and A is the matrix diag$\{ a_1, a_2, ..., a_m \}$ with $a_i = \lim_{t\to 0} \frac{\gamma_i(t)}{\eta w(t)}$ with $\gamma_i(t)$ being the value of γ_i during the time interval t.

Proof

With the same assumptions as before, we can write

$$\frac{W(B) - W(B-1)}{\eta(B)} = (I - U(B))W(B)\mathbf{x(B)x(B)^T} - (I - U(B))W(B)\mathbf{x(B)x(B)^T}W(B)^T(I - U(B))^TV(B)^T \quad (72)$$

If we also assume that

$$lim_{p\to\infty}\eta(p) = 0$$

then the sequence of $w_{ji}(T)$ asymptotically approaches a continuous-time function and the left-hand side of Equation (72) approaches its derivative. Then we can replace Equation (72) with the corresponding averaged differential equation

$$\frac{dW}{dt} = (I-U)WC - (I-U)WCW^T(I-U)^TV^T \quad (73)$$

where C is the covariance matrix of the stationary distribution producing the x_k values. Now, under certain assumptions about the rate η it can be shown that the

solution of the stochastic algorithm approaches the solution of the differential equation 73 with probability 1.

Similarly, for the weight updates of the U weights,

$$u_{ij}(B) = u_{ij}(B-1) + \gamma_i(B)z_i(B)z_j(B)$$

Therefore,

$$\frac{U(B)-U(B-1)}{\eta(B)} = \frac{\gamma_i(B)}{\eta(B)} z_i z_j$$

If we make the further assumption that $\lim_{B\to 0} \frac{\gamma_i(B)}{\eta(B)}$ exists, we can take the limit of the above stochastic equation giving

$$\frac{dU}{dt} = AQ$$

where A= diag{ $a_1, a_2, ..., a_m$ } with $a_i = \lim_{t\to 0} \frac{\gamma_i(t)}{\eta(t)} > 0$, and Q the m*m matrix with elements $q_{ij} = \langle z_i z_j \rangle, i \neq j$, and $q_{ii} = 0$ for all i,j. The angled brackets indicate an ensemble average. We will, for the time being, assume that the a_i values are constant during the learning process. We will return to this assumption in section 4.2.1 Now ,

$$\begin{aligned} Q &= \langle \mathbf{z}\mathbf{z}^T \rangle && (74)\\ &= \langle (I-U)W\mathbf{x}\mathbf{x}^T W^T (I-U)^T \rangle && (75)\\ &= (I-U)WCW^T(I-U)^T && (76) \end{aligned}$$

where C_{ij} is $\langle x_i x_j \rangle$ for all i,j. Hence,

$$\frac{dU}{dt} = A(I-U)WCW^T(I-U)^T$$

The transform from x to z is G where $G(t) = (I - U(t))W(t)$ where $U(t)$ is the value of U at time t etc. Then,

$$\begin{aligned} \frac{dG}{dt} &= (I-U(t))\frac{dW(t)}{dt} - \frac{dU(t)}{dt}W(t) && (77)\\ &= (I-U)\frac{dW}{dt} - \frac{dU}{dt}W && (78) \end{aligned}$$

Theorem 8 *U=0, the m*m zero matrix, is a solution of* $\frac{dG}{dt} = 0$, *and* $\mathbf{g_i} = \mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$ *is the corresponding solution for G and W where* $\mathbf{c_i}$ *are the eigenvectors of the covariance matrix of the input data in order . i.e. if* $\gamma_{u_1} < \gamma_{u_2} < ... < \gamma_{u_m}$, *then* $\mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$ *where* $\mathbf{c_i}$ *is that eigenvector with corresponding eigenvalue* λ_i *where* $\lambda_1 > \lambda_2 > ... > \lambda_m$ *and, as before,* $a_i = \lim_{t\to 0} \frac{\gamma_i(t)}{\eta w(t)}$

Proof

$$\frac{dG}{dt} = (I-U)\frac{dW}{dt} - \frac{dU}{dt}W$$

First we note that as $U \to 0$,

$$\begin{aligned}\frac{dG}{dt} &= \frac{dW}{dt} - \frac{dU}{dt}W \\ &= (I-U)WC - (I-U)WCW^T(I-U)^TV^T \\ &\quad -A(I-U)WCW^T(I-U)^TW \\ &= GC - GCG^TV^T - AGCG^TW \\ &\to WC - (I+A)WCW^TW\end{aligned}$$

at the point of convergence of V and W.

Note the similarity between these equations and those which are required for Oja's Weighted Subspace Theorem; therfore, we conjecture that a solution of $\frac{dG}{dt} = 0$ at U=0 is $\mathbf{g_i} = \mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$ the i^{th} eigenvector of C in normal order. Here we show that the stated values are solutions; stability will be proved later.

$$\begin{aligned}\frac{dG}{dt} &= (I-U)\frac{dW}{dt} - \frac{dU}{dt}W \\ &= (I-U)((I-U)WC - (I-U)WCW^T(I-U)^TV^T) - A(I-U)WCW^T(I- \\ &\to WC - WCW^TV^T - AWCW^TW \\ &= \Lambda W - KV^T - AKW\end{aligned}$$

where K is diagonal matrix whose $(i,i)^{th}$ element is $\lambda_i|\mathbf{w_i}|^2$ with λ_i the i_{th} eigenvalue and Λ is the diagonal matrix whose $(i,i)^{th}$ element is λ_i.

Then taking $\mathbf{g_i}$ as the i^{th} vector of G i.e. going into the i^{th} interneuron and using the fact that $\mathbf{w_i} = \mathbf{v_i}$, we have

$$\frac{d\mathbf{g_i}}{dt} = \lambda_i\mathbf{w_i} - k_i\mathbf{w_i} - a_ik_i\mathbf{w_i}$$

$$= (\frac{\lambda_i}{\sqrt{1+a_i}} - \frac{\lambda_i}{1+a_i}\frac{1}{\sqrt{1+a_i}} - \frac{a_i\lambda_i}{1+a_i}\frac{1}{\sqrt{1+a_i}})\mathbf{c_i}$$
$$= (\frac{\lambda_i(1+a_i) - \lambda_i - a_i\lambda_i}{\sqrt{1+a_i}(1+a_i)})\mathbf{c_i} = 0$$

So the stated values are stationary points of the system.

Note: We can in fact go further, in that if U=0 then

$$\frac{d\mathbf{g_i}}{dt} = \lambda_i\mathbf{w_i} - k_i\mathbf{w_i} - a_ik_i\mathbf{w_i}$$
$$= (\lambda_i - \lambda_i|\mathbf{w_i}|^2 - a_i\lambda_i|\mathbf{w_i}|^2)\mathbf{w_i}$$
$$= \lambda_i(1 - |\mathbf{w_i}|^2 - a_i|\mathbf{w_i}|^2)\mathbf{w_i}$$

so if $\frac{d\mathbf{g_i}}{dt} = 0$ then $|\mathbf{w_i}|^2(1+a_i) = 1$ i.e. $|\mathbf{w_i}| = \pm\frac{1}{\sqrt{1+a_i}}$.

Theorem 9 *At the solutions U=0,* $\mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$ *of* $\frac{dG}{dt} = 0$, *then*

$$\frac{du_{ij}}{dt} = 0 \text{ for all } i,j$$

if $\lambda_i \neq 0$ *i.e. the* i^{th} *eigenvalue is not zero.*

Proof

$$\frac{dU}{dt} = A(I-U)WCW^T(I-U)^T$$
$$= AWCW^T$$

Now $\mathbf{w_i}C\mathbf{w_j}^T = 0$ for all $i \neq j$ and $\mathbf{w_i}C\mathbf{w_i}^T = \lambda_i|\mathbf{w_i}|^2$. Therefore WCW^T is a diagonal matrix of the form $\text{diag}\{k_1, k_2, .., k_m\}$ where $k_i = \lambda_i|\mathbf{w_i}|^2$, λ_i being the i^{th} eigenvalue. Then

$$\frac{dU}{dt} = AK = \text{diag}\{a_1k_1, a_2k_2, ..., a_mk_m\}$$

Therefore, $\frac{du_{ij}}{dt} = 0$ for all $i \neq j$.

Theorem 10 *The solutions* $u_{ij} = 0$, $\mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$ *for all i,j of* $\frac{dG}{dt} = 0$ *ensure all variables are stationary at this point.*

Proof

At the stated points of solution of $\frac{dG}{dt} = 0$ then $\frac{dU}{dt} = 0$. But

$$\begin{aligned} \frac{dG}{dt} &= (I-U)\frac{dW}{dt} - \frac{dU}{dt}W \\ &= \frac{dW}{dt} \end{aligned}$$

So $\frac{dG}{dt} = \frac{dU}{dt} = \frac{dW}{dt} = \frac{dV^T}{dt} = 0$

Theorem 11 *If* $\mathbf{c_i}$ *are the unit length eigenvectors of C, the solutions*

$$\mathbf{v_i} = \mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$$

$$\mathbf{u_i} = \mathbf{0}, \textit{ where } \mathbf{0} \textit{ is the m*1 zero vector}$$

of the equations governing the dynamics of this network are asymptotically stable for all i.

Proof

First consider the $\mathbf{u_i}$. We have already shown that $U = 0 \Longrightarrow \frac{dU}{dt} = 0$. Consider a disturbance of ϵ in U=0. We have

$$\begin{aligned} \frac{dU}{dt} &= A(I-(0+\epsilon))WCW^T(I-(0+\epsilon))^T \\ &= AK - A\epsilon WCW^T - AWCW^T\epsilon + O(\epsilon^2) \\ &= -A\epsilon WCW^T - AWCW^T\epsilon + O(\epsilon^2) \text{ (off diagonal)} \\ &= -A\epsilon K - AK\epsilon + O(\epsilon^2) \end{aligned}$$

Since A and K are both diagonal matrices with entries ≥ 0, if $\epsilon > 0$, the rate of change of U is negative i.e. U must decrease. If $\epsilon < 0$, the rate of change of U is positive i.e. U will increase.

Now consider the W weights. We have proved that the stated values are solutions; we must still prove asymptotic stability. Note that at the stated points of convergence,

$$\begin{aligned} G &= (I-U)W = W \\ \frac{dG}{dt} &= (I-U)\frac{dW}{dt} - \frac{dU}{dt}W \end{aligned}$$

Now since G=W at the points stated, any instantaneous disturbance in W will have an equal instantaneous effect on G. Therefore, we will investigate the effect of a disturbance in W on G in order to derive the asymptotic stability of W. We do this through investigating the effects of the disturbance on U and W. Let there be a disturbance of E in the converged weights W. Then,

$$\begin{aligned}\frac{dU}{dt} &= A(W+E)C(W+E)^T \\ &= AWCW^T + AECW^T + AWCE^T + AECE^T \\ &\approx (AWCW^T) + AECW^T + AWCE^T\end{aligned}$$

ignoring terms of $O(E^2)$. Thus,

$$\begin{aligned}\frac{dU}{dt}(W+E) &= (AWCW^T + AECW^T + AWCE^T)(W+E) \\ &\approx AWCW^TW + AECW^TW + AWCE^TW + AWCW^TE \\ &= (AWCW^TW) + AECW^TW + AWCE^TW + AKE\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{dW}{dt} &= (W+E)C - (W+E)C(W+E)^TV^T \\ &\approx WC + EC - WCW^TV^T - ECW^TV^T - WCE^TV^T \\ &= (WC - WCW^TV^T) + EC - ECW^TV^T - WCE^TV^T\end{aligned}$$

So still ignoring terms of $O(E^2)$,

$$\begin{aligned}\frac{dG}{dt} &= \frac{dW}{dt} - \frac{dU}{dt}W \\ &= (WC - WCW^TV^T - AWCW^TW) + EC - ECW^TV^T - WCE^TV^T \\ &\quad -AECW^TW - AWCE^TW - AWCW^TE \\ &= EC - ECW^TV^T - WCE^TV^T - AECW^TW - AWCE^TW - AKE \\ &= EC - AKE - (I+A)(ECW^T + WCE^T)W\end{aligned}$$

Now, considering a disturbance of ϵ in the direction of c_j of the weight w_i,(i.e. a disturbance of ϵ_j) we note first that the matrix$(I+A)(ECW^T+WCE^T)$is a diagonal matrix with its j^{th} element $(1+a_j)\frac{2\lambda_j\epsilon}{\sqrt{1+a_j}}$. So considering the rate of change of g_i in

the direction of $\mathbf{c_j}$

$$\begin{aligned}\frac{d\mathbf{g_i}}{dt} &= \lambda_j\epsilon_\mathbf{j} - a_j k_j \epsilon_\mathbf{j} - (1+a_j)\frac{2\lambda_j\epsilon}{\sqrt{1+a_j}}w_j \\ &= (\lambda_j\epsilon - a_j\frac{\lambda_j}{1+a_j}\epsilon - (1+a_j)\frac{2\lambda_j\epsilon}{\sqrt{1+a_j}\sqrt{1+a_j}})\mathbf{c_j} \\ &= -\lambda_j(1+\frac{a_j}{1+a_j})\epsilon\mathbf{c_j} \qquad (79)\end{aligned}$$

Since C is symmetric, $\lambda_j > 0$. Further, the learning rates γ_j were such that $a_j > 0$. Then, Equation (79) shows that if $\epsilon > 0$, which would cause G to grow, the system will self-organise to cause G to shrink; if $\epsilon < 0$ the system will self-organise to cause G to grow. Since we have shown that U=0 is a stable solution, then the solutions of the W vectors must also be stable.

Now consider the $\mathbf{v_i}$. The proof that the stated values are solutions is implicit in the section above. To show asymptotical stability, let there be a disturbance of $\epsilon > 0$ in V. Then

$$\begin{aligned}\frac{dV^T}{dt} &= (I-U)WC - (I-U)WCW^T(I-U)^T(V+\epsilon)^T \\ &= WC - WCW^T(V+\epsilon)^T \\ &= WC - WCW^TV^T - WCW^T\epsilon^T \\ &= -WCW^T\epsilon^T \\ &= -K\epsilon^T \\ &< 0\end{aligned}$$

since every element of K is greater than 0. Similarly, if $\epsilon < 0$, we have $\frac{dV^T}{dt} > 0$.

Thus all the stated values are stable points of the system.

4.2.1 The GW Anomaly

There is an apparent anomaly in the above equations. The solution of $\frac{dG}{dt} = 0$ is $\mathbf{g_i} = \mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$ whereas the solution of $\frac{dW}{dt} = 0$ occurs at $\mathbf{w_i} = \mathbf{c_i}$. Further, as U$\rightarrow$ 0, G $\rightarrow$W. This suggests a less stable system than before and this is indeed the case. Thus in order to minimise instability, it is necessary to ensure that the a_i values are low. Experimental results suggest a value of 0.1 is sufficient to ensure stable

convergence to Principal Components. However, we note that, at $\mathbf{w_i} = \frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$, U=0 and so

$$\begin{aligned} \frac{dW}{dt} &= (I-U)WC - (I-U)WCW^T(I-U)^TW^TV^T \\ &\rightarrow WC - WCW^TV^T \\ \text{and so } \frac{d\mathbf{w_i}}{dt} &= \frac{\lambda_i}{\sqrt{1+a_i}}\mathbf{c_i} - \frac{\lambda_i}{1+a_i}\frac{1}{\sqrt{1+a_i}}\mathbf{c_i} \\ &= \frac{\lambda_i}{\sqrt{1+a_i}}\frac{a_i}{1+a_i}\mathbf{c_i} \end{aligned}$$

Therefore, for any $a_i \neq 0$, there will be a tendancy for the weights to grow away from the global optimum. However, as seen in the equations governing $\frac{dG}{dt}$, this cause instantaneous change in U which will drive the W weights in the opposite direction. In order to produce a damped system, the values of a_i should be small.

One possible response to this anomaly is to insist that as we are taking a limit to infinity, the a_i values can only be ≥ 0 i.e. to allow equality. However, this is not the experimental situation where a strict ratio is maintained as the terms decrease to 0 nor does it help the analysis as we then have a diagonal matrix which is not of full rank and would not then provide the differential decay necessary for convergence to the Principal Components.

The approach chosen here is to choose the values of a_i appropriately small so that the term $\frac{1}{\sqrt{1+a_i}} \approx 1$. Under this constraint the system has been found experimentally to be stable.

The final point to note is that in this system the decay of the learning rate to 0 may be essential to the fixed stability of the system; if the learning rates are not allowed to decay to zero, the very dynamical nature of the convergence will continue.

4.2.2 Simulations

The results of a typical experiment on the same type of data as in the previous chapters are indicated in Table 5. Here, the first interneuron has the smallest learning rate i.e. $\alpha_{u_1} < \alpha_{u_2} < \alpha_{u_3}$. Further, while the initial values of the **w** and **v** weights were O(0.0001) those of the **u** weights were O(0.00001).

V			W		
1.000	0.012	0.000	**1.000**	0.012	-0.000
0.030	**1.000**	0.008	0.030	**1.000**	0.009
0.005	-0.003	**0.994**	0.004	-0.003	**0.994**
0.004	0.007	-0.023	0.004	0.007	-0.023
-0.002	0.000	-0.000	-0.002	0.000	0.000

Interneuron No:	1	2	3
Angle (radians)	0.0011	0.0006	0.0004

Table 5: Results show the weights of a typical set of V and W weights from the Parallel Learning Algorithm with the angle in radians between the **v** and **w** vectors.No. of iterations=40000. Initially, $\alpha_w = \alpha_v = 0.0001; \alpha_{u_1} = 0.000005; \alpha_{u_2} = 0.00001; \alpha_{u_3} = 0.000015$

V			W		
0.651	0.188	1.0312	0.650	0.188	1.031
-0.151	0.984	-0.092	-0.150	0.984	-0.092
0.824	0.049	-0.305	0.842	0.049	-0.305
-0.019	0.006	0.006	-0.018	0.006	0.006
-0.001	-0.001	-0.001	-0.001	-0.001	-0.002

Table 6: Results of the same network as before with homogeneous U learning rates

In order to show that it is the different learning rate which causes the convergence to Principal Components the same experiment was rerun with all the U weights having the same learning rate; the results of this are shown in Table 6. While there may appear to be a soft PCA taking place, this effect vanishes in larger networks. This effect - that increased size removes the tendancy to perform a 'soft' PCA - has been found in other models in this section and therefore slightly larger networks have been used in obtaining other corroborative empirical results.

4.3 Differential Activation Functions

In this section we investigate 3 models of Peer Inhibitory Interneurons which use activation functions instead of learning rate to break the symmetry of the system. We will not repeat the explicit derivations of the last section for each of the 3 models as the mathematics is usually very similar; however specific points of interest will be

identified and analysed. First each of the 3 models is introduced and experimental results are given; points of interest in each are identified. Then a comparison of the models is made and additional models which have similar properties are outlined.

Unless stated otherwise, the empirical data is obtained from a network with 12 inputs, 7 interneurons and input data with variance of x_1 > variance of x_2 > ....A is a diagonal matrix with $A_{11} > A_{22} >$... The network here is slightly larger than in previous sections in order to highlight various interesting empirical results which are not so obvious in smaller networks.

Restricting ourselves to models where only the interneurons use activation functions and restricting such activation functions to multiplicative factors, (so that we still have a linear system) there are several possible models; we will identify 3 separate classes of models by determining the characteristics of 3 of these models. We will use the same conventions in naming vectors as before. Note, in particular, that there are still no self-connections for the interneurons i.e. the main diagonal of U is composed of zeros. In this section, all **u** weights will learn at the same rate η_U but there will be differential activation functions (multiplicative factors) on the interneurons. For simplicity, we assume that $\eta_W = \eta_V = \eta_U = \eta$. (This does not affect our results and provides a simpler mathematical model)

4.3.1 Model 1 - Lateral Activation Functions

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z} \tag{80}$$

$$\mathbf{z}' = \mathbf{W}\mathbf{y} = \mathbf{W}\mathbf{x} \tag{81}$$

$$\mathbf{z} = \mathbf{z}' - \mathbf{A}\mathbf{U}\mathbf{z}' \tag{82}$$

$$\Delta\mathbf{W} = \eta\mathbf{z}\mathbf{y}^T \tag{83}$$

$$\Delta\mathbf{V}^\mathbf{T} = \eta\mathbf{z}\mathbf{y}^T \tag{84}$$

$$\Delta\mathbf{U} = \eta\mathbf{z}\mathbf{z}^T \tag{85}$$

Then, omitting details, we have $G = (I - AU)W$ and

$$\frac{dW}{dt} = (I - AU)WC - (I - AU)WCW^T(I - AU)^TV^T \tag{86}$$

$$\frac{dU}{dt} = (I - AU)WCW^T(I - AU)^T \quad (87)$$

$$\frac{dG}{dt} = (I - AU)\frac{dW}{dt} - A\frac{dU}{dt}W \quad (88)$$

Then if U converges to 0 at which point G=W=V^T,

$$\begin{aligned}\frac{dG}{dt} &= (I - AU)\{(I - AU)WC - (I - AU)WCW^T(I - AU)^T V^T\} \\ &\quad -A\{(I - AU)WCW^T(I - AU)^T\}W \\ &= (I - AU)\{GC - GCG^T V^T - AGCG^T W\} \\ &\rightarrow GC - (I + A)GCG^T W \text{ as U} \rightarrow 0\end{aligned}$$

at convergence,which should be compared with the equations in the previous section.

However the dynamics of the two models should not be assumed to be the same; note for example the different format of the equations governing the behaviour of the U values. This will be shown to be important in the investigation below. The w_i weights (almost)converge to the eigenvectors of the input data's covariance matrix. The underlying rationale for this network is that each interneuron has different susceptibility to the inhibition from its peers.

The results shown in Table 7 are from a 12-input, 7 interneuron network, with $a_i = 1.58 - 0.2 * (i - 1)$ for i=1,...,7. We note that while almost all the Principal Components have been certainly identified, the second and third interneurons have not identified precisely their respective Principal Components. The vectors seem to be almost correct and to satisfy $\mathbf{w_2}.\mathbf{w_3} = 0$ yet are not in the direction of the eigenvectors themselves. In fact by appropriate choice of the parameters a_i, this effect can be eliminated; however,

1. We wish to develop a network which will not require any fine tuning as it is used in different situations

2. The analysis of this fault provides insight into the network behaviour

The reason for this fault lies in the convergence of the U values. In the model of the last section the learning rule for the U values was shown to be

$$\frac{dU}{dt} = A(I - U)WCW^T(I - U)^T$$

Notice that as U→0, this learning rule continues to be dominated by the A matrix whereas, in Model 1, the effect of the A matrix vanishes as U→0 (see Equation 87).

The importance of $\frac{dU}{dt}$ is due to the fact that the value of $\frac{dU}{dt}$ is a major component in the decay term in $\frac{dG}{dt}$. Thus as U tends to zero and hence $\frac{dU}{dt} \rightarrow 0$, the decay term tends to zero. In the previous model this decay term maintained its differential effect as it decreased but in Model 1, as U $\rightarrow$ 0, the decay loses its directional impact - it becomes homogeneous.

More formally, consider the convergence to a solution of the system which is not an eigenvector. Let $\mathbf{w_i}$ have converged to $a\mathbf{c_i} + b\mathbf{c_j}, a, b \neq 0$ and let $\mathbf{w_j}$ have converged to $c\mathbf{c_i} + d\mathbf{c_j}, c, d \neq 0$. Both c and d are necessarily not zero as $\mathbf{w_i}.\mathbf{w_j} = 0$. Then we can show that the $(i,j)^{th}$ element of WCW^T can be shown to be

$$\begin{aligned}
\mathbf{w_i} C \mathbf{w_j}^T &= (\lambda_i a\mathbf{c_i} + \lambda_j b\mathbf{c_j}).(c\mathbf{c_i} + d\mathbf{c_j}) \\
&= \lambda_i ac + \lambda_j bd \\
\text{Similarly, } \mathbf{w_i} C \mathbf{w_i}^T &= \lambda_i a^2 + \lambda_j b^2 \\
\mathbf{w_j} C \mathbf{w_i}^T &= \lambda_i ac + \lambda_j bd \\
\mathbf{w_j} C \mathbf{w_j}^T &= \lambda_i c^2 + \lambda_j d^2
\end{aligned}$$

Now the i^{th} row of (I-AU) is $[\, -a_i u_{i1} ... 1 ... - a_i u_{ij} ... - a_i u_{im} \,]$ where the 1 is in the i^{th} position. Similarly with the j^{th} row. Then,

$$\begin{aligned}
\frac{du_{ij}}{dt} &= (I - AU)_i W C W^T (I - AU)_j^T \\
&= u_{ij}^2 (a_i a_j (\lambda_i ac + \lambda_j bd)) - u_{ij} (a_j (\lambda_i a^2 + \lambda_j b^2) + a_i (\lambda_i c^2 + \lambda_j d^2)) + (\lambda_i ac + \lambda_j bc
\end{aligned}$$

Thus $\frac{du_{ij}}{dt}$ at $u_{ij} = 0$ is equal to $(\lambda_i ac + \lambda_j bd)$ which is exactly zero for $b = c = 0$ i.e. the eigenvectors.

But, as $u_{ij} \rightarrow 0$, a situation arises where there is no particular impulse for the change of u_{ij} in any particular direction provided the constraint $ac + bd = 0$ is satisfied. The symmetry of the formula shows that $\frac{du_{ij}}{dt} = \frac{du_{ji}}{dt}$; thus the differential term in $\frac{dg_i}{dt}$ also vanishes at this point and so the weights,having approached the eigenvectors, need not converge precisely to any eigenvector - the driving force of differential weight decay has vanished.

For the system analysed in the previous section, we have

$$\frac{du_{ij}}{dt} = a_i u_{ij}^2(\lambda_i ac + \lambda_j bd) - u_{ij}(a_i(\lambda_i a^2 + \lambda_j b^2) + a_j(\lambda_i c^2 + \lambda_j d^2)) + a_i(\lambda_i ac + \lambda_j bd)$$

Note the asymmetry in this rule in that even as $u_{ij} \to 0, \frac{du_{ij}}{dt} \neq \frac{du_{ji}}{dt}$ Therefore this system will maintain a preferred degree of slope no matter how small U becomes. As noted earlier, the value of $\frac{du_{ij}}{dt}$ is precisely the value of the differential decay term in $\frac{dg_i}{dt}$ and so W will continue to converge taking account of the network's asymmetry, no matter how small U becomes.

It is possible to somewhat circumvent this problem by choosing values of a_i which are sufficiently different to make the term containing u_{ij} significant till U is closer to zero; however this is an heuristic and an *a priori* decision on the size of the a_i cannot be made.

A further difficulty with this Model is that we now have the activations fed to the interneurons being processed differently depending on their origins: all interneurons are responding equally to the fed-forward activations from the summing neurons but are responding differentially to the activations from their peers. This seems unrealistic for a biological model and requires an engineered model to have meta-information (as to whether to use an activation function or not). Model 2 is designed to rectify this.

4.3.2 Model 2 - Lateral and Feedforward Activation Functions

Our equations are almost the same as in the last section but note that every input to Z carries an activation function times the weighted inputs.The rationale behind this model is a belief that all inputs to an interneuron should be treated equally.

$$\mathbf{y} = \mathbf{x} - \mathbf{Vz} \tag{89}$$

$$\mathbf{z}' = \mathbf{AWy} = \mathbf{AWx} \tag{90}$$

$$\mathbf{z} = \mathbf{z}' - \mathbf{AUz}' \tag{91}$$

$$\mathbf{\Delta W} = \eta \mathbf{z} \mathbf{y}^T \tag{92}$$

$$\mathbf{\Delta V^T} = \eta \mathbf{z} \mathbf{y}^T \tag{93}$$

$$\mathbf{\Delta U} = \eta \mathbf{z} \mathbf{z}^T \tag{94}$$

	1	2	3	4	5	6	7
1	-0.005	-0.001	0.004	0.006	-0.023	-0.025	**0.999**
2	0.001	0.005	0.012	0.008	-0.014	**0.999**	0.029
3	-0.011	0.011	0.022	-0.045	**-0.998**	-0.023	-0.031
4	-0.003	0.021	0.058	**-0.999**	0.024	0.010	-0.003
5	0.001	**0.146**	**-0.996**	-0.031	0.019	-0.002	0.009
6	0.040	**0.998**	**0.119**	-0.004	-0.012	-0.019	-0.002
7	**0.998**	0.029	0.023	0.004	-0.016	0.013	0.014
8	-0.016	-0.013	0.004	0.004	0.003	0.011	-0.007
9	0.004	0.006	-0.003	0.004	-0.003	-0.001	-0.20
10	0.013	0.005	-0.001	0.002	-0.001	0.003	-0.005
11	-0.001	-0.004	-0.001	-0.001	-0.003	0.001	0.004
12	0.001	-0.001	-0.003	-0.003	-0.004	-0.002	0.001

Table 7: Model 1 Results.
The results are from a network with 12 inputs/summing neurons and 7 interneurons; each column is the vector of weights into each interneuron. In most cases the actual Principal Components have been identified since ideally we would wish an array whose only non-zero elements would be a diagonal (top-right to bottom-left) line of 1s.

Then we have

$$\mathbf{z} = (I - AU)\mathbf{z}' = (I - AU)AW\mathbf{x}$$

Therefore

$$\begin{aligned} G &= (I - AU)AW \\ \frac{dG}{dt} &= (I - AU)A\frac{dW}{dt} - A\frac{dU}{dt}AW \end{aligned}$$

As before, we can show that

$$\frac{dW}{dt} = (I - AU)AWC - (I - AU)AWC((I - AU)AW)^T V^T \quad (95)$$

$$\frac{dU}{dt} = (I - AU)AWC((I - AU)AW)^T \quad (96)$$

and so that

$$\begin{aligned} \frac{dG}{dt} &= (I - AU)A(I - AU)AWC - (I - AU)A(I - AU)AWC((I - AU)AW)^T V^T \\ &\quad -A(I - AU)AWC((I - AU)AW)^T AW \\ &\rightarrow A^2WC - A^2WC(AW)^T V^T - A^2WC(AW)^T AW \text{ as U} \rightarrow 0 \\ &= A\{(AW)C - (AW)C(AW)^T V^T - (AW)C(AW)^T AW\} \quad (97) \end{aligned}$$

	1	2	3	4	5	6	7
1	**0.438**	**-0.359**	**0.763**	0.001	-0.039	-0.013	0.004
2	**0.661**	**0.199**	**-0.418**	**-0.176**	-0.070	0.017	0.008
3	0.022	**0.709**	**0.275**	**0.478**	**0.112**	-0.012	-0.001
4	0.076	**-0.266**	**-0.182**	**0.852**	**0.273**	0.006	0.047
5	0.007	0.010	0.023	**-0.206**	**1.063**	**0.394**	0.010
6	-0.027	0.009	0.016	0.060	**-0.248**	**1.248**	**0.101**
7	0.009	0.006	0.010	-0.007	-0.007	-0.062	**1.611**
8	0.003	0.006	-0.010	-0.006	0.002	-0.009	0.006
9	-0.011	0.010	-0.013	-0.002	0.001	-0.006	0.010
10	-0.003	0.004	-0.004	0.005	-0.002	0.002	0.013
11	0.003	0.001	0.003	0.002	0.001	-0.005	-0.002
12	0.000	0.001	0.002	0.002	0.001	0.002	0.002

Table 8: Model 2 Results
Results from a network with 12 inputs/summing neurons and 7 interneurons. Each vector into each interneuron(the columns above) has almost all of its weight int the first 7 directions. The actual Principal Components have not, in general, been identified.

The factor A which multiplies the whole of the right side of Equation 97 acts on the whole of that side equally i.e. does not have the differential decay effect necessary to force convergence to Principal Components. It does however have the effect that the first vector $\mathbf{w_1}$ has the highest learning rate and so will tend to adapt to those directions which contain the greatest variance before the others do. This results in a "fuzzy" PCA. The first and last terms are precisely those of the Subspace Algorithm i.e. will cause (AW), and hence W, to converge to the Principal Subspace though not to the Principal Components themselves. The results of a simulation based on the usual set of data are shown in Table 8.

A second drawback of this model is that the activation function in Eqn 91 is applied only to the effect of the other inhibitory interneurons. i.e. the interneurons are calculating their final output values after the activation function has been applied. This may not be appropriate in a biological mode.

4.3.3 Model 3 - Feedforward Activation Functions

Now we only have an activation function on the first calculation of the z values:

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z} \tag{98}$$

$$\mathbf{z}' = \mathbf{AWy} = \mathbf{AWx} \tag{99}$$

$$\mathbf{z} = \mathbf{z}' - \mathbf{U}\mathbf{z}' \tag{100}$$

$$\mathbf{\Delta W} = \eta \mathbf{z}\mathbf{y}^T \tag{101}$$

$$\mathbf{\Delta V^T} = \eta \mathbf{z}\mathbf{y}^T \tag{102}$$

$$\mathbf{\Delta U} = \eta \mathbf{z}\mathbf{z}^T \tag{103}$$

Then we have

$$\mathbf{z} = (I - U)\mathbf{z}' = (I - U)AW\mathbf{x}$$

Therefore

$$G = (I - U)AW$$

$$\frac{dG}{dt} = (I - U)A\frac{dW}{dt} - \frac{dU}{dt}AW$$

As before, we can show that

$$\frac{dW}{dt} = (I - U)AWC - (I - U)AWC((I - U)AW)^T V^T \tag{104}$$

$$\frac{dU}{dt} = (I - U)AWC((I - U)AW)^T \tag{105}$$

and so that

$$\begin{aligned}\frac{dG}{dt} &= (I - U)A(I - U)AWC - (I - U)A(I - U)AWC((I - U)AW)^T V^T \\ &\quad -(I - U)AWC((I - U)AW)^T AW \\ &\rightarrow A^2WC - A^2WC(AW)^T V^T - AWC(AW)^T AW \text{ as } \mathrm{U} \rightarrow 0 \\ &= A\{(AW)C - (AW)C(AW)^T V^T\} - (AW)C(AW)^T(AW) \\ &= A\{(AW)C - (AW)C(AW)^T A^{-1}(AW) \\ &\quad -A^{-1}(AW)C(AW)^T(AW)\}\end{aligned} \tag{106}$$

The rationale behind this model is that each interneuron has equal inhibitory effect on the others but have differential responses to inputs.

The central term causes convergence to the Principal Subspace but within that subspace causes no convergence to the Principal Components themselves. The last term is the one which causes convergence to the actual Principal Components.

In more detail, consider the system as governed by

$$\begin{aligned}\frac{dG}{dt} &= (I-U)A(GC - GCG^TV^T - ((I-U)A)^{-1}GCG^TAW) \\ &\rightarrow A(GC - GCG^TV^T - A^{-1}GCG^TG) \text{ as U}\rightarrow 0\end{aligned}$$

We note that as A is diagonal and of full rank, it has an inverse, which is also diagonal, and each element of the inverse, $(A^{-1})_{ii} = a_i^{-1}$.

Then, as before, the central term will have no effect on convergence once the weights have converged to the Principal Subspace. Within that subspace convergence of the weights is governed by the equation

$$\frac{dG}{dt} = A(GC - A^{-1}GCG^TG)$$

This causes $\mathbf{g_i}$ to converge to $\frac{1}{\sqrt{A_{ii}^{-1}}}\mathbf{c_k} = \sqrt{a_i}\mathbf{c_k}$ where k=m-i. Note that if $a_i > a_j$ then $a_i^{-1} < a_j^{-1}$ and so this model causes convergence in the "opposite direction" to that normally associated with the A values.See Table 9. Now, $\mathbf{g_i} = (I-U)A\mathbf{w_i} \rightarrow a_i\mathbf{w_i}$; therefore,

$$\begin{aligned}a_i\mathbf{w_i} &= \sqrt{a_i}\mathbf{c_i} \\ \text{i.e. } \mathbf{w_i} &= \frac{1}{\sqrt{a_i}}\mathbf{c_i}\end{aligned}$$

Now G is simply a mathematical construct to help us understand the model; the actual learning processes take place in the modification of the W and U weights; in particular, the values of the W weights are determined by the convergence of $\frac{dW}{dt}$. At

$$\mathbf{w_i} = \frac{1}{\sqrt{a_i}}\mathbf{c_i}$$

$$\begin{aligned}\frac{d\mathbf{w_i}}{dt} &= a_i\mathbf{w_i}C - a_i\mathbf{w_i}C(a_i\mathbf{w_i})^T\mathbf{v_i}^T \\ &= a_i\lambda_i a_i^{-\frac{1}{2}}\mathbf{c_i} - a_i(a_i^{-\frac{1}{2}}\mathbf{c_i})C(a_i a_i^{-\frac{1}{2}}\mathbf{c_i})^T(a_i^{-\frac{1}{2}}\mathbf{c_i})\end{aligned}$$

	1	2	3	4	5	6	7
1	**0.794**	-0.024	0.014	-0.009	-0.004	-0.002	0.001
2	0.020	**0.849**	0.003	-0.007	0.014	0.013	0.005
3	-0.027	-0.012	**0.918**	0.047	0.028	-0.008	-0.010
4	-0.005	0.010	-0.021	**1.007**	-0.015	0.018	-0.001
5	0.002	-0.002	-0.017	-0.004	**1.129**	0.016	0.013
6	-0.005	-0.018	0.011	0.000	-0.007	**1.308**	-0.040
7	0.009	0.013	0.014	-0.001	-0.025	0.021	**1.613**
8	-0.007	0.010	-0.002	-0.004	-0.001	-0.010	0.006
9	-0.017	-0.001	0.002	-0.004	-0.004	0.006	-0.013
10	-0.003	0.001	0.002	-0.003	0.001	0.006	0.003
11	0.003	-0.002	0.003	0.001	-0.003	-0.003	0.002
12	0.001	0.001	0.004	0.003	0.001	0.002	0.001

Table 9: Model 3
The results are from a network with 12 inputs/summing neurons and 7 interneurons; each column is the vector of weights into each interneuron. In all cases the actual Principal Components have been identified. Note the different direction of 'slope' of the bold figures (see text).

$$\begin{aligned} &= \lambda_i a_i^{\frac{1}{2}} \mathbf{c_i} - \lambda_i a_i^{\frac{1}{2}} \mathbf{c_i} . a_i^{\frac{1}{2}} \mathbf{c_i} (\lambda_i a_i^{-\frac{1}{2}} \mathbf{c_i}) \\ &= \lambda_i a_i^{\frac{1}{2}} \mathbf{c_i} - \lambda_i a_i a_i^{-\frac{1}{2}} \mathbf{c_i} \\ &= 0 \end{aligned}$$

In other words, that solution of the overall system dynamics, $\frac{dG}{dt} = 0$ is also a solution of $\frac{dW}{dt} = 0$. The system will converge in harmony.

4.3.4 Summary

We present a summary comparison of the models using the rate of change of the various weights to guide the comparison.

$\frac{dW}{dt}$

Note first that in all three models, $\frac{dW}{dt}$ will cause convergence to the Principal Subspace but not to the actual Principal Components themselves. We repeat the equations here for convenience:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7
x	1.58	1.38	1.18	0.98	0.78	0.58	0.38
$\sqrt{x}$	1.26	1.17	1.09	0.99	0.88	0.76	0.62
$\frac{1}{\sqrt{x}}$	0.79	0.85	0.92	1.01	1.14	1.31	1.62

Table 10: Experimental values of A_i and corresponding values of $f(A_i)$ for the functions shown

	w_1	w_2	w_3	w_4	w_5	w_6	w_7
Model 1	0.998	1.000	0.998	0.999	0.999	0.999	1.000
Model 2	0.792	0.861	0.930	1.013	1.130	1.309	1.614
Model 3	0.795	0.849	0.919	1.008	1.130	1.308	1.614

Table 11: Lengths of the relevant vectors from the 3 Models

Model 1 $\frac{dW}{dt} = (I - AU)WC - (I - AU)WCW^T(I - AU)^TV^T$

Model 2 $\frac{dW}{dt} = (I - AU)AWC - (I - AU)AWC((I - AU)AW)^TV^T$

Model 3 $\frac{dW}{dt} = (I - U)AWC - (I - U)AWC((I - U)AW)^TV^T$

All three equations are of the form

$$\frac{dW}{dt} \rightarrow GC - GCG^TV^T \text{ as U} \rightarrow 0$$

and as before it can be shown that $V^T = KG$ for some diagonal matrix K. Therefore, all of these equations will cause the G vectors to converge to the Principal Subspace but not to the Principal Components themselves. We note that if $a_i w_i$ has converged to an eigenvector then w_i has converged to the same eigenvector. Further, these equations will determine the size of the W vectors; since each vector,g_i is of length 1, we have, noting that $\lim_{t\rightarrow\infty} u_{ij} = 0$

Model 1 $|w_i| = 1$

Model 2 $|w_i| = a_i^{-\frac{1}{2}}$

Model 3 $|w_i| = a_i^{-\frac{1}{2}}$

This analysis is corroborated by Tables 10 and 11 We will demonstrate that the

stated solution is correct for Model 2 - the other models can be similarly[3] analysed. We have

$$\begin{aligned}\frac{dW}{dt} &= (I-AU)AWC-(I-AU)AWC((I-AU)AW)^TV^T \\ &\rightarrow AWC-AWC(AW)^TV^T \text{ as U} \rightarrow 0\end{aligned}$$

Now let $\mathbf{w_i} = a_i^{-\frac{1}{2}}\mathbf{b_i}$ where $\mathbf{b_i} = \sum_j b_j\mathbf{c_i}$ i.e.$\mathbf{b_i}$ is a unit length combination of the vectors $\mathbf{c_i}$, with $\mathbf{c_i}$ the eigenvectors of C as usual. Then,

$$\begin{aligned}\frac{dW}{dt} &= AWC-AWC(AW)^TV^T \\ \frac{d\mathbf{w_i}}{dt} &= a_i(a_i^{-\frac{1}{2}}\sum_j b_j\mathbf{c_j}C-a_i(a_i^{-\frac{1}{2}}\sum_j b_j\mathbf{c_j})Ca_i(a_i^{-\frac{1}{2}}\sum_j b_j\mathbf{c_j})^T(a_i^{-\frac{1}{2}}\sum_j b_j\mathbf{c_j}) \\ &= a_i^{\frac{1}{2}}\sum_j \lambda_j b_j\mathbf{c_j}-a_i^{\frac{1}{2}}\sum_j \lambda_j b_j\mathbf{c_j}(a_i^{\frac{1}{2}}\sum_j b_j\mathbf{c_j})(a_i^{-\frac{1}{2}}\sum_j b_j\mathbf{c_j}) \\ &= a_i^{\frac{1}{2}}\sum_j \lambda_j b_j\mathbf{c_j}-a_i^{\frac{1}{2}}\sum_j \lambda_j b_j\mathbf{c_j}(\sum_j b_j\mathbf{c_j})^2 \\ &= 0\end{aligned}$$

since $\mathbf{b_j}$ is a unit length vector and so $(\sum_j b_j\mathbf{c_j})^2 = 1$;

$\frac{dU}{dt}$

Neither will $\frac{dU}{dt}$ cause convergence to the actual eigenvectors.

Model 1 $\frac{dU}{dt} = (I-AU)WCW^T(I-AU)^T$

Model 2 $\frac{dU}{dt} = (I-AU)AWC((I-AU)AW)^T$

Model 3 $\frac{dU}{dt} = (I-U)AWC((I-U)AW)^T$

Note that all equations have the general form $\frac{dU}{dt} = GCG^T$. We consider only the case U=0, since it can be shown that, at U=0 in all models, $\frac{dU}{dt} = 0$[4]. Then since A is a diagonal matrix, convergence to non-zero diagonal elements is achieved whenever

[3]Indeed, more simply since we can use the fact that these models cause convergence to the eigenvectors .

[4]This is not to be taken that we assume that $\frac{dU}{dt} = 0 \Longrightarrow U = 0$

the rows and columns of W are orthogonal. (If $a_i\mathbf{w_i} \perp a_j\mathbf{w_j}$ then $\mathbf{w_i} \perp \mathbf{w_j}$). Consider Model 3, at U=0 , $\mathbf{w_i} = a_i^{-\frac{1}{2}}\mathbf{c_i}$; then

$$\begin{aligned}
\frac{du_{ij}}{dt} &= (I-U)A\mathbf{w_i}C((I-U)A\mathbf{w_j})^T \\
&\rightarrow A\mathbf{w_i}C(A\mathbf{w_j})^T \text{ as U} \rightarrow 0 \\
&= a_i\lambda_i\mathbf{w_i}.a_j\mathbf{w_j} \\
&= \lambda_i a_i a_j \mathbf{w_i}.\mathbf{w_j} \\
&= \lambda_i a_i a_j \delta_{ij} a_i^{-\frac{1}{2}} a_j^{-\frac{1}{2}} \\
&= \lambda_i \delta_{ij} a_i^{\frac{1}{2}} a_j^{\frac{1}{2}}
\end{aligned}$$

where δ_{ij} is the Kronecker delta. Therefore, off the main diagonal, $\frac{du_{ij}}{dt} = 0$. So the equations governing the growth of U and W merely ensure that the columns of W form an orthogonal basis of the Principal Subspace of the covariance matrix of the input data.

$\frac{dG}{dt}$

$\frac{dG}{dt}$ is the equation which causes convergence to the Principal Components. $\frac{dG}{dt}$ is the manifestation of the interaction between the dynamical development of U and that of W.

Recall that G is defined as

Model 1 $G = (I - AU)W$

Model 2 $G = (I - AU)AW$

Model 3 $G = (I - U)AW$

If we assume convergence at U=0, then we have

Model 1

$$\begin{aligned}
\frac{dG}{dt} &= (I-AU)\{GC - GCG^TV^T\} - AGCG^TW \\
&\rightarrow GC - (I+A)GCG^TW \\
&\approx GC - (I+A)GCG^TG
\end{aligned}$$

This causes convergence of $\mathbf{g_i}$ to $\frac{1}{\sqrt{1+a_i}}\mathbf{c_i}$; however note the caveats made in section 4.3.1

Model 2

$$\begin{aligned}\frac{dG}{dt} &= (I-AU)A\{GC-GCG^TV^T\}-AGCG^T(AW)\\ &\rightarrow A\{GC-GCG^TV^T-GCG^T(AW)\}\end{aligned}$$

There is no specific parameter which will force the weights to the actual Principal Components themselves; both decay terms cause convergence to the Principal Subspace but within that subspace are non-directional.

Model 3

$$\begin{aligned}\frac{dG}{dt} &= (I-U)A\{GC-GCG^TV^T\}-GCG^TAW\\ &\rightarrow A\{GC-GCG^TV^T-A^{-1}GCG^TAW\}\\ &\approx A\{GC-GCG^T(A^{-1}G)-A^{-1}GCG^TG\}\end{aligned}$$

This causes convergence to $\frac{1}{\sqrt{a_i^{-1}}}\mathbf{c_{m-i}}$. The essential point to note is that that vector associated with the smallest value of A corresponds to that vector with the largest eigenvalue.

4.3.5 Other Models

Clearly the models identified above are not the only possible models; however, all models investigated have been found to be of one of the three classes defined by the above 3 models e.g.

Model 4 Our equations are almost the same as in Model 2 but note that the second outputs of the interneuron are calculated after the subtraction of the inputs from their peers :

$$\mathbf{y} = \mathbf{x}-\mathbf{Vz} \tag{107}$$

$$\mathbf{z}' = \mathbf{AWy} = \mathbf{AWx} \tag{108}$$

$$\mathbf{z} = \mathbf{A}(\mathbf{z}' - \mathbf{U}\mathbf{z}') \tag{109}$$

$$\Delta\mathbf{W} = \eta\mathbf{z}\mathbf{y}^T \tag{110}$$

$$\Delta\mathbf{V}^\mathbf{T} = \eta\mathbf{z}\mathbf{y}^T \tag{111}$$

$$\Delta\mathbf{U} = \eta\mathbf{z}\mathbf{z}^T \tag{112}$$

Then we have

$$\mathbf{z} = A(I-U)\mathbf{z}' = A(I-U)AW\mathbf{x}$$

Therefore

$$\begin{aligned} G &= A(I-U)AW \\ \frac{dG}{dt} &= A(I-U)A\frac{dW}{dt} - A\frac{dU}{dt}AW \end{aligned}$$

As before, we can show that

$$\frac{dW}{dt} = A(I-U)AWC - A(I-U)AWC(A(I-U)AW)^TV^T \tag{113}$$

$$\frac{dU}{dt} = A(I-U)AWC(A(I-U)AW)^T \tag{114}$$

and so that

$$\begin{aligned} \frac{dG}{dt} &= A(I-U)AA(I-U)AWC - A(I-U)AA(I-U)AWC(A(I-U)AW)^T \\ &\quad -AA(I-U)AWC(A(I-U)AW)^TAW \\ &= A(I-U)A(GC - GCG^TV^T - (A(I-U))^{-1}GCG^T(AW) \\ &\rightarrow A^2\{(A^2W)C - (A^2W)C(A^2W)^TV^T - A^{-1}(A^2W)C(A^2W)^TAW\} \end{aligned}$$

This model acts similarly to Model 3 in that the A^{-1} causes convergence. All effects, however, are even more pronounced: the differential learning rates of the feedforward functions $\mathbf{g_i}$ are even more exaggerated, and the differences in the size of vectors are larger. The rationale behind this model is that each interneuron will calculate its activation at all times based on the sum of the inputs at that time. This is possibly the most realistic biological model; it requires no meta knowledge and is local, simple and parallel.

Model 5

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z} \tag{115}$$
$$\mathbf{z}' = \mathbf{W}\mathbf{y} = \mathbf{W}\mathbf{x} \tag{116}$$
$$\mathbf{z} = \mathbf{A}(\mathbf{z}' - \mathbf{U}\mathbf{z}') \tag{117}$$
$$\Delta\mathbf{W} = \eta\mathbf{z}\mathbf{y}^T \tag{118}$$
$$\Delta\mathbf{V}^\mathbf{T} = \eta\mathbf{z}\mathbf{y}^T \tag{119}$$
$$\Delta\mathbf{U} = \eta\mathbf{z}\mathbf{z}^T \tag{120}$$

Then we have

$$\mathbf{z} = A(I-U)\mathbf{z}' = A(I-U)W\mathbf{x}$$

Therefore

$$G = A(I-U)W$$
$$\frac{dG}{dt} = A(I-U)\frac{dW}{dt} - A\frac{dU}{dt}W$$

As before, we can show that

$$\frac{dW}{dt} = A(I-U)WC - A(I-U)WC(A(I-U)W)^TV^T \tag{121}$$
$$\frac{dU}{dt} = A(I-U)WC(A(I-U)W)^T \tag{122}$$
$$\frac{dG}{dt} \rightarrow A\{GC - GCG^TV^T - GCG^TW\} \tag{123}$$

This model acts like Model 2. It finds the Principal Subspace but not the Principal Components themselves as there is no differential decay in the model.

Model 6

$$\mathbf{y} = \mathbf{x} - \mathbf{V}\mathbf{z} \tag{124}$$
$$\mathbf{z}' = \mathbf{A}\mathbf{W}\mathbf{y} = \mathbf{A}\mathbf{W}\mathbf{x} \tag{125}$$
$$\mathbf{z} = \mathbf{A}(\mathbf{z}' - \mathbf{A}\mathbf{U}\mathbf{z}') \tag{126}$$
$$\Delta\mathbf{W} = \eta\mathbf{z}\mathbf{y}^T \tag{127}$$
$$\Delta\mathbf{V}^\mathbf{T} = \eta\mathbf{z}\mathbf{y}^T \tag{128}$$
$$\Delta\mathbf{U} = \eta\mathbf{z}\mathbf{z}^T \tag{129}$$

Then we have

$$\mathbf{z} = A(I - AU)\mathbf{z}' = A(I - AU)AW\mathbf{x}$$

Therefore

$$G = A(I - AU)AW$$

$$\frac{dG}{dt} = A(I - AU)A\frac{dW}{dt} - AA\frac{dU}{dt}AW$$

As before, we can show that

$$\frac{dW}{dt} = A(I - AU)AWC - A(I - AU)AWC(A(I - AU)AW)^T V^T \quad (130)$$

$$\frac{dU}{dt} = A(I - AU)AWC(A(I - AU)AW)^T \quad (131)$$

$$\frac{dG}{dt} \rightarrow A^2\{GC - GCG^T V^T - GCG^T AW\} \quad (132)$$

Again there is no asymmetry in the learning process and so the model will act like Model 2 - it finds the Principal Subspace but not the Principal Components.

4.4 Emergent Properties of the Peer-Inhibition Network

A possible criticism of envisaging biological neural nets as performing a Principal Component Analysis is that it leads to a situation whereby one neuron is in charge of all information passing in a particular direction; therefore, if it is in any way damaged, the information in that direction which should be passed on will be lost.

An interesting property of large Peer-inhibitory networks is that such so-called "grandmother" cells take a very long while to form: the network quickly self-organises till each interneuron's weights are maximally sensitive to 4/5 directions but it then takes a very long while to converge to a single Principal Component. A typical set of weights is shown in Tables 13 - 16. It should be seen that each weight is gradually converging to a particular Principal Component; what is more difficult to show is that the direction of each Principal Component is maximally associated with the

weights of approximately 4 or 5 interneurons after 50000 iterations and the weights only gradually thereafter converge to a single Principal Component. The full matrix would appear as a "fuzzy" diagonal of bold-faced type.

4.5 Conclusion

In this chapter, we have used the negative feedback effect of each interneuron on the other interneurons in an attempt to ensure that the weights into each interneuron converge to the actual Principal Components themselves. We have shown that it is not enough to simply feedback the activations of the interneurons as they are calculated - some measure of asymmetry must be introduced into the network.

Two main methods of introducing such asymmetry have been shown to be successful: interneurons using different learning rates and interneurons using different activation functions. While both of these methods have been shown to be successful, the results of the analysis and experiments with different activation functions show that simply to introduce an asymmetry into the network without a theoretical understanding of the consequences could lead to unpredictable consequences: in the case of activation functions, it has been shown that the same activation function can have the desired effect, no effect or the opposite effect to that which might be predicted depending on where it is introduced.

Nevertheless, several models have been shown to be extremely successful at finding Principal Components of input data and hence of transmitting the maximum amount of information with the least possible amount of hardware. The inherent parallelism of the network should make possible a very fast implementation of the network on parallel hardware.

Table 12: Each row represents the first 7 components of the first 5 interneuron weights in a network of 100 inputs and 50 interneurons after 50000 iterations; all weights not shown are less than 0.1, most considerably less...

	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7
Interneuron 1	**0.737**	**0.405**	**0.464**	0.063	**0.154**	**0.114**	-0.079
Interneuron 2	0.072	**0.323**	**-0.637**	**0.360**	**0.557**	-0.030	-0.021
Interneuron 3	0.046	**0.638**	**-0.375**	**-0.461**	**-0.429**	-0.045	**0.297**
Interneuron 4	**0.340**	**-0.259**	**-0.255**	**0.533**	**-0.475**	-0.015	**-0.440**
Interneuron 5	**0.506**	**-0.466**	**-0.366**	**-0.443**	**0.123**	**0.322**	**0.170**

Table 13: The same network as above after 100000 iterations...

	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7
Interneuron 1	**0.898**	**0.325**	**0.256**	-0.021	0.081	-0.025	-0.048
Interneuron 2	0.021	**0.361**	**-0.690**	**0.379**	**0.466**	-0.032	0.059
Interneuron 3	-0.095	**0.626**	**-0.319**	**-0.496**	**-0.471**	0.043	-0.091
Interneuron 4	**0.266**	**-0.289**	**-0.378**	**0.490**	**-0.653**	-0.086	**-0.226**
Interneuron 5	**0.329**	**-0.525**	**-0.488**	**-0.525**	**0.101**	**0.190**	**0.209**

Table 14: The same network as above after 200000 iterations...

	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7
Interneuron 1	**0.922**	0.022	**0.162**	**-0.130**	-0.032	-0.015	0.023
Interneuron 2	**0.207**	**0.191**	**-0.850**	**0.339**	**0.248**	0.083	-0.053
Interneuron 3	-0.082	**0.845**	-0.082	**-0.457**	**-0.273**	0.004	0.060
Interneuron 4	0.034	**-0.184**	**-0.257**	**0.180**	**-0.914**	**-0.121**	-0.016
Interneuron 5	-0.010	**-0.479**	**-0.398**	**-0.737**	0.005	**0.237**	**0.131**

Table 15: The same network as above after 300000 iterations...

	Input 1	Input 2	Input 3	Input 4	Input 5	Input 6	Input 7
Interneuron 1	**0.961**	**-0.133**	**0.139**	**-0.167**	-0.037	-0.002	0.041
Interneuron 2	**0.218**	**0.164**	**-0.902**	**0.308**	0.091	0.035	-0.015
Interneuron 3	0.063	**0.949**	0.078	**-0.265**	**-0.101**	0.056	0.016
Interneuron 4	-0.032	-0.063	**-0.102**	0.034	**-0.971**	**-0.157**	0.067
Interneuron 5	**-0.137**	**-0.219**	**-0.367**	**-0.860**	0.011	**0.190**	**0.103**

Table 16: At the other end of the matrix/table, the interneurons' weights are converging only slightly more slowly

	Input 50	Input 49	Input 48	Input 47	Input 46	Input 45	Input 44
Interneuron 45	0.013	**0.273**	-0.061	**0.123**	**-0.717**	-0.077	**0.297**
Interneuron 46	**0.129**	**-0.172**	**0.122**	**0.271**	-0.017	**-0.868**	0.039
Interneuron 47	0.065	**0.317**	-0.072	**-0.859**	-0.090	**-0.313**	0.018
Interneuron 48	-0.088	0.058	**0.963**	**-0.114**	**-0.105**	**0.121**	-0.093
Interneuron 49	**0.731**	**0.558**	0.080	**0.223**	**0.270**	**0.108**	0.002
Interneuron 50	**0.617**	**-0.654**	0.007	**-0.237**	**-0.297**	**0.170**	-0.015

Chapter 5

Biological Asymmetries

5.1 Back to the Biological Drawing Board

The initial rationale for the development of ANNs came from attempting to emulate biology. We have been trying in the previous chapters to create an engineered PCA machine which does not stray too far from a possible biological model. However, the models of the last section have moved quite a distance from a conceivable biological model, not simply in terms of the increase of complexity of the model but also, more importantly, in the necessity for structured time intervals. In the previous chapter, we required a 3-phase operation in transferring activations:

Phase 1 The activation is fed forward; the interneurons calculate their activations.

Phase 2 The interneurons' activations are fed to the other interneurons; the interneurons recalculate their activations.

Phase 3 The interneurons activations are fed back to the summing neurons; the summing neurons calculate their activations

and it is only after all 3 phases have taken place that any learning will take place.

All biological evidence suggests that nature does not countenance such complex dependencies; she tends to throw lots of simple robust power at problems.

Let us restate the situation as we currently find it:

1. The interneuron network has been shown to converge to the Principal Subspace (i.e. not to the Principal Components themselves but to some basis of the PCA subspace) when simply set running in parallel mode.

2. We can force the network to learn the actual Principal Components if we introduce asymmetry into the network either by creating interneurons in a phased manner(temporal asymmetry) or by creating asymmetry in a Peer-Inhibition network by means of differential learning rates (acceleration asymmetry) or differential activation functions (efficiency asymmetry).

3. In introducing any of the asymmetries to the network, we corrupt some of the attractive properties of the basic network which we have identified as

   - Simplicity
   - Homogeneity
   - Locality of information use
   - Parallelism

In this Chapter[1] , we consider the effects of two naturally-occuring asymmetries and then use one in an application of our negative feedback neural network.

5.2 Non-negative Weights

There is one obvious asymmetry used in nature which we have not used as yet: it is believed that signals from neurons may be excitatory or inhibitory but not both i.e. a neuron's output can excite (positively) other neurons or it can inhibit (negatively) other neurons; what cannot happen is that it excites some and inhibits others. The results reported in previous Chapters were based on a model where the weights were allowed to take any value positive or negative and so a neuron could be exciting some neurons while inhibiting others. In fact, it is possible for a neuron to switch from excitatory activation to inhibitory as its weight changes from positive to negative. If

[1]Some of this work has appeared in (Fyfe, 1993a; Fyfe, 1993e; Fyfe and McGregor, 1994).

	Input 1	Input 2	Input 3	Input 4	Input 5
Interneuron 1	**0.552**	0	0.004	0.000	0.001
Interneuron 2	**0.700**	0	0.005	0.000	0.001
Interneuron 3	0.035	**0.991**	0.004	0.000	0.000
Interneuron 4	**0.450**	0	0.003	0.000	0.001

Table 17: A 5 input, 4 interneuron network with the same type of input data as previously

we allow only non-negative weights i.e. ensure that if a weight, while learning, never takes a negative value, we have the following interesting situation:

Assume that two weights of our converged network have values $ac_i + bc_j$ and $cc_i + dc_j$, with the same notation as before. Then since the weights converge to an orthogonal basis of the space, $ac + bd = 0$. Now if none of the terms a,b,c or d can be negative, then at least 2 must be zero (one from each term ac and bd). In other words, this constraint swings the weight vectors through the weight space to the actual Principal Components themselves. Since we are not directing the process, situations where several sets of weights converge to the same Principal Component tend to appear. An extreme example is shown in Table 17 in which we report the results of a simulation on the same type of data as previously but where the basic VW interneuron network was set up and the weights allowed to learn concurrently.

Clearly, the weights of interneurons 1,2 and 4 have all converged to the same Principal Component. Note that the weights marked only "0" have been stopped from becoming negative.

5.2.1 Other data sets

However, there is one clear difficulty with this program - if we are calculating Principal Components from a general data set, there must be a negative term in at least one of the Principal Components' coordinates. (In order to have orthogonal directions, the inner product of the components must be zero and hence there must be at least one negative component).

To further investigate the network's potential, data from a distribution whose

Direction	1	2	3	4	5	Value
First PC	**0.584**	**0.811**	0.000	-0.002	-0.002	59.3
Second PC	-0.006	0.001	**-0.469**	**-0.617**	**-0.632**	33.7
Third PC	**-0.811**	**0.584**	-0.010	0.005	0.011	7.1
Fourth PC	0.012	-0.008	**-0.876**	**0.235**	**0.421**	2.4
Fifth PC	0.002	-0.001	**0.111**	**-0.751**	**0.650**	0.5

Table 18: Principal Components of the new data calculated using a standard statistical package

Interneuron 1	0.005	0.000	**0.465**	**0.616**	**0.635**
Interneuron 2	**0.391**	**0.518**	0.001	0.000	0.000
Interneuron 3	**0.324**	**0.467**	0.001	0.000	0.000
Interneuron 4	**0.296**	**0.409**	0.001	0.000	0.000

Table 19: A 5-4 interneuron circuit operating on the data of the previous table

Principal Components are shown in Table 18 was used as input to the network: it should be clear that there is a sharp division in the data between the first two directions and the last three. It might seem to be possible for the network to converge to a mixture of the above weights e.g. the directions {0.584,0,0.469,0,0} and {0,0.811,0,0.617,0.632} span the subspace of the first two Principal Components. This does not happen; the network converges to the first 2 Principal Components themselves (see analysis in the next Section).

It is impossible for the network using the positive weight constraint to converge to any direction containing a negative component i.e. from the third onwards. To find out how the network would respond to a situation where there were more degrees of freedom than possible directions to be found, we used the network with these 5 inputs and 4 interneurons (with the constraint that no weights are allowed to become negative). The results are shown in Table 19.

It is clear that the first interneuron has found the second Principal Component while the second, third and fourth interneurons have found the first Principal Component. This is a general finding with this type of network with the non-negative weight constraints.

This form of information extraction may be of importance if the data has been

Inter	Input	Weight	Inter	Input	Weight	Inter	Input	Weight
0	9	1.000	17	23	0.999	34	10	0.374
1	19	1.000	18	3	0.308	35	20	0.999
2	33	0.998	19	4	0.455	36	16	0.999
3	8	0.554	20	28	0.998	37	25	0.999
4	12	1.000	21	5	1.000	38	6	0.999
5	1	0.370	22	27	0.998	39	0	0.641
6	1	0.490	23	35	0.993	40	18	0.999
7	0	0.611	24	26	0.998	41	31	0.998
8	2	0.999	25	30	0.999	42	22	1.000
9	32	0.997	26	14	0.999	43	7	0.674
10	15	0.999	27	3	0.951	44	0	0.142
11	8	0.481	28	10	0.870	45	10	0.320
12	29	0.998	29	11	0.797	46	21	0.999
13	34	0.995	30	17	0.999	47	7	0.738
14	0	0.441	31	8	0.679	48	11	0.602
15	13	1.000	32	1	0.451	49	1	0.648
16	4	0.890	33	24	0.999			

Table 20: Results from an interneuron network with 100 inputs labelled 0-99, and 50 interneurons labelled 0-49, e.g. the weights into interneuron 0 have converged to (input) direction 9 and the weight in that direction was 1.000

preprocessed in order to have isolated the "texture" data from the "colour" data from the "smell" data etc.. This type of distributed data-processing is known to happen in biological neural networks. However, this type of data-processing cannot be an initial data-processing function. The information must first be differentiated into disjoint dimensions: if there is any overlap between the dimensions in which the data exists, no more than one Principal Component per data set is possible.

We note that the length of the total vector of weights into interneurons 2, 3 and 4 is one unit.

Restricting ourselves to our specialised data, we can show that the principal directions are found: in Table 20, we show the weights from a network with 100 inputs of the same specialised form as before and 50 interneurons. All weights not shown were under 0.015 after 100000 iterations. We note that

- The weights into each interneuron converged to a single Principal Component

- Some of the directions with largest eigenvalues, (those of the first 12 Principal Components) were covered by more than 1 interneuron. Maximally, directions 0 and 1 were covered by the weights of 4 interneurons
- The weights in each direction still (approximately) had length 1
- There is no half-way house with this network's converged weights- the weights into different interneurons are either totally orthogonal or in completely the same direction.

The last 2 points are potentially important in considering an interneuron network as a possible explanation of biological networks' information management processes. If such recognition is spread over a group of neurons such as is shown here, this provides a robustness in the network which has been missing up till now.

Further, since the total weight in any direction still has length 1, then directions which are represented by more than one interneuron are not overemphasised in any data processing.

Experiments with larger sizes of networks have shown that the above effects increases with size.

5.2.2 Theoretical analysis

Consider a network with 4 inputs and 2 interneurons. Let the eigenvector of the input data with the largest eigenvalue be $\mathbf{a} = \{a_1, a_2, 0, 0\}$ and the second eigenvector be $\mathbf{b} = \{0, 0, b_3, b_4\}$. Then, in the situation described in the last section, if $\mathbf{w_i}$ is the vector of weights into interneuron i, then $\mathbf{w_1}$ converges to $\mathbf{a}$ and $\mathbf{w_2}$ to $\mathbf{b}$ or vice-versa. We show that this is a stable solution.

Using angled brackets to indicate the ensemble average, the expected input is

$$\langle x \rangle = k_a \mathbf{a} + k_b \mathbf{b} = \{k_a a_1, k_a a_2, k_b b_3, k_b b_4\} \quad (133)$$

$$\langle z_1 \rangle = k_a (a_1^2 + a_2^2) \quad (134)$$

$$\langle y_1 \rangle = k_a a_1 - k_a a_1 (a_1^2 + a_2^2) \quad (135)$$

So, the expected change in the weight between y_1 and z_1 is

$$\begin{aligned}\langle \Delta w_{11} \rangle &= \eta \langle y_1 z_1 \rangle \\ &= \eta\{(a_1(1-(a_1^2+a_2^2))k_a^2(a_1^2+a_2^2)\} \end{aligned} \tag{136}$$

So, since $a_i \neq 0, \forall i$ and $k_a > 0$, $\langle \Delta w_{11} \rangle = 0 \iff a_1^2 + a_2^2 = 1$. Since the eigenvector a has length 1, the converged weights are stable.

Now consider a network whose weights have converged to values incorportating both eigenvectors e.g. let $\mathbf{w_1}$ have converged to $\{a_1, 0, b_3, 0\}$ and $\mathbf{w_2}$ have converged to $\{0, a_2, 0, b_4\}$. Then a similar argument to the above leads to

$$\langle \Delta w_{11} \rangle = (k_a a_1^2 + k_b b_3^2) a_1 (k_a - (k_a a_1^2 + k_b b_3^2))$$

So $\langle \Delta w_{11} \rangle = 0 \iff k_a = k_b \frac{b_3^2}{1-a_1^2} = k_b \frac{b_3^2}{a_2^2}$. This equation imposes constraints on the input data relating the internal proportion of each eigenvector in each direction to the relative size of each eigenvalue.

Thus, while it is possible to construct data to satisfy these criteria, it is not generally the case that data-sets will comply with the constraints.

Further note that this equation is only one of 4 derivable from the system. We can show that the system requires $\frac{k_a}{k_b} = \frac{b_3^2}{a_2^2} = \frac{b_4^2}{a_1^2}$ for stability. This will not generally be true.

5.3 Using Distance Differences

Another possible model is suggested by the innate asymmetry in real biological neural networks in terms of the distances between neurons. This will manifest itself as different times to respond to a signal depending on the distance which the signal must travel (assuming that there is some uniformity in the speed of information transfer).

This differential is used in a new model where different interneurons take different lengths of times to respond to the input signal $\mathbf{x}$. Therefore while the activation from the input neurons is transmitted *to* all interneurons at the same time, each interneuron's response takes a different length of time to feedback to the input neurons. Thus

the negative feedback is felt and used in a phased manner and learning takes place immediately the returned signal is received. Therefore, we embed the learning process in the feedback loop, so that we now postulate a learning and activation-transmission process which takes place in the order in which the following equations are given.

$$\text{initial value of } \mathbf{y} = \mathbf{y}(0) \;=\; \mathbf{x} \tag{137}$$

$$\mathbf{z} \;=\; W\mathbf{y} \tag{138}$$

$$\mathbf{y}(t) \;=\; \mathbf{y}(t-1) - \mathbf{v_i}(t-1)z_i \tag{139}$$

$$\Delta\mathbf{w_i}(t) \;=\; \eta(t)z_i(t)\mathbf{y}(t) \tag{140}$$

$$\Delta\mathbf{v_i}(t) \;=\; \eta(t)z_i(t)\mathbf{y}(t) \tag{141}$$

where e.g. $\mathbf{v_i}(t-1)$ indicates the value of the vector of weights $\mathbf{v_i}$ at the time t-1.

Other than the first two steps,(the acceptance of the initial activation $\mathbf{x}$ and its forward transmission to the interneurons) the process (defined by Equations (139), (140) and (141)) is repeated for each interneuron in turn. This corresponds to the feedback from the interneurons being received at different times (perhaps depending on the physical distance which the activation must traverse, perhaps depending on the efficiency of transmission of the interneuron). This process results in the weights of the first (fastest) interneuron learning the first Principal Component, the second fastest interneuron learns the second Principal Component etc.. Experimental results from a network with 5 inputs and 3 interneurons are given in Table 21. In order to demonstrate the effect of the network, we have carried out our simulations on the same type of data as previously. Clearly the first 3 principal components have been found by the 3 interneurons.

Note that the crucial difference between this model and previous models is the embedding of the learning process in the activation reception process. When this is done, the resulting network is more similar to a Sanger-type (Sanger, 1990) network rather than an Oja-type network. The k^{th} interneuron is learning to extract the maximum amount of information which is left after the previous (k-1) interneurons have extracted their information.

V			W		
1.000	0.006	-0.010	**1.000**	0.006	-0.010
-0.000	**-1.000**	0.013	-0.000	**-1.000**	0.013
0.012	0.023	**1.000**	0.012	0.023	**1.000**
0.000	-0.003	0.004	0.000	-0.002	0.004
-0.002	-0.004	-0.001	-0.002	-0.004	-0.001

Table 21: Results of the Differential Distance Model; each column shows the converged weights between one interneuron and the input neurons after learning on data from independent zero mean Gaussians with descending variances

5.3.1 Equivalence to Sanger's Algorithm

Sanger's algorithm has, as a learning rule

$$\Delta w_{ij} = \eta y_i (x_j - \sum_{k=1}^{i} y_k w_{kj})$$

in a totally feedforward architecture, where the outputs at y are given by

$$y_i = \sum_j w_{ij} x_j$$

We can show that the interneuron network using the rules determined by Equations 137 - 141 is equivalent to Sanger's algorithm:

Let the y values be indexed with the time of feedback from the interneurons. Then,

$y_j(0)$ is the initial value of y_j at time 0. i.e. $y_j(0) = x_j$

$y_j(1)$ is the value of y_j after receiving the feedback activation from the first (and hence closest) interneuron. i.e. $y_j(1) = y_j(0) - v_{1j} z_1$. Note that the time values are only ordinal indices - they do not imply equal intervals between feedback activations.

Similarly, if $y_j(2)$ is the value of y_j after receiving feedback from the first 2 interneurons, then

$$y_j(2) = y_j(1) - v_{2j} z_2 = y_j(0) - \sum_{k=1}^{2} v_{kj} z_k \quad (142)$$

In general, if $y_j(i)$ is the value of y_j after receiving feedback from the first i interneurons,

$$y_j(i) = y_j(0) - \sum_{k=1}^{i} v_{kj} z_k = x_j - \sum_{k=1}^{i} v_{kj} z_k \quad (143)$$

Therefore,

$$\begin{aligned}\Delta v_{ij} = \Delta w_{ij} &= \eta y_j(i) z_i \\ &= \eta(y_j(0) - \sum_{k=1}^{i} v_{kj} z_k) z_i \\ &= \eta z_i(x_j - \sum_{k=1}^{i} v_{kj} z_k)\end{aligned}$$

which is exactly Sanger's formulation (see Chapter 2).

5.4 The Interneuron Coding Network

The various algorithms (such as that above) describing learning within the interneuron network have been shown to extract the maximum information from sets of stochastic data. The next obvious question is to decide what a network should do with such information when it has been extracted. Some form of coding would be helpful in classifying such data.

The interneuron coding network was developed in appreciation of the way in which Carlson (Carlson, 1990) amended the basic network of Rubner and Schulten (Rubner and Schulten, 1990), a PCA network, in order to create a coding network.

While we have wished to emulate his success, we have the continuing design ethos based on the retention of as many of the attractive features of the basic interneuron network as possible - those of simplicity, homogeneity, locality of information use and parallelism.

Our aim is to create a network which will take a set of raw data and code it so that different sections of the data are coded differently and such that data which have most similarity are most alike in codes i.e. a topology-preserving network. A binary code is easiest to implement with a simple threshold. Since we require several bits for each codeword, we suggest a network such as shown in Figure 10.

This network is shown with only one input and 3 coding interneurons. Raw data at the $\mathbf{x}$ input is converted to a binary coded vector $\mathbf{z}$ at the coding interneurons.

There are only two differences between this network and those investigated previously: each interneuron is connected to only one input and each interneuron has

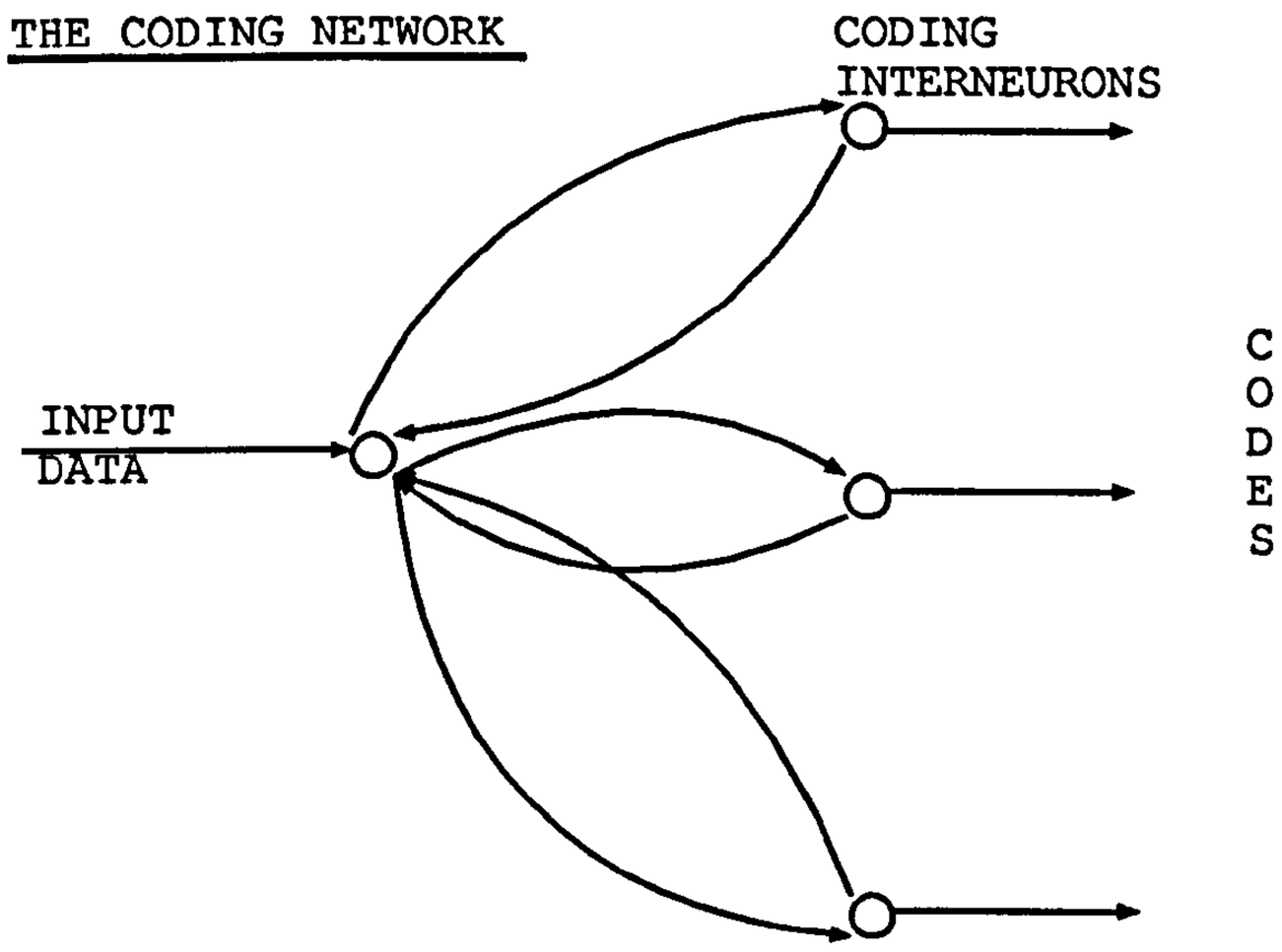


Figure 10: The Coding Network. Raw data is fed in from the left; binary codes emerge from the right

a threshold above which its weighted inputs must sum in order to force a positive firing; if the threshold is not reached, the interneuron will have a negative activation. In detail,

- Each interneuron,in turn, receives a weighted sum (in this case of only 1) of the y values; however, each interneuron has a threshold above which its activation will have a positive value and below which the activation will be negative. We choose the threshold for *all* interneurons to be 0. Then, if z_i is the activation of this interneuron,

$$\begin{aligned} z_i &= 1 \text{ if } w_i x > 0 \\ &= -1 \text{ if } w_i x < 0 \end{aligned}$$

- This activation is then returned (weighted) and subtracted from the y values.

$$y(t+1) = y(t) - v_i z_i$$

where y(t) is the value of y at time t (and y(0)=x).

- The weights, w_i and v_i are updated according to the same rule as previously, i.e. a simple Hebbian learning rule

$$w_i = w_i + \eta y z_i$$

$$v_i = v_i + \eta y z_i$$

Note that the z_i values are either 1 or -1; changing the weights is the sole method of learning in the system to ensure that appropriate codes are found.

We wish to emphasise that we have not programmed a threshold nor any specific function which monitors the variance of the data and adjusts the network's response appropriately.

5.4.1 Results

A typical set of results is shown in Table 22. These results are for an 6 interneuron network which is learning from a set of x-values generated from a uniform distribution[2] between 2 and 4. The network used a learning rate of 0.01 and ran for 10000 iterations

Several points are worth noting:

- First the coding seems fairly inefficient in that the first figure is always 1. This is due to our insistence that all means are zero. Thus the first code element is always 1 for inputs > 0 (see Section 5.4.2)

- If we wish a code where the first interneuron performs maximum discrimination, (i.e. in the above example, all inputs less than 3 would be coded as -1, all inputs > 3 would be coded as $+1$) we would use a threshold which will also learn; a rule such as

$$\theta_j = \theta_j + \alpha w_j . x$$

[2]We use a uniform distribution here to make it clear why each weight has converged to the value it has converged to

Decimal							Decimal						
2.0	1	0	0	0	0	0	3.0	1	0	1	1	1	1
2.1	1	0	0	0	0	1	3.1	1	1	0	0	0	1
2.2	1	0	0	0	1	1	3.2	1	1	0	0	1	1
2.3	1	0	0	1	0	0	3.3	1	1	0	1	0	0
2.4	1	0	0	1	1	0	3.4	1	1	0	1	1	0
2.5	1	0	1	0	0	0	3.5	1	1	0	1	1	1
2.6	1	0	1	0	0	1	3.6	1	1	1	0	0	1
2.7	1	0	1	0	1	1	3.7	1	1	1	0	1	0
2.8	1	0	1	1	0	0	3.8	1	1	1	1	0	0
2.9	1	0	1	1	1	0	3.9	1	1	1	1	1	0

Table 22: A section of coding for vectors produced by a 1-input 6-interneuron network for input data from a uniform distribution between 2 and 4. Learning rate = 0.01, number of trials = 10000. We have replaced -1 with 0 to highlight the binary nature of the code

Interneuron	1	2	3	4	5	6
Weight (w)	3.005	0.505	0.254	0.123	0.063	0.031

Table 23: The weights which the above network learned using only simple Hebbian learning

where θ_j is the threshold for the j^{th} interneuron is an entirely local rule and easy to implement; however, in keeping with our design philosophy of maintaining simplicity, we have not implemented that here.

- The code seems to be topology preserving - similar inputs have similar outputs (see Section 5.4.4).

- Experiments have shown that larger networks have no difficulty in providing more detailed codes and require only a slight increase in time as each element of the coding is done on the error remaining after the previous interneurons have performed their coding.

- A topological feature map using the interneuron network has one major advantage over e.g. a Kohonen feature map: it can easily be re-implemented to show a hierarchy of subfeatures (see Section 5.5) by adding a new level of coding interneuron

- Lower valued digits are automatically coded more slowly and hence are less prone to an unusual input creating large changes

5.4.2 Statistics and Weights

We will investigate what the weights are actually learning by considering their values at convergence. In general, we are using simple Hebbian learning, so

$$\Delta w_i = \eta y(i) z_i$$

where the subscript denote the i^{th} interneuron and y(i) denotes the value of y at time i. The proof that $v_i = w_i$ is similar to that shown previously and will not be repeated here. We investigate 2 interesting cases before looking at the general case:

1. A zero mean symmetric distribution.

   - Consider w_1, the weight to the first interneuron. Then,

$$\begin{aligned} \Delta w_1 &= \eta y(1) z_1 \\ &= \eta(x - w_1 z_1) z_1 \\ &= \eta(x z_1 - w_1) \end{aligned} \tag{144}$$

   since $z_1^2 = 1$. Hence at convergence

$$\langle \Delta w_1 \rangle = 0 \iff \langle w_1 \rangle = \langle x z_1 \rangle \tag{145}$$

   With a zero mean symmetric distribution, $z_1 = -1$ when x is negative and $z_1 = +1$ when x is positive; therefore $z_1 x = |x|$ and so

$$w_1 = \langle |x| \rangle$$

   at convergence i.e. w_i converges to the expected value of the absolute value of the input data, i.e. the mean absolute value. This has the effect of mapping the two halves of the distribution to a tighter (bipolar) distribution which is mostly contained within the interval $[-\overline{|x|}, \overline{|x|}]$. See Figure 11.

2. A positive, compact distribution.

   By compact, we mean a distribution which contains no holes.

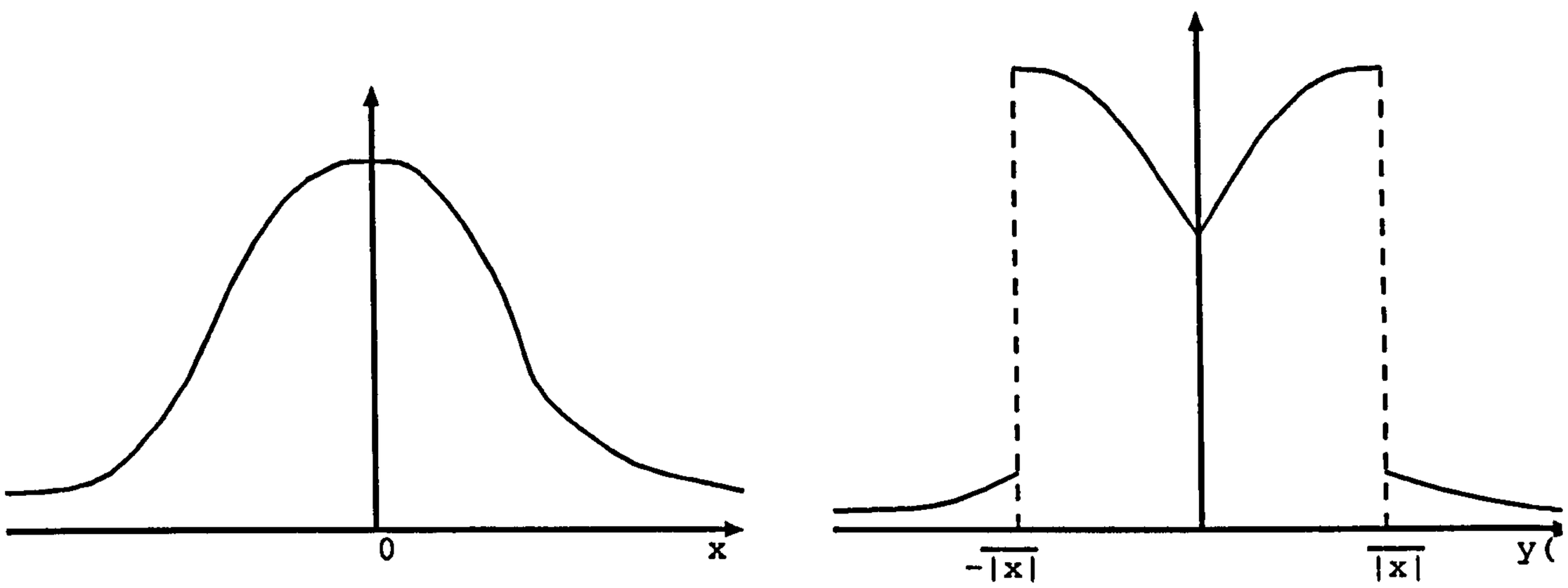


Figure 11: The original zero mean, symmetric distribution (as seen by the first interneuron) is transformed into a bipolar distribution (which is seen by the second interneuron) by the activity of the first interneuron. The bipolar distribution is almost totally between the negative and positive expected absolute values of the input distribution.

- Consider w_1; as before, we can show that, at convergence,

$$\langle \Delta w_1 \rangle = 0 \iff \langle w_1 \rangle = \langle x z_1 \rangle \tag{146}$$

With this distribution, $z_1 = +1$ for all input values of x. So

$$w_1 = \langle x \rangle$$

i.e. the mean value of the input data

- Now consider w_2, the weight to the second interneuron. Then,

$$\begin{aligned} \Delta w_2 &= \eta y(2) z_2 \\ &= \eta (y(1) - w_2 z_2) z_2 \\ &= \eta (y(1) z_2 - w_2) \end{aligned} \tag{147}$$

since $z_2^2 = 1$. But

$$\begin{aligned} y(1) &= x - w_1 z_1 \\ &= x - \langle x \rangle \end{aligned}$$

Now if $x > \langle x \rangle$, i.e. $x > w_1$, then $z_2 = 1$ while if $x < \overline{x}$, i.e. $x < w_1$, then $z_2 = -1$ Thus w_2 at convergence equals $\langle |x - \langle x \rangle| \rangle$. Therefore, at

convergence, w_2 is bisecting the residual area of the distribution after w_1 has subtracted out the mean. So w_2 for this distribution is performing the task which w_1 performed for the symmetric distribution.

3. In general, for values of x drawn from any distribution

$$\begin{aligned} \Delta w_i &= \eta y(i) z_i \\ &= \eta(y(i-1) - w_i z_i) z_i \\ &= \eta(y(i-1) z_i - w_i) \\ \text{Therefore, } w_i &\rightarrow \langle y(i-1) z_i \rangle \text{ at convergence} \end{aligned}$$

and if $y(i-1) > 0$ then $z_i = 1$
while if $y(i-1) < 0$ then $z_i = -1$.
Therefore, at convergence, $w_i = \langle |y(i-1)| \rangle$

In general, the z values correspond to the coding taking place while the y values represent the error after the coding has taken place.

The coding of a uniform distribution is shown in Table 22. We use a uniform distribution this time to make it easy to corroborate that the network is performing an efficient coding; note from Table 23 that $w_1 = \langle |x| \rangle = \langle x \rangle$ and $w_2 = \langle |x - \langle x \rangle| \rangle$ etc..

5.4.3 Reconstruction Error

There are 2 possible sources of reconstruction error: from the Principal Component Analysis and from the coding scheme in each PC direction. Each must be considered independently.

Reconstruction Error from Coding

We show that, if we use the codes produced by this method to reconstruct the original magnitude of the vector in each direction, we can make the expected absolute reconstruction error from a final code (sometimes called the mean quantization error) arbitrarily small by simply adding new coding interneurons.

Let us assume that we cannot i.e that there exists an $\epsilon > 0$ such that all mean absolute reconstruction errors are greater than ϵ. We note from the above that this is equivalent to showing that the value of $w_i > \epsilon$ for all i.

Note that for a finite distribution, the maximum possible absolute error after the first coding is

$$E_{Max} = max(|x_{Max} - \langle|x|\rangle|, |x_{Min} - \langle|x|\rangle|, \langle|x|\rangle)$$

where x_{Max}, x_{Min} is the largest (resp. smallest) possible member of the distribution. Thus E_{Max} is finite.

Consider a particular input x. From the results in the last section, because the system is creating the coding at z and subtracting the weighted coding at y, the value $y(i)$ is simply the error between the code found by the first i coding interneurons and the input x after i codings have taken place. Therefore $\langle|y(i)|\rangle$ is the mean absolute error after i codings of the input. Now,

$$\begin{aligned} y(i) &= y(i-1) - w_i z_i \\ &= y(i-1) - \langle|y(i-1)|\rangle z_i \end{aligned}$$

If $y(i-1) < 0$, $z_i = -1$ and

$$\begin{aligned} y(i) &= y(i-1) + \langle|y(i-1)|\rangle \\ &= -|y(i-1)| + \langle|y(i-1)|\rangle \end{aligned}$$

while if $y(i-1) > 0$, $z_i = +1$ and

$$\begin{aligned} y(i) &= y(i-1) - \langle|y(i-1)|\rangle \\ &= |y(i-1)| - \langle|y(i-1)|\rangle \end{aligned}$$

Therefore the amplitude of y(i) is the difference between the absolute value of y(i-1) and the mean absolute value of y(i-1). Thus

$$\begin{aligned} |y(i)| &= |\{|y(i-1)| - \langle|y(i-1)|\rangle\}| \\ &= |\{|y(i-1)| - w_i\}| \\ &< ||y(i-1)| - \epsilon| \end{aligned}$$

since all terms are positive. Therefore the absolute error at each stage is decreasing by more than ϵ. Therefore the mean absolute error is also decreasing by at least ϵ at each stage. Now the initial maximum error is E_{Max} which is finite and the absolute error is decreasing by a finite amount each time and so cannot remain above ϵ for all time.

Therefore, we can make the quantization error arbitrarily small by continuing the coding for a sufficient number of coding interneurons.

Reconstruction Error from the PCA

Since the expected error in the Principal Component representation of the data after having projected N-dimensional input data onto M PCs is given by

$$E = \sqrt{\sum_{i=M+1}^{N} \lambda_i} \tag{148}$$

where λ_i is the i^{th} eigenvalue in normal order, the error may be made arbitrarily small by increasing M. In particular, for full rank data, it may be necessary to have M=N.

5.4.4 Topology Preservation -1

In stating that we have a topology preserving coding, we mean that

1. similar inputs should be projected onto similar outputs and

2. similar outputs should be the representations of similar inputs.

This is only approximately true, in general, of feature maps using neural nets e.g. a Kohonen (Kohonen, 1984) map attempts to project the input space onto a network in such a way that the most essential neighbourhood relationships between data in the input space are preserved. Yet input data can be constructed which do not permit a 2-D (or 3-D) mapping to adequately represent all topological equivalences in the data. We first give an intuitive notion of topology preservation here; a more formal proof is given in the next Section.

Consider an n-unit coding of a set of input values. Let a particular point x be represented by $\mathbf{z_i}$ where $\mathbf{z_i} = \{z_{i1}, z_{i2}, ..., z_{in}\}$. The subscript i denotes an ordering of the z values (i.e. of the coding) such that $\mathbf{z_i} < \mathbf{z_j}$ for all $i < j$.

Then any input $x + \Delta x$ is represented by the vector $\mathbf{z_i}$ or by $\mathbf{z_{i-1}}$ or $\mathbf{z_{i+1}}$ for all Δx such that $|\Delta x| < w_n$, the n^{th} weight. i.e. similar inputs are represented by similar outputs.

Now consider two distinct input values, x and u, which are represented by the same $\mathbf{z_j}$. Then

$$x = \sum_{i=1}^{n} w_i z_{ji} + \Delta x$$

$$u = \sum_{i=1}^{n} w_i z_{ji} + \Delta u$$

where Δx and Δu are the errors in the representations after the first n codings have taken place. Therefore,

$$\begin{aligned} x - u &= \Delta x - \Delta u \\ &\leq 2w_n \end{aligned}$$

From the previous section, we know that the value of w_n can be made arbitrarily small and so a single code can be made to represent only similar values. Clearly a similar argument will show that if the values, x and u, are represented by contiguous codes, the values x and u can only be at most $3w_n$ apart. Thus similar codes represent similar values.

5.4.5 Topology Preservation - 2

We will use the results from Section 5.4.3 in this proof: recall that for every $\epsilon > 0$, there exists an n such that $w_n < \epsilon$.

Let M be the metric space defined by that (sub)set of the real numbers defined by the probability distribution of the raw data and the metric, d, defined by the usual Euclidean distance metric.

Let M_1 be the metric space defined by the set of codes (of the real numbers in M) and the metric, d_1, defined by the usual Euclidean distance metric (now calculating in binary).

We do not prove that there is a topology-preserving function which maps M to a particular instance of M_1; however, it is possible to prove that there exists a mapping from the set M to a member of the family $\{M_1^1, M_1^2, M_1^3, ..., M_1^n, ...\}$ where M_1^n is the coding which has length n (i.e. formed by using n coding interneurons). In other words, we can make our mapping as close to a topology preserving mapping as possible by chosing n appropriately.

We shall create an ordering, $\{C_i\}$ of the codes in M_1^n based on the size of their binary values. Thus $C_i < C_j$ for $i < j$. Note that for this coding on M_1^n,if f(x) is the function which codes the inputs i.e. $f : M \rightarrow M_1$, then $(C_i - C_{i-1}) = w_n$, the weight of the n^{th} level of the coding. Note also that the greatest distance between values coded by the same code is also w_n.

- First consider the mapping $f : M \rightarrow M_1$.

  Then take any point $c \in M$. $\forall \epsilon > 0$, we require to prove that $\exists \delta > 0$ such that

  $$d_1(f(c), f(x)) < \epsilon$$

  $$\forall x \in M : d(c, x) < \delta$$

  Choose the coding M_1^n such that $w_n < \frac{1}{2}\epsilon$. Let $\delta = w_n$. Let f map c to class C_i, the i^{th} class of M_1^n. Then for all x such that $d(c, x) < \delta = w_n$, f(x) is either in class C_i or in one of its neighbours C_{i-1} or C_{i+1}. So f(x) is within $2w_n$ of f(c) i.e. $f(x) \in (f(c) - \epsilon, f(c) + \epsilon)$, when d(x,c) $< 2w_n$. i.e.

  $$d_1(f(c), f(x)) < \epsilon \text{ when } d(c, x) < \delta$$

- Now consider the mapping $f : M_1 \rightarrow M$.

  Then take any point $C_i \in M_1$. Given any $\epsilon > 0$, we must prove that

  $$\exists \delta > 0 : d(f(C_i), f(x)) < \epsilon, \forall x \in M : d_1(C_i, x) < \delta$$

Choose n such that $w_n < \frac{1}{2}\epsilon$; this defines the actual representative of M_1 as M_1^n. We chose δ to be equal to 1. Then $\forall x \in (C_i - \delta, C_i + \delta)$ is equivalent to $x \in C_{i-1}, C_i$ or C_{i+1}

Now the maximum distance between the values which code to C_{i-1} or C_{i+1} and those which map to C_i is $2w_n$ i.e. 2 times the remaining error after the n^{th} coding. i.e. $d(f(x), f(c_1)) \leq 2w_n < \epsilon$ for all x in the declared interval.

Now any particular interneuron coding network is not truly topology preserving; however, it can be made arbitrarily close to such a network by increasing the length of the coding. Therefore any particular coding is performing an approximate topology-preserving mapping.

We define an ϵ_0-*topology preserving network* as an interneuron network in which for all points $c \in M$, $\forall \epsilon > \epsilon_0, \exists \delta > 0 : d_1(f(c), f(x)) < \epsilon, \forall x \in M : d(c, x) < \delta$ where $f(x), f(c)$ are the binary codes.

Note that this is equivalent to defining $w_n = \frac{1}{2}\epsilon_0$. Then each network in the sequence of interneuron networks of increasing discrimination is an ϵ_0-topology preserving network with the value of ϵ_0 defined as $2w_n$ for the specific mapping M_1^n.

5.5 A Hierarchical Feature Map

We now propose the complete network shown in Figure 12 and use it in implementing the ideas of the previous sections. The left half of the network is the basic interneuron network described in Section 5.3; the right half of the network is the interneuron coding network described in Section 5.4. The first section will extract the maximum information from the raw input data i.e. data will be projected onto those directions which contain maximum information; the second section will code the data along each dimension independently.

All parts of the system use unsupervised learning. Our learning rule continues to be simple Hebbian learning with no weight decay or clipping of weights.

Since the network is topology preserving in each direction, it is topology preserving in the space spanned by these directions. Therefore the construction can be viewed

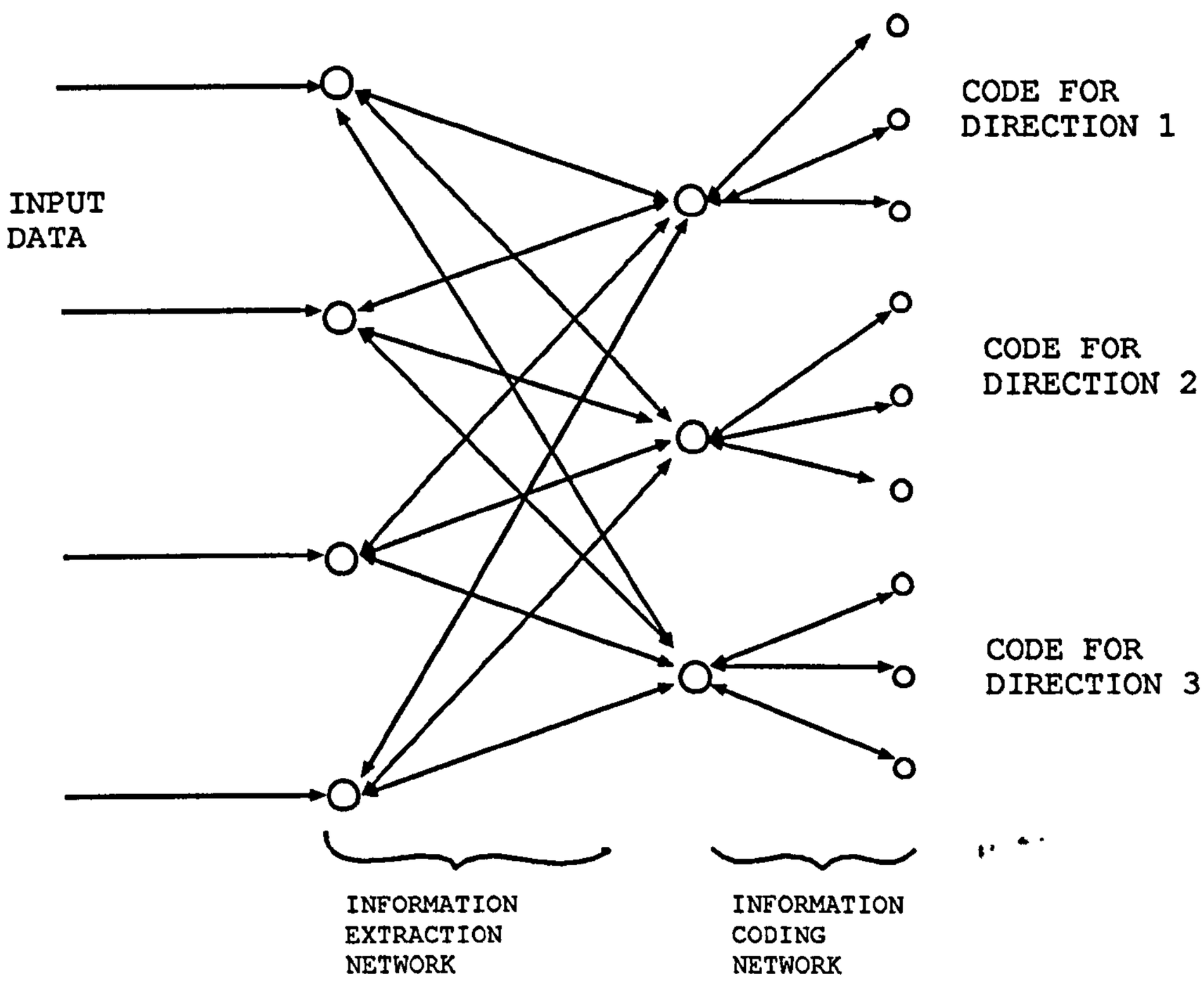


Figure 12: The complete information extraction and coding network. The double headed arrow represents both the feedforward weights and the feedback weights

as forming a feature map which is topology preserving in the major information directions of the data. While it is possible to create continuous multi-dimensional maps which are topologically different from their projection onto this subspace, each such anomaly will tend to swing the principal components in the direction of the anomaly suggesting that such anomalies can at most be a minor part of the input data. Further, as will be shown, augmenting this type of feature map to take account of such features is a simple process.

The inherently modular nature of the network (see Figure 12) allows us to consider the effects of augmenting the network as a purely local process. This modular nature is a direct consequence of the Principal Component Analysis performed by the first section which leads to orthogonal input vectors for the second section.

5.5.1 An Example

Kohonen and Ritter (Ritter and Kohonen, 1989) used the data shown in Table 24 to illustrate the emergence of a "semantic map" using a Kohonen feature map. They trained a Kohonen network to learn to associate the name of each animal with the attributes with which it might be associated. They then showed that by entering the name (and only the name) of the animal into the network,

- different nodes responded maximally to different animals
- the spatial organisation (on a 2-D grid) grouped similar animals together

We have chosen to repeat this experiment with the interneuron network as it might be thought that organising such a set of data might be difficult for a statistics-based network such as the interneuron network. Figure 13 shows that the first 2 principal components are sufficient to make the major differentiation into animals and birds and within each group there is some differentiation into subgroups. However, the advantage of using the interneuron network as an encoding network is that further differentiation is possible by looking at other directions and the third principal component shows clearly the differentiation into subgroups taking place in this direction. This is equivalent to a semantic interpretation of data which allows us to be aware

	dove	hen	duck	goose	owl	hawk	eagle	fox	dog	wolf	cat	tiger	lion	horse	zebra	cow
small	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
medium	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
big	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
2 legs	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
4 legs	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
hair	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
hooves	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
mane	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
feathers	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
hunt	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
run	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
fly	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
swim	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 24: Animal names and their attributes

of similarities in attributes within a group while being aware of differences in other attributes between members of the same group. This is the feature which allows us to call the map a hierarchical feature map.

Following Kohonen and Ritter, each animal was represented by a 29-bit vector, the first 16 bits of which were 0 except for the bit which identified the particular animal. The other bits were the animal's attributes as shown in Table 24. The output data were plotted in the obvious manner - each output vector was identified as a binary number (1/0) and converted to decimal to give the coordinates in each direction.

While we do not believe that this is a good (or particularly useful) application of a data-driven network, we believe that this example highlights the hierarchical nature of the coding possible with an interneuron network.

5.5.2 The Principal Component Analysis

Another advantage of the proposed method is that to some extent, it allows us to look inside the results in order to investigate how the results came about. This is particularly interesting in examples such as the current one where the data have strong everyday connotations for us.

Table 25 shows the first 3 Principal Components of the covariance matrix of the data of Table 24 as identified by the interneuron weights of the middle layer. It is easily verified that the vectors form an orthonormal basis of a 3 dimensional subspace of the data.

- The first Principal Component is most strongly identifying animal type features: animals tend to be big, have 4 legs and hair; some have hooves or a mane;

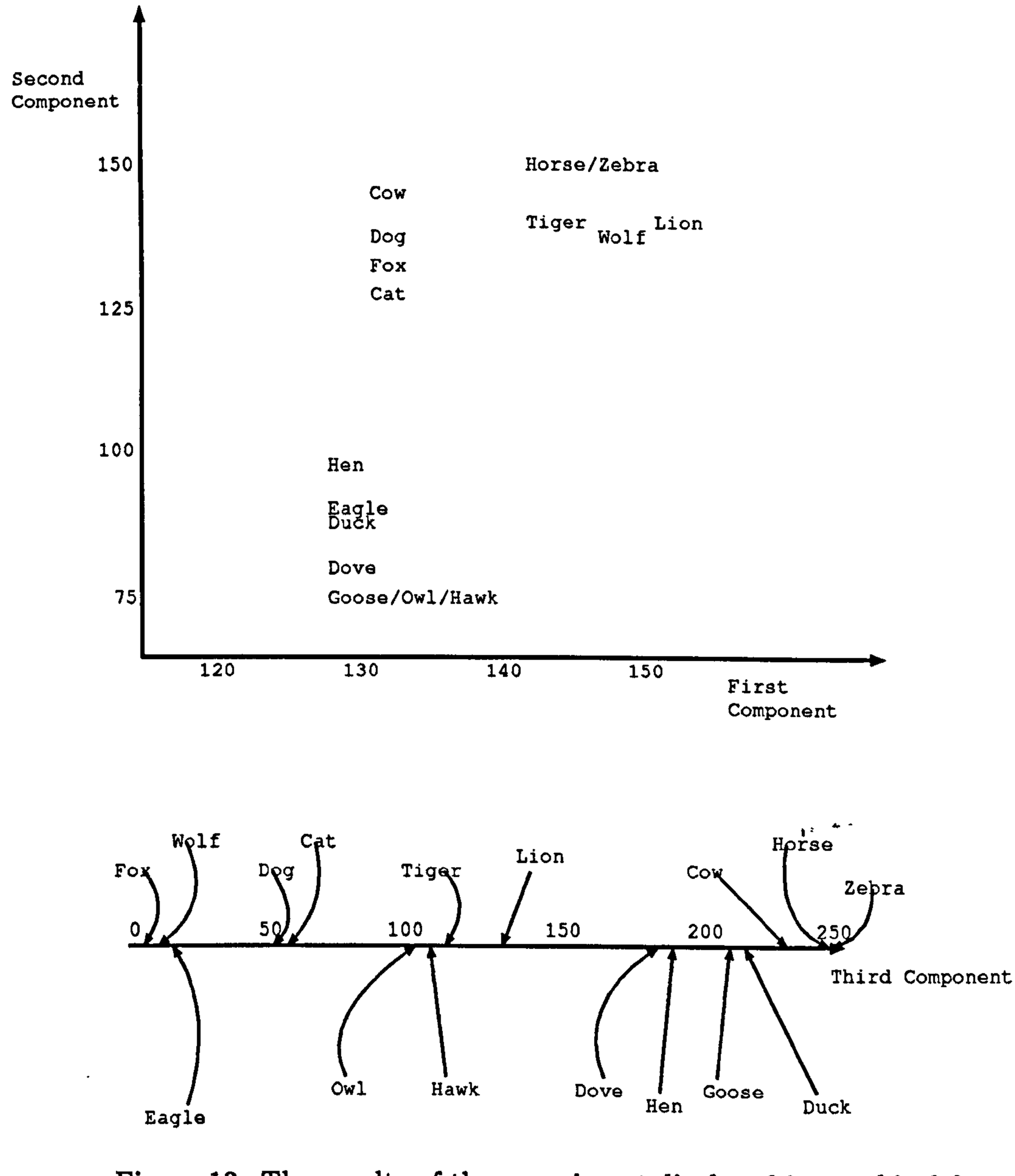


Figure 13: The results of the experiment displayed in graphical form

Attribute/class	First PC	Second PC	Third PC
dove	0.017	-0.071	0.054
hen	0.013	-0.057	0.059
duck	0.014	-0.062	0.083
goose	0.018	-0.077	0.078
owl	0.027	-0.077	-0.022
hawk	0.027	-0.077	-0.022
eagle	0.028	-0.060	-0.116
fox	0.044	0.008	-0.155
dog	0.044	0.021	-0.066
wolf	0.061	0.020	-0.118
cat	0.044	-0.009	-0.062
tiger	0.057	0.021	-0.019
lion	0.065	0.026	0.005
horse	0.059	0.036	0.140
zebra	0.059	0.036	0.140
cow	0.041	0.024	0.102
small	0.161	-0.431	0.166
medium	0.177	-0.012	-0.457
big	0.281	0.143	0.369
2 legs	0.146	-0.482	0.113
4 legs	0.474	0.183	-0.034
hair	0.474	0.183	-0.034
hooves	0.159	0.096	0.383
mane	0.243	0.116	0.167
feathers	0.146	-0.482	0.112
hunt	0.354	-0.149	-0.512
run	0.345	0.159	0.081
fly	0.118	-0.364	-0.029
swim	0.032	-0.139	0.161

Table 25: The first 3 Principal Components of the input data's covariance matrix. It is easily verified that they form an orthonormal basis of the 3 dimensional subspace

somewhat more hunt and run.

- The second Principal Component completes the job: the birds all are represented by a negative component as are the small, medium, 2 legs, feathers, hunt, fly and swim attributes. Also, it can be seen that the more prototypical bird-like features have larger absolute values e.g. |fly| > |swim| since the prototypical bird is more likely to fly than swim. Note also that the cat has a small negative value brought about by its prototypical bird-like attribute of smallness. Thus, in the map (Figure 13), the cat appears closest to the birds in the direction of the second PC.

- The Third Principal Component seems to be mainly differentiating the hunters from the non-hunters, though differentiation in size and between fliers and swimmers is also taking place

Note that the components defining e.g. horse and zebra are identical in all 3 directions as there is nothing in the input data provided which will allow us to discriminate between these groups. Similarly, the attributes "4 legs" and "hair" are identically represented as they are identically distributed in the information we have given.

5.5.3 Augmenting a Map

The desire to augment a map may be brought about by 2 circumstances:

1. The map is too crude as too little information has been extracted from the original input data.

   To extract more information from the raw data, we must find a new Principal component along which to project the data. Therefore, we must create a new data extraction interneuron i.e. in the central layer of the network. Now, by adding our new interneuron at the end of the learning process described in Equations (139), (140) and (141), we are not disturbing the learning in any of the other directions which have already been found. Therefore the new direction can be found without disturbing the principal components already

found; therefore the existing codes are not disturbed and the coding of the new dimension can be done independently of the existing codes.

2. The map is too crude as too little discrimination has taken place in a particular direction

   Note again that the modular nature of the map allows the discrimination in each direction to be modified independently of that in the other directions. Thus, in the coding within a particular direction, we may simply add a new coding interneuron into the network and, provided it learns after all the others have learned, it will simply learn to bisect the remaining information after the others have subtracted their activations. In other words, a new coding interneuron will simply provide increased discrimination within that direction and will affect neither the coding in other directions nor the existing coding in its direction.

Experimental results have confirmed this analysis.

Note that the potential for improving a map after it has been constructed is an improvement on the Kohonen map whose parameters must be specified in advance.

5.5.4 Reparing a Damaged Map

The situation described in this section is more complex than that in the last. Again 2 possibilities must be considered:

1. Where damage has occurred in the coding layer.

   Here we consider what happens when a neuron is damaged, loses its learning (its weight assumes a small random number) yet still remains in the same physical position, with the same physical potential for learning. Experimental results have shown that even in the worst case, that of damaging the coding interneuron which extracts the greatest information (i.e. the first coding interneuron), relearning is very fast. The network has the advantage that, when it loses an interneuron, all of the weights which learn after the damaged interneuron's activities, increase in the order in which they are learning - there is no complex "dog-fight" which is sometimes seen in artificial neural networks. All of the

other interneurons whose weights had increased on a pro-rata basis gradually lose their gains to the recovering interneuron till the previously existing weight values are regained. If the new interneuron is deemed to be irrepairable, i.e. does not recover its ability to learn, the other interneurons simply step up their information coding capacities in order i.e. each weight increases until it is performing exactly the same discrimination that the weight above it was previously performing.

2. Where damage has occurred in the information extraction layer

   This is potentially the most disruptive damage in the network: when 1 interneuron in the central layer is damaged, the interneurons after it in the learning sequence are able to take advantage of its incapacity to capture its previously held information. This will obviously have an effect on those coding interneurons which will have to readjust their weights in order to accommodate the new information-bearing capacity of the first layer interneurons. Each weight will tend to increase as it is now working for an interneuron which has captured a little more of the available information in the network. However, this is a minimal rearrangement: each coding neuron is most similar in weight to its equivalent coding neuron for the previous direction. All coding neurons will augment their weights in unison and no realignment of coding is necessary.

   All increases and decreases in weight values are done as though synchronised; there is never an example of the complex untwisting which we see in developing Kohonen nets.

Again experimental results have confirmed this analysis.

Note again that the interneuron map is an improvement on the Kohonen map in which the destruction of a single competitive neuron reduces that network's ability to respond appropriately to the region which was optimally represented by that neuron.

5.5.5 Summary

We have described a novel robust topology-preserving neural network which seems to have significant advantages over current networks with equivalent properties. We must emphasise that both parts of the network are necessary:

- Without the first section, the second section would merely divide up the information as a whole. In other words, the amplitude of the message would determine the coding and no account would be taken of its content.

- Without the second section, the first section would find those directions of maximum information but an observer would not know e.g. if an output of 1.0 came from a direction with zero mean and variance 0.5 or mean 1 and variance 10. The second section of the network is essential to calibrate the mapping.

5.6 A Single Type of Neuron

We have however increased the complexity of our network considerably in that we now have 2 different types of neuron in the network. In addition, it is well known that biological neurons are not accurately modelled by either simple linear summation neurons nor by neurons which have step functions as activation functions. Instead a degree of non-linearity of response has been found which is usually modelled by a sigmoidal function.

Using tanh() (See Figure 14) as an activation function for the interneurons allows a unified model of the above two types of interneurons to be created:

- tanh() is approximately linear in its middle section. We may adjust the range of middle section over which it is linear by using the parameter λ in $z_i = tanh(\lambda \sum_j w_{ij} y_j)$. To get a large linear section requires a small value of λ. Experimental results have confirmed that $\lambda = 0.1$ is sufficiently small to approximate a linear function with which Principal Components can be found as before.

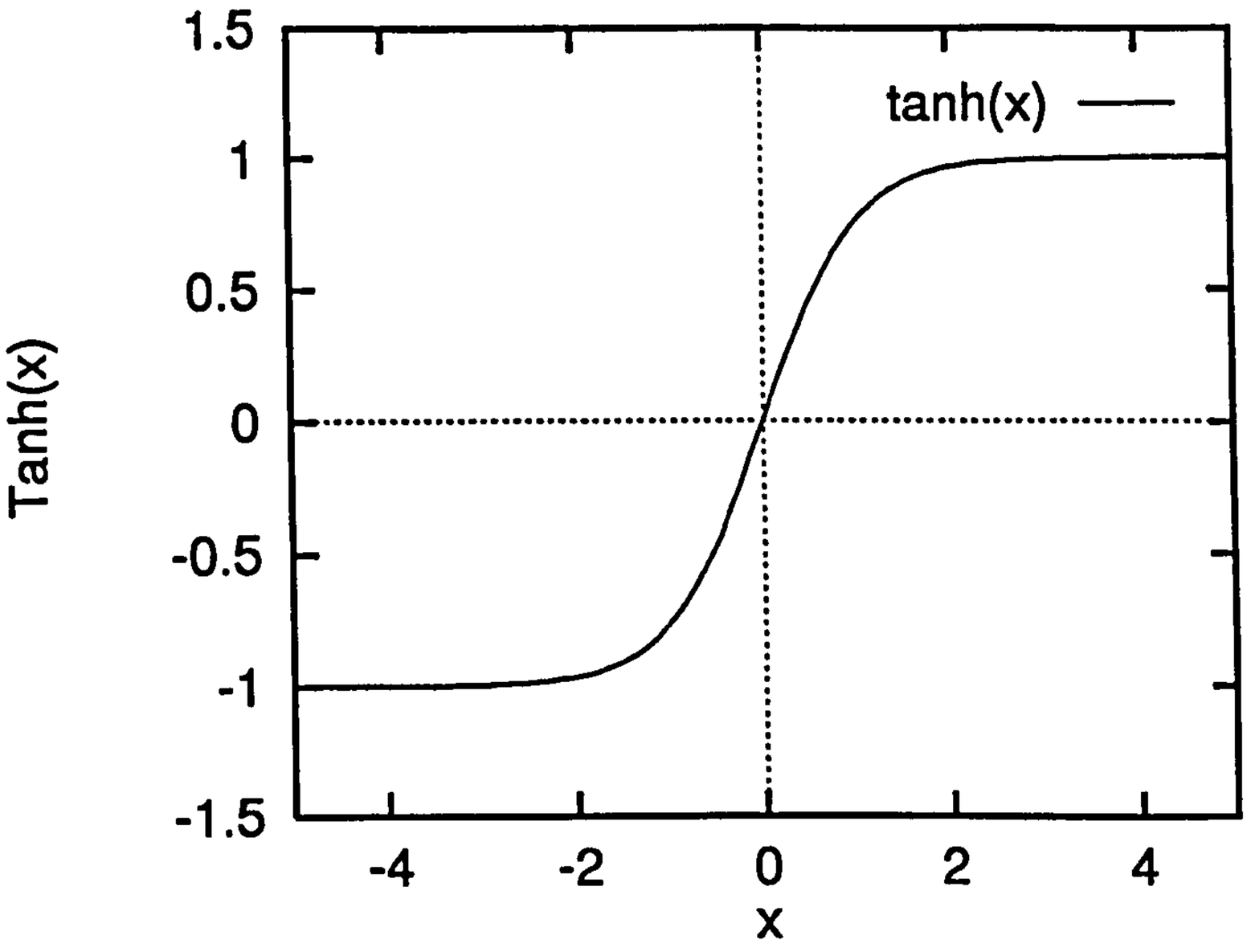


Figure 14: The Tanh() function

- tanh() may be made more dichotomous by adjusting the parameter, λ, upwards. A value of O(10) is sufficient to create an interneuron which performs a smoothed coding of the input data (but see below). The larger the value of λ, the more step-like the function becomes.

Therefore a single type of interneuron with different λ parameters can perform both the information extraction and the information coding described above. Both sets of interneurons will have an activation function, $\tanh(\lambda \sum_j w_{ij} y_j)$. The sole difference is that interneurons in the first layer have the parameter $\lambda = 0.1$, while those in the second have the parameter $\lambda = 10$. The first value is, of course, dependent on the distribution of the input data and is based on distributions with single figure standard deviation and mean; the second depends on the amount of discrimination (length of the code) required.

Using such an activation function with the information extracting interneurons has several implications:

1. The output values, z, at these interneurons are all in the range (-1,1)

2. Thus the weights in the first part of the network will grow much larger than previously

3. The learning rate in this network can be made much larger than before as the z values are constrained to this small range. Previous networks used a learning rate of O(0.0001); with a tanh() activation function a learning rate of O(0.1) is possible.

However, there is a drawback to the use of this activation function in the coding layer: with the values stated, codes for the first 3 coding interneurons in each direction agree with those found with the step function network. However, for the later coding interneurons a lack of discrimination develops leading to very imprecise codes. By the 6^{th} coding interneuron, we have lost all claim to have a topology-preserving network.

To show that this cannot be wholly explained by the foreshortening of the outputs from the first section of the network to the range (-1,1), we have used the same data which produced the coding shown in Table 22 and multiplied the outputs of the first section by 10 to put them in the same approximate range as before. The activation values of the sixth digit (i.e sixth coding interneuron) are shown in Figure 15. Clearly

- the code cannot be called topology preserving in the neighbourhood of the changes from 1 to -1.

- the number of differentiated codes is much smaller than it should be

This effect can be ameliorated by increasing the value of λ from 10 to 100. The results of this can be seen in Figure 16. Clearly, however, using this activation function means that we have lost the ability to subclass at will almost infinitely often. Whether this is important in a biological context must still be investigated.

5.7 Conclusion

We have in this Chapter introduced asymmetry into our interneuron network in 2 physically realistic ways: by considering the fact that weights cannot change from

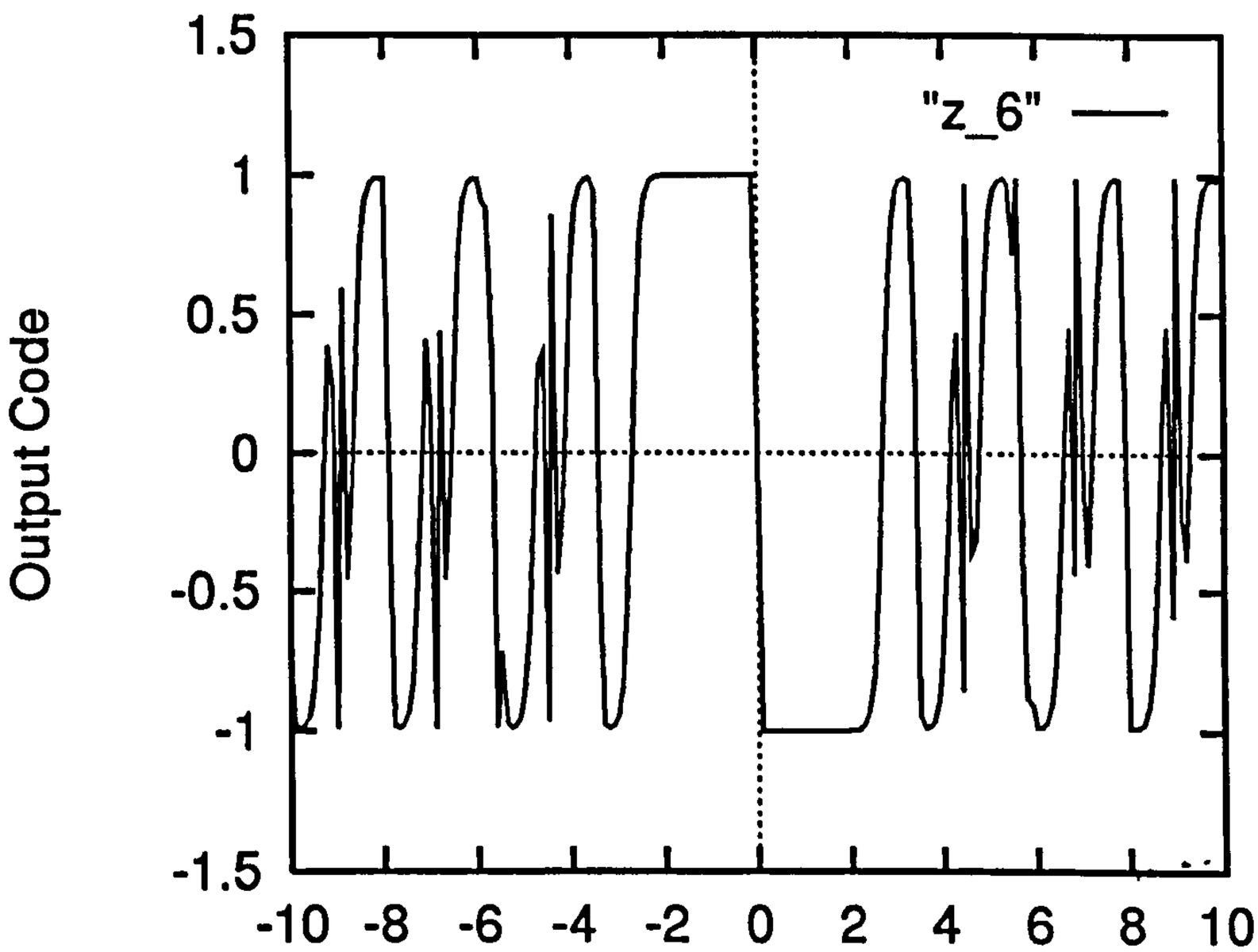


Figure 15: The coding produced by the sixth coding interneuron on 10*the output of the first principal component when both are using tanh(). $\lambda = 10$.

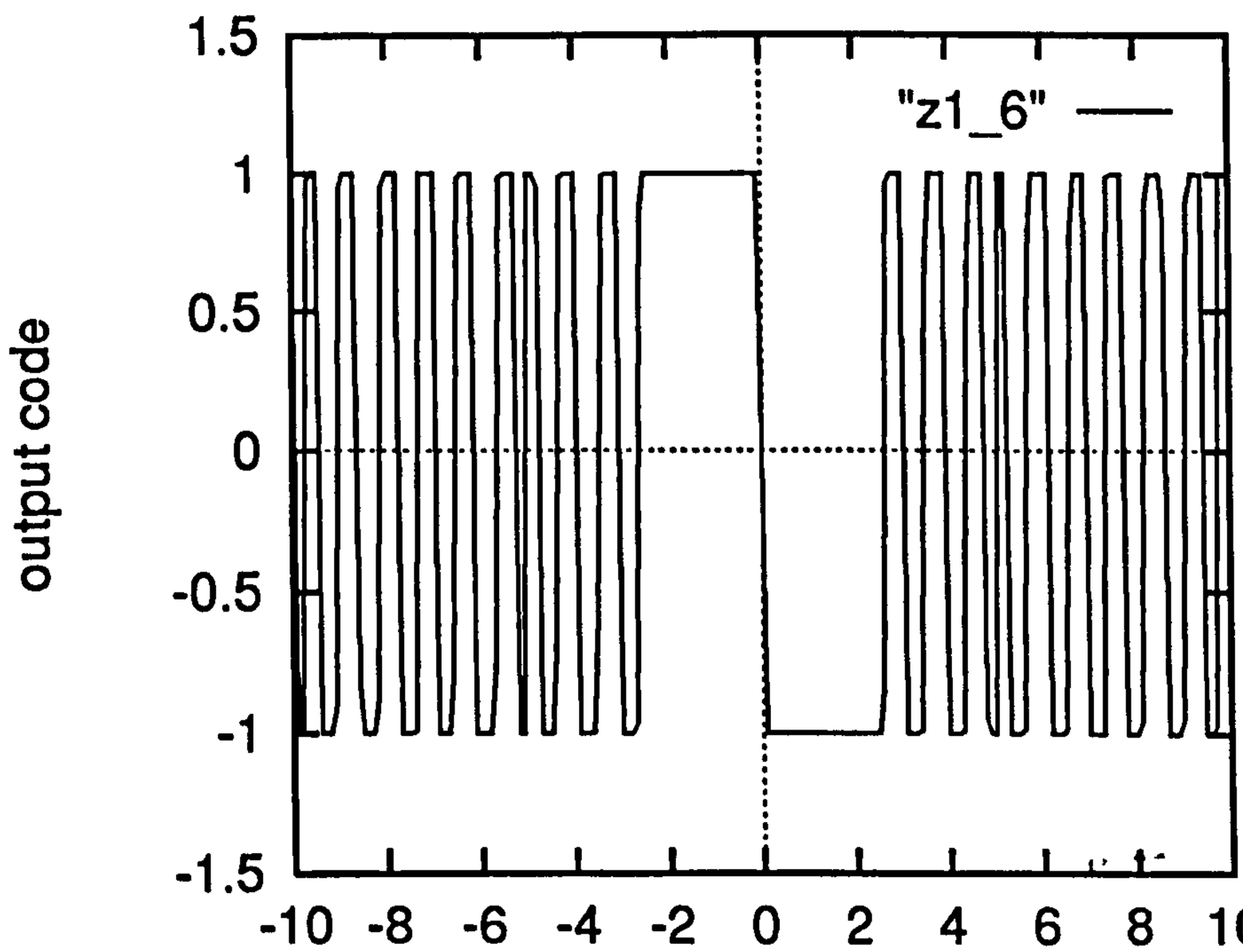


Figure 16: The coding produced by the sixth coding interneuron on 10* the output of the first principal component when both are using tanh(). $\lambda = 100$.

positive to negative during the learning process and by considering the physical distance between neurons and allowing activation transfer to be a linear function of this distance. Both were then shown to allow the network to perform an actual Principal Component Analysis on the input data.

We then for the first time introduced a non-linear feature into the network and developed a new method of creating a topology-preserving feature map composed of a 2 stage process involving projection of the input data onto the directions of maximum information followed by a discrimination process within each direction.

The simple interneuron network performs the projection while the interneuron coding network performs the discrimination. Both use only simple Hebbian learning with no normalisation, weight decay or clipping of weights.

An attempt to produce a biologically feasible unification of the two types of interneurons was partially successful in that a single type of interneuron with different values of a parameter in its activation function has been shown to be capable of performing both tasks necessary to produce the feature map; the single addition of an activation function $tanh(\lambda x)$ allowed us to dispense with the threshold in the coding interneurons. This network requires parameter tuning if it is to perform more than a crude discrimination in each direction.

The success of our first non-linearity suggests that this may be an interesting line of further research. We will see that this is indeed the case in the next Chapter.

Chapter 6

Non-linear Structure Extraction

6.1 Introduction

Cross-fertilisation between the fields of artificial neural networks and statistics has recently proved fruitful. In unsupervised learning, the realisation that simple neural network architectures are capable of performing classical statistical analysis has allowed insight into the operation of simple Hebbian neural networks and allowed the results of neural networks to be related to human psychophysical performance. Principal Component networks have been the major outcomes of this research. Here we use the same neural network architecture as in previous Chapters and show that it has other important statistical properties.

Principal Component Analysis(PCA) has proved to be a powerful tool for the investigation and analysis of large data sets. However, some structure in data sets is not identifiable by means of the linear associations (correlations) among the variables; such effects as clustering or definition of edges of data-sets are easily identified using the human eye on low dimension projections of data but are not achievable by using the tools of classical multivariate analysis. For example, Figure 17 shows two elipsoids representing the shapes of two data clusters characterised by features X1 and X2; the first Principal Component is along the direction X1, yet the structure in the data - the two clusters - is not visible in the projection onto this direction. This problem increases in severity as the dimensionality of the data increases. The success of PCA

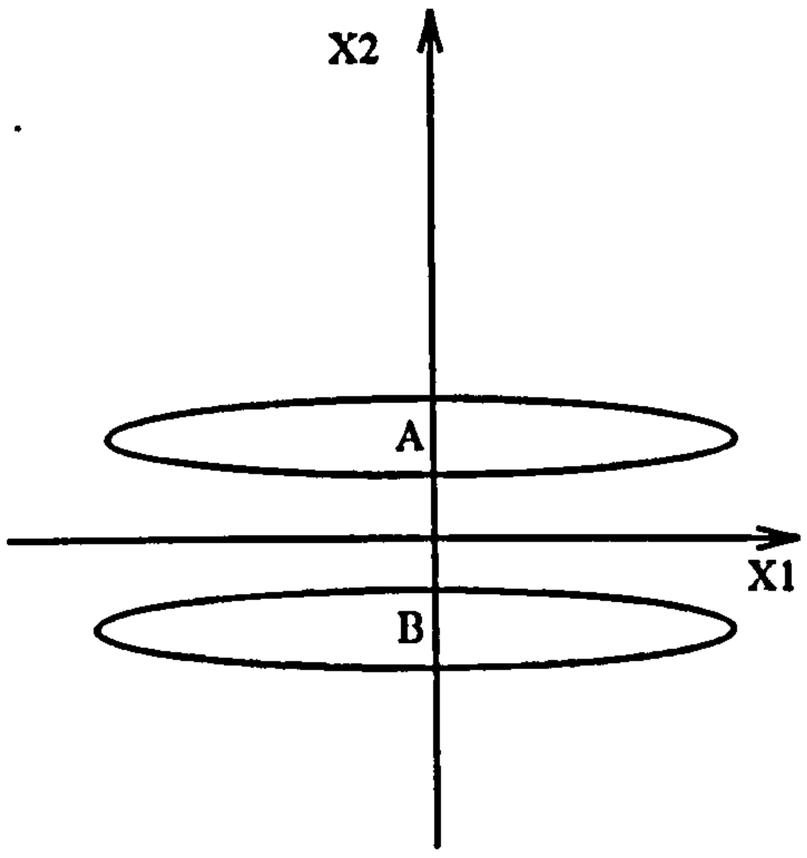


Figure 17: The two elipsoids indicate the shapes of the clusters of data points characterised by two features, X1 and X2. The clusters are clearly visible to the human eye, yet are completely hidden when projected onto the first Principal Component direction, X1.

has, in part, been because those directions which contain most of the variance in a data set will tend to contain most of the structure in the data set. However, this relationship is not logically necessary.

Exploratory Projection Pursuit (EPP)[1] defines a recent form of exploratory data analysis methods which attempt to find "interesting" directions in high dimensional data (for reviews see (Huber, 1985; Jones and Sibson, 1987)). We introduce a non-linearity to our PCA network and show that it is capable of performing an EPP.

6.2 Exploratory Projection Pursuit

The group of methods based on Projection Pursuit is based on one central idea: rather than solving the difficult problem of identifying structure in high dimensional data, project the data onto a low dimensional subspace and look for structure in the projection. However not all projections will reveal the data's structure equally well. Therefore we define an index that measures how "interesting" a given projection is,

[1]Some of this work has already appeared in (Fyfe and Baddeley, 1994) and will appear in (Fyfe and Baddeley,).

and then represent the data in terms of the projections that maximise the index and are therefore maximally "interesting". We will initially restrict our attention to one dimensional subspaces i.e. we will identify an index for each line in the space and attempt to maximise the index in order to make projections of the raw data onto the line as interesting as possible.

Clearly the choice of index is the crucial factor in Projection Pursuit, and the index is specified by our desire to identify interesting directions. Therefore we must define what we mean by "interesting directions".

6.2.1 Interesting Directions

Friedman (Friedman, 1987) notes that what constitutes an interesting direction is more difficult to define than what constitutes an uninteresting direction. The idea of "interestingness" is usually defined in relation to the oft-quoted observation of Diaconis and Freedman((Diaconis and Freedman, 1984)) that most projections of high-dimensional data onto arbitrary lines through most multi-dimensional data give almost Gaussian distributions. This would suggest that if we wish to identify "interesting" features in data, we should look for those directions α, projections onto which are as non-Gaussian as possible. Thus, we will look for an $I(\alpha)$, an index function of the direction α, which is maximum when the projection of the distribution onto α is furthest from Gaussian.

Two common measures of deviation from a Gaussian distribution are based on the higher order moments of the distribution (see Figure 18). Skewness is based on the normalised third moment of the distribution and basically measures if the distribution is symmetrical. Kurtosis is based on the normalised fourth moment of the distribution and measures the heaviness of the tails of a distribution. A bimodal distribution will often also have a negative kurtosis and therefore kurtosis can signal that a particular distribution shows evidence of clustering. Whilst these measures have there drawbacks as measures of deviation from normality (particularly their sensitivity to outliers), their simplicity makes them ideal for explanatory purposes.

In passing, we note that if we know what type of interesting structure we expect to find in the data set, instead of moving away from the uninteresting Gaussian

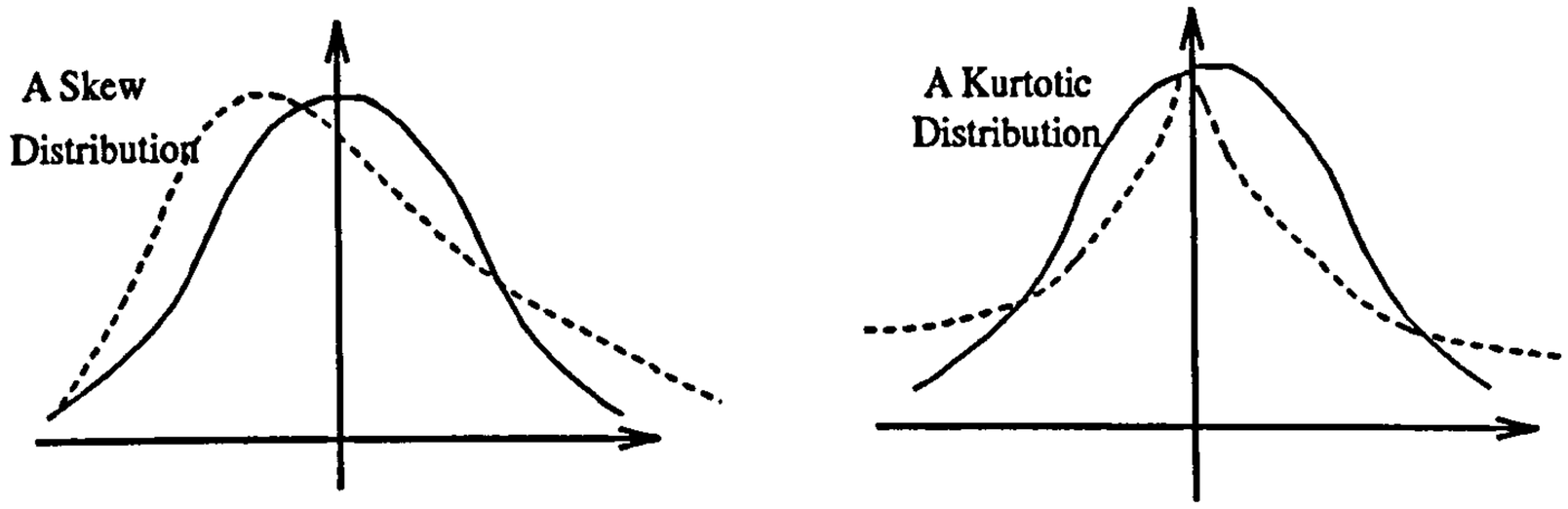


Figure 18: Deviations from Gaussian distributions: the dotted line on the left represents a negatively skewed distribution; that on the right represents a positively kurtotic distribution; in each case, the solid line represents a Gaussian distribution

distribution, we could move towards the interesting direction.

6.3 The Data and Sphering

Because a Gaussian distribution with mean a and variance x is no more or less interesting than a Gaussian distribution with mean b and variance y - indeed this second order structure can obscure higher order and more interesting structure - we remove such information from the data. This is known as "sphering". That is, the raw data is translated till it has mean 0, projected onto the principal component directions and multiplied by the inverse of the square root of its eigenvalue to give data in all directions which has mean zero and is of unit variance. We may think of this as moving the data along each axis till it is centred over the origin and then compressing or expanding each direction till it has the same spread. This removes all potential differences due to first and second order statistics from the data. To do this, the eigenvalue-eigenvector decomposition of the covariance matrix[2] is performed. i.e. for input data X, we find the covariance matrix

$$\Sigma = \langle (X - \langle X \rangle)(X - \langle X \rangle)^T \rangle = UDU^T \tag{149}$$

where U is the eigenvector matrix and D is the diagonal matrix of eigenvalues and the T denotes the transpose of the matrix. New samples drawn from the distribution

[2]In practise, we make no distinction between statistics generated by samples from the distribution and those of the distribution itself

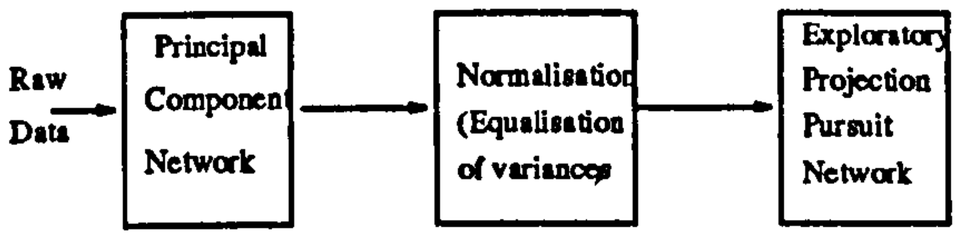


Figure 19: A block diagram of the Exploratory Projection Pursuit operation

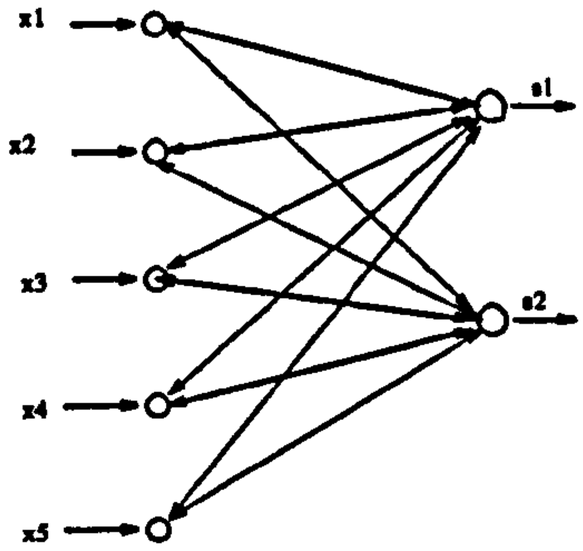


Figure 20: The negative feedback network.Activation transfer is as before except that now a non-linear function of the interneuron activations is calculated and the weights are adjusted through simple Hebbian learning

are then transformed to the principal component axes to give variables y where

$$y_i = \frac{1}{\sqrt{D_i}} \sum_{j=1}^{n} U_{ij}(X_i - \langle X_i \rangle), \text{for } 1 \leq i \leq m \tag{150}$$

where n is the dimensionality of the input data and m ($\leq n$) is the dimensionality of the sphered data. Typically $m << n$ and so this operation makes high-dimensional data more manageable. It is important to note that any linear combinations of the y-values also retains these properties of the mean and variance e.g. see (Mardia et al., 1979), Corollary 3.2.1.3. This is the data in which we wish to find interesting directions.

A block diagram of the operation is shown in Figure 19.

6.4 The Projection Pursuit Network

Figure 20 shows the network which we will use to perform Exploratory Projection Pursuit: as in previous Chapters, (the sphered) data is fedforward from the input

neurons (the x-values) to the output neurons (the interneurons). Here a linear summation is calculated to give the activation of the output neurons and this activation is fed back to and subtracted from the input neurons. A function of the output activations is calculated and used in the simple Hebbian learning procedure. We have for N dimensional input data and M output neurons

$$s_i = \sum_{j=1}^{N} w_{ij} x_j(t) \tag{151}$$

$$x_j(t+1) \leftarrow x_j(t) - \sum_{k=1}^{M} w_{kj} s_k \tag{152}$$

$$r_i = f(\sum_{j=1}^{N} w_{ij} x_j(t)) = f(s_i) \tag{153}$$

$$\Delta w_{ji} = \eta_t r_i x_j(t+1) \tag{154}$$

$$= \eta_t f(\sum_{k=1}^{N} w_{ik} x_k(t)) \{x_j(t) - \sum_{l=1}^{M} w_{lj} \sum_{p=1}^{N} w_{lp} x_p(t)\} \tag{155}$$

where r_i is the value of the function f() on the i^{th} output neuron and $x_j(t)$ is the j^{th} input at time t. Thus (155) may be written in matrix form as

$$\Delta \mathbf{W}(t) = \eta(t)[\mathbf{I} - \mathbf{W}(t)\mathbf{W}^T(t)]\mathbf{x}(t) f(\mathbf{x}^T(t)\mathbf{W}(t)) \tag{156}$$

where t is an index of time and I is the identity matrix.

The set of network rules described above is a generalisation of those for the interneuron network which performs PCA. For this reason we feel able to link the Exploratory Projection Pursuit network to current work on Non-linear PCA.

6.4.1 Non-linear PCA

Recently, the topic of "non-linear PCA" has been receiving a great deal of attention from the neural net community e.g. (Shapiro and Prugel-Bennett, 1992; DeMers and Cottrell, 1993; Oja et al., 1991; Karhunen and Joutsensalo, 1993b; Karhunen and Joutsensalo, 1993a; Oja and Karhunen, 1993; Karhunen and Joutsensalo, 1994; Karhunen and Joutsensalo, 1992). The impetus for such a development is the recognition that neural networks are ideally suited to non-linear adaption because of their

incremental methods of learning: while closed form solutions may exist for linear processes such as PCA, such methods are simply not possible for non-linear algorithms.

Following (Karhunen and Joutsensalo, 1994), we can derive (156) as an approximation to the maximisation of a function, J, of the weights $J(\mathbf{W}) = \sum_{i=1}^{M} \langle g[\mathbf{x}^T \mathbf{w}_i] | \mathbf{w}_i \rangle$.

We must ensure that the optimal solution is kept bounded; otherwise there is nothing to stop the weights from growing without bound. Formally,

$$\text{Let } J(\mathbf{W}) = \sum_{i=1}^{M} \langle g[\mathbf{x}^T \mathbf{w}_i] | \mathbf{w}_i \rangle + \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \lambda_{ij} [\mathbf{w}_i^T \mathbf{w}_j - a_{ij}] \tag{157}$$

where the last term enforces the constraints $\mathbf{w}_i^T \mathbf{w}_j = a_{ij}$ using the Lagrange multipliers λ_{ij}. As usual, we differentiate this equation with respect to the weights and with respect to the Lagrange multipliers. This yields respectively, at a stationary point,

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \langle \mathbf{x} g'(\mathbf{x}^T \mathbf{W}) | \mathbf{W} \rangle + \mathbf{W} \mathbf{\Lambda} = 0 \text{ and} \tag{158}$$

$$\mathbf{W}^T \mathbf{W} = \mathbf{A} \tag{159}$$

where $g'(\mathbf{x}^T \mathbf{W})$ is the elementwise derivative of $g(\mathbf{x}^T \mathbf{W})$ with respect to $\mathbf{W}$, A is the matrix of parameters a_{ij} (often the identity matrix) and Λ is the matrix of Lagrange multipliers. Equations (158) and (159) define the optimal points of the process. Pre-multiplying (158) by $\mathbf{W}^T$ and inserting (159), we get

$$\mathbf{\Lambda} = -\mathbf{A}^{-1} \mathbf{W}^T \langle \mathbf{x} g'(\mathbf{x}^T \mathbf{W}) | \mathbf{W} \rangle$$

and using this value and reinserting this optimal value of Λ into (158) yields the equation,

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = [\mathbf{I} - \mathbf{W} \mathbf{A}^{-1} \mathbf{W}^T] \langle \mathbf{x} g'(\mathbf{x}^T \mathbf{W}) | \mathbf{W} \rangle \tag{160}$$

We wish to use an instantaneous version of this in the gradient ascent algorithm

$$\Delta \mathbf{W} \propto \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

to yield

$$\Delta \mathbf{W} = \mu [\mathbf{I} - \mathbf{W} \mathbf{A}^{-1} \mathbf{W}^T] \mathbf{x} g'(\mathbf{x}^T \mathbf{W}) \tag{161}$$

We will be interested in the special case where the W values form an orthonormal basis of the data space and so $\mathbf{A} = \mathbf{I}$, the identity matrix. Therefore, we can equate (161) with (156).

Karhunen and Joutsensalo point out that the algorithm is approximative since the expression for $\mathbf{\Lambda}$ is derived from the optimum solution and used from the beginning of the algorithm. As we shall see in Section 6.4.4, the implications of the approximation are profound: to be used in a gradient ascent algorithm, $\frac{\partial J}{\partial W}$ must be continuous and with positive slope in the iteration intervals. We shall see that these constraints can only be justified, in general, on the set of points where the second constraint in (157) is satisfied *a priori*.

6.4.2 The Projection Pursuit Indices

Now for projection pursuit, we wish to maximise a specific index. But note that from the derivation in the last section, when we wish to maximise an index function we must use its derivative in the learning algorithm: the function $f()$ in (155) is equivalent to the function $g'()$ in (161). Thus to maximise a projection pursuit index e.g. for skewness, we could use a learning process like that described in (161) noting that to maximise the skewness index we must use the derivative of the index in the learning process.

Firstly, notice that from (151), we have $\mathbf{s} = \mathbf{x}^T\mathbf{W}$; hence, $g(\mathbf{s}) = g(\mathbf{x}^T\mathbf{W})$ and $\frac{dg}{d\mathbf{w}} = \mathbf{x}^T\frac{dg}{d\mathbf{s}}$. But since changes to the parameters of the system are made during a single presentation of $\mathbf{x}$, we may take $\frac{dg}{d\mathbf{s}} \propto \frac{dg}{d\mathbf{w}}$ during that particular presentation.

We wish to emphasise the properties of the negative feedback network rather than those of specific indices. Thus we choose to report on the network's development in relation to the simplest possible indices. The indices which we investigate in this section are either directly based on the higher moments of the input data or are functions of them (see Figure 18):

- To measure skewness in a Normal distribution, $N(\mu, \sigma)$ we use

$$g(s) = \frac{\langle (s-\mu)^3 \rangle}{\sigma^3}$$

where s is a random variable drawn from the distribution with mean μ and standard deviation σ. Now our data-distributions have all been sphered i.e. $\langle x \rangle = 0; \langle (x - \langle x \rangle)^2 \rangle = 1$ and our weights, w_i are normalised and therefore every direction s_i has the same first and second moments. Thus $g(s) = s^3$ is a measure of the skewness of the distribution. Thus in the algorithm, (156) we use

$$f(s) = k * s^2 \propto \frac{d}{ds} s^3$$

Now in all Normal directions, this measure will be zero but in a direction with a skewed distribution, there will be a non-zero skew value.

- Similarly, kurtosis[3] is measured by

$$g(s) = \frac{\langle (s - \mu)^4 \rangle}{\sigma^4}$$

Therefore as above, to measure a kurtotic deviation, we could use

$$f(s) = k * s^3 \propto \frac{d}{ds} s^4$$

- We can also use functions (see Section 6.5.4) whose expansions are dominated by either odd or even powers of s to measure kurtosis or skewness respectively.

The first 2 are the simplest possible measures of departure from Normality yet are generally not used because of their susceptibility to outliers. For this reason, we have experimented with the third set of measures. We will use the naive sample-based versions of the measures making no adjustments for any potential differences between sample and distribution moments (see e.g.(Mardia et al., 1979) for a discussion of such differences). We further treat each test as measuring only one facet although we are aware that tests for skewness and kurtosis are distributionally dependent (see e.g. the discussion in (Horswell and Looney, 1992)).

[3]Typically, 3 is subtracted from this measure in order to make the kurtosis of a truly Normal distribution 0. However, in the experiments reported herein, we have simply used the stated measure

Traditional statistical methods require a computationally-intensive recalculation of the distribution's moments from a reasonable sample of data points from the distribution each time a measure must be recalculated. However, it will be shown that a Hebbian learning rule for neural networks based on a measure of the instantaneous moments does in fact find that direction of maximum interest in the sense of Section 6.2.1.

6.4.3 Principal Component Analysis

The negative feedback network introduced here is identical to that used previously as a Principal Component network. The transfer of activation is exactly the same as described in this Chapter however there was previously no non-linear activation function at the output neurons. This is equivalent to a network with $f(x) = x$, the identity transformation. Now since f is the derivative of the function we wish to maximise, we can see that the PCA network is maximising the second moment of the distribution i.e. as we know, PCA is finding that direction with greatest variance. In fact, in the simulations described below, we use the above network twice - the first time to project the data onto the eigenvectors corresponding to the Principal Components and the second time to carry out Exploratory Projection Pursuit.

The fact that the same network structure is capable of performing a PCA as well as EPP is unsurprising since Huber (Huber, 1985) has shown that PCA may be viewed as a particular case of Projection Puruit. Thus, for the PCA network, we are choosing $f(x) = x \propto \frac{d}{dx}(x^2)$ and so the original network is seen to be maximising the second order statistics of the distribution i.e. finding the eigenvectors corresponding to maximal eigenvalues.

This suggests that Oja's Subspace Algorithm can be derived in terms of a gradient ascent procedure. However, recall from Chapter 3 that Baldi and Hornik (Baldi and Hornik, 1988) have shown that this algorithm is not derivable from such a procedure. The reason for this apparent contradiction is found in the approximation assumptions used in the derivation of the algorithm and will be discussed in the next section.

6.4.4 Convergence of the Algorithm

The derivation of the algorithm was based on gradient ascent using (157). Therefore, this equation must define a function of W which is twice differentiable with respect to W and whose second derivative is monotonic.

Consider first the convergence of the algorithm on the set of points restricted to the surface $\|\mathbf{w}\| = c$, where $\|.\|$ denotes the Euclidean norm and c is a constant. On this set, the second term of the equation, $\frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\lambda_{ij}[\mathbf{w}_i^T\mathbf{w}_j - \delta_{ij}]$, is a constant and we may then return to the original maximisation, which we will denote $J'(\mathbf{W}) = \sum_{i=1}^{M}\langle(g[\mathbf{x}^T\mathbf{w}_i]|\mathbf{w}_i)\rangle$ on this set. Note first that each of the functions used in this Chapter is twice differentiable.

Used as an instantaneous algorithm, we have, for the presentation of a single pattern x

- For kurtosis, $J'(\mathbf{W}) = \sum_{i=1}^{M}(\mathbf{x}^T\mathbf{w}_i)^4$. Then $\frac{\partial^2 J'}{\partial \mathbf{w}_{ij}^2} = 12(\mathbf{w_i}.\mathbf{x})^2 x_j^2 \geq 0$. Thus the function $f(s) = ks^3, k > 0$ will converge to that direction with maximum kurtosis when the convergence takes place on the set of all points which satisfy $\|\mathbf{w}\| = c$. Similarly the function, $f(s) = ks^3, k < 0$ will always cause convergence to those directions with minimum kurtosis. Therefore to test kurtosis in a situation where the form of the data is unknown, we can (in parallel) test for both positive and negative kurtotic distributions e.g. with $f(s) = k_1 s^3, k_1 > 0$ and $f(s) = k_2 s^3, k_2 < 0$.

- For skewness, $J'(\mathbf{W}) = \sum_{i=1}^{M}(\mathbf{x}^T\mathbf{w}_i)^3$. Now $\frac{\partial^2 J'}{\partial \mathbf{w}_{ij}^2} = 6(\mathbf{w_i}.\mathbf{x})x_j^2$. Thus if $\Delta\mathbf{W} \propto \frac{\delta J'(W)}{\delta W}$, we have a gradient ascent rule if $\langle(\mathbf{w}_i.\mathbf{x})\rangle$ is greater than 0 i.e. we will ascend till we converge to the direction with greatest positive skewness. If $\langle(\mathbf{w}_i.\mathbf{x})\rangle$ is less than 0, we will descend till we converge to the direction with most negative skewness. Thus one function can be used to test for both positive and negative skewness. It is important to recall that this is an exploratory data investigation tool: we do not care if the structure has positive or negative skewness - only that it is deviating from a Gaussian distribution.

However, this does leave open the possibility that there exists a stage in the

convergence when skewness in two directions reaches a stable point of convergence which is a mixture of two optimal states, though we have never seen this situation experimentally.

Therefore the algorithm may be viewed as gradient ascent on the hypersphere satisfying $\|\mathbf{w}\| = c$. Now we must consider the convergence of the algorithm in general; consider (158) with respect to a particular vector of weights into output neuron, i, for a function $g(s) = s^k$. Then we have

$$\frac{\partial J}{\partial \mathbf{w_i}} = k\mathbf{x}(\mathbf{w_i}.\mathbf{x})^{k-1} + \mathbf{W}\lambda_\mathbf{i} \tag{162}$$

$$\frac{\partial^2 J}{\partial \mathbf{w_i^2}} = k(k-1)Diag\{\mathbf{x}\mathbf{x}^T\}(\mathbf{w_i}.\mathbf{x})^{k-2} + \lambda_{ii}I \tag{163}$$

where $Diag\{.\}$ is an operator which sets all off-diagonal entries to 0 and $\lambda_\mathbf{i}$ is the vector of Lagrange coefficients for the direction $\mathbf{w_i}$. Now since the data is sphered, $< (x_j^2) >= 1$. Thus we only have a positive gradient in those directions, $\mathbf{w_i}$, which satisfy

$$(\mathbf{w_i}.\mathbf{x})^{k-2} > \frac{-\lambda_{ii}}{k(k-1)} \tag{164}$$

Recalling that λ_{ij} determines the relative weight accorded to the function J' and the constraint $[\mathbf{w_i^T w_j} = \delta_{ij}]$ and we can see that the use of the final converged value of λ_{ij} in the converging algorithm causes a more serious problem than merely being an approximation. The algorithm is not guaranteed to converge.

In practice, this has not been found to be a problem. One possible heuristic would be to start the weights normalised and then converge across the surface. However there is the possibility that the convergence process will be slower using this. Empirically, little difference has been found between starting with small (near 0) random weights and starting with normalised vectors.

6.5 Simulations and Results

The neural network shown in Figure 20 was set up with 10 inputs, 10 interneurons and (initially) 1 output neuron. Input data was drawn from 9 independent zero-mean Gaussians while in the tenth direction data was also drawn from a Gaussian

distribution but was modified in some way to make it interesting.

The modifications were designed to be simple yet ensure that there existed differences in the high order statistics between this interesting direction and all other directions. Therefore

1. To create a skewed distribution, we multiplied e.g. all values less than the mean of the distribution in this direction by a positive constant, generally in the range 0.8 to 1.2.

2. To create a distribution showing positive kurtosis (leptokurtic data) which appears as a value greater than the Normal distribution's value of 3 in our tables, we randomly selected samples (typically 20% of the total available) and substituted small random numbers in the range -0.25 to 0.25. This gives a narrowly peaked distribution.

3. To create a distribution showing negative kurtosis (platykurtic data) which is quantified as values under 3 in our tables, we randomly divided all the samples of the distribution into two disjoint sets and added a constant value to all samples in one set and subtracted the same value from all samples in the other set. This "twin-peaked" distribution has a negative kurtosis since it has distinct shoulders.

6.5.1 One Interesting Direction

As an example of using the generalised interneuron network, we show the results of a simulation in Figure 22. The Gaussian distribution in one direction was amended in order to create a distribution which was sharply peaked (see Figure 21) i.e. had a positive kurtosis. This distribution is generally one of the most difficult to find as it is only visible in one very tightly defined direction. The results of using the Kurtosis Index, $f(s) = s^3$, are shown in Figure 22.

Statistics from the data to which this network is responding is shown in Table 26. Note that since the data is first sphered, it is not possible to base differential learning on first or second order statistics which are respectively 0 and 1 in each

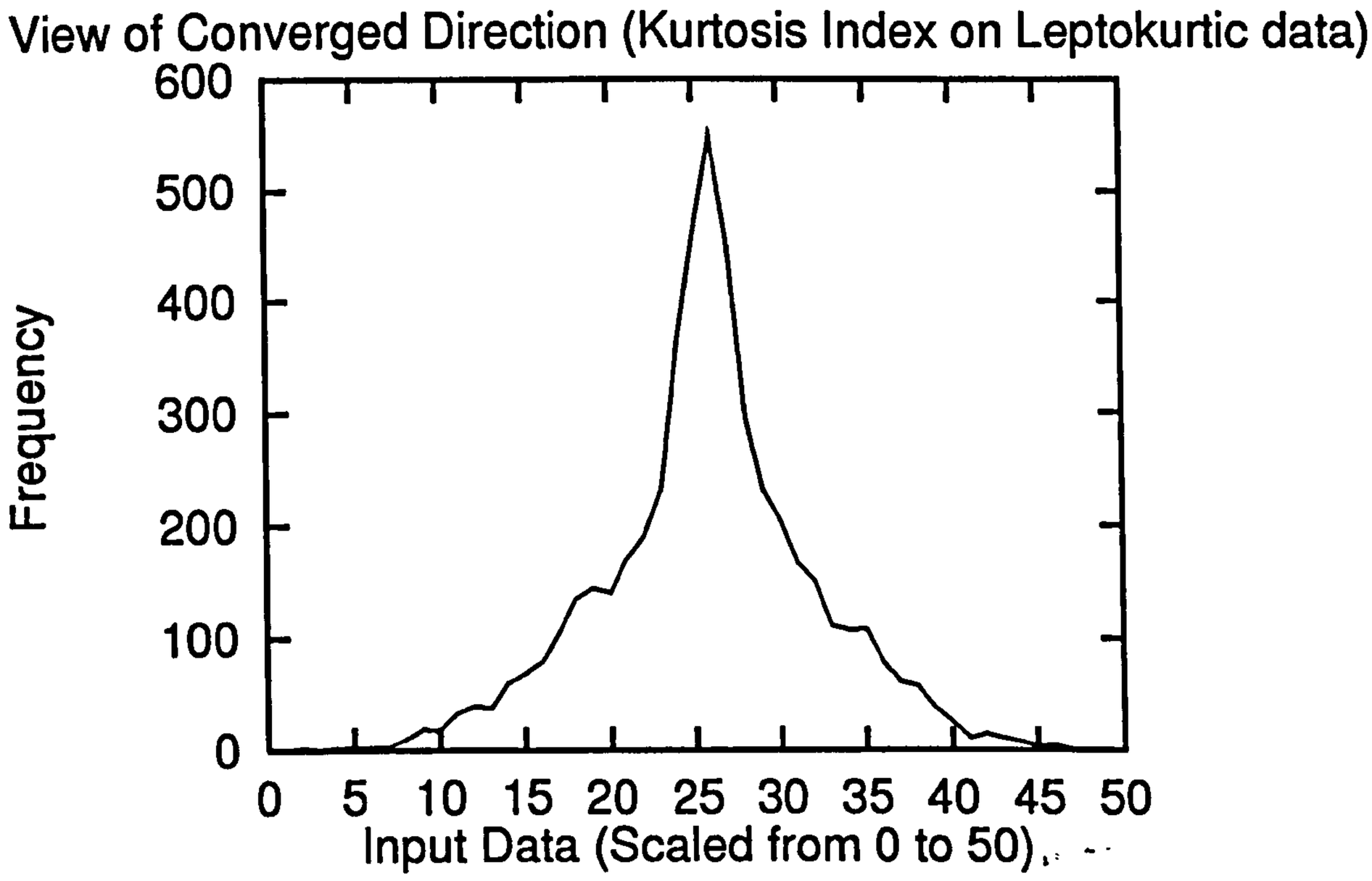


Figure 21: The input data were projected onto the direction in which the net converged. The distribution is scaled from 0 to 50. Thus the view here shows the distribution as it would be seen by an observer looking at the projection of the data in the "interesting" direction

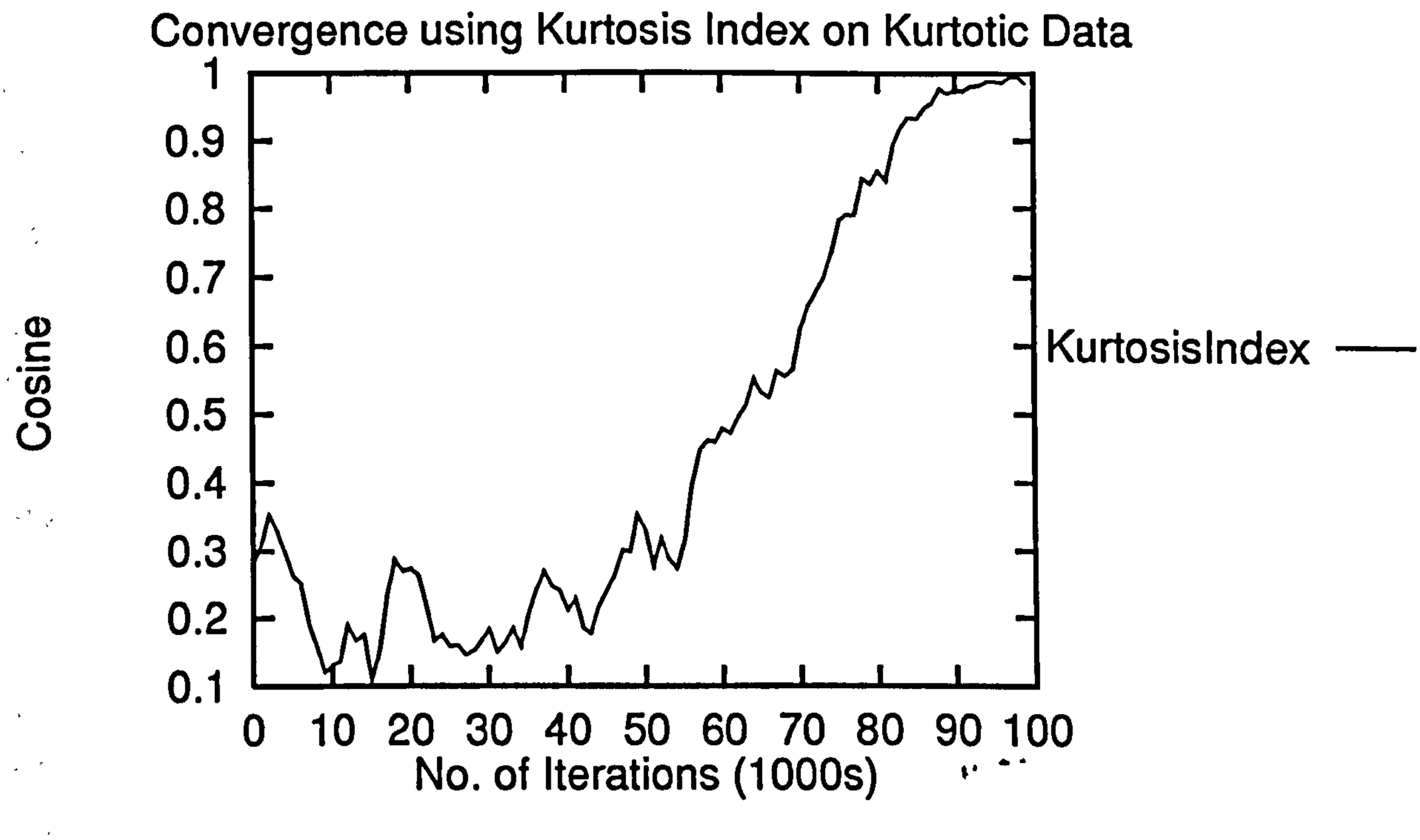


Figure 22: The convergence trajectory of the weights to the "interesting" (highly-peaked) direction. The vertical axis shows the cosine of the angle between the current direction and the optimal direction.

Direction	Mean	Variance	Cube	Fourth Power
1	0.000	1.015	0.055	3.082
2	-0.019	0.968	-0.045	2.761
3	0.007	0.987	0.015	2.890
4	0.004	0.981	0.024	2.893
5	-0.006	0.990	-0.022	2.949
6	-0.006	0.998	-0.017	3.016
7	-0.017	1.017	-0.037	3.103
8	0.011	0.992	0.058	2.889
9	-0.013	1.023	-0.087	**4.001**
10	-0.010	0.988	-0.000	2.960

Table 26: Statistics of the first set of 10-dimensional data: only one direction(9) has been modified to create a positively kurtotic distribution. Note that the sphering has produced a zero mean, variance 1 distribution in every direction

direction. It is our finding that the sphering must be reasonably accurate otherwise the learning process responds to the low order statistics - a finding consistent with the statistics literature though not mentioned in neural network implementations of projection pursuit.

6.5.2 More Than One Interesting Direction

Since the projection pursuit method is designed to find interesting directions worthy of human investigation and since humans can visually investigate functions over a plane, we are often interested in finding 2 independent directions in a data set which contain interest. We first consider the situation when each interesting direction has the same type of interest. Experiments have shown that, in such situations, the network usually finds one direction more interesting than the others; the weights will converge to that direction on which the projection of the data has the largest deviation from the statistics of a normal distribution, e.g. when the kurtosis index is 3.67 in direction 4 and 3.66 in direction 7, the kurtosis index function invariably causes convergence to direction 4.

However, it may be appropriate to find all interesting directions. This situation

Direction	Mean	Variance	Cube	Fourth Power
1	-0.000	1.011	-0.007	3.025
2	0.009	0.995	0.005	2.965
3	0.012	1.012	0.024	3.126
4	0.018	0.978	0.079	2.874
5	0.007	1.014	0.035	**3.813**
6	-0.015	0.963	-0.054	2.715
7	0.002	0.974	0.059	2.751
8	-0.017	0.990	0.012	2.830
9	0.017	0.961	**-0.263**	2.965
10	0.009	1.003	0.047	2.982

Table 27: Statistics of the distribution showing that direction 5 is interesting because of its 4^{th} moment while direction 9 is interesting because of its 3^{rd} moment

sometimes (Friedman, 1987) is dealt with by "structure removal" using a transformation of the intersting direction to create a Normal distribution in that direction. This method has the disadvantage that such transformations may affect the Normality of other solution projections. However we note that the learning process used here not only finds but removes the interesting directions i.e. the residuals at $\mathbf{x}$ consist of the original data minus the projections onto the learned interesting directions. Thus we suggest running the network till one interesting direction is found; then set these weights and restart learning with a new output neuron. This has been found to be very effective.

6.5.3 Differing Types of Interesting Directions

When the data contains directions which are interesting in different ways, we can investigate the data in different ways simultaneously. We can for example construct a network as before but with M output neurons each of which is searching for different characteristics in the sphered input data.

For example, we repeat the above experiment but now we change the data so that we have one direction which is positively kurtotic and one direction which is negatively skewed. The statistics of this data are shown in Table 27 and the convergence of the appropriate directions are shown in Figure 23

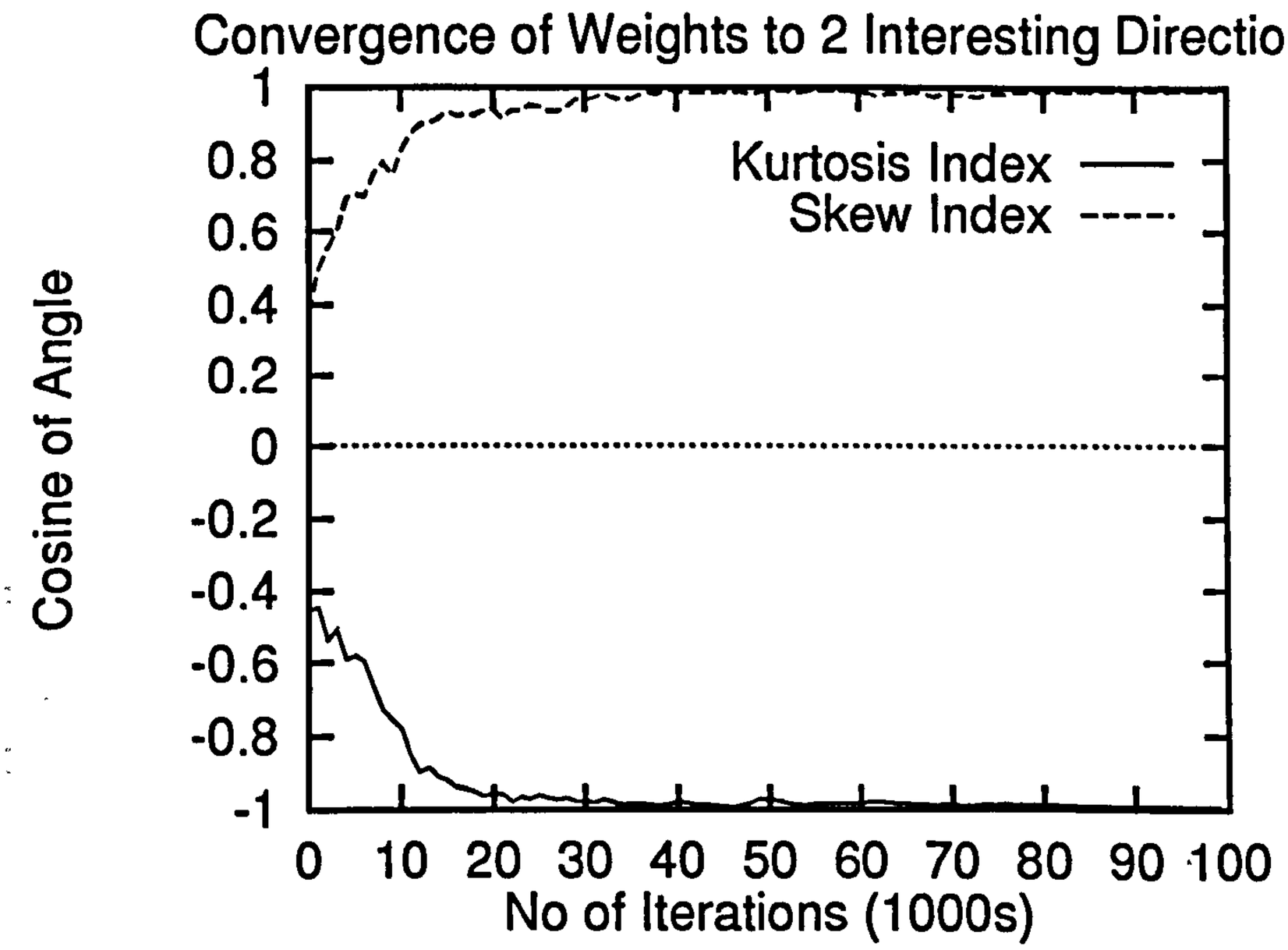


Figure 23: Convergence of the weights into the neurons using the indices for Positive Kurtosis and Negative Skewness to the appropriate direction

These results are from a network created with 4 output neurons all of which converged to one or other of the interesting directions. So our final algorithm is:

1. Sphere the input data
2. Create a set of output neurons with M different indices and train this network. For example, for the indices we have so far considered we may simply have 2 output neurons which use skew and kurtosis indices respectively.
3. Visually examine (either individually, as lines, or in pairs, as planes) the directions found in order to identify humanly interesting directions
4. Remove those neurons whose weights have not converged to interesting directions
5. Repeat Sections 3-5 training the new neurons on the residuals after all current neurons have removed their projections.

6.5.4 Using Hyperbolic Functions

As an example of using hyperbolic functions we perform an experiment similar to the last one but report the convergence of

- $f(s) = tanh(s)$. Since $f(s) = tanh(s)$ has an expansion of $s - \frac{s^3}{3} + \frac{2s^5}{15} - \ldots$, it is an odd function. It can then be used to measure the kurtotic deviation from the normal distribution. In detail, using tanh(s) as the learning function, f, in (156) maximises the integral of that function; thus, using $f(s) = tanh(s)$ in the stochastic algorithm maximises

$$\langle(\int tanh(s)ds)\rangle = \langle(\int(s - \frac{s^3}{3} + \frac{2s^5}{15} - \ldots)ds)\rangle \quad (165)$$

$$= \langle(\frac{s^2}{2} - \frac{s^4}{12} + \frac{2s^6}{90} - \ldots)\rangle \quad (166)$$

$$= \langle(\frac{s^2}{2})\rangle - \langle(\frac{s^4}{12})\rangle + \langle(\frac{2s^6}{90})\rangle - \ldots \quad (167)$$

$$= K - \langle(\frac{s^4}{12})\rangle + \langle(\frac{2s^6}{90})\rangle - \ldots \quad (168)$$

since s is a linear combination of sphered data.

Now for small s, the most important parts of the series is the first few terms and that subsequent terms alternate between reinforcing the effect of the second term (being negative) and detracting from its effect. $\frac{s^4}{12} > \frac{2s^6}{90}$ in the interval (-1.93, 1.93) which contains almost exactly 95% of the sphered data in Normal directions. This proportion will be slightly different for a non-normal direction but the major conclusion must be that for the overwhelming majority of the data points the driving force of the learning is the cubed term in the expansion of tanh(s). Therefore if we use $tanh$(s) in our algorithm, we are minimising s^4 i.e. finding directions with least kurtosis.

We used this index to force the weights to converge to direction 4 whose statistics are shown in Table 28. A view of the projection of a sample of the distribution onto the direction found using this index is shown in Figure 25.

- Similarly we can use $f(s) = sech^2(s)$ which is the derivative of tanh(s) and is an even function, to find deviations in skewness from the normal distribution. The weights using this index converge to direction 7 in Table 28. A view of the projection of a sample of the distribution onto the direction found using this index is shown in Figure 24.

We emphasise that these 2 directions were found in parallel. In fact, not only does the convergence of the two indices not interfere with each other, the convergence of each may actually help the other if we use as input data for the second output neuron the residuals at $\mathbf{x}$ after the first neuron has subtracted its projection. This has the effect of decreasing the dimension of the space of input data in which the second neuron must search for interesting directions. Note also that these are by no means the only functions which can be used. For example, $tan^{-1}(s)$ can also be used for searching for kurtotic distributions as its expansion is also odd.

Direction	Mean	Variance	Cube	Fourth Power
1	0.003	0.997	0.042	2.999
2	-0.009	1.012	-0.068	3.137
3	-0.004	0.993	0.026	2.990
4	-0.003	1.007	-0.019	**1.732**
5	0.018	0.991	0.098	3.020
6	-0.000	0.971	-0.004	2.761
7	-0.005	0.985	**0.222**	3.053
8	-0.000	0.999	-0.021	2.954
9	0.010	1.000	0.075	2.988
10	0.002	0.988	-0.005	2.902

Table 28: Statistics of the sphered data on which the hyperbolic indices were optimised

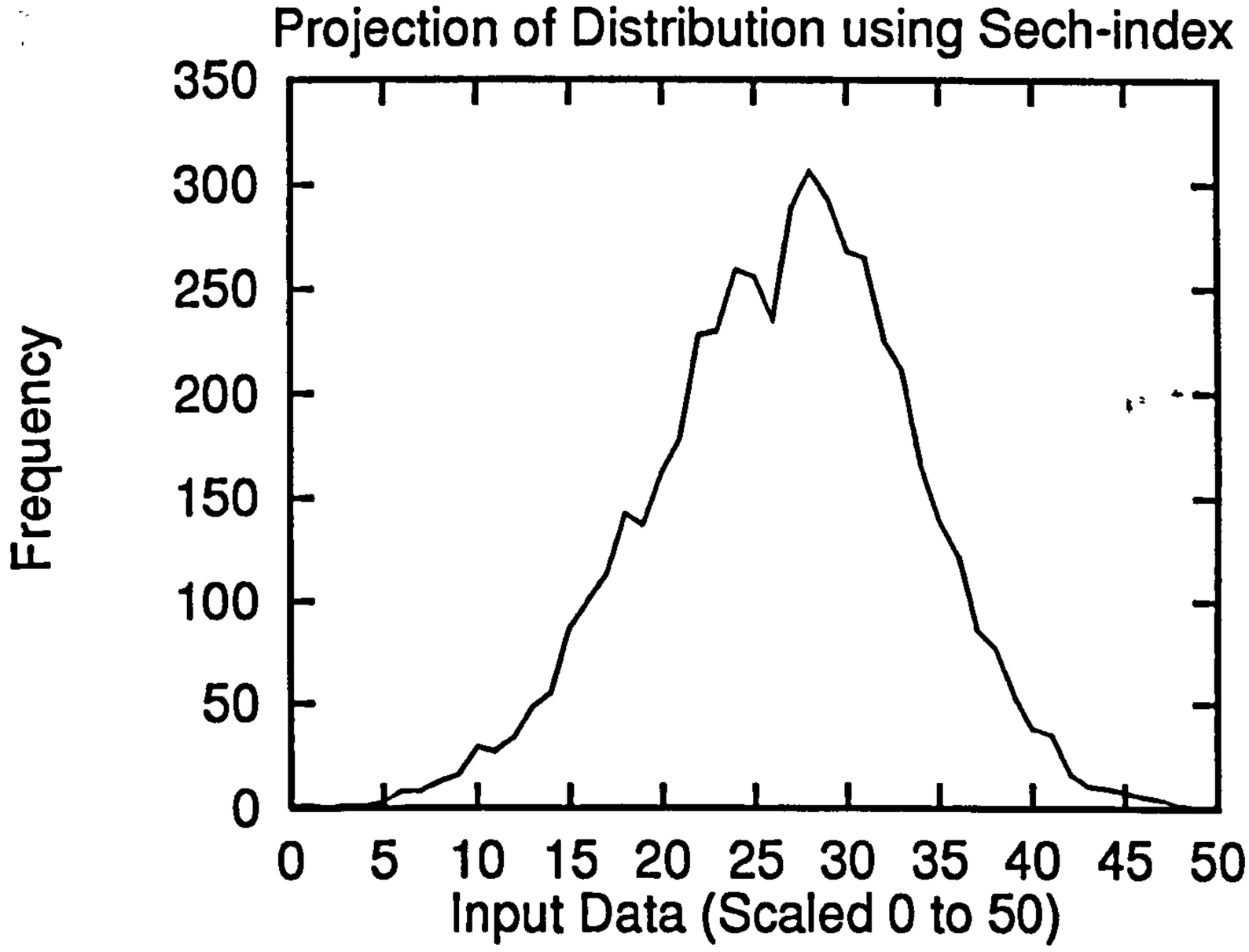


Figure 24: Projection of data onto the interesting direction found using the sech-index. The direction is skewed

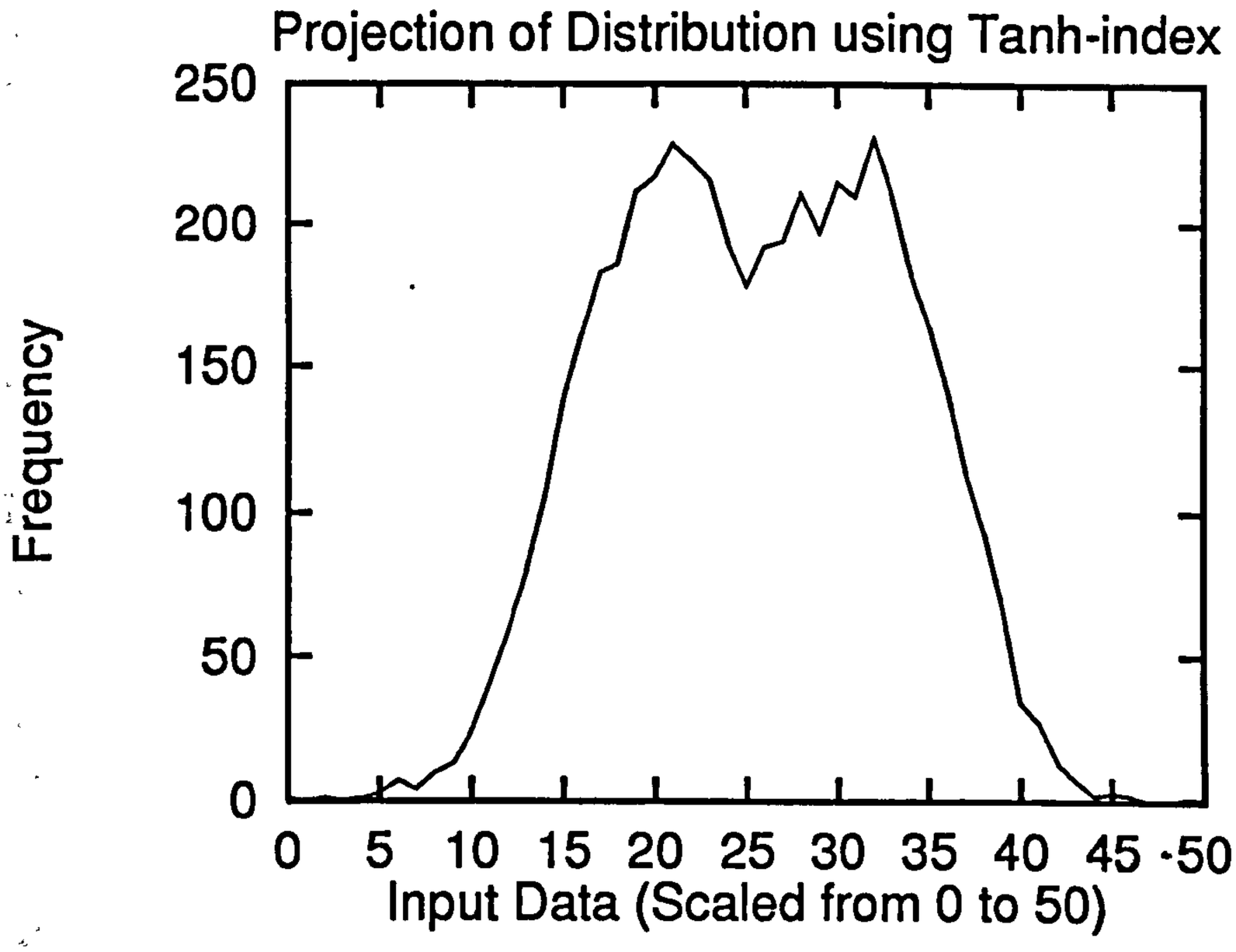


Figure 25: Projection of data onto the interesting direction found using the tanh-index. The direction is interesting because of its kurtosis

6.6 Other Indices

With the method and network for Projection Pursuit now established, we can investigate other possible indices. We will investigate indices based on Information Theory and two specific indices: Friedman's Index and Intrator's Index. These have aroused a great deal of interest in their respective communities and are thought to be substantially the best currently on offer (though we must add the caveat that such assessments are usually made with respect to specific data sets). The most frequently referenced indices from the statistics community are those from Friedman (Friedman, 1987) and Hall (Hall, 1989). Both indices use polynomial approximations to analytically-deduced indices of interestingness; such polynomials are usually introduced for reasons of computational efficiency. We chose to investigate Friedman's rather than Hall's as the former index is thought to be more generally effective than the latter (e.g. see (Sun, 1993) for a recent comparison).

In the neural network community, the series of articles written by Intrator(e.g. (Intrator, 1992; Intrator, 1993b; Intrator and Cooper, 1992)) stand virtually unchallenged as implementations of the Projection Pursuit methodologies. Other articles (e.g. (Hinton and Nowlan, 1990)) do not specifically mention Projection Pursuit though often appearing to use a PP methodology. An interesting implementation of PP methodologies using radial basis function nets is given in (Zhao, 1992).

6.6.1 Indices based on Information Theory

As noted by Marriot(in Discussion of (Jones and Sibson, 1987)), " a moment criterion, or any criterion dominated by third and fourth cumulants, will miss clustered projections that happen to be roughly symmetrical and nearly mesokurtic"; this has lead to a search of alternative measures of non-normality.

Entropy measures suggest themselves as a means of measuring the divergence of the distribution from Normality; however using such measures requires having the neuron retain a memory of previous inputs in order to calculate a relative frequency approximation to the entropy.

Nevertheless, it does suggest using in (155) the measure $f(s) = -\log \phi(s)$, where

$\phi(s)$ is the probability that it would have received input s had s come from a Normal distribution.This is equivalent to maximising

$$\begin{aligned}\langle -\int log\phi(s)ds\rangle &= \langle -\int \log\{\frac{1}{\sigma\sqrt{2\pi}}\exp\frac{-(s-\mu)^2}{2\sigma^2}\}ds\rangle \\ &= \langle -\int \log\frac{1}{\sigma\sqrt{2\pi}}ds + \int\frac{(s-\mu)^2}{2\sigma^2}ds\rangle \\ &= -\int \log\frac{1}{\sqrt{2\pi}}ds + \int\frac{s^2}{2}ds \\ &= s(k_2s^2 - k_1) \text{ where } k_1, k_2 \text{are constants} \end{aligned} \quad (169)$$

since the data has been sphered. It is clear that this index is maximising a function of two components with the second acting as a non-central pivot about which the other is maximised. Since the dominant term is s^3 the function finds skew directions; however the extra pivoting power around k_1 permits convergence to twin-peaked (negatively kurtotic) directions also.

It is of interest also, in this case to investigate the properties of $\frac{\partial J}{\partial \mathbf{w}}$ which are information theory based.

Thus when we use the measure $f(s) = -\log\phi(s)$, the expected value of the function used in the learning rule is

$$\begin{aligned} D &= \frac{\partial J}{\partial \mathrm{w}} \\ &= \frac{1}{N}\sum_{n=1}^{N} f(s_n) \\ &\approx \sum_{m=1}^{M} p(s_m)f(s_m) \\ &= -\sum_{m=1}^{M} p(s_m)\log\phi(s_m) \end{aligned}$$

where the data has been binned into M boxes and the mean value of s in the m^{th} box is s_m. $p(s_m)$ is the relative frequency of the sample in box m and is taken as an estimate of the probability of having an input s in box m. Now

$$D = -\sum_{m=1}^{M} p(s_m)\log\phi(s_m)$$

$$\begin{aligned} &= \sum_{m=1}^{M} p(s_m) \log \frac{p(s_m)}{\phi(s_m)} - \sum_{m=1}^{M} p(s_m) \log p(s_m) \\ &= D(p(s)||\phi(s)) + H_p \end{aligned} \tag{170}$$

where $D(a||b)$ is the relative entropy between the distributions a and b and H_a is the Shannon Entropy of the distibution a. Note that since we are using a discrete (binned) version of $p(x)$, we may assume that $H_p \geq 0$.

This since relative entropy is convex and entropy is concave we may expect local minima as an inevitable fact of life.

6.6.2 Friedman's Index

Friedman (Friedman, 1987) has developed an index which has attracted wide interest within the community of statistics users. Sun (Sun, 1993) has performed a useful and informative comparison of Friedman's and Hall's indices and has concluded that Friedman's is generally better. Hall (Hall, 1989) agrees. Briefly, Friedman's index is based upon the transformation of the projection of the sphered data, $\mathbf{x}^T\mathbf{w} \rightarrow R = 2\Phi(\mathbf{x}^T\mathbf{w}) - 1$ where $\Phi(X)$ is the standard normal cumulative density function

$$\Phi(X) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{X} \exp(-\frac{t^2}{2}) dt \tag{171}$$

If X follows a standard normal distribution, then R will be uniformly distributed in the interval [-1,1]. Therefore we take as a measure of the distance of $\mathbf{x}^T\mathbf{w}$ from the normal distribution, the integral squared distance of the variable R from the uniform distribution,

$$\int_{-1}^{+1} [p(R) - \frac{1}{2}]^2 dR = \int_{-1}^{+1} p^2(R) dR - \frac{1}{2} \tag{172}$$

Friedman expands R in Legendre polynomials so that

$$\int_{-1}^{+1} p^2(R) dR - \frac{1}{2} = \int_{-1}^{+1} [\sum_{j=0}^{\infty} a_j P_j(R)] p(R) dR - \frac{1}{2} \tag{173}$$

where the Legendre polynomials are

$$\begin{aligned} P_0(R) &= 1 \\ P_1(R) &= R \\ P_j(R) &= \frac{(2j-1)RP_{j-1}(R) - (j-1)P_{j-2}(R)}{j}, \forall j > 1 \end{aligned}$$

The coefficients a_j are given by

$$a_j = \frac{2j+1}{2}\int_{-1}^{+1} P_j(R)p(R)dR = \frac{2j+1}{2}\langle[P_j(R)]\rangle \tag{174}$$

From this, we have an easily calculated projection index

$$g(\mathbf{w}) = \sum_{j=1}^{J}(2j+1)\langle[P_j(R)]\rangle^2/2 \tag{175}$$

for any direction w, which is applied as

$$\hat{g}(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{J}(2j+1)[\frac{1}{N}\sum_{i=0}^{N} P_j(2\Phi(\mathbf{x}^T\mathbf{w})-1)]^2 \tag{176}$$

to give the sample version of the index. Note that this must be maximised under the constraint that $WW^T = 1$ to ensure a finite solution.

However, we require the derivative of $g(\mathbf{w})$ for our instantaneous measure, f(s), and so we use

$$f(\mathbf{s}) \propto \sum_{j=1}^{J}(2j+1)\langle P_j(R)\rangle\langle[P_j'(R)\exp -(\mathbf{x}^T\mathbf{w})^2/2]\rangle$$

Now $P_j'(R)$ is also easily calculated via the recursion relation

$$\begin{aligned} P_1'(R) &= 1 \\ P_j'(R) &= RP_{j-1}'(R) + jP_{j-1}(R) \end{aligned}$$

Thus the instantaneous version of Friedman's Index used in our neural net implementation is

$$f(\mathbf{s}) = \sum_{j=1}^{D}(2j+1)P_jP_j'\exp(-(\mathbf{x}^T\mathbf{w})^2)$$

Simulations on data such as used to test the polynomial indices have shown that such an index finds directions with either skew or kurtotic deviations from Normality with great reliability and accuracy.

6.6.3 Intrator's Index

Intrator (Intrator, 1992; Intrator, 1993b; Intrator and Cooper, 1992; Intrator and Gold, 1993; Intrator, 1990; Intrator, 1993a) has constructed a model for Exploratory Projection Pursuit derived from the Bienenstock, Cooper, Monro (Bienenstock et al., 1982) model of cortical plasticity. This model has a learning function of

$$\frac{dw}{dt} = \mu(t)(\mathbf{x}.\mathbf{w})(\mathbf{x}.\mathbf{w} - \frac{4}{3}\theta_w)\mathbf{x} \tag{177}$$

where $\theta_w = \langle(\mathbf{x}.\mathbf{w})^2\rangle$ provides a moving threshold which yields the dynamic flexibility necessary for stability.

While no mention of sphering is made in the reported simulations using the BCM neuron, it is our finding that if Projection Pursuit is attempted using neural networks on unsphered data *with either method*, the learning rules respond to the lower moments in the input data. We have thus in the simulations on which our findings are based performed sphering as an essential data preparation step.

Though the format of the BCM learing rule is not immediately transferable into the format (156), the driving force of the Hebbian learning (as opposed to the decay term) is identical to the negative feedback network learning rules when the skew index is used. Unsurprisingly it is our finding that the BCM neuron finds positively skewed directions with the same facility as the negative feedback network. However, we do not find that this network can find negatively skewed or kurtotic distributions.

An extension of the model introduced to improve the learning rule's robustness with respect to outliers involves introducing a sigmoid function non-linearity to a network loss function, L

$$L(\mathbf{w}) = -\mu\{\frac{1}{3}\sigma^3(\mathbf{w}.\mathbf{x}) - \frac{1}{4}\langle\sigma^2(\mathbf{w}.\mathbf{x})\rangle\sigma^2(\mathbf{w}.\mathbf{x})\} \tag{178}$$

which leads to the learning rule

$$\frac{dw}{dt} = \mu(t)\sigma(\mathbf{x}.\mathbf{w})(\sigma(\mathbf{x}.\mathbf{w}) - \frac{4}{3}\theta_w)\sigma'(\mathbf{x}.\mathbf{w})\mathbf{x} \tag{179}$$

where now $\theta_w = \langle\sigma^2(\mathbf{x}.\mathbf{w})\rangle$.

We do not believe that the significance of the sigmoid has been appreciated: our experiments on the artificial data using the BCM neuron show that this extension not

only improves stability (the basic rule is far from stable) but also permits convergence to negatively kurtotic distributions. A Taylor series expansion (similar to that given for the tanh() neurons in the negative feedback network) shows that the reason for this new capability is found in the nature of the sigmoid.

6.7 Early Vision Processing

We complete this Chapter with an example of Exploratory Projection Pursuit on image data. We choose to report on this area as not only does the experiment show Exploratory Projection Pursuit working on real data but also links in with biological speculations on the nature of early sensory processing.

6.7.1 The Statistics of Natural Images

A step performed early in image processing is the whitening of images which equalises the power(variance) over every frequency in the images. Edge detection is another of the early steps: it is generally held that in detecting edges we are representing the world in a way that is a prerequisite for making sense of the high-dimensional space which corresponds to visual data.

Barlow and Tolhurst (Barlow and Tolhurst, 1992) have shown that in whitened images there is an excess of kurtosis in an image when 9 pixels are sampled in a line compared to that when 9 random pixels or 9 pixels in a square are sampled. This might suggest that edges in images create highly kurtotic distributions.

We have an extremely simple and effective method to find those directions in high-dimensional spaces which exhibit most kurtosis. Instead of whitening our images we sphered the data (as above) as this also equalises the variance in each direction.

The images consisted of a collection of pictures of natural scenes. Pictures were taken on a 35mm camera with a 50mm lens. The photographs were then digitised using a Hewlett Packard Scanjet Plus at the 75 dpi setting. The central 256 by 256 pixel region was then sampled and the grey levels normalised so that the lowest value

was 0 and the largest was 255. No attempt was made to account for non linearities in the processing. In an attempt to remove the artificial effects of composition (other than ensuring correct vertical alignment) we used the following techniques: the camera view finder was not used, the camera was pointed at random, and pictures were taken at set time intervals. This procedure produced three sets of 36 exposures. Out-of-focus pictures were removed together with pictures showing lens obstructions. Nine of these images in each group were randomly selected. This process was used to create 27 images: 9 of which are taken from the office environment, 9 from the countryside and 9 from a city centre.

The kurtosis-extracting neural network technique was applied to samples of natural images in order to estimate the filter with highest output kurtosis. Specifically the following was done:

- From random locations within all 27 images, 32x32 samples were extracted from the images. This process could generate up to 1,354,752 different samples from the collection of natural images.
- We used the Principal Component network to find the first eight principal components of the samples. Thus since our input vector is 1024 dimensional, our PCA network is performing a substantial dimensionality reduction. The cut-off after 8 principal components was chosen after an initial investigation showed that the standard deviations of the principal components fell in the ratios

$$\sigma_1 : \sigma_2 : \sigma_3 : \sigma_4 : \sigma_5 : \sigma_6 : \sigma_7 : \sigma_8 = 10 : 3.9 : 3.1 : 2.3 : 2.2 : 1.6 : 1.6 : 1.5 \quad (180)$$

  with no other PC having a standard deviation greater than 0.6 on this scale.
- The second layer of the neural network was used to perform gradient ascent on the output kurtosis. During this search, the learning rate was annealed from 0.001 to 0.
- After 100000 iterations[4], the first four moments i.e. the mean, standard deviation, skewness, and kurtosis of the output distribution of the derived filters was measured.

[4]Convergence typically took less than $\frac{1}{10}$ th of this time

- This process was repeated ten times to assess the reliability of the network and its dependence on initial conditions.

6.7.2 Results

The statistics of 10000 samples of the output of the first (principal component extracting) network after sphering are given in Table 29. As can be seen, the principal components are far from Gaussian(where the skewness and kurtosis would both be zero [5]); note also that almost all directions show positive kurtosis. This might be thought to be in line with our expectations as many images contained many lines and not only confirms the findings of Barlow and Tolhurst but also shows that it was not their method of sphering which created the kurtosis in the data. The first 8 principal components are shown diagramatically in Figure 26. Comparing Table 29 and Figure 26 we see that the kurtosis values for vertical directions are greater than those for horizontal directions.

We report on the results of 10 simulations and within each simulation, we use a decomposition method: since the first neuron finds *and subtracts* one direction, the second neuron searches in the image space left after the first has subtracted its space. Note that this orthogonality is quite different from that seen on the images. The filters are shown in graphical form in Figure 26 and the kurtosis values corresponding to each direction is given in Table 30. As can be seen, the network consistently finds directions of high kurtosis. We see again that the vertical filters show higher kurtosis than the horizontal and, in general, the filters of each type which the kurtosis-seeking network found have greater kurtosis than those principal components of the same type.

As has been stated, the algorithm was evolved as an Exploratory Projection Pursuit method and the exploratory nature of the algorithm is well illustrated in the differences in the converged directions which were found during different simulations. Clearly the method is dependent on the initial conditions of the network and the actual images seen during training.

---

[5] In this section we perform the more general step of subtracting 3 from the raw kurtosis figure so that comparison can be more easily made with other research findings

Direction	First	Second	Third	Fourth
1	0.002	1.006	0.958	0.659
2	-0.003	1.003	0.343	2.876
3	0.004	0.958	0.013	7.676
4	0.012	0.970	0.214	4.500
5	-0.013	1.026	-1.112	21.638
6	-0.012	1.026	0.215	5.547
7	0.001	0.948	0.268	6.356
8	0.005	1.028	-0.207	14.924

Table 29: The statistics of the output of the first layer of the neural network. As can be seen, the outputs are normalised in both mean and standard deviation. Note that the statistics of the images are far from Gaussian: almost every direction shows both kurtosis and skewness; note also that all directions show positive kurtosis.

Simulation	1	2	3	4	5	6	7	8	9	10
First	22.8	24.4	25.9	24.0	5.7	16.2	7.4	24.9	17.8	27.4
Second	15.0	20.5	31.7	19.0	18.4	18.3	17.9	19.1	22.6	16.8
Third	7.4	7.1	7.1	7.8	9.9	8.9	6.7	8.3	7.3	9.4

Table 30: The kurtosis for the directions found by 10 simulations using 3 kurtosis-seeking neurons each.

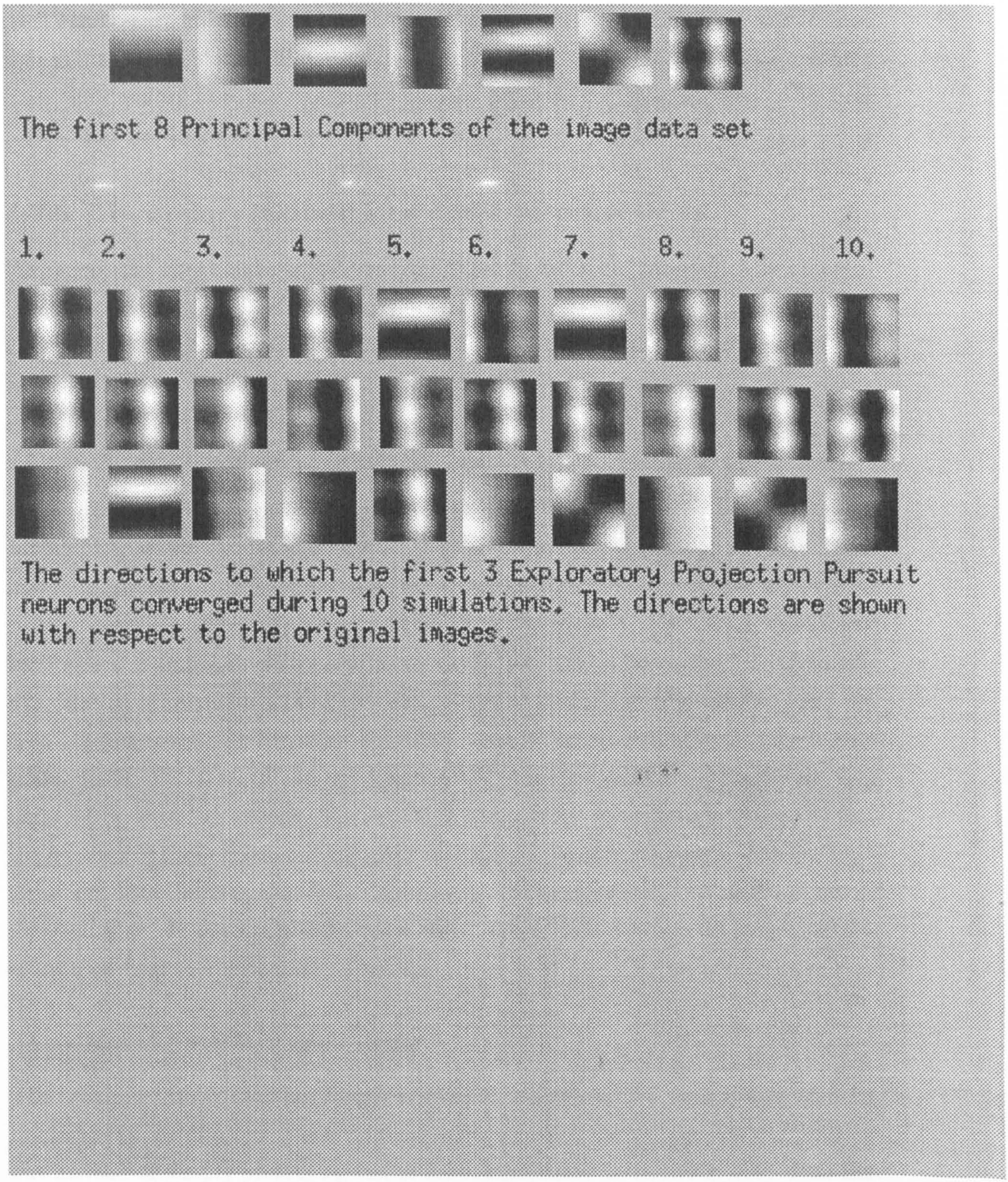


Figure 26: The directions found by the network with respect to the original images.

There are 15 directions whose orientation is vertical, 8 horizontal and 7 top-right/bottom-left blobs. It is clear from the statistics that the vertical directions have the greatest kurtosis which presumably accounts for the fact that a vertical orientation was found most often.

6.7.3 Boundary Conditions and Pre-processing

We repeat the above experiment with the same network but using slightly different input data to the PCA network: firstly we consider the effect of pre-processing the data with a logarithmic function and secondly the effect of a Gaussian mask on the data.

The effect non-linear pre-processing

Field (Field, 1994) performed his experiments upon log preprocessed images. This can be justified for two reasons. Firstly, human physiology appears to be more linear in the logarithm of contrast as opposed to simple contrast. Secondly, log preprocessing may help alleviate problems of differing illumination. By taking logarithms, local ratios in image intensity are transformed into local differences of image intensity. Ratios of intensity should be more robust to changes in illumination than are absolute differences. To investigate if logarithmic pre-processing affects the resulting components, we repeated the experiment, but this time, all image intensities were replaced by their logarithms. The resulting components are shown in Figure 27 and the kurtosis of these directions are given in Table 31. Firstly it appears that the PCA directions have been unchanged. Secondly the network is still finding directions of great kurtosis. It appears also that the vertical directions continue to have an excess of kurtosis over the horizontal directions.

The effect of boundary conditions

The previous results may have been artificially affected by the square samples from image used. To find out if this was the case, we repeated the previous experiment but instead of simply presenting the 32x32 samples the the network, they were first

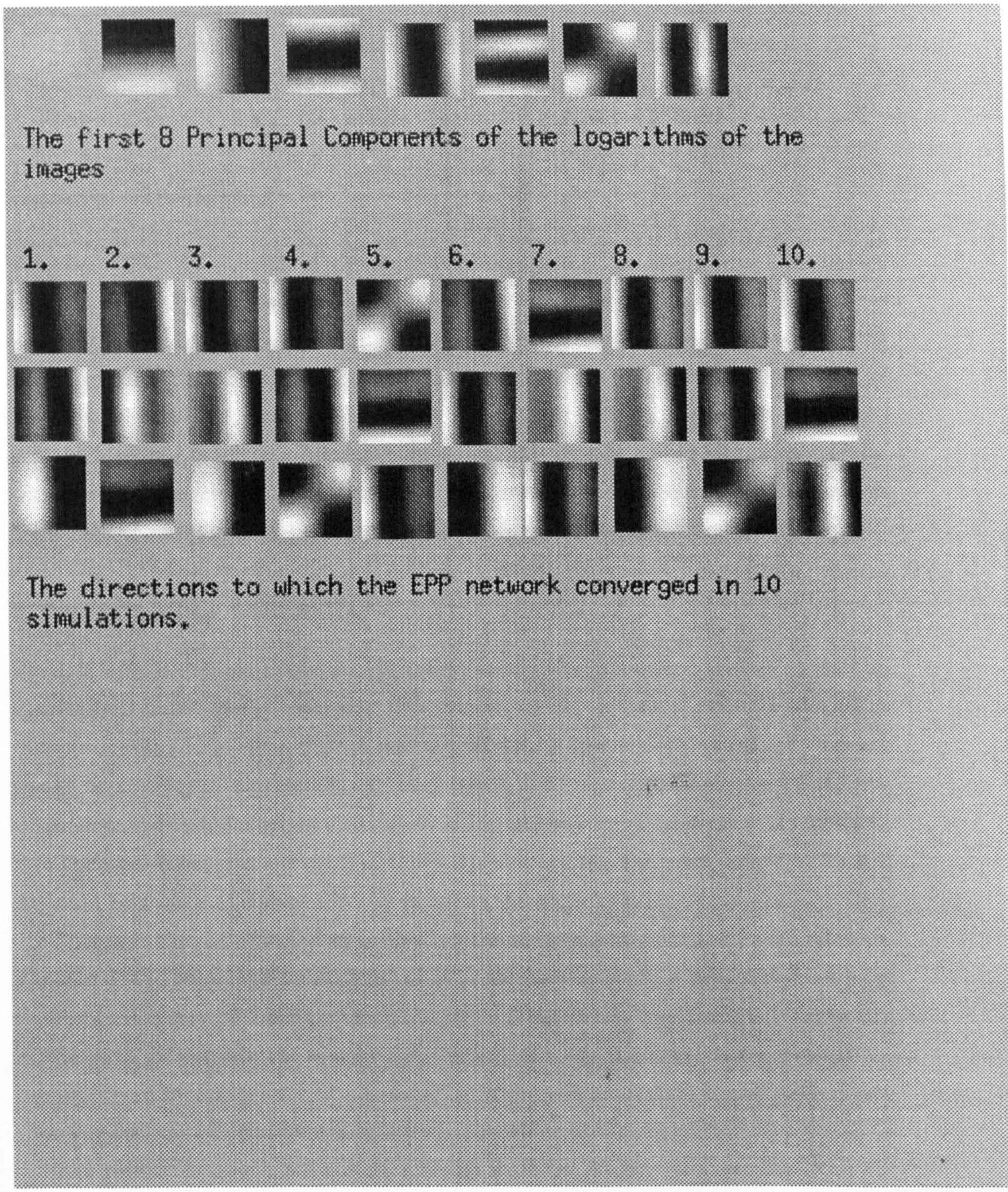


Figure 27: The directions found when the network is trained on logarithmically-preprocessed images.

Simulation	1	2	3	4	5	6	7	8	9	10
First	12.3	15.3	14.2	13.1	7.6	17.1	7.3	14.3	11.3	15.3
Second	14.8	12.5	14.6	13.7	6.6	13.0	15.2	13.2	16.3	7.0
Third	9.0	5.6	9.1	7.6	15.6	9.3	11.9	8.0	5.6	14.4

Table 31: The kurtosis for the directions found by the kurtosis seeking network after logarithmic preprocessing

windowed with a Gaussian of standard deviation of 6. The filters found are shown in Figure 28. As can be seen, the network still mainly converges first to vertical directions and such directions continue to be very strongly kurtotic. However, having found and subtracted one vertical and one horizontal direction it then finds a third direction which also shows great kurtosis but little pattern in its shape. We conjecture that such a pattern is due to the extremely local nature of the mask and the particular set of images met during training. Experiments with different width Gaussians have confirmed the major findings that vertical (more kurtotic) directions are preferred.

6.7.4 Coding Methods

Field (Field, 1994) has made an important distinction between compact codes and sparse distributed codes: we may define a compact code as a code in which the dimensionality of the input data is reduced whereas a sparse distributed code will use the same number of dimensions but only a few of these will be non-zero at any one time. Any code produced by a sparse distributed coder may be viewed as belonging to a reduced dimension subspace of the original space but the whole set of codes will not require a basis of dimensionality equal to the original space.

Therefore the code formed by projecting the data onto the first few Principal Components of the data may be thought of as a compact code since such a code reduces the dimensionality of the representation while it retains as much of the information in the data as possible. A sparse distributed code, on the other hand, retains (or perhaps even increases) the dimensionality of the representation but in such a way that any individual code uses only a few dimensions of the channel.

The context for such a distinction is the work by Barlow (Barlow, 1989) on the

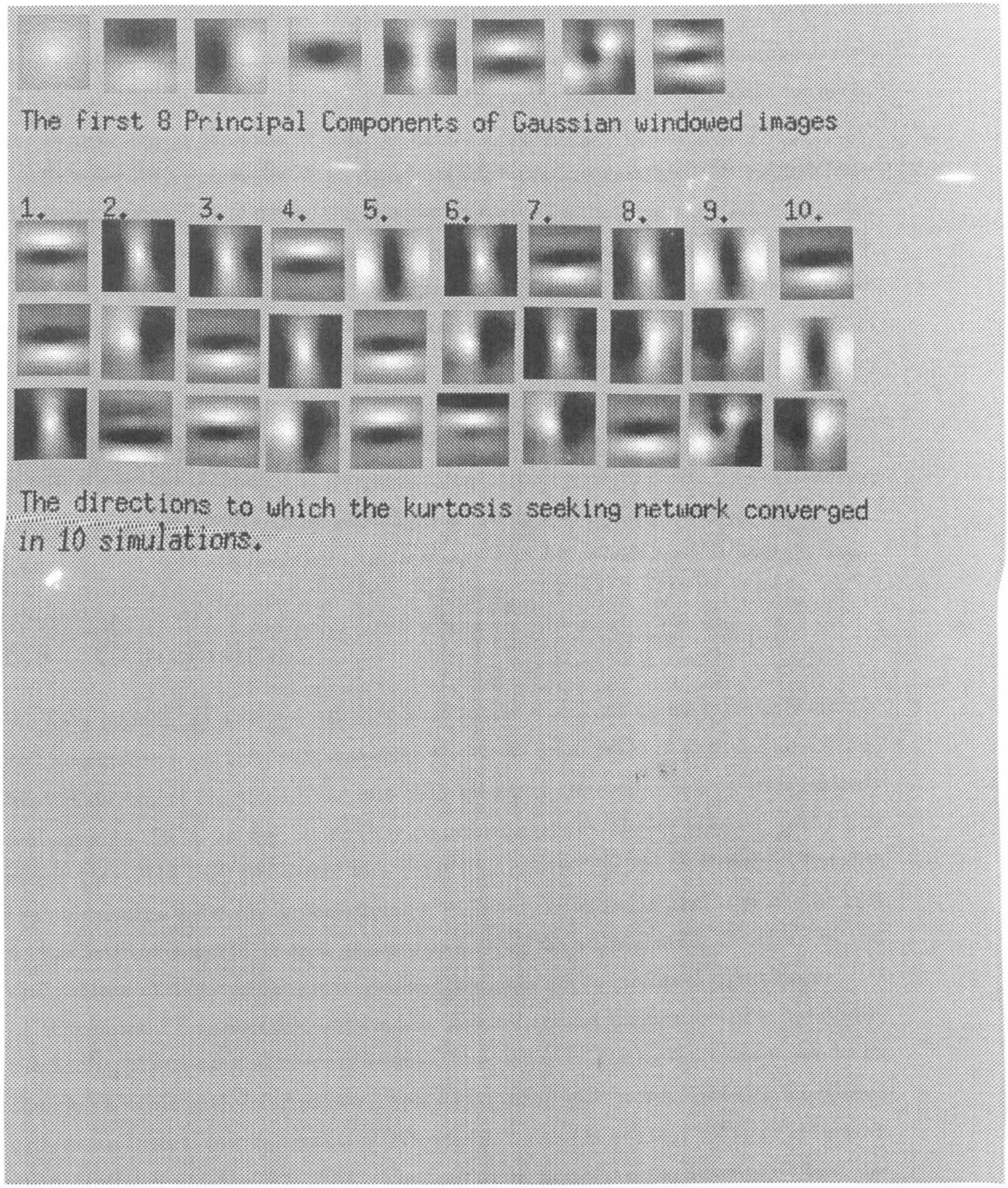


Figure 28: The directions found when the network is trained on gaussian-filtered images

early visual processing system. Barlow has suggested that a major task of this system is to use redundancies in the infinite information available to it in order to make sense of the data. Thus a PCA will reduce the redundancy inherent in the data by using the correlations in the data in order to create a compressed code.

However, in a sparse distributed code, while overall each individual cell may have the same probability of firing, the chances of two cells firing together are very much reduced. Thus, the chances of false "suspicious coincidences" are very much reduced. The statistics of such a code are strongly kurtotic - each code has a great number of low firing cells corresponding to the random occasional firing of neurons while at the same time there are a few cells, those which correspond to the signal, firing very strongly.

We have, in our Projection Pursuit Network, the first Artificial Neural Network which is capable of finding such sparse codes. Note also that the kurtosis-seeking neuron tends to identify those directions in the input data in which a line is most likely: i.e. the network becomes sensitized to directions which tend to contain lines.

6.8 Conclusion

We have introduced a neural network architecture which, using an extremely simple architecture and learning rule, has been shown to be capable of performing sophisticated statistical functions. The fact that the same network structure is capable of performing a PCA as well as EPP is unsurprising since Huber (Huber, 1985) has shown that PCA may be viewed as a particular case of Projection Puruit. Thus, for the interneuron network, in performing a PCA we are choosing $g(x) = x \propto \frac{d}{dx}x^2$ and so the original network is seen to be maximising the second order statistics of the distribution i.e. finding the eigenvectors corresponding to maximal eigenvalues.

The initial PP indices discussed in this Chapter are the simplest possible indices for the finding of non-normal interesting directions; however, the method was shown to be equally valuable with Information Theory indices or with more sophisticated indices such as an instantaneous version of Friedman's index (Friedman, 1987) or Intrator's index (Intrator and Cooper, 1992). The important point to note, however, is that the

method may be used with any function denoting a criterion which we wish to optimise. Our feeling is that information indices may prove to be the most effective indices in analysing a variety of distributions: we envisage further research into indices which maximise mutual information (Becker, 1992) or maximise the effects of contextual information subject to externally-imposed conditions e.g. (Kay and Phillips, 1994).

The advantage of using Projection Pursuit concepts is that it provides a framework for understanding and integrating previous neural network models which have tended to introduce non-linearity in an *ad hoc* fashion. However, we note that the format reviewed in this Chapter, required the function to be optimised to be differentiable; this need not be the case for the general neural network model. For example, Shapiro and Prügel-Bennet (Shapiro and Prugel-Bennett, 1992) have introduced a non-linearity - a power law - into Oja's Subspace Algorithm but also used a threshold below which the neuron will not fire. Since they set the threshold to be zero, the analysis of convergence of a second order network is understandable in PP terms yet the fact that the threshold may be changed suggests a direction for future research of PP indices.

Chapter 7

Conclusion

This thesis has discussed the role of negative feedback as an organising principle in the development of Arificial Neural Networks which use Hebbian learning to self-organise. We began by emphasising the holistic nature of our algorithms and we wish to reiterate this point here - every network is stable due to the interaction of the negative feedback of activation and the positive feedback of the learning rule which, in *all* cases, has been simple Hebbian learning. This method provides an alternative to classical methods of limiting weight growth in Hebbian learning networks and has the additional property that convergence takes place to statistically important sets of weights.

Chapter 3 introduced the interneuron network and showed that Hebbian learning alone in this context was sufficient to cause convergence to the Principal Components of the input data. In our continuing search to develop models which might possibly be models of actual biological networks we showed that a network in which the forward and backward weights were independent was also capable of performing a PCA. An algorithm which found the actual PCs was developed.

Chapter 4 investigated the case where the negative feedback of activation from each interneuron was allowed to affect all other interneurons. By introducing asymmetry into this process, we were able to devise a network whose weights converge to the actual Principal Components concurrently. Several different models were developed and analysed. However such models seem to be unrealistic models of biological

information processing.

Thus, in Chapter 5 we investigated situations which may reflect conditions in carbon-based neural nets: first by adopting Dale's Law and not allowing weights to change from excitatory to inhibitory or vice versa. In doing this, we showed that the weights converge to the actual principal components in a parallel fashion; this seems an important gain in return for a minor constraint, yet we showed that there are limits to the data for which this convergence can be guaranteed. A second condition was to take into consideration the actual physical distance between neurons which would manifest itself as a differential in the times when negative feedback would be felt. Such a net was also shown to converge to the PCs, a slight modification to which produced an extremely simple binary coder. An interesting application of this net with artificial data saw the generation of a topology preserving neural network.

In Chapter 6, we introduced non-linearity. The major insight of Chapter 6 is the embedding of the work which is being done on non-linear Principal Component type neural networks in the statistical theory of Projection Pursuit. In Chapter 6, we used an Exploratory Projection Pursuit network to find "interesting" structure in high-dimensional data spaces and investigated the statistics of images using this model.

7.1 Future directions

Future research on Artificial Neural Networks must inevitably concentrate on non-linearities. As we have pointed out, when closed form solutions of problems exist, they are almost always more efficient. The real promise of ANNs is in their application to non-linear problems. The non-linearities introduced in Chapter 6 are clearly not the only ones possible and the strength of the emergent properties of the networks described in that Chapter suggests that it would be wrong to dismiss any other non-linearity *a priori*. There are two obvious directions of continuing research in this area:

1. Research into different types of functions in the Exploratory Projection Pursuit networks described in Chapters 6.

2. Research into the introduction of non-linearity in different manners: we note that Karhunen and Joutsensalo have discussed a different form of non-linearity which has interesting signal separation properties.

The specific extensions which are of most interest to the author are the introduction of non-linearity in the regression form of the negative feedback network and in the limiting case where the non-linearity becomes a dichotomy the use of the network as a classifier.

7.2 Cybernetics

While we have not discussed biological neural networks in detail in this thesis, we have continued to attempt to create models which have biological relevance and believe that the mathematical models developed here are potentially of interest to neurophysiologists. Barlow (Barlow, 1989) has proposed that insight can be gained into the operation of low level vision operations by viewing them as statistical processes. For example, Principal Component Analysis results in filters that match some aspects of pysychophysical performance: the operation of the retina can be viewed, to a reasonable approximation, at high signal to noise levels as removing correlations between variables. Whether a PCA process is actually being implemented is a question for empirical research.

Barlow has also recently (Barlow and Tolhurst, 1992) suggested that edge detection is a consequence of the need for an economic representation of images in natural neural networks. He notes that, in whitened images, the edges are completely visible though whitening removes the correlations between pairs of images. Thus, if edges are examples of "suspicious coincidences", to investigate the statistics of edges in whitened images, we must go beyond second order statistics. Barlow has calculated the kurtosis excess (over the normal distribution's value) from sets of points and shows that when he choses a set of points which lie in a straight line that kurtosis for the set is much larger than average. However he feels that a search for fourth order statistics is too complex to be biologically feasible.

Yet we have shown here that an extremely simple neural network *is* capable of

finding kurtotic distributions. This would be an easy implementation of "edge detectors".

Thus we have described a set of models model which self-organise on very simple principles and which have demonstrably powerful statistical properties and seem equally valid as a model for new information processing devices and as a model for describing the processing of biological processes.

It is for reasons such as these that we feel able to evoke the spirit of cybernetics: our research has been into a model of information processing which, being simple and robust, seems capable of being a model for both biological information processing and engineered information processes. We have seen the construction of mathematical models as more important than any actual implementation but have nevertheless shown that all models reported herein can be easily implemented on current computers.

Bibliography

Ambrose-Ingerson, J., Granger, R., and Lynch, G. (1990). Simulation of paleocortex performs hierarchical clustering. *Science*, 247:1344–1348.

Amit, D. J. (1989). *Modelling Brain Function.* Cambridge University Press.

Anderson, J. A. (1968). A memory model using spatial correlation functions. *Kybernetik*, 5:113–119.

Armstrong, W., Dwelly, A., Liang, J., Lin, D., and Reynolds, S. (1991). Some results concerning adaptive logic networks. ftp from archive.cis.ohio-state.edu, /pub/neuroprose.

Baldi, P. and Hornik, K. (1988). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58.

Barlow, H. and Tolhurst, D. (1992). Why do you have edge detectors. JOSA Meeting, Albuquerque.

Barlow, H. B. (1989). Unsupervised learning. *Neural Computation*, 1:295–311.

Barto, A., Bradtke, S., and Singh, S. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57, University of Massachusetts.

Barto, A., Sutton, R., and Watkins, C. (1989). Learning and sequential decision making. COINS 89-95, University of Massachussetts.

Bates, E. and Elman, J. (1992). Connectionism and the study of change. Technical Report 9202, Centre for Research in Language, University of California, San Diego, La Jolla, CA 92093-0526.

Becker, S. (1992). *An Information-theoretic Unsupervised Learning Algorithm for Neural Networks.* PhD thesis, Gradute Department of Computer Science, University of Toronto.

Biehl, M. (1993). An exactly solvable model of unsupervised learning. (preprint).

Biehl, M. and Mietzner, A. (1993). Statistical mechanics of unsupervised structure recognition. ftp from archive.cis.ohio-state.edu, /pub/neuroprose. neuroprose.

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

Brause, R. (1993a). Transform coding by lateral inhibited neural nets. In *Proceedings of IEEE Tools with Artificial Intelligence.*

Brause, R. W. (1993b). A symmetrical lateral inhibition network for pca and feature decorrelation. In Gielen, S. and Kappen, B., editors, *ICANN93*, pages 486–490. Springer Verlag.

Carlson, A. (1990). Anti-hebbian learning in a non-linear neural network. *Biological Cybernetics*, 64:171–176.

Carpenter, G. A. (1989). Neural network models for pattern recognition and associative memory. *Neural Networks*, 2:243–257.

Carpenter, G. A. and Grossberg, S. (1987a). Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930.

Carpenter, G. A. and Grossberg, S. (1987b). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115.

Cohen, M. A., Grossberg, S., and Stork, D. G. (1988). *Evolution, Learning, Cognition, and Advanced Architectures*, chapter Speech Perception and Production by a Self-organizing Neural Network, pages 217–231. World Scientific Publishing.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley-Interscience Publication.

DeMers, D. and Cottrell, G. (1993). Non-linear dimensionality reduction. ftp archive.cis.ohio-state.edu, /pub/neuroprose.

Dennis, S. and Wiles, J. (1993). Integrating learning into models of human memory:the hebbian recurrent network. Technical report, University of Queensland.

Dennis, S., Wiles, J., and Humphries, M. (1992). What does the environment look like? setting the scene for interactive models of human memory. Technical Report 249, University of Queensland.

Diaconis, P. and Freedman, D. (1984). Asymptotics of graphical projections. *The Annals of Statistics*, 12(3):793–815.

Diamantaras, K. I. (1992). *Principal Component Learning Networks and Applications*. PhD thesis, Princeton University.

Dwelly, A. (1990). An implementation of adaptive logic networks. ftp from archive.cis.ohio-state.edu, /pub/neuroprose.

Elman, J. (1991). Incremental learning, or the importance of starting small. Technical Report 9101, University of California,San Diego, La Jolla, CA 92093-0126.

Elman, J. (1992). Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*.

Fahlman, S. and Lebiere, C. (1991). The cascade correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University.

Field, D. J. (1994). What is the goal of sensory coding. *Neural Computation*, 6:559–601.

Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209.

Friedman, J. H. (1987). Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266.

Fritzke, B. (1991). Let it grow- self-organising feature maps with problem dependent structure. In *ICANN-91*. Elsevier Science.

Fritzke, B. (1993a). *Advances in Neural Information Processing Systems 5*, chapter Kohonen Feature Maps and Growing Cell Structures - a Performance Comparison. Morgan Kaufmann.

Fritzke, B. (1993b). Vector quantization with a growing and splitting elastic net. In *Proceedings of the International Conference on Artificial Neural Networks.*

Fyfe, C. (1993a). The beneficial effect of distance on learning in interneurons. In *Neuronet'93.*

Fyfe, C. (1993b). A fully parallel pca network. In *IEEE/IEE Workshop on Natural Algorithms in Signal Processing.*

Fyfe, C. (1993c). Interneurons which identify principal components. In *Recent Advances in Neural Networks, BNNS93.*

Fyfe, C. (1993d). Pca properties of interneurons. In *From Neurobiology to Real World Computing, ICANN 93.*

Fyfe, C. (1993e). Positive weights in interneurons. In *Neural Computing, Research and Applications, the Third Irish Neural Networks Conference.*

Fyfe, C. (1993f). A simple homogeneous parallel pca network. In *IJCNN 93.*

Fyfe, C. and Baddeley, R. Non-linear data structure extraction using simple hebbian networks. *Biological Cybernetics.* (to appear).

Fyfe, C. and Baddeley, R. (1994). A projection pursuit neural network. In *Irish Neural Net Conference.*

Fyfe, C. and McGregor, D. R. (1994). A novel topology-preserving network. In *Irish Neural Net Conference.*

Grossberg, S. (1968). Some nonlinear networks capable of learning a spatial pattern of arbitrary complexity. *Proceedings of the National Academy of Sciences(USA)*, 59:368–372.

Grossberg, S. (1984). Unitization, automaticity, temporal order, and word recogntion. *Cognition and Brain Theory*, 7:263–283.

Grossberg, S. (1988a). Non-linear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1:17–61.

Grossberg, S. (1988b). Nonlinear neural networks:principles, mechanisms, and architectures. *Neural Networks*, 1:17–61.

Grossberg, S. and Schmajuk, N. A. (1989). Neural dynamics of adaptive timing and temporal discrimination during associative learning. *Neural Networks*, (2):79–102.

Hall, P. (1989). On polynomial-based projection indices for exploratory projection pursuit. *The Annals of Statistics*, 17(2):589–605.

Hebb, D. O. (1949). *The Organisation of Behaviour.* Wiley.

Hertz, J., Krogh, A., and Palmer, R. G. (1992). *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing.

Hinton, G. and Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review*, 98(1):74–95.

Hinton, G. E. and Nowlan, S. (1990). The bootstrap widrow-hoff rule as a cluster-formation algorithm. *Neural Computation*, 2(3):355–362.

Hopfield, J. (1982). Neural networks and physical systems with collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79:2554–2558.

Horswell, R. L. and Looney, S. W. (1992). A comparison of tests for multivariate normality that are based on measures of multivariate skewness and kurtosis. *Journal of Statistical Computing Simulations*, 42:21–38.

Huang, S. and Huang, Y. F. (1993). Principal component vector quantization. *Journal of Visual Communication and Image Representation*, 4(2):112–120.

Huber, P. J. (1985). Projection pursuit. *Annals of Statistics*, 13:435–475.

Intrator, N. (1990). A neural network for feature extraction. In *NIPS 2*, pages 719–726. Morgan Kaufman.

Intrator, N. (1992). Feature extraction using an unsupervised neural network. *Neural Computation*, 4(1):98–107.

Intrator, N. (1993a). Combining exploratory projection pursuit and projection pursuit regression with application to neural networks. *Neural Computation*, 5:443–455.

Intrator, N. (1993b). On the use of projection pursuit constraints for training neural networks. In Spatz, B., editor, *NIPS 5*, pages 3–10. Morgan Kaufmann.

Intrator, N. and Cooper, L. N. (1992). Objective function formulation of the bcm theory of visual cortical plasticity: Statistical connections, stability conditions. *Neural Networks*, 5:3–17.

Intrator, N. and Gold, J. I. (1993). Three-dimensional object recognition using an unsupervised bcm network: The usefulness of distinguishing features. *Neural Computation*, 5:61–74.

Jolliffe, I. (1986). *Principal Component Analysis.* Springer-Verlag.

Jones, M. C. and Sibson, R. (1987). What is projection pursuit. *The Royal Statistical Society.*

Jonker, H. (1992). *Information Processing and Self-Organisation in Neural Networks with Inhibitory Feedback.* PhD Thesis.

Karhunen, J. and Joutsensalo, J. (1992). Nonlinear hebbian algorithms for sinusoidal frequency estimation. In Aleksander, I. and Taylor, J., editors, *Artificial Neural Networks, 2*, pages 1099–1103. North-Holland.

Karhunen, J. and Joutsensalo, J. (1993a). Learning of robust principal component subspace. In *1993 International Joint Conference on Neural Networks*, pages 2409–2412.

Karhunen, J. and Joutsensalo, J. (1993b). Nonlinear generalizations of principal component learning algorithms. In *IJCNN'93*, pages 2599–2602.

Karhunen, J. and Joutsensalo, J. (1994). Representation and separation of signals using nonlinear pca type learning. *Neural Networks*, 7(1):113–127.

Kay, J. and Phillips, W. (1994). Activation functions, computational goals and learning rules for local processors with contextual guidance. Technical Report CCCN-15, Centre for Cognitive and Computational Neuroscience, University of Stirling.

Kehagias, A. (1991). Stochastic recurrent networks: Prediction and classification of time series. ftp from archive.cis.ohio-state.edu, /pub/neuroprose.

K.Miller and MacKay, D. (1992). The role of constraints in hebbian learning. *Neural Computation.*

Kohonen, T. (1974). An adaptive associative memory principle. *IEEE Transactions on Computers*, C-23:444–445.

Kohonen, T. (1984). *Self-Organization and Associative Memory.* Springer-Verlag.

Kosko, B. (1991). *Pattern Recognition by Self-Organising Neural Nets*, chapter Adaptive Bidirectional Associative Memories, pages 207–236.

Levin, A. (1990). Optimal dimensionality reduction using hebbian learning. In *Connectionist Summer School*, pages 141–144.

Linsker, E. (1988). Self-organization in a perceptual network. *IEE Computer*, pages 105–117.

Linsker, R. (1986a). From basic network principles to neural architecture. In *Proceedings of National Academy of Sciences.*

Linsker, R. (1986b). From basic network principles to neural architecture. In *Proceedings of National Academy of Sciences.*

Lipschutz, S. (1968). *Theory and Problems of Linear Algebra.* McGraw-Hill.

Mardia, K. V., Kent, J., and Bibby, J. (1979). *Multivariate Analysis.* Academic Press.

Martinetz, T. (1993). Competitive hebbian learning rule forms perfectly topology preserving maps. In Gielen, S. and Kappen, B., editors, *ICANN93*, pages 427–434. Springer Verlag.

McClelland, J., Rumelhart, D. E., and Group, T. P. R. (1986). *Parallel Distributed Processing*, volume Volume 1 and 2. MIT Press.

Murphy, P. C. and Sillito, A. M. (1987). Corticofugal feedback influences the generation of length tuning in the visual pathway. *Nature*, 329:727–729.

Oja, E. (1982). A simplified neuron model as a principal component analyser. *Journal of Mathematical Biology.*

Oja, E. (1989). Neural networks, principal components and subspaces. *International Journal of Neural Systems.*

Oja, E. and Karhunen, J. (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications*, 106:69–84.

Oja, E. and Karhunen, J. (1993). Nonlinear pca: Algorithms and applications. Technical Report A18, Helsinki University of Technology.

Oja, E., Ogawa, H., and Wangviwattana, J. (1992a). Pca in fully parallel neural networks. In Taylor, A. ., editor, *Artificial Neural Networks,2.*

Oja, E., Ogawa, H., and Wangviwattana, J. (1992b). Principal component analysis by homogeneous neural networks, part 1: The weighted subspace criterion. *IEICE Trans. Inf. & Syst.*, E75-D:366–375.

Oja, E., Ogawa, J., and Wangviwattana, J. (1991). Learning in nonlinear constrained hebbian networks. In Kohonen, T., Makisara, K., Simula, O., and Kangas, J., editors, *Artificial Neural Networks*, pages 385–390. Elsevier Science Publishers.

Palmieri, F. (1993). Linear self-association for universal memory and approximation. In *World Congress on Neural Networks*, pages 2–339– 2–343. Lawrence Erblaum Associates.

Palmieri, F., Zhu, J., and Chang, C. (1993). Anti-hebbian learning in topologically constrained linear networks: A tutorial. *IEEE Transactions on Neural Networks*, 4(5):748 – 761.

Pece, A.,(1992). *Redundancy reduction of a Gabor representation: a possible computational role for feedback from primary visual cortex to lateral geniculate nucleus.* North-Holland.

Plumbley, M. (1991). *On Information Theory and Unsupervised Neural Networks.* PhD thesis, University of Cambridge.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1988). *Numerical Recipes in C.* Cambridge University Press.

Prugel-Bennett, A. and Shapiro, J. L. (1993). Statistical mechanics of unsupervised hebbian learning. *Journal of Physics A : Math. Gen.*, 26:2343–2369.

Ritter, H. and Kohonen, T. (1989). Self-organising semantic maps. *Biological Cybernetics*, 61:241–254.

Robinson, D. A. (1987). *Vision, Brain, and Cooperative Computation*, chapter : Why Vusuomotor Systems Don't Like Negative Feedback and How They Avoid It, pages 89–107. MIT Press.

Rubner, J. and Schulten, K. (1990). Development of feature detectors and self-organisation. *Biological Cybernetics.*

Rubner, J. and Tavan, P. (1989). A self-organising network for principal-component analysis. *Europhysics Letters*, 10(7):693–698.

Sanger, T. (1990). Analysis of the two-dimensional receptive fields learned by the generalized hebbian algorithm in response to random input. *Biological Cybernetics.*

Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal.*

Shapiro, J. L. and Prugel-Bennett, A. (1992). Unsupervised hebbian learning and the shape of the neuron activation function. In Aleksander, I. and Taylor, J., editors, *Artificial Neural Networks, 2*, pages 179–183. North-Holland.

Shultz, T. and Schmidt, W. (1991). A cascade-correlation model of balance scale phenomena. In *Thirteenth Annual Conference of the Cognitive Science Society*, pages 635–640. Erblaum.

Steinbuch, K. (1961). Die lernmatrix. *Kybernetik*, (1):36–45.

Sun, J. (1993). Some practical aspects of exploratory projection pursuit. *SIAM Journal of Scientific Computing*, 14(1):68–79.

Thrun, S. (1992). Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University.

von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14:85–100.

White, R. W. (1993). Competitive hebbian learning 2:an introduction. In *World Congress on Neural Nets*, pages 557–560.

Wiener, N. (1948). *Cybernetics*. MIT Press.

Williams, R. J. (1985). Feature discovery through error-correcting learning. Technical Report 8501, Institute for Cognitive Science, University of California, San Diego.

Willshaw, D. and von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organisation. *Proceedings of the Royal Society of London B*, 194:431–445.

Willshaw, D. J., Buneman, O., and Longuet-Higgins, H. (1969). Non-holographic associative memory. *Nature*, 222:960–962.

Xu, L. (1993). Least mean square error reconstruction principle for self-organizing neural-nets. *Neural Networks*, 6(5):627 – 648.

Zhao, Y. (1992). *On Projection Pursuit Learning*. PhD thesis, MIT.