Department of Engineering
University of Cambridge

Low Entropy Coding with Unsupervised Neural Networks

George Francis Harpur
Queens' College

February 1997

A dissertation submitted for
the degree of Doctor of Philosophy
at the University of Cambridge

Summary

The discovery of the *independent components* of a series of data is the 'holy grail' of unsupervised learning systems. It results in a code that is optimally compressible, easily processed, and a system that generates such a code can be said to have 'understood' its data. While fully independent representations are extremely hard to achieve in most real-world situations, any system that reduces the *redundancy* in its data can bring similar benefits.

This thesis describes an information-theoretic framework for unsupervised learning, and identifies the usefulness of maximising information transfer while simultaneously attempting to minimise the sum of output entropies. A recurrent neural network model is developed with the aim of fulfilling these goals.

The model is first introduced as a purely linear network that minimises the differences between the original input and a reconstruction generated from the output values. The basic model is then modified to help it produce efficient codes by the addition of various constraints, such as through the use of penalty functions. These encourage outputs to have *low entropy* distributions, and in particular are used to promote *sparseness*. Modifications which allow the network to use nonlinear mixture models, including one which takes into account the effects of occlusion, are also explored.

The practical applicability of the work is illustrated with examples based on visual and speech data. The ability of the network to automatically generate *wavelet codes* from natural images is demonstrated. These bear a close resemblance to *2-D Gabor functions*, which have previously been used to describe physiological receptive fields, and as a means of producing compact image representations.

Keywords: neural networks, unsupervised learning, self-organisation, feature extraction, information theory, redundancy reduction, sparse coding, imaging models, occlusion, image coding, speech coding.

Declaration

This dissertation is the result of my own original work, except where reference is made to the work of others. No part of it has been submitted for any other university degree or diploma. Its length, including captions, footnotes, appendix and bibliography, is approximately 58 000 words.

Acknowledgements

I would like first and foremost to thank Richard Prager, my supervisor, for giving me a wonderful balance of freedom to pursue my own ideas and guidance to prevent me from wandering lost. Thanks are due also to Bruno Olshausen for many enlightening discussions, and to Mark Plumbley, John Daugman, Arthur Pece, Roland Baddeley and Horace Barlow for their help and inspiration. I am greatly indebted to many members of the SVR group, in particular to Jonathan Lawn, Chris Dance, Mark Gales and Simon Blackburn for useful advice, David Lovell for some excellent proof-reading, and the Davids, Gary, Dan, Marty, Mike, John, Phil, James, Tina, Steve and others for making the lab such an enjoyable place to work. I would like to gratefully acknowledge the financial support of the EPSRC, the Engineering Department, and Queens' College during the course of the past three years.

Finally, I would like to thank my family for their constant support, and Katy for her understanding and encouragement, particularly in the final gruelling stages of writing this thesis.

Notation and Abbreviations

Every effort has been made to keep notation consistent throughout this thesis. All commonly used symbols and abbreviations are listed below. For the sake of simplicity, various abuses of notation are made, but the intended meaning should be clear from the context. In particular, the separate probability density functions $p_X(x)$ and $p_Y(y)$ for random variables X and Y will commonly be written simply as $p(x)$ and $p(y)$. Similarly, identical symbols will often be used to represent both a set of data and particular samples from that set.

In describing the number of calulations $g(n)$ required by an algorithm for a problem of size n, the standard 'big-O' notation is used, so that an algorithm is said to be $\mathcal{O}(f(n))$ if there exist constants c and n_0 such that $g(n) \leq cf(n)$ for all $n > n_0$.

General

$\mathbf{X}$	a matrix composed of elements x_{ij}
$\mathbf{x}$	a column vector composed of elements x_i
$\|x\|$	absolute value of scalar x
$\|\|\mathbf{x}\|\|$	length (L_2-norm) of vector $\mathbf{x}$
$\|\|\mathbf{x}\|\|_r$	L_r-norm of vector $\mathbf{x}$
$\mathbf{x}^{\mathrm{T}}$	transpose of $\mathbf{x}$
$\mathbf{X}^{-1}$	inverse of $\mathbf{X}$
$\mathbf{X}^{+}$	pseudoinverse of $\mathbf{X}$
$\det\mathbf{X}$	the determinant of $\mathbf{X}$
$\mathbf{I}$	the identity matrix
$f'(x)$	the derivative of f with respect to its argument
[( ])	inclusive and exclusive lower and upper bounds on a range of numbers respectively, so that for example $[a, b)$ represents a range containing numbers x, where $a \leq x < b$
$\mathcal{E}[x]$	the expected value of x
$\hat{x}$	'true' or optimal value for x
δ_{ij}	the Kronecker delta symbol ($\delta_{ij} = 1$ if $i = j$, 0 otherwise)

Information theory

$H(X)$	entropy of random variable X
$H(X, Y)$	joint entropy of X and Y
$H(X\|Y)$	conditional entropy of X given Y
$I(X; Y)$	mutual information between X and Y
$D_{KL}(p \,\|\|\, q)$	Kullback-Leibler divergence (relative entropy) between p.d.f.s p and q

Network model

m	number of input units
n	number of output units
$\mathbf{x}$	input vector
$\widetilde{\mathbf{x}}$	reconstructed input

$\mathbf{r}$	output from first layer (residual)
$\mathbf{a}$	output vector
$\mathbf{W}$	weight matrix
$\mathbf{w}_i$	weight vector of ith unit (transpose of the ith row of $\mathbf{W}$)
E_r	reconstruction error
E_R	expected reconstruction error over all data
E_p	penalised reconstruction error
E_P	expected penalised error over all data
μ	activation rate
η	learning rate
Ω	overall penalty term
ω	penalty function (for an individual output)
λ	penalty parameter
b_{ij}	mixture weighting
d_i	object depth

Abbreviations

CVQ	Cooperative vector quantisation
EM	Expectation maximisation
HMM	Hidden Markov model
ICA	Independent component analysis
ICL	Iterative competitive learning
LP	Linear programming
MAP	Maximum *a posteriori*
MFCC	Mel-frequency cepstral coefficient
MLP	Multi-layer perceptron
MSE	Mean-squared error
PCA	Principal component analysis
p.d.f.	Probability density function
PP	Projection pursuit
REC	Recurrent error-correction
SEC	Symmetric error-correction
SSE	Sum-squared error
VQ	Vector quantisation
w.r.t.	With respect to
WSJ	Wall Street Journal
WTA	Winner-take-all

Contents

List of Figures

List of Tables

Causes and effects are discoverable, not by reason but by experience.
— David Hume, *An Inquiry Concerning Human Understanding* (1784)

Felix qui potuit rerum cognoscere causas.
— Virgil, *Georgics* (*ca.* 29 B.C.)

Chapter 1

Introduction

A beginner's guide to pattern recognition: describe an object by one hundred binary features that between them represent all of its distinguishing characteristics and which are statistically independent of each other. Congratulations! You have solved the problem of pattern recognition: you are now almost certain to be able to recognise this object, from amongst several thousand others described in the same way, even if it turns out that a significant proportion of your features were in fact determined incorrectly. You will be able to perform the recognition in a fraction of a second on a modern digital computer.[1]

The problem, of course, lies in obtaining a set of features that fulfil the above criteria, and in particular ones that achieve *independence*. Nevertheless, cast in this way, pattern recognition may be seen entirely as a problem of *representation* — if we can model the data in the right way, subsequent processing becomes easy. However we should not be under any pretence that finding a good representation is a simple task. If it were, then the field of artificial intelligence would not still be in its infancy after nearly fifty years.

This thesis takes a small step on the journey towards making systems that are able to automatically learn good representations from complex data. To do this, it borrows some ideas from the only system we know of that is able to perform such tasks — the brain. At the same time, it tries to stay within the confines of the best theories we have to describe uncertainty, namely statistics and information theory. The approach is limited, and contains many simplifying assumptions that cannot fully be justified. The results, however, are interesting.

1.1 Related work and original content

The work in this thesis draws on many of the ideas commonly used in the field of neural networks, ideas that are well-described by several recent books (Bishop, 1995; Ripley, 1996; Haykin, 1994). Much of the inspiration for this work comes from the concept of *redundancy reduction* put forward by Attneave (1954) and Barlow (1959, 1961, 1989), and which is founded in information theory (Shannon and Weaver, 1949). Similar ideas are captured in the framework of *minimum description length* (Rissanen, 1978, 1985; Zemel, 1993; Hinton and Zemel, 1994). Further links between information theory and neural networks have been

[1] This idea is based on concepts discussed by Daugman (1993), to which the reader is referred for further details.

made by several authors (Linsker, 1988, 1990, 1992; Plumbley, 1991; Atick, 1992; Redlich, 1993a; Becker and Hinton, 1992; Deco and Obradovic, 1996).

This work is also related to the concepts of both *independent component analysis* and *source separation* (Jutten and Herault, 1991; Comon, 1994) which have been applied to network models by Bell and Sejnowski (1995), Deco and Obradovic (1995), Oja (1995) and Karhunen *et al.* (1995). The same idea may be described as *factorial coding* (Schmidhuber, 1992; Redlich, 1993b; Ghahramani, 1995). Several authors have noted the importance of *sparse* coding in achieving these goals (Földiák, 1990; Field, 1994; Földiák and Young, 1995; Fyfe and Baddeley, 1995a).

The work in this thesis is based on an iteratively-activated neural network model that uses *negative feedback* and is similar to one set out by Pece (1992) on the basis of previous work by Daugman (1988). The properties of the activation are explored in detail, and learning is added with a simple Hebbian rule. The basic model so obtained is closely related to one developed independently and concurrently by Olshausen and Field (1996a,b) for generating sparse image codes.

This work builds an understanding of the model, in terms of its dynamics, its applicability, and practical issues relating to its implementation on computer systems. The motivation and methods for *constraining* the model in various ways are investigated in depth, and theoretical explanations are developed. The network is also modified to allow inclusion of nonlinear mixture models, as an extension of the ideas put forward for a binary network by Saund (1994, 1995).

The network's performance is evaluated in a number of experiments with both artificial and real data. The image-coding experiments of Olshausen and Field (1996a,b) are reproduced and extended, with some key differences in the approaches made clear. The network is also applied to the coding of handwritten characters and speech waveforms.

Some of the preliminary work for this thesis has previously been published by Harpur and Prager (1994, 1995, 1996).

1.2 The neural network paradigm

Neural network research saw an explosion of interest in the late-1980s. Against this backdrop, the field struggled to establish itself as a serious and theoretically justifiable challenger to more traditional statistical techniques. This has largely been successful, and to such an extent that distinctions between neural networks and statistics have in many cases become blurred. It is quite possible, for example, to find papers in recent neural network conferences and journals where little or no reference is made to any kind of network architecture.

There does, however, remain a distinct role for the neural paradigm, *i.e.* a model based on a network of simple but highly interconnected processing units. Formulating a system in this way can bring several distinct benefits:

- A neural network provides a solid conceptual model for what can otherwise be quite abstract computational techniques. Viewing a complex minimisation, for example, as the dynamics of a system whose components are simple to understand is potentially a very powerful method for giving new insight into such processes.

- Once framed as a network model, various extensions to existing techniques often become apparent. Frequently, for example, networks begin life as small linear models for which the theory is already well known. Subsequently they can then be generalised quite naturally to various high-dimensional and nonlinear cases, at which point they may well become more powerful than previously existing techniques.

- Neural networks (at least in their 'purest' form) are inherently *localised* models, that is, each unit's operation depends almost exclusively on the signals provided by incoming connections. A high degree of localisation is closely connected with low computational complexity, since limiting the number of signals needed by a unit also limits the amount of processing it has to perform. At the same time, a highly localised model lends itself naturally to a highly parallelised implementation.

- Artificial neural networks were originally inspired by their biological counterparts, and there is potentially great benefit to be had from retaining this link. The problems people attempt to address with neural networks are often ones whose solutions have eluded researchers for many years, and yet are also ones that biological systems appear to solve with great ease. It seems sensible therefore not to overlook any clues that are to be had from reverse-engineering a system that already has the answers we require.

All of these factors have played a part in the work described here. Both the basic model and its various extensions were originally conceived in the form of networks, and only subsequently described in mathematical terms. A primary goal of the work has been to favour simplicity wherever possible, and, where successful, this has largely been the result of viewing the methods in terms of their network implementations. Perhaps most significantly, some important steps have been inspired by physiology. For example the decision to perform large experiments with just a few iterations of the recurrent model (Chapter 7), which theoretically seemed unlikely to work, was taken purely because the brain, if recurrent, must be similarly constrained.

1.3 Overview of the chapters

Chapter 2 introduces some key concepts in information theory, discusses the general nature of the modelling problem, and shows how many important modelling issues may be expressed in terms of information theory and coding.

In Chapter 3 we shall review some of the pre-existing unsupervised learning methods, in particular those based on neural network models. The primary emphasis will be on systems that are forerunners to this work, and on pointing out their strengths and weaknesses. Chapter 4 presents a recurrent network model that is intended to overcome some of these limitations. The network is linked to an inhibitory feedback mechanism that has been proposed as a computational model for the brain by several leading researchers. Its operation is described, first in mathematical terms, and then by means of several simple examples. The model is put forward as a flexible means of generating *complete* representations of a perceptual environment.

The network model is enhanced in Chapter 5 by imposing various constraints aimed at making the representations *efficient* as well as complete. Low entropy, and in particular *sparseness*, are identified as a means to achieve this in many cases. These ideas are again developed mathematically and related to probability and information theory, and subsequently illustrated with some simple examples.

Chapter 6 extends the network model to cases where sub-patterns in the environment may no longer be said to mix linearly. In particular, we shall look at situations where individual features are binary-like, and explore their interaction in terms of *imaging models*. Work on modelling *occlusion* in visual environments is also presented, and the validity of these ideas is demonstrated with experimental results.

In Chapter 7 the theory that has been developed previously is applied to real-world data from vision and speech. Automatic generation of *wavelet codes* from natural images is demonstrated, as is the production of other 'feature-based' codes from handwritten characters and continuous speech data. The success of the networks in these tasks is measured in terms of the information-theoretic framework that was set out in Chapter 2.

We shall conclude in Chapter 8 with a discussion of the contributions made by this thesis, some comparisons with other work in related areas, and suggestions for future developments in this line of research.

Chapter 2

Framework

The task that this work seeks to address is that of generating good models of complex environments. We shall attempt to define what is meant by a 'good' model in terms of *information theory*, and shall begin therefore with a look at some of the key concepts from this important field. The properties of models within this framework are described, and the aims of a general-purpose modelling system are reduced to two information-theoretic goals. We shall also discuss the nature of such a system in terms of the codes it generates, and shall look at some of the difficulties inherent in the modelling process.

2.1 An introduction to information theory

Information theory provides a useful framework in which to express many of the ideas presented in this thesis, so it will be helpful to examine some of the main concepts of this theory before progressing. Further details may be found in standard texts on information theory (*e.g.* Ash, 1965; Cover and Thomas, 1991).

2.1.1 Information and entropy

The *information* associated with an event of probability p is defined as $\log(1/p)$. As a measure of information, this simple formula achieves two intuitively appealing properties:

1. The more unlikely an event is, the more information it conveys, while an event that is a certainty ($p = 1$) carries no information at all.

2. The *total* information conveyed by two independent events (with probabilities p_1 and p_2) is the same whether we consider them as a single event with probability $p_1 p_2$ or simply add the individual information measures, since $\log[1/(p_1 p_2)] = \log(1/p_1) + \log(1/p_2)$.

For complete systems, it is useful to have a model of an information-generating mechanism. The simplest of these is the *discrete memoryless information source*, and we shall assume all sources to be of this type. The source X emits a sequence of symbols from a finite alphabet $\mathcal{A}_X = \{x_1, x_2, \ldots, x_n\}$, each symbol occurring with a statistically independent probability

given by the function $p_X(x)$ for $x \in \mathcal{A}_X$. We are often interested in *the mean information obtained per symbol*. This is simply

$$\begin{aligned} H(X) &= \mathcal{E}[\log\{1/p(x)\}] \\ &= \sum_{x\in\mathcal{A}_X} p(x)\log\{1/p(x)\} \\ &= -\sum_{x\in\mathcal{A}_X} p(x)\log p(x) \end{aligned} \tag{2.1}$$

where we have used $p(x)$ as shorthand for $p_X(x)$, and $\mathcal{E}[\cdot]$ is the expectation operator. This quantity is known as the *entropy* of the source. Since probability requires that $p(x) \leq 1$, we know that $\log p(x) \leq 0$, and so $H(X)$ is guaranteed to be non-negative. If the logarithm of (2.1) is in base 2, then the entropy is measured in *bits*; if it is in base e (*i.e.* a *natural* logarithm), the entropy is said to be expressed in *nats*.

At one extreme, where all n possible values for X are equally likely, the entropy $H(X)$ has its maximum value ($\log n$). At the other, where there is only one value of X with non-zero probability (*i.e.* X is *deterministic*), $H(X) = 0$. We therefore have the idea that entropy is associated with the 'peakiness' of a distribution — the more sharply it is peaked, the lower the entropy.

The choice to measure entropy in bits is not purely accidental. If, for example, the source X has four equiprobable values, its entropy, from (2.1), is given by

$$H(X) = 4 \times \frac{1}{4} \times \log_2 4 = 2 \text{ bits}$$

corresponding to the number of bits required to represent such a variable in a binary code. Less intuitive is that if the values were *not* equiprobable, the entropy would be lower, and need not be a whole number of bits. In coding terms, the entropy is in fact giving us the *mean* length of the shortest (*i.e.* most efficient) possible code for representing samples of X. Entropy, therefore, is a measure of the *compressibility* of an information source.

2.1.2 Joint and conditional entropy

If we now consider a *pair* of sources X and Y with alphabets $\mathcal{A}_X$ and $\mathcal{A}_Y$ respectively, we may define some quantities relating to the dependency between them. Firstly, the *joint entropy*, written as $H(X,Y)$, comes directly from the joint probability distribution $p(x,y)$:

$$\begin{aligned} H(X,Y) &= -\mathcal{E}[\log p(x,y)] \\ &= -\sum_{x\in\mathcal{A}_X}\sum_{y\in\mathcal{A}_Y} p(x,y)\log p(x,y). \end{aligned}$$

Secondly, we may define the *conditional entropy* as[1]

$$\begin{aligned} H(Y|X) &= -\mathcal{E}[\log p(y|x)] \\ &= -\sum_x p(x)\sum_y p(y|x)\log p(y|x) \\ &= -\sum_{x,y} p(x,y)\log p(y|x). \end{aligned}$$

[1]For simplicity the ranges of summations are omitted from this point onwards, and joint summations are combined where possible.

This value gives us the information we can expect, on average, from a sample of Y if we already know the corresponding sample from X. Entropy, joint entropy and conditional entropy obey the simple relation

$$H(X,Y) = H(X) + H(Y|X). \tag{2.2}$$

This is easily proved by starting from the standard relation $p(x,y) = p(x)p(y|x)$ and taking the logarithms and expectations of both sides. Repeated application of (2.2) extends this relation to a *chain rule* for many variables $X_1, X_2, \ldots, X_n$:

$$H(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1}, \ldots, X_1). \tag{2.3}$$

2.1.3 Relative entropy and mutual information

We may define the *relative entropy* or *Kullback-Leibler divergence* of two probability distributions $p(x)$ and $q(x)$ as

$$\begin{aligned} D_{KL}(p \,||\, q) &= \mathcal{E}_p\left[\log \frac{p(x)}{q(x)}\right] \\ &= \sum_x p(x) \log \frac{p(x)}{q(x)} \end{aligned}$$

where $\mathcal{E}_p[\cdot]$ denotes the expectation over the distribution p. The relative entropy may be interpreted as a measure of *distance* between the distributions p and q. Strictly, it is not a distance metric because it is not symmetric and does not obey the triangle inequality, but it does have the property that $D_{KL}(p \,||\, q) \geq 0$, with equality if and only if $p = q$.

The *mutual information* $I(X;Y)$ between two sources X and Y may be expressed as the relative entropy between their joint distribution and the product of their marginal distributions:

$$\begin{aligned} I(X;Y) &= D_{KL}(p(x,y) \,||\, p(x)p(y)) \\ &= \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}. \end{aligned} \tag{2.4}$$

It is a standard statistical property that if X and Y are statistically independent then $p(x,y) = p(x)p(y)$. From the properties of relative entropy noted above, therefore, zero mutual information is a necessary and sufficient condition for statistical independence. Where X and Y form part of the same code, mutual information is a useful measure of the *redundancy* between them.

Splitting the mutual information (2.4) into two terms, we find that

$$\begin{aligned} I(X;Y) &= -\sum_{x,y} p(x,y) \log p(x) + \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(y)} \\ &= -\sum_{x} p(x) \log p(x) + \sum_{x,y} p(x,y) \log p(x|y) \\ &= H(X) - H(X|Y). \end{aligned} \tag{2.5}$$

We could equally have split the terms in x and y the other way round, so we see that $I(X;Y) = I(Y;X)$. The relationships (2.2) and (2.5) between the various entropic quantities for two random variables are summarised in Figure 2.1.

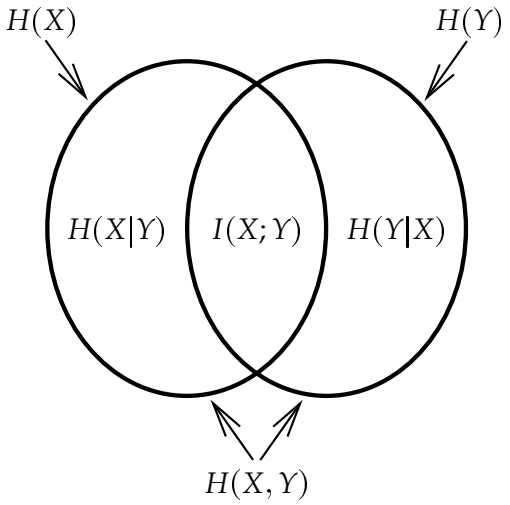


Figure 2.1: Representation of the relationship between different entropic quantities for two random variables.

2.1.4 Differential entropy

We have so far only looked at the entropy of discrete random variables. To extend this to the case of a *continuous* random variable X with probability density function $p(x)$, we simply replace the summation of (2.1) with an integral, and define the *differential entropy* as

$$h(X) = -\int_S p(x) \log p(x)\, dx$$

where S is the set where $p(x) > 0$ (known as the *support set* of X).

As an example, consider the case of a continuous random variable X with a uniform distribution over the range 0 to $a\,(>0)$. This distribution has a density of $1/a$ between 0 and a, and of 0 elsewhere, so the differential entropy of X is

$$h(X) = -\int_0^a (1/a) \log(1/a)\, dx = \log a.$$

We note that for $a < 1$, $h(X) < 0$, and so observe that, unlike the entropy of discrete random variables, differential entropy can be negative. Furthermore, as $a \to 0$, $h(X) \to -\infty$, so differential entropy is not as well-behaved as its discrete counterpart.

Another important case is the normal (or Gaussian) distribution. It is not hard to show that the differential entropy for a normally distributed random variable X_G with variance σ^2 is

$$h(X_G) = \frac{1}{2} \log 2\pi e \sigma^2 \tag{2.6}$$

and it is well known that the normal distribution *maximises* the entropy over all distributions of equal variance (*e.g.* Deco and Obradovic, 1996, p. 20). We shall use this fact in Section 2.1.5 below.

There are further problems with differential entropy besides its potential negativity. In the discrete case, a random variable may be arbitrarily scaled without affecting its entropy (as seen from (2.1) because $H(X)$ is a function of $p(x)$ alone). Differential entropy, however,

is not invariant under scaling. It is straightforward to show (*e.g.* Cover and Thomas, 1991, p. 233) that

$$h(aX) = h(X) + \log|a|. \tag{2.7}$$

This property extends to vector-valued random variables, so that for a transformation matrix $\mathbf{A}$ of full rank,

$$h(\mathbf{A}\mathbf{x}) = h(\mathbf{x}) + \log|\det \mathbf{A}|.$$

In particular we note that the differential entropy of $\mathbf{x}$ is conserved by the transformation $\mathbf{A}$ if its determinant has magnitude one, or in other words if the mapping is *volume-conserving*.

The other measures of entropy that we have seen for discrete random variables extend as one would expect to the continuous case, and the good news is that relative entropy and mutual information keep the same properties, in particular their non-negativity, in the continuous domain.

2.1.5 Negentropy

The scaling properties of differential entropy mean that comparisons between entropies of continuous-valued variables should be made only with care. One method is to use the *standardised negentropy*, defined as

$$j(X) = -h(X) + h(X_G)$$

where X_G is a normally distributed random variable with the same variance σ^2 as X. Using (2.6) we may therefore write the negentropy as

$$j(X) = -h(X) + \log\Big((2\pi e)^{1/2}\sigma\Big).$$

Using (2.7), we see that

$$\begin{aligned} j(aX) &= -h(X) - \log|a| + \log\Big((2\pi e)^{1/2}|a|\,\sigma\Big) \\ &= j(X) \end{aligned}$$

and so negentropy is scale-invariant. At the same time, since the normal distribution maximises entropy for a fixed variance, it is guaranteed to be non-negative. These properties extend straightforwardly to the multivariate case (Comon, 1994).

Another, equivalent, means of making entropy comparisons significant, and one that will be used later, is simply to scale variables to have equal variance before calculating their differential entropy.

2.2 Modelling terminology

Before moving on to discuss the properties of modelling systems, the terminology that will be used should be clarified. The input to a system is also referred to as the *environment*, or just the *data*. The code-generating process itself is variously called a *model*, *system*, or *network*, and its output is also referred to as a *representation* or *code*. This notation is summarised in Figure 2.2. A slight ambiguity results because the concept of a 'model' exists on two levels — that of the processing system itself, and its internal state — but the meaning should be clear from the context.

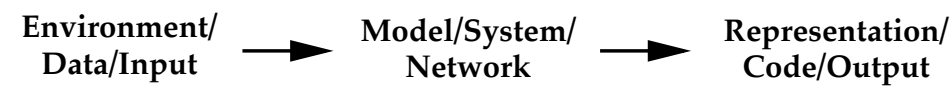


Figure 2.2: A summary of the terms used in this work to describe modelling systems, showing the various descriptions given to the input, the system itself, and its output.

2.3 What makes a good model?

Since we wish to evaluate models, we need to have some idea of what it is that makes one model of the world better than another. Possible ways to model images include, for example, creating a set of pixel values, a frequency-based representation composed of Fourier coefficients, or a grid of binary digits representing the location of edges. Given a number of models, how are we to assess their relative merits? We may identify several factors that could be helpful:

Completeness. How well is the input represented by the model? If the model is complete (or lossless), *all* aspects of the input are represented, allowing complete reconstruction of the original from the model's representation. When this is not possible, the model is *lossy*. This may be caused by noise or imprecision within the processing system, or by the model having insufficient expressive power to fully represent the input. The latter case need not be a problem if the information lost is not useful for subsequent processing, for example when it is due only to 'sensor noise' on the inputs.

Simplicity. There may be any number of models that are complete, or that come as close as each other to being complete. In such circumstances, which should we favour? The principle of Occam's razor tells us to choose the simplest. In determining what we mean by the simplicity of a model there are two different (and conflicting) factors to consider: the complexity of the *representation* (the number of code elements produced by the model, for example), and the complexity of the model itself (perhaps measured by the amount of processing required to compute the code, or to attempt a reconstruction from it). How these are balanced will depend upon the relative amounts of transmission bandwidth versus processing power available to the system. There is also often a trade-off between simplicity and completeness: sometimes we may wish to sacrifice the ability to represent the input fully in order to keep the representation simple. This is the approach taken by lossy compression algorithms.

Conformity. Good models correspond closely with reality. We would expect, for example, a high-level model of a visual environment to contain some notion of *objects*. While it appears difficult to specify exactly how the model should conform with reality,[2] we may be able to measure its success by the degree to which it is able to *generalise*, *i.e.* to model new data from the same environment. A system that merely stored all past data, for example, is likely to perform particularly badly in this respect.

Fitness for purpose. A model is typically only a means to an end, a basis for describing the world or deciding on a course of action. Therefore its value must largely be determined

[2] Any determined attempt to do so is likely to lead to a deep philosophical discussion about the nature of reality, something that is *definitely* beyond the scope of this work!

by its usefulness in performing this higher goal. Coding an image in terms of edges, for example, may be sufficient if the system as a whole is required to find the boundaries of objects, but is not a good choice if the overall aim is to recognise different textures. This idea applies even when the model is complete — there is little point in using the Fourier transform of an image, for example, if the system must extract positional information from the data.

The above descriptions give a qualitative idea of what is desired of a good model, but do not allow us to quantify our success in these areas. However, concepts from information theory can help us here. We shall assume for simplicity that we are dealing with *discrete* systems, but the ideas are equally applicable to the continuous case, providing that we take into account the scaling problems set out in Sections 2.1.4 and 2.1.5.

Completeness. The completeness of a model may be measured in terms of the *mutual information* $I(X;A)$ between the raw data X and the model's representation A (Section 2.1.3). We shall refer to this as the *information transfer* (from input to output) of the modelling system. If we assume, as we shall in the remainder of this work, that the system itself is noise-free, then any information at the output can *only* have come from the input, *i.e.* $H(A|X) = 0$. Hence, from (2.5), we see that $I(X;A) = H(A)$, and so maximising information transfer can be achieved simply by maximising the output entropy. This is a technique known as 'infomax' (Linsker, 1988, 1992; Bell and Sejnowski, 1995).

Simplicity. The simplicity of the *representation* produced by a model can be measured in terms of its entropy. The simplicity of the model itself can, subject to finding a suitable way of describing it, be measured by the length (*i.e.* number of bits) of such a description. Combining the two into a single expression forms the basis for the *minimum description length* principle (Rissanen, 1978, 1985). This introduces a problem of how to model the model, and finding a suitable way to describe a model can be tricky. In a neural network this is typically done using a measure of the number, and possibly size, of the weights, and forms the theoretical justification for the technique of weight-elimination (Weigend *et al.*, 1991). We shall not attempt to quantify model complexity in this work, concentrating instead on the entropy of the representation: the lower the entropy, the simpler the representation. We have just seen, however, that for a system to generate complete models, it should *maximise* the entropy of its output, so the conflict between completeness and simplicity that we noted above is also clearly apparent in terms of entropy. A solution, which need not compromise the completeness of the model, is to consider the entropy of each element of the output *separately*, and attempt to minimise their sum. The minimum, for a fixed overall output entropy, is obtained when each of the elements is statistically independent. Representations that achieve this are known as *minimum entropy codes*, and we shall discuss these in greater detail below.

Conformity. If the signals taken in by our senses had maximum entropy, they would appear to be random noise, like 'snow' on a television screen; we would be unable to generalise, and thus unable to make sense of the world and make predictions about it. It is

in fact *dependencies* that form the basis for objects, concepts, and so on, and the process that we call 'understanding' is, essentially, the act of discovering these dependencies through experience. In saying that we wish a model to 'conform with reality,' we are asking that it builds a representation based on the dependencies of its environment. If the model's representation has independent elements, then it must necessarily have detected and removed the dependencies in its input to achieve this. Therefore the sum of output entropies, as a measure of independence, can again be used to gauge the model's success in this regard.

Fitness for purpose. A limitation of information theory is that it cannot, by itself, distinguish between useful and useless information, *i.e.* signal and noise. For a learning system to perform a particular task, external influences are required to indicate which information is important, and how well the system is performing. In nature, this is provided ultimately by evolutionary pressures, and more immediately by sensations such as pleasure and pain. For an artificial system, it is up to us to provide similar forms of pressure, typically in the form of *supervised* or *reinforcement* learning. While information theory cannot help directly in tailoring a system to a particular task, it can help at a preprocessing stage, which is primarily what we shall consider in this work — if a preprocessor can produce a representation whose elements are independent, then a higher, task-specific, level of processing need only *select* those elements that are of interest. Yet again, an independent (minimum entropy) code appears to be important. This point is discussed in more detail in the next section.

From four properties of 'good' models, we have produced just two information-theoretic goals: the desire to have models that are complete means we should attempt to maximise total output entropy, and the properties of simplicity, conformity and usefulness can all be captured by minimising mutual information between the separate elements of the output. In the following sections we shall look at some of these issues in more detail.

2.4 Noise as information

A property of information theory that is slightly counter-intuitive is that the more random a signal is, the higher its entropy, and hence the higher its potential information content. It is common to assume that 'noise' is *entirely* random, and yet our normal notion of noise is as something that carries zero information. The way out of this apparent contradiction is that what we commonly refer to as 'information' is in fact the *mutual* information between some received signal and a subject of interest. Noise *is* information, but it is information about something in which we have no interest.

We therefore see that the concepts of signal and noise can be task-specific: the variation between different people's speech is noise to a speech-recognition system, for example, but is precisely the signal required by a speaker-identification system; or similarly much of the information rejected as noise by a visual edge-detector forms the signal for a texture-segmentation system.

In this thesis, we shall consider only systems that are *unsupervised*, *i.e.* where there is no external agent to say which information is interesting and which is not. This is why we

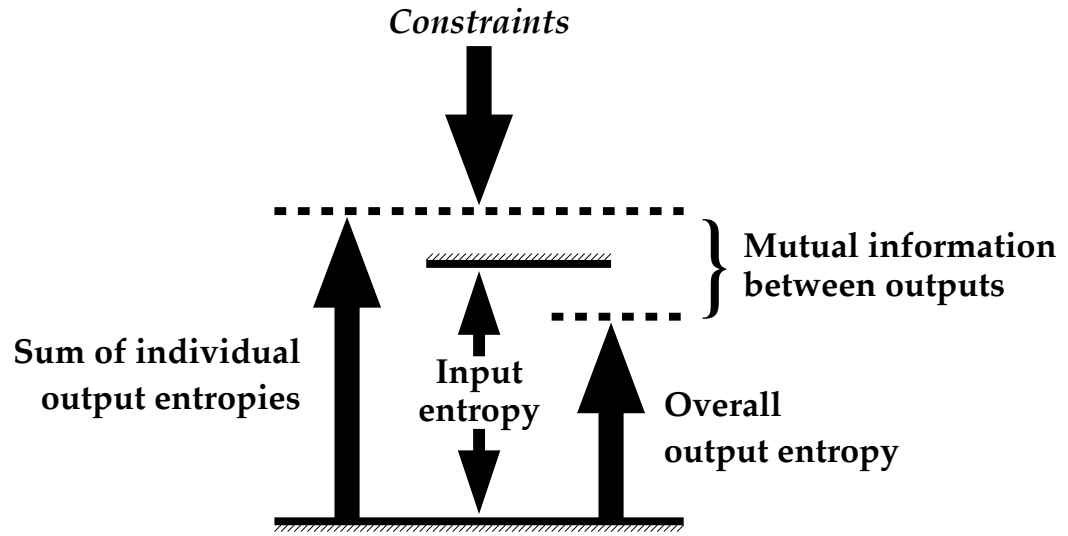


Figure 2.3: A representation of the relationship between several measures of entropy in a (discrete) information-processing system. The input entropy is fixed by the environment, and (if the system is assumed noiseless) provides an upper bound on the overall output entropy. The sum of the entropies of each of the output values may be above or below the level of input entropy, but will always be at least as great as the overall output entropy, with any difference accounted for by mutual information between the outputs. The downward arrow represents some means of bounding the individual output entropies by the application of constraints. In an ideal system, both dotted lines will converge on the solid line representing input entropy, giving a complete factorial code.

must aim to capture *all* information, and leave it to later, task-specific processing to decide which to use and which to ignore. As mentioned above, this separation is made as easy as possible by building an independent representation, since signal and noise are, by definition, independent of each other.[3] The information in a speech signal relating to the words spoken, for example, is largely independent (except at a very high level) of the information relating to the person who is speaking them.

2.5 Entropy in an information-processing system

The entropic quantities we have identified as important in a modelling system are depicted in Figure 2.3. We shall again assume that all variables are discrete. Denoting the input by the vector $\mathbf{x}$, the overall input entropy $H(\mathbf{x})$ provides a fixed reference, shown in the centre of the diagram. Denoting the output by the vector $\mathbf{a} = (a_1, \ldots, a_n)^{\mathrm{T}}$, the information in the output that relates to the input is given by

$$I(\mathbf{a}; \mathbf{x}) = H(\mathbf{a}) - H(\mathbf{a}|\mathbf{x}).$$

But, as we have already noted, if the system itself is assumed to be noiseless, then $H(\mathbf{a}|\mathbf{x}) = 0$, and

$$I(\mathbf{a}; \mathbf{x}) = H(\mathbf{a}).$$

[3] If this were not the case then the 'noise' should be retained because it must contain relevant information that could be useful in subsequent processing.

Maximising the overall output entropy is therefore equivalent to maximising information transfer.

The overall output entropy is given by the sums of the individual elements' entropies minus any mutual information between them. This expression can be obtained by combining (2.3) and (2.5) to give

$$H(\mathbf{a}) = \sum_{i=1}^{n} H(a_i) - \sum_{i=1}^{n} I(a_i; a_{i-1}, \ldots, a_1).$$

We are already attempting to maximise $H(\mathbf{a})$. This can simultaneously *minimise* the mutual information term if $\sum_{i=1}^{n} H(a_i)$ is suitably constrained. In other words, if some upper bound or downward pressure can be placed on the sum of output entropies, then maximising overall output entropy will 'squeeze out' the mutual information, as depicted in Figure 2.3. In the work of Bell and Sejnowski (1995), for example, a fixed upper bound on individual entropies is introduced by limiting the range of values than outputs can take on. In Chapter 5 we shall look at a number of other constraints that either introduce bounds or put downward pressure on output entropies.

If we successfully squeeze out all mutual information between outputs, the redundancy of the code becomes zero and

$$H(\mathbf{a}) = \sum_{i=1}^{n} H(a_i).$$

Accordingly, we find that

$$p(\mathbf{a}) = \prod_{i=1}^{n} p(a_i)$$

and so the code elements are statistically independent. We have generated a *factorial* or *minimum entropy* code.

There is, unsurprisingly, some confusion over whether entropy should be maximised or minimised, since some work advocates one, and some the other. The answer, we know see, is that we should do both! The obvious contradiction is resolved by observing that we are referring to two different quantities — we wish to maximise mutual information between input and output, but minimise mutual information between the individual output values.

The fundamental nature of the problem of generating independent codes means that there is no shortage of methods that attempt to solve it. The name used generally depends upon the field in which the technique originated. In the statistical community, for example, the problem is usually known as *factor analysis* (Halman, 1976; Watanabe, 1985), in signal processing as *independent component analysis* (Jutten and Herault, 1991; Comon, 1994), in geophysics as *minimum entropy deconvolution* (Wiggins, 1978; González *et al.*, 1995; Satorius and Mulligan, 1992), and in physiology as *redundancy reduction* (Attneave, 1954; Barlow, 1959, 1961; Atick, 1992) or *minimum entropy coding* (Barlow, 1989; Barlow *et al.*, 1989). In the somewhat eclectic field of neural networks, all of these terms may be found!

2.6 Low entropy codes

The title of this thesis avoids the term 'minimum entropy,' and uses instead the phrase 'low entropy coding.' This is in recognition of the fact that for most real data, and with simple

models such as will be used here, there is no hope of producing fully independent codes.

The aim, in essence, is to perform low-level *feature detection*. 'Features' represent the dependencies in a set of data. Visual features such as lines, for example, account for the fact that the intensity at one point in an image tends to be highly dependent on the intensities of those around it. At the same time we expect the dependency *between* features to be low (otherwise they should be amalgamated into one or more larger features).

A feature-based code should therefore have low entropy. It is unlikely, however, to have *minimum* entropy, because dependencies often extend to very high levels of processing. In a typical visual environment, for example, small edge segments are interdependent because they combine to form straight or curved lines, which themselves combine to form the boundaries of distinct objects. At a higher level still, objects themselves have complex dependencies, since, for example, solid objects are required to support one another, but cannot occupy the same region of space.

As another example, word-sounds, or phones, are useful low-level features of speech, but are not fully independent because they combine to form words, which are joined to form phrases, sentences, entire passages, and so on. The dependencies even extend to the *meaning* that the speech is intended to confer.

The generation of low entropy codes is a valid goal, because redundancy reduction is a useful step towards a full 'understanding' of the data, even if it leaves further dependencies, or what Barlow (1989) terms 'suspicious coincidences,' to be discovered in subsequent processing.

2.7 Local, distributed and sparse coding

Having said that we wish to generate codes with low redundancy, it is useful to have some idea of what form such codes might take. There has in particular been much debate as to the nature of codes in the brain, and correspondingly about ways in which these might be reproduced in artificial neural networks.

At one end of the spectrum come *local* codes, where only one element (or an isolated cluster of elements) of the code are 'active' (non-zero, for example) for any particular pattern. The advantage of such a representation is that it is very easy to tell which pattern occurred, but the principal disadvantage is that for any reasonably complex environment, a huge number of elements are required if the code is to be anywhere near complete. It is also difficult to generalise between patterns that are classed as distinct under a local code. A local code cannot have minimum entropy because the fact that one element's being active implies that all others are inactive shows there is a dependency between them.

At the other extreme come *fully distributed* codes, where all elements are used to represent each pattern. These tend to be much more compact in terms of the number of elements required, but it is no longer possible to determine directly which particular pattern is being represented, implying a need for further processing. There is is general no guarantee that there will not be strong dependencies between the code elements, but a fully distributed code *could* have minimum entropy in certain cases.

In between local and distributed codes we have the situation where only a fraction of

the code elements are actively used to represent a typical pattern. Such codes are often termed *sparse*. They have the potential to give 'best of both worlds' solutions: codes that are not hampered by a combinatorial explosion in size, but are nevertheless able to represent separate components of the data directly.

A sparse code may also have very low redundancy, providing that the underlying independent causes of the data it encodes are themselves sparse. This seems to be a reasonable hypothesis in many cases. If, for example, we say that to a first approximation individual objects form the independent components in images, then an object-based code will be both independent and sparse, since there are many possible objects, but only a few appear in a typical image. Similarly, by making the simplifying assumption that words are the independent components of sentences, then a word-based representation of a sentence will also be both independent and sparse, since of the many possible words, only a few appear in any one sentence.

Further discussions about the relative merits of the different types of representation are given by Thorpe (1995) and Zemel (1993). The usefulness of sparse coding, and its validity as a model of brain function, have been examined by several authors (Barlow, 1983, 1994; Földiák, 1990, 1992; Földiák and Young, 1995; Field, 1987, 1994; Olshausen and Field, 1996a; Rolls and Tovee, 1995). We shall return to the issue of sparse coding in Chapter 5.

2.8 Why the modelling problem is hard

Modelling is a problem in which one attempts to infer *causes* from observations of the *effects* that they produce. This is often known as an *inverse* problem, as distinct from the *forward* problem of determining effects from causes.

Forward problems are (relatively) easy to address. In a deterministic system, there is a single mapping from any set of causes to their effects. This uniqueness is a key condition for a *well-posed* problem (Hadamard, 1923).[4] Difficulties only arise when deciding how to make simplifying approximations to the mapping and how to deal with incomplete knowledge of the causes.

Inverse problems, by contrast, are typically *ill-posed* (Bertero *et al.*, 1988): there are often many possible explanations for a particular observation, so the solution is no longer unique. In such circumstances, it becomes necessary to consider which explanations are the most *likely*, thus introducing the extra complexity of a probabilistic element that was not present in the equivalent forward problem.

Computer graphics is an example of a forward problem: one takes a description of some objects, their positions, the lighting and so on, and calculates the patterns of light that result. The corresponding *inverse* problem is computer vision, where one takes a pattern of light and attempts to explain it in terms of the underlying causes. The relative maturity of the two fields gives a fairly good indication of their relative complexities: while computer graphics is able to produce realistic pictures and animations from world models, computer vision is still nowhere near the stage of being able to reverse the process and use those images to regenerate the underlying model.

[4] Under the original definition, there is also a requirement for continuity which we shall not consider here.

2.9 Some basic modelling problems are NP-complete

Another way of characterising the difficulty of modelling problems is in terms of their computational complexity. Many can be cast as *combinatorial optimisation problems*, where the time taken to find the optimal solutions increases exponentially with the size of the problem.

Algorithms that run on a serial computer in a time equal to some polynomial in the size of the problem are commonly placed in a class P. Another class, NP, contains those whose solutions can be *verified* in polynomial time, which certainly includes all of P. It is believed, although not proven, that NP is a strict superset of P, and that there are therefore problems in NP but not in P that cannot be solved in polynomial time on any serial computer.

The hardest of the problems in NP are known as NP-complete, and it has been shown (Cook, 1971) that if any one of these were solved in polynomial time, then *all* NP problems could be solved in polynomial time.

Many of the problems that have been shown to be NP-complete are what we are terming here 'modelling problems.' One example is the set basis problem (Stockmeyer, 1975; Garey and Johnson, 1979, p. 222). It can stated as:

> Consider a set X of binary strings of a finite length m, and a positive integer $n \leq |X|$. The problem is to find a set W of binary strings of length m with $|W| = n$ such that, for each $x \in X$, there is a subset of W whose logical OR is exactly x.

In fact we shall later tackle a problem that is exactly of this form (Section 4.12.1). Another NP-complete modelling problem is a form of *clustering* (Garey and Johnson, 1979, p. 281):

> Consider a finite set X, an integral distance $d(x, y)$ for each pair x, $y \in X$, and two positive integers K and B. The problem is to determine a partition of X into disjoint sets $X_1, X_2, \ldots, X_K$ such that, for $1 \leq i \leq K$ and all pairs x, $y \in X_i$, $d(x, y) \leq B$.

The NP-completeness of modelling tasks such as these is in one sense disheartening because it implies that we cannot in general obtain complete, tractable solutions to these problems. At the same time, the absence of a tractable general solution justifies approaches that are to some extent task-specific, rely on heuristics, or are not guaranteed to give optimal results.

2.10 Discussion

This chapter has attempted to demonstrate first that modelling problems can be set in the context of information theory, and second that they are difficult to solve. A good general-purpose modelling system will generate representations that are maximally *complete* (by maximising mutual information between input and output) while also being maximally *efficient* (by minimising mutual information between code elements).

Systems that attempt to build models and develop codes using the raw data alone are often termed *unsupervised* or *self-organising* systems. We shall look at some of these in the next chapter. Although most are not based directly on information theory, they all to some extent embody the ideas of completeness and efficiency.

Chapter 3

Self-Organisation and Unsupervised Learning

This chapter gives an overview of some of the techniques already in existence for performing unsupervised learning. No attempt will be made to cover the whole range of this enormous field. We shall instead concentrate primarily on those techniques that provide the basis for work set out in subsequent chapters. This means that the main emphasis is on *neural* techniques in particular. Broader and more detailed surveys of unsupervised techniques for neural networks are given by Becker (1989), Haykin (1994), Ripley (1996) and Hertz *et al.* (1991), and the classic work by Duda and Hart (1973) is a good starting point for more general statistical methods.

Although most of the methods we shall look at do not use directly the information-theoretic ideas set out in the previous chapter, the aim is the same — to automatically produce a good model of the data. The language used, however, is often slightly different. It is common to think of the original data as residing in a space, known as the *data space*. The process can be thought of as transforming this into a *feature space* which, we hope, will be more amenable to subsequent processing. Where the feature space has fewer dimensions that the original data space, the process is known as *dimensionality reduction*. We shall begin with a well-known technique for performing such a mapping.

3.1 Principal component analysis

One of the longest-standing and most popular statistical methods of dimensionality reduction is principal component analysis (also known as the Karhunen-Loève transformation) first introduced by Pearson (1901). The method is described in depth by Jolliffe (1986). In the presentation given here, we shall look at certain aspects of the method in detail because closely related mathematical expressions will appear in the description of the network in Chapter 4.

The aim of PCA is to map vectors $\mathbf{x}$ in an m-dimensional space onto vectors $\mathbf{a}$ in an n-dimensional space, with $n \leq m$. Often the inequality is strict (producing a subspace projection of the original points) since PCA is a useful method for dimensionality reduction. When $n = m$, the process forms the basis for *sphering*, a method that will be discussed further in

Section 3.8.1.

For simplicity we shall assume that the data have zero mean:

$$\mathcal{E}[\mathbf{x}] = \mathbf{0}.$$

If this were not the case, we could simply subtract the mean from the data before proceeding. Alternatively, it is quite possible to perform a similar analysis without making this assumption (*e.g.* Bishop, 1995, p. 310).

Any point $\mathbf{x}$ in the original m-dimensional space may be represented as a linear combination of (any) m orthonormal vectors $\mathbf{u}_i$:

$$\mathbf{x} = \sum_{i=1}^{m} a_i \mathbf{u}_i. \tag{3.1}$$

Since the vectors $\mathbf{u}_i$ are orthonormal, they obey the relation

$$\mathbf{u}_i^{\mathrm{T}} \mathbf{u}_j = \delta_{ij} \tag{3.2}$$

and hence, by premultiplying (3.1) by $\mathbf{u}_i^{\mathrm{T}}$, we find that the coefficients a_i are given by

$$a_i = \mathbf{u}_i^{\mathrm{T}} \mathbf{x}. \tag{3.3}$$

Now we restrict the representation in (3.1) to a linear combination of just n orthonormal vectors, with $n < m$. Since we have restricted the number of degrees of freedom, it will no longer be possible in the general case to represent all of the data exactly — the approximation $\widetilde{\mathbf{x}}$ to the original is given by

$$\widetilde{\mathbf{x}} = \sum_{i=1}^{n} a_i \mathbf{u}_i. \tag{3.4}$$

The aim now is to choose the coefficients a_i and basis vectors $\mathbf{u}_i$ such that they give the minimum possible sum-squared error (SSE) between the original vectors $\mathbf{x}$ and our reconstructions $\widetilde{\mathbf{x}}$. This gives an error function

$$E_r = ||\mathbf{x} - \widetilde{\mathbf{x}}||^2 \tag{3.5}$$

which should be minimised by our selection of coefficients a_i (for a particular pattern $\mathbf{x}$ and set of basis vectors $\mathbf{u}_i$). Similarly, the mean error over the entire data set is given by

$$E_R = \mathcal{E}[||\mathbf{x} - \widetilde{\mathbf{x}}||^2] \tag{3.6}$$

which should be minimised by our selection of basis vectors $\mathbf{u}_i$. The choice of sum-squared difference as the error metric is a natural one; some theoretical justifications for its use are given in Chapter 4.

Minimisation of (3.5) is easy. The function is quadratic and has a single, global minimum. The derivative of E_r with respect to a_i is given by

$$\begin{aligned} \frac{\partial E_r}{\partial a_i} &= -2\,(\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}} \mathbf{u}_i \\ &= -2\,(\mathbf{x} - \sum_{k=1}^{n} a_k \mathbf{u}_k)^{\mathrm{T}} \mathbf{u}_i. \end{aligned}$$

Setting this to zero, we find that

$$\mathbf{x}^{\mathrm{T}}\mathbf{u}_i - \sum_{k=1}^{n} a_k \mathbf{u}_k^{\mathrm{T}}\mathbf{u}_i = 0$$

which by application of the orthonormality relation (3.2) reduces directly to (3.3). Thus we see that the optimal coefficients are the same whether the orthonormal basis is complete or partial. Given this, we can substitute (3.1) and (3.4) into (3.6) to give

$$\begin{aligned} E_R &= \mathcal{E}\left[\left\| \sum_{i=n+1}^{m} a_i \mathbf{u}_i \right\|^2\right] \\ &= \sum_{i=n+1}^{m} \mathcal{E}[a_i^2]. \end{aligned}$$

This is simply the sum of the variances[1] for the projection coefficients in the directions that have been discarded in our approximation (3.4). What this tells us is that in order to minimise E_R we must find and discard those directions in which the data has minimum variance or, equivalently, use only the directions of greatest variance.

The task of finding the optimal directions $\mathbf{u}_i$ for any particular value of n is underconstrained. Any orthogonal basis that spans the n-dimensional space containing the greatest variance in the data (often called the *principal subspace*) will suffice. If however we retain the right to choose n *after* the analysis, then the directions $\mathbf{u}_i$ must be ordered such that the first is in the direction of greatest variance, the second the next greatest (while being orthogonal to the first) and so on. These directions are known as the first, second, *etc.*, principal components.

By substituting the values of a_i given by (3.3) we note that the variances of the coefficients a_i, which we shall call σ_i^2, are

$$\begin{aligned} \sigma_i^2 &= \mathcal{E}[a_i^2] \\ &= \mathcal{E}[(\mathbf{u}_i^{\mathrm{T}}\mathbf{x})(\mathbf{x}^{\mathrm{T}}\mathbf{u}_i)] \\ &= \mathbf{u}_i^{\mathrm{T}}\mathcal{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}]\mathbf{u}_i \\ &= \mathbf{u}_i^{\mathrm{T}}\mathbf{\Sigma}\mathbf{u}_i \end{aligned} \tag{3.7}$$

where $\mathbf{\Sigma}$ is the covariance matrix, defined (because $\mathbf{x}$ has zero mean) as the expectation of the outer product of $\mathbf{x}$ with itself:

$$\mathbf{\Sigma} = \mathcal{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}].$$

From this definition, the covariance matrix is clearly real and symmetric. It is a well-known theorem of linear algebra (*e.g.* Strang, 1988, p. 296) that a real symmetric matrix such as $\mathbf{\Sigma}$ can be factored as

$$\mathbf{\Sigma} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathrm{T}}$$

where $\mathbf{Q}$ is a matrix whose columns $\mathbf{q}_i$ are the orthonormal eigenvectors of $\mathbf{\Sigma}$ and $\mathbf{\Lambda}$ is a diagonal matrix where the diagonal elements λ_i are the corresponding eigenvalues. We

[1] Because we have assumed that the data have zero mean, any projection will also have zero mean, so that the variance of a coefficient a_i over the data is simply its mean-square value.

may assume without loss of generality that the columns of $\mathbf{Q}$ have been permuted such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$.

We can now write (3.7) as

$$\sigma_i^2 = \mathbf{u}_i^{\mathrm{T}} \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^{\mathrm{T}} \mathbf{u}_i \tag{3.8}$$

and note that *if* it happened that for some i, $\mathbf{u}_i = \pm \mathbf{q}_i$, then (3.8) would reduce, by application of the orthonormality relation (3.2), to

$$\sigma_i^2 = \lambda_i$$

for that value of i.

The directions $\mathbf{u}_i$ we seek are the stationary points of (3.7), subject to the constraint that $\mathbf{u_i}$ has unit length, *i.e.* that $\mathbf{u}_i^{\mathrm{T}} \mathbf{u}_i - 1 = 0$. Using a Lagrange multiplier ρ_i, these can be expressed as the stationary points of

$$L(\mathbf{u}_i, \rho_i) = \mathbf{u}_i^{\mathrm{T}} \mathbf{\Sigma} \mathbf{u}_i - \rho_i (\mathbf{u}_i^{\mathrm{T}} \mathbf{u}_i - 1).$$

Differentiating with respect to $\mathbf{u}_i$ and setting the result to zero yields

$$\mathbf{\Sigma} \mathbf{u}_i - \rho_i \mathbf{u}_i = 0.$$

But this is precisely the equation satisfied by the eigenvectors $\mathbf{q}_i$ of $\mathbf{\Sigma}$. Also, since we put the eigenvectors in order of decreasing eigenvalue and require the directions $\mathbf{u}_i$ to be in order of decreasing variance, we find that, for all i, $\mathbf{u}_i = \pm \mathbf{q}_i$ and $\sigma_i^2 = \lambda_i = \rho_i$.

The principal components of a data set are therefore given by the eigenvectors of its covariance matrix, taken in order of decreasing eigenvalue. It is not hard to see from (3.8) that transformation of the data into the principal component coefficients produces a representation with a diagonal covariance matrix, and the coefficients are therefore *decorrelated*, a property that we shall discuss further in Section 3.3.

There is a useful algorithm, known as *singular value decomposition*, for performing PCA (Press *et al.*, 1986, Chap. 2). This method is commonly used, and is practical and efficient, but there has also been considerable interest in the ability of neural models to discover principal components. Some of these models are discussed in the following section.

3.2 Neural networks as principal component analysers

There are close links between many unsupervised neural networks and the method of PCA outlined above. Indeed several neural models have been designed explicitly for their ability to find principal components. Some of these are discussed in this section, as a prelude to seeing how we might develop networks that go beyond the limited capabilities of PCA as a pattern-processing method. Fuller reviews may be found in Oja (1992), Haykin (1994), Baldi and Hornik (1995) and, from an information theoretic viewpoint, Plumbley (1991).

3.2.1 A single linear neuron with Hebbian learning

The linear model of a neuron i is simply one that takes its m inputs x_j, multiplies them by the strength of the 'connection weight' w_{ij} from each, and sums the result to give an output a_i:

$$a_i = \sum_{j=1}^{m} w_{ij} x_j$$

or equivalently in vector notation

$$a_i = \mathbf{w}_i^{\mathrm{T}} \mathbf{x}. \tag{3.9}$$

The neuron (henceforth less specifically termed a 'unit') 'learns' by changing its weight values. In self-organised networks, the rules governing weight changes are typically based on the 'Hebb synapse':

> 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 (Hebb, 1949, p. 62).

The idea is attractive both because of its simplicity (being based purely on a correlation between firing patterns) and also because of its *locality* (all the information required to change the strength of the coupling between two cells is already available at that point without requiring any additional mechanism to deliver it). Although the Hebb synapse was only a hypothesis in 1949, it has since been supported by considerable neurophysiological evidence (see Brown *et al.*, 1990, for a review). Its simplest mathematical interpretation is as

$$\delta w_{ij} = \eta x_j(t) a_i(t) \delta t \tag{3.10}$$

where δw_{ij} is the change (in time δt) to the weight of the connection from unit ('cell') i to unit j, $x_j(t)$ and $a_i(t)$ are the outputs at time t of units j and i respectively and η is the rate at which learning occurs. In a continuous-time system, we could integrate (3.10) to determine weight values, but many neural networks, including those presented in this thesis, use a discrete-time model, so that we may rewrite (3.10) in the form

$$\Delta w_{ij} = \eta x_j a_i \tag{3.11}$$

where we have assumed unit time steps, and for simplicity removed the explicit time dependency of the signals. Combining all inputs x_j in a single equation, we may write

$$\Delta \mathbf{w}_i = \eta a_i \mathbf{x}. \tag{3.12}$$

The use of this rule is normally known as *Hebbian learning*. If we consider a single linear unit with output a and weights $\mathbf{w}$ (Figure 3.1), a reasonable goal is for the unit to differentiate its inputs as much as possible, or in other words to maximise its output variance σ^2. If, as in Section 3.1, we assume data $\mathbf{x}$ with zero mean, we have

$$\sigma^2 = \mathcal{E}[a^2] = \mathcal{E}[(\mathbf{w}^{\mathrm{T}}\mathbf{x})^2] = \mathbf{w}^{\mathrm{T}} \boldsymbol{\Sigma} \mathbf{w}$$

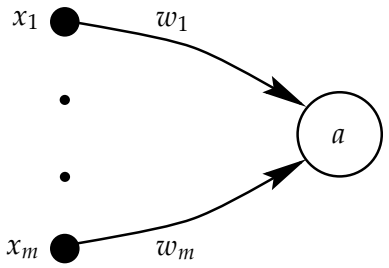


Figure 3.1: A single-unit 'network.' In this and subsequent network diagrams, a black circle represents an input signal, and a large white circle a processing unit, labelled by its output value. Signals are propagated along the connecting lines in the directions shown, with their values multiplied by the associated weight. In this example, since all connections are in one direction, the network is termed *feedforward*. Black dots indicate a number of similar elements that have been omitted from the diagram for simplicity.

where $\mathbf{\Sigma}$ is the covariance matrix of the data, as previously. We also see that

$$\frac{\partial \sigma^2}{\partial \mathbf{w}} = 2\mathcal{E}[a\mathbf{x}]$$

which is precisely the direction taken (on average, assuming small steps) by (3.12). We therefore find that, for a single linear unit and zero-mean data, Hebbian learning attempts to maximise the unit's output variance. Without some constraint, this will result in the weight values growing towards infinity.

If the problem is modified by adding the constraint that $||\mathbf{w}|| = 1$, then we already know from Section 3.1 that the optimal solution occurs when $\mathbf{w} = \pm\mathbf{q}_1$ where $\mathbf{q}_1$ is the normalised eigenvector of $\mathbf{\Sigma}$ with the largest eigenvalue. If the normalisation is applied as part of each weight update, as suggested by Oja (1982), we have that

$$w_j(t+1) = \frac{w_j(t) + \eta a(t) x_j(t)}{\sqrt{\sum_{k=1}^{m} [w_k(t) + \eta a(t) x_k(t)]^2}}$$

which, under the assumption that η is small, may be expanded as a power series in η to give

$$w_j(t+1) = w_j(t) + \eta a(t)[x_j(t) - a(t) w_j(t)] + \mathcal{O}(\eta^2)$$

where $\mathcal{O}(\eta^2)$ represents second- and higher-order terms in η. This gives (since η is small) the learning rule

$$\Delta w_j = \eta a(x_j - a w_j) \tag{3.13}$$

often known as 'Oja's rule.' Under reasonable assumptions about the data, it may be shown (Oja, 1982; Oja and Karhunen, 1985) that if the weight vector does not start out perpendicular to $\mathbf{q}_1$ (such directions represent local maxima of variance), and if $\eta \to 0$ at a suitable rate, then $\mathbf{w}$ will indeed always converge to $\pm\mathbf{q_1}$ as desired.

Further analyses of this and other feature extraction algorithms are given by Kuan and Hornik (1991) and Hornik and Kuan (1992). Full proofs of convergence is cases such as this can be somewhat cumbersome, but the common underlying goal is one of achieving 'on-line' learning, where the data points may be considered individually and sequentially. The

techniques are closely related to a stochastic approximation algorithm proposed by Robbins and Monro (1951). In particular, where the learning rule is the negative *gradient* of an error function, it can be shown that an on-line learning process is guaranteed to converge to a minimum providing that the learning rate obeys certain properties (see, for example Bishop, 1995, p. 46). This process is often known as *stochastic gradient descent*, and will be used in the learning rules introduced in Chapter 4.

3.2.2 Networks of PCA units

We have seen that a single Hebbian unit will attempt to maximise its output variance, and that a simple modification to the Hebbian rule will additionally cause the weight vector to converge to unit length, thereby enabling it to find the first principal component of the data. Clearly, if we were to produce a fully-connected feedforward network of many such units, then all would converge to the same solution, so how can we construct a network that extracts more than just the first principal component?

One simple solution makes use of the orthogonality of the principal components. A single unit finds the first principal component of the data — if it subtracts this component from the data and feeds the remainder to a second unit, this will find the first principal component of *its* input, which will in fact be the *second* principal component of the original data. This process may be repeated for as many units as we require principal components. It is implemented simply by modifying the inputs x_j in Oja's rule (3.13) to

$$x'_j(i) = x_j - \sum_{k=1}^{i-1} w_{kj} a_k$$

so that the effective input to the ith unit (for the purpose of learning) is the original input minus the projection onto the subspace defined by all lower-numbered units. This procedure was proposed by Sanger (1989) and is known as the 'generalised Hebbian algorithm.'

An alternative approach is for the units in a network to compete via *lateral inhibition*, *i.e.* adaptive inhibitory connections between the output units of the network that are intended to prevent all units from becoming identical and extracting only the first principal component. One such network was proposed by Földiák (1989) in which the feedforward weights are updated using Oja's rule (3.13) and the lateral weights follow a simple *anti-Hebbian* rule, equivalent to normal Hebbian learning (3.11) but with the sign reversed. Because each unit is connected to every other, this method is more symmetrical than Sanger's essentially hierarchical approach. A hierarchy of units is useful in the particular case of PCA, where there is a strict ordering of directions, but is less applicable to more general forms of feature extraction.

There are in fact a large number of neural PCA algorithms in a very similar vein to those already mentioned proposed by, among others, Oja and Karhunen (1985), Rubner and Tavan (1989), Chauvin (1989b), Kung and Diamantaras (1990), Leen (1991) and Oja (1992). Baldi and Hornik (1995) provide a useful analysis and comparison of these within a common framework.

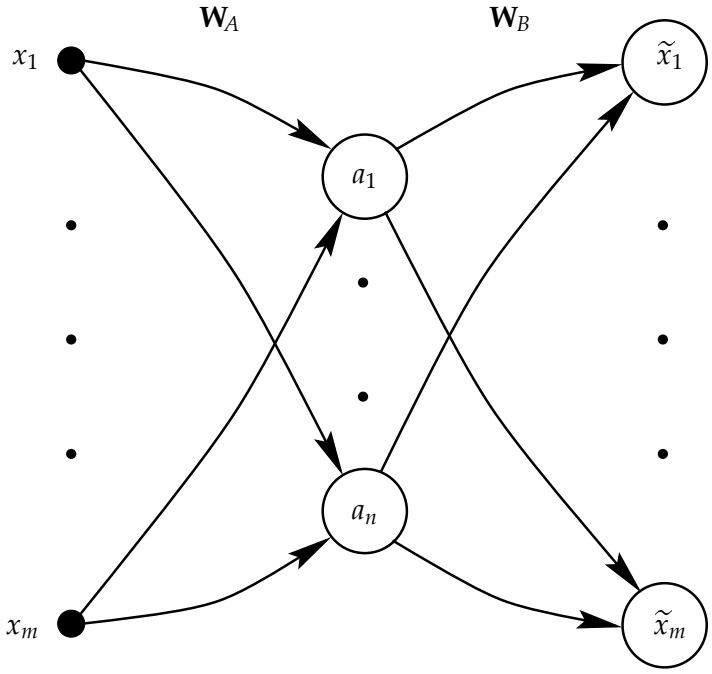


Figure 3.2: A feedforward autoencoder network. The first layer of connection weights are represented by the matrix $\mathbf{W}_A$, and the second by $\mathbf{W}_B$.

3.2.3 The linear feedforward auto-encoder

Another approach to performing PCA with a neural network is to make use of a multi-layer feedforward model commonly known as the multi-layer perceptron (MLP). The MLP is usually used as a *supervised* system, *i.e.* for every training input $\mathbf{x}$ presented to the network, there is an associated 'target' $\mathbf{t}$, with the aim being to minimise some measure of error between $\mathbf{t}$ and the network's actual outputs $\mathbf{y}$. In the case of the auto-encoder, the targets are identical to the inputs ($\mathbf{t} = \mathbf{x}$), and therefore, since the raw data $\mathbf{x}$ form the only input to the system, the network could reasonably be called 'unsupervised.' Because, however, it is produced by a simple modification to a supervised model, the MLP auto-encoder is more commonly known as a 'self-supervised' method.

The simplest form of the network is shown in Figure 3.2, where we have m inputs, one *hidden* layer of n units and m outputs, with $n < m$. By forcing the data through a 'bottleneck' of n units, and attempting to reconstruct the original from this representation alone, the hope is that the intermediate representation $\mathbf{a}$ will give a compressed version of the data.

Assuming linear units (3.9), and representing the first and second layers of weights in matrix form as $\mathbf{W}_A$ and $\mathbf{W}_B$ respectively, we see that

$$\mathbf{a} = \mathbf{W}_A \, \mathbf{x}$$

and

$$\widetilde{\mathbf{x}} = \mathbf{W}_B \, \mathbf{a} = \mathbf{W}_B \mathbf{W}_A \, \mathbf{x}.$$

We have already noted (in Section 3.1) that under the standard mean-square-error (MSE) metric (3.6), and assuming zero-mean data, the optimum combined mapping $\mathbf{W}_B \mathbf{W}_A$ is one that forms an orthogonal projection onto the (m-dimensional) principal subspace of the data.

It can be shown (Bourlard and Kamp, 1988; Baldi and Hornik, 1989) that the network of Figure 3.2, trained using the standard back-propagation learning rule (Rumelhart *et al.*, 1986) will indeed produce such a projection.[2] While the optimal mapping from input to output (taken as a whole) is unique, the solution in terms of individual weight vectors, and hence of the coding produced in the hidden layer, is highly under-constrained. The optimal weight values may be expressed as

$$\begin{aligned} \mathbf{W}_A &= \mathbf{C}\mathbf{Q}_n^{\mathrm{T}} \\ \mathbf{W}_B &= \mathbf{Q}_n\mathbf{C}^{-1} \end{aligned} \tag{3.14}$$

where the columns of $\mathbf{Q}_n$ represent the first n principal components of the data, and $\mathbf{C}$ is *any* invertible $n \times n$ matrix. In particular we note that the weight vectors responsible for producing the internal representation $\mathbf{a}$ (the rows of $\mathbf{W}_A$) need be neither normal nor mutually orthogonal. The lack of such constraints could be seen as a drawback, but we shall see later (Chapter 5) that this leaves the door open to other, more useful constraints.

We may also note from (3.14), however, that in the special case where $\mathbf{C} = \mathbf{I}$, the weights do indeed represent the orthonormal principal components directly, and further the weight matrices $\mathbf{W}_A$ and $\mathbf{W}_B$ are transposes of one another. Using this fact, Baldi (1989) suggests a learning algorithm that is likely to be faster, where the weights are initialised randomly, but such that $\mathbf{W}_B = \mathbf{W}_A^{\mathrm{T}}$, and weight updates are performed symmetrically so that this relation continues to hold during training. The result is a direct extraction of the principal components, and it is possible to show (Baldi and Hornik, 1995) that the resulting algorithm is a multi-unit generalisation of Oja's rule (3.13).

3.2.4 The SEC (Symmetric Error Correction) network

If we are going to link weights in different layers of the autoencoder, as just described, we could equivalently imagine 'folding over' the network of Figure 3.2 so that the outputs are combined with the inputs (where we have the target values anyway) and the signals are passed first forward to the 'hidden' units and then back again to the inputs using the same weights in each direction. A simple network architecture incorporating this form of feedback, called the SEC (Symmetric Error Correction) network was described by Williams (1985). A pattern $\mathbf{x}$ is presented to the network, and propagated forward to give the output $\mathbf{a} = \mathbf{W}\mathbf{x}$ (where $\mathbf{W}$ represents the weight values), and then back to the input using the same weights to give a reconstructed input $\widetilde{\mathbf{x}} = \mathbf{W}^{\mathrm{T}}\mathbf{a}$. The inputs then compute the difference between the feedback signal and the original input, giving an error signal (residual) of $\mathbf{r} = \mathbf{x} - \widetilde{\mathbf{x}}$. The individual weight vectors are then 'corrected' using

$$\Delta\mathbf{w}_i = \eta\, a_i \mathbf{r} \tag{3.15}$$

which is identical to the basic Hebbian rule (3.12), but using the residual $\mathbf{r}$ in place of the original input $\mathbf{x}$. By substituting values for $\mathbf{r}$ and $\widetilde{\mathbf{x}}$, (3.15) may be rewritten in matrix form as

$$\Delta\mathbf{W} = \eta\, \mathbf{a}(\mathbf{x} - \mathbf{W}^{\mathrm{T}}\mathbf{a})^{\mathrm{T}}. \tag{3.16}$$

[2] In fact the requirement for zero-mean data can be lifted by giving the units adjustable *biases* (as is usual in the MLP model), but for simplicity all units are assumed here to have zero bias.

A direct comparison of (3.16) with (3.13) shows that the former is a straightforward multi-dimensional generalisation of the latter. Indeed this learning rule is identical both to the 'subspace' algorithm described by Oja (1989) and to the symmetrically updated version of the autoencoder (Baldi, 1989) outlined above.

It is interesting that starting from different conceptual models, several researchers have arrived at precisely the same algorithm. The reason for dwelling in particular on the SEC network, however, is that it demonstrates the usefulness of feedback. It could well be argued that the SEC network is the most 'natural' realisation of the algorithm, since it requires neither the linking of weights that are conceptually (and actually in a physical realisation) distant, nor the 'nonlinear internal feedback' included in Oja's model. Furthermore, by adopting the feedback model, we find that learning can follow the simplest possible learning rule, based solely on the correlation of local signals (3.15).

The value of feedback as a conceptual and physical model was noted by Hinton and McClelland (1988) in a proposed method of 'recirculation,' which is essentially a nonlinear generalisation of the SEC network. These ideas have also been incorporated into more recent models such as the Helmholtz machine (Dayan *et al.*, 1995). The generalisation and enhancement of this type of feedback forms the conceptual basis for the network model introduced in Chapter 4.

3.2.5 Do PCA neural networks have a role?

While interesting from a biological perspective, neural PCA models in their simplest form have little practical use since they will almost always be outperformed by the method of singular value decomposition.

Nevertheless, PCA networks could well have practical uses when extended in some way. Xu and Yuille (1993, 1995), for example, look at the issue of building models that robustly find principal components, even in the presence of outliers.

There is also scope for further constraining PCA networks in an attempt to find components that are fully independent, rather than just decorrelated. We shall look at some of these later in this chapter.

3.3 Some common misconceptions about PCA

Certain deficiencies of PCA are well known. In particular, because the 'interestingness' of particular projections is based entirely on variance, it is highly sensitive to scaling of the data. Because of this, it is common to advocate the normalisation of data before performing PCA, but the very fact that scaling changes the results obtained suggests that the method is not capturing the 'true' features of the data.

In their well-known work, Duda and Hart (1973) pointed out the deficiencies of techniques such as PCA that consider only second order statistics by giving examples of very different distributions that have identical second order statistics. Similar examples are recreated in Figure 3.3. Despite this, and although many are well aware of the limitations, several misconceptions about PCA are prevalent amongst those who know of and maybe even use

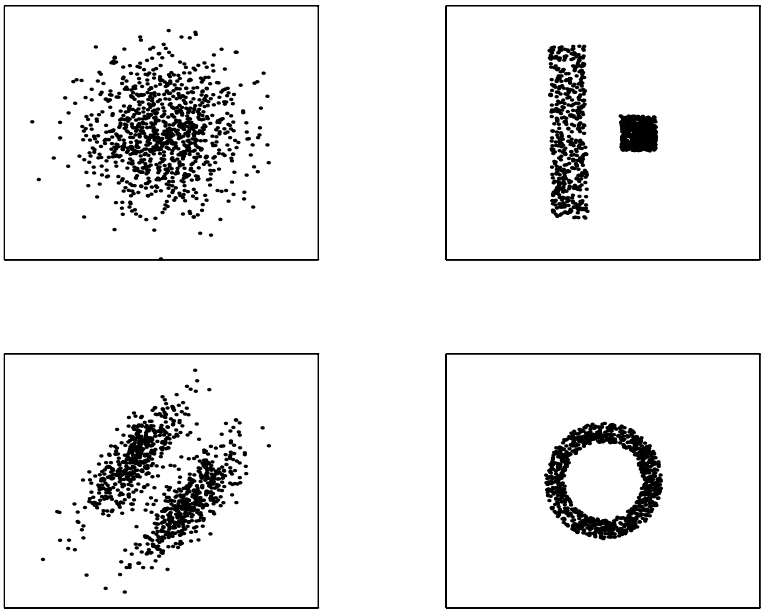


Figure 3.3: Four data sets with identical second-order statistics (after Duda and Hart, 1973). In fact their distributions are also all *spherical* (see Section 3.8.1) so that PCA is unable to distinguish between *any* projection of these data onto orthogonal axes centred at the mean.

the method, but have not examined it in detail. Textbooks generally do little to dispel these myths, and a few even perpetuate them. Some of the misconceptions are discussed below:

1. "*A requirement for finding a decorrelated representation is that the features used are orthogonal.*" While it is true that PCA produces a decorrelated representation by using orthogonal 'features,' there are in general a whole continuum of mappings that result in decorrelated coefficients. PCA produces a unique solution from this continuum by imposing the constraint of orthogonality. This constraint is very useful from a computational point of view because it permits the sequential reduction employed by singular value decomposition (Section 3.1) and hierarchical PCA networks (Section 3.2.2). In terms of the solutions produced, when PCA is used for dimensionality reduction, the constraint is somewhat arbitrary because the principal subspace is the same whether the basis we choose is orthogonal or not. For feature detection, however, it means that PCA can only find the 'true' features if they happen to be orthogonal. In general there is no reason to suppose this to be the case.

2. "*PCA gives us a representation with decorrelated values, which means that they are statistically independent.*" The English words 'correlation' and 'dependence' are virtually synonymous. Unfortunately, 'correlation' in statistical literature is usually (and often implicitly) used in relation only to second-order statistics, so that the diagonality of a random variable's covariance matrix is a sufficient condition for describing its elements as 'decorrelated.' This is a much weaker requirement than true statistical independence, so that while statistical independence certainly implies decorrelation, the reverse implication is not valid. Such an implication is often made, however, resulting in statements such as:

> The singular value decomposition defines a linear transformation from the data to a new representation with statistically independent values ... (Wandell, 1995, p. 255).

3. "*If we ignore statistics of higher than second order, we still get a reasonable approximation to independent features, similar to the way, for example, that we find a reasonable approximation to a function by ignoring higher-order terms in its Taylor expansion.*" There is some truth in this, in that a consideration of statistics only up to second order reduces the space of all possible features to a subspace that *contains* the independent features (if they exist within the model in question). This does not necessarily mean, however, that picking a point from this subspace (either at random or by following some arbitrary constraint) will necessarily result in a useful approximation to the independent components. With any two-dimensional data, for example, and for *any* direction chosen at random as a first feature, there is a second such that the resulting transformation gives decorrelated coefficients.

4. "*Granted, higher-order statistics are important: to address these we must consider nonlinear models.*" This is not necessarily true. In the case, for example, where the data $\mathbf{x}$ is generated according to $\mathbf{x} = \hat{\mathbf{W}}\hat{\mathbf{a}}$ for any invertible 'mixing matrix' $\hat{\mathbf{W}}$ and a random vector $\mathbf{a}$ with statistically independent elements, there is (of course) a linear transformation that extracts the independent components from the data, given simply by $\hat{\mathbf{W}}^{-1}$. Finding $\hat{\mathbf{W}}^{-1}$ from the data alone (*i.e.* without knowing $\hat{\mathbf{W}}$ in advance) requires the use of various nonlinear constraints on the model, but it is somewhat misleading to say that the model itself must (in such cases) be nonlinear. The term 'constrained linear model' might be more appropriate. This problem is often termed *blind deconvolution*, and we shall see some examples of it in Section 5.9.1.

These ideas may be illustrated with some simple examples. Consider a set of data generated from two independent random variables, $\hat{a}_1$ and $\hat{a}_2$, both with zero-mean distributions of unit variance. The variables may be combined to form a two-element column vector $\hat{\mathbf{a}}$. This is then premultiplied by a mixing matrix $\hat{\mathbf{W}}$ to give a vector $\mathbf{x}$. The matrix $\hat{\mathbf{W}}$ is given by:

$$\hat{\mathbf{W}} = \begin{pmatrix} \sigma_1 \cos\hat{\theta}_1 & \sigma_1 \sin\hat{\theta}_1 \\ \sigma_2 \cos\hat{\theta}_2 & \sigma_2 \sin\hat{\theta}_2 \end{pmatrix}.$$

Figure 3.4(a) shows a random sample of data $\mathbf{x}$ generated in this way for $\sigma_1 = 1, \sigma_2 = 2, \hat{\theta}_1 = 18^\circ, \hat{\theta}_2 = 60^\circ$ and underlying random variables $\hat{a}_i$ with Gaussian distributions. The orientations $\hat{\theta}_1$ and $\hat{\theta}_2$ are shown with solid lines, and the principal components with dashed lines. In the Gaussian case, it is not possible (with any technique) to recover the original orientations: the principal components give the major and minor axes of the elliptical iso-probability contours, and this is the best we can hope to do. Real distributions are very rarely Gaussian, however, and there is plenty of scope for improvement with non-Gaussian data. Figure 3.4(b) shows an example where the underlying distributions are more highly peaked than a Gaussian, sometimes termed *super-Gaussian*. They were generated using a *Weibull distribution*.

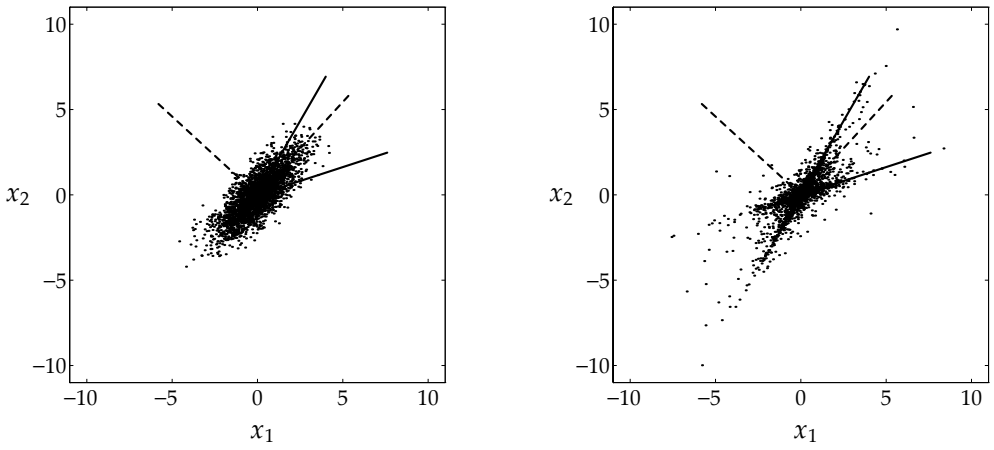


Figure 3.4: Demonstrations of (a) the adequacy of PCA as a feature-detector when data are normally distributed, and the impossibility of recovering the original components in this case, and (b) the *inadequacy* of PCA in this respect when data are non-normally distributed. Both sets of data have been created using a random sample from the linear superposition of two one-dimensional distributions whose probability density functions are even. The distributions are at orientations of 18° and 60° to the horizontal, as shown by the solid lines, and their variances are 1 and 2 respectively. The dashed lines show the directions of the two principal components. (a) was produced using two normal distributions, and (b) using two double-tailed Weibull distributions (3.17), each with parameters $\alpha = 1$ and $\beta = \frac{2}{3}$.

The p.d.f. of the (double-tailed) Weibull distribution is given by

$$p(x) = \alpha^{-\beta}\beta\,|x|^{(\beta-1)}\exp\left[-\left(\frac{|x|}{\alpha}\right)^{\beta}\right]. \tag{3.17}$$

The use of this particular p.d.f. has no special significance — it merely provides (for $0 < \beta \leq 1$) a reasonably generic super-Gaussian unimodal distribution from which it is easy to generate random samples (Dagpunar, 1988, p. 47). With α fixed at 1 and $\beta = 1$, the distribution is identical to a Laplacian (double-tailed exponential), and becomes more highly peaked as the value of β is decreased towards zero.

In this example, the underlying orientations (the *independent* components) are clearly visible, but the *principal* components lie in very different directions. The situation is further illustrated in Figure 3.5. This shows the (numerically calculated) zero-contours of the correlation between two coefficients a_1 and a_2 obtained by projecting the data of Figure 3.4(b) onto vectors at angles θ_1 and θ_2 to the horizontal.

We see that there is a continuum of possible pairs of vectors for which a_1 and a_2 are decorrelated. PCA constrains the solution to two points by allowing only orthogonal sets of vectors, and to a single point by ordering the components by decreasing variance.

However, the coefficients given by the PCA solution are *not* statistically independent, and it is clear from Figure 3.4(b) that the vectors do not correspond to the 'true' features of the data. A method that *is* able to determine the original angles $\hat{\theta}_1$ and $\hat{\theta}_2$ is often termed *independent component analysis*, and so these solutions are marked ICA in Figure 3.5. We could, if desired, order by variance to obtain a unique solution, but this is a minor consideration.

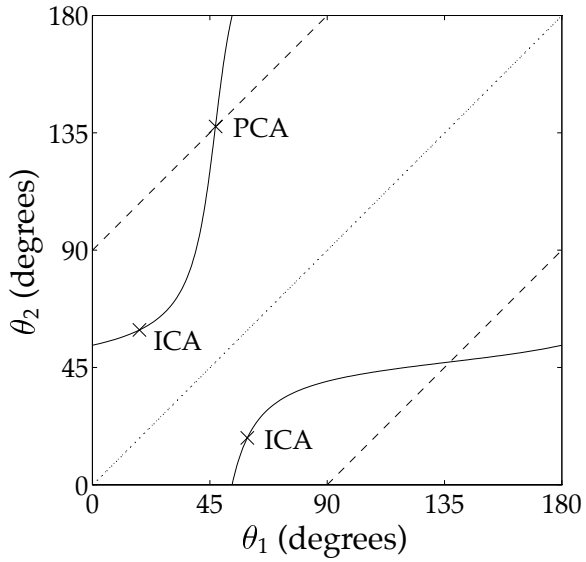


Figure 3.5: Graph of θ_1 and θ_2, the angles to the horizontal of two basis vectors, $\mathbf{w}_1$ and $\mathbf{w}_2$, with the solid lines showing the values for which $\mathcal{E}[a_1 a_2] = 0$, where a_1 and a_2 are the coordinates obtained by projecting the data of Figure 3.4(b), or equivalently of Figure 3.4(a), onto $\mathbf{w}_1$ and $\mathbf{w}_2$. Since the directions are interchangeable, the graph has reflective symmetry about the line $\theta_1 = \theta_2$ (shown dotted). The dashed lines show the values of θ_1 and θ_2 for which $\mathbf{w}_1$ and $\mathbf{w}_2$ are orthogonal. The solution given by PCA, and the two possible ICA solutions are marked with crosses.

We shall return to the issue of ICA later in this chapter (Section 3.8.2), and methods for obtaining the desired solution for this particular example will be given in Chapter 5.

3.4 Projection and kernel methods

The technique of PCA may be described as a *projection method* (Zemel, 1993) — the codes that it produces are created by projecting the data onto what might loosely be termed 'interesting' directions in the input space. This and other such methods generally produce *distributed* representations, where each input vector is coded as a combination of some or all of the component directions, with the output values representing the coordinates in the new space.

In contrast to this are a class of unsupervised techniques that we shall term *kernel methods*, where *local* representations are formed by fitting each input vector to a single template, class, or *cluster*. Another way of characterising the difference between the two types of method is that while in a projection method, the individual outputs *cooperate* to build the representation, in a kernel method they *compete*.

In the following section we shall turn briefly from projection to kernel methods, before examining why the best representations may often come from a combination of the two.

3.5 Vector quantisation and competitive learning

Vector quantisation (VQ) is a simple but effective unsupervised clustering technique (Gray, 1984). Each data vector is represented by a vector, known as a *codeword*, chosen from a lim-

ited *codebook*. The codebook can be developed using the *LBG* learning algorithm (Linde, Buzo, and Gray, 1980). This codes each training vector using the *nearest* (in Euclidean distance) available codeword and then updates each codeword to be the centroid of all the training vectors that mapped to it. This process is iterated until a minimum of average *distortion* between training vectors and the corresponding codewords is reached. Since Euclidean distance is used, the objective function, as for PCA, is simply the mean-squared error (3.6).

Several neural network models work along similar lines. These use *competitive* learning, where the standard feedforward model is augmented by direct connections between outputs that cause them to compete for activation. These have negative weights so that a strong output from one unit suppresses those of others around it, and the structure is therefore known as *lateral inhibition*.

In its strongest form, the competition is so great that only a single unit (the 'winner') is active at the end of the process, producing what is known as a *winner-take-all* (WTA) network. While it may in theory be desirable to think in terms of lateral connections, in computer simulations it is usually more practical simply to 'pick' the winner after feedforward activation, for example by choosing the unit with the largest output value. Learning is usually achieved by moving the weight vector of the winning unit *closer* to the input vector (Rumelhart and Zipser, 1985). When the distance measure used is Euclidean, this is simply an on-line version of vector quantisation.

A common extension to this idea is to arrange the output units in a grid formation, and for the learning rule to be applied not only to the winning unit but also to its neighbours. This leads to a model known as the *self-organising map* (Kohonen, 1982, 1990). It creates *topographic maps* that retain proximity relations in the data. This is a useful computational simplification of a biological model proposed by von der Malsburg (1973), where cells excite their close neighbours, but inhibit the activity of those that are more distant.

3.6 The limitations of single-cause models

All of the models in the previous section attempt to ascribe each input pattern to a single cause, *i.e.* while they contain 'templates' for many possible features, they represent each pattern using only one of them. In so doing, the models make a firm *decision* to place each pattern in a single class or cluster. This unsupervised form of classification can have significant advantages over cooperative projection methods, whose representations do not directly allow subsequent processing to see what *type* of input has occurred. However, there are several important limitations of these single-cause models:

- The techniques are beset by the *curse of dimensionality* (Bellman, 1961). If, for example, the clusters of data are sited at fixed intervals over the entire input space, their numbers increase exponentially with the number of input dimensions. Compare this with projection methods such as PCA where the required number of outputs increases at worst linearly. For large problems, single-cause models may require an infeasible number of outputs to produce a complete or near-complete representation. Alternatively, if the number of outputs is kept small, large amounts of information may be lost by the mapping.

- We have identified the decision-making property of a clustering technique as a potential advantage over projection-based methods, but it can also be a weakness. A decision implies a loss of information, a feature that is not necessarily desirable at an early stage of processing (Section 2.3). Any ambiguity in the inputs, for example, is often best passed on to higher levels of processing, where additional information may be available to resolve it. This does not completely remove the role of an unsupervised preprocessing step: the ideal system is arguably one that massages data into a form that allows any subsequent decision-making process to be as simple as possible.

- The problems with a modelling scheme that generates a single winner also extend into learning. In most unsupervised systems, parameter updates are a function of the output values. Therefore if an incorrect decision ('guess') is made in determining the winning output, the parameter updates will also be wrong. Such mistakes could quite possibly throw the whole learning process off course. This idea that 'wrong winners lead to wrong learning' is discussed further by Marshall (1995).

- The generalisation capabilities of a single-cause model are much weaker than those where there are many active outputs. A pair of output vectors with different winners are no more or less alike than any other such pair, so the generalisation properties lie only in the extent to which different input values are mapped to the same winning output. In a projection-based representation, by contrast, the similarity of two output vectors is readily measured, for example by their separation in the output space. Having said that, the self-organising map (Section 3.5) largely overcomes this limitation by introducing a measure of proximity between output elements.

These are much the same issues that we came up against when discussing the relative merits of local and distributed codes in Section 2.7, although there we were interested primarily in representations rather than models. The close links reflect the fact that a code is tied intimately to the model that generated it. Just as we concluded previously that a sparse representation could give the best compromise solution, so here we find that both extremes of single-cause kernel methods and fully distributed projection methods have their limitations, and that the best systems may lie in the middle ground. This leads us to look at *multiple-cause* models.

3.7 Multiple-cause models

It is common for there to be multiple separate causes, appearing in many different combinations, underlying the input patterns. In this case, the above limitations can largely be overcome by adopting a representation that can contain several 'active' elements, for example by extending VQ to use multiple codewords per pattern, or WTA networks to have multiple winners. This section looks at a few examples of such systems.

3.7.1 Iterative competitive learning

The primary difficulty with multiple-cause models is that of striking the right balance between competition (needed to prevent many outputs from encoding the same features) and

cooperation (needed to build a representation from a combination of features).

Rumelhart and Zipser (1985) used a fixed partitioning to separate a network into independent competitive 'pools' of units, but in the absence of competition between pools, there is a tendency for each to perform the same clustering of the input space. A simple solution to this problem is to activate the pools sequentially. This technique was used by Mozer (1991) in an algorithm known as *iterative competitive learning* (ICL). Each stage of the activation process produces a single winner, and the corresponding feature is subtracted from the data before it is fed to the next stage. This method is equivalent to an extension of VQ known as *multi-step* VQ. The main problem with this sequential approach is its assumption that features can be extracted unambiguously in a fixed order of priority. Where single patterns contain several features of roughly equal importance (as is more common in real data), sequential techniques cannot be guaranteed to find the best overall representation.

3.7.2 Cooperative vector quantisation

Another extension to the standard VQ algorithm is the method of *cooperative vector quantisation* (CVQ) (Zemel, 1993; Hinton and Zemel, 1994). This uses several vector quantisers in parallel, whose individual outputs, when combined, attempt to reconstruct the input.

The difficulty of training such a system is tackled by using an objective function based on the *minimum description length* principle (Rissanen, 1978, 1985). The system's objective function is a combination of the reconstruction error and an 'activity cost,' based on the outputs' deviation from their expected values. We shall see very similar ideas used in Chapter 5.

3.7.3 EXIN networks

Marshall (1995) identifies four factors that an unsupervised system should deal with if it is to produce a useful coding of a complex perceptual environment.

1. Context. Units that encode an individual feature of the environment should be sensitive (where necessary) to the context in which that feature appears. Cohen and Grossberg (1986) illustrate the idea with the word 'myself.' Contained within the word are several others such as 'my,' 'self,' and 'elf.' If we are to have one unit responding, say, to 'myself' and a separate one to 'elf,' then the latter should respond to 'elf' in isolation, but not when preceded by 'mys.' To achieve this type of coding, the system must take the full context, and not just sub-patterns, into account.

2. Uncertainty. In real environments, patterns can be ambiguous and noisy. Where a pattern is not matched by one or more of the system's stored 'prototypes,' the uncertainty should be represented in the system's output. In this way it is possible to resolve ambiguities should further information subsequently become available. Conversely, if hard decisions are made at this stage of processing (by using binary outputs, for example), it is much more difficult to correct mistakes or resolve ambiguities later.

3. Multiplicity. Complex patterns are typically composed of subparts, possibly in many different combinations. In such cases, it is sensible for the system to represent each subpattern separately, and allow multiple outputs to be active when several patterns occur

simultaneously. This overcomes the limitations of single-cause models, as discussed in Section 3.6.

4. Scale. The system should be sensitive to features at a variety of scales. In competitive networks, this may require careful balancing of the weights to ensure that outputs representing 'large' features (those extending over many inputs, for example) do not receive an unjustified advantage in competition. At the same time, according to Cohen and Grossberg's (1986) principle of *perceptual grouping*, the system should choose a representation based on the largest (and therefore fewest) features that are compatible with its model of the world.

Marshall introduces a system called an 'EXIN' (excitatory and inhibitory) network that addresses these issues. It uses excitatory feedforward and inhibitory lateral connections, with separate learning rules for each, and its activation is governed by a set of coupled differential equations.

While the EXIN network's efficacy can be demonstrated on small examples, its practicality is limited by the high computational cost of finding numerical solutions to the differential equations governing its activation. In the absence of a clear objective function, it is also difficult to ascertain, except empirically, the system's effectiveness and stability.

3.8 Projection methods beyond PCA

We shall return now to techniques of extracting features by making projections of the data. We have already discussed the limitations of PCA in this respect, but there are other methods that build upon its capabilities by introducing nonlinearities and/or taking higher order statistics into account. The first step of several of these techniques is, in essence, to perform PCA itself in a process known as *sphering*.

3.8.1 Data sphering

The coefficients obtained from a projection onto the principal components are *decorrelated*, *i.e.* second-order dependencies are removed. It is common to make use of this property before going on to consider higher order statistics by first *sphering* the data.[3]

Sphering is achieved (Fukunaga, 1990) simply by a linear scaling of the principal component projection, such that each coefficient also has unit variance. For zero-mean data $\mathbf{x}$, the transformation is given by:

$$\mathbf{z} = \mathbf{\Lambda}^{-1/2}\mathbf{U}^{\mathrm{T}}\mathbf{x}$$

where $\mathbf{U}$ is the matrix whose columns represent the principal component directions (eigenvectors of the covariance matrix), and $\mathbf{\Lambda}$ is a diagonal matrix with elements equal to the corresponding eigenvalues. The covariance of the sphered data $\mathbf{z}$ is therefore simply the unit (identity) matrix.

[3]The technique is also known as *whitening*, but we shall avoid this term since it is sometimes used to describe methods that remove *all* dependencies (not just those of second order) and is therefore in danger of breeding the same confusion as exists between the terms 'decorrelated' and 'independent.'

If the data are truly a linear mixture of independent underlying components, then although sphering will not itself extract the components, they may be obtained by an *orthogonal* projection of the sphered data. This is a very useful property of sphering: an example is given in Figure 3.6(a), which shows a sphered version of the data set previously seen in Figure 3.4(b). We see that the process has transformed the data such that the underlying distributions are now at right angles to each other.

This property is used as a preprocessing step by several higher-order statistical techniques since it means that they only need consider orthogonal projections. There are problems, however. Several arguments against the use of sphering are put forward in the discussion following a paper by Jones and Sibson (1987). These principally concern the lack of robustness to outliers. An example is given in Figure 3.6(b), which shows a sphered version of the same distribution as in (a), but after the addition of two outlying points. We see that the desired directions are now far from orthogonal.

The danger is that if a method *relies* on using sphered data as its input, then it is heavily dependent on the effectiveness of the sphering process. While techniques may be employed to increase the robustness to outliers (*e.g.* Tukey and Tukey, 1981), these often have only limited success.

A further problem occurs where dependencies between components are not fully linear. An example is shown in Figure 3.6(c), where both linear and nonlinear terms have been used to generate the data. The nonlinearity is apparent as a curve in one of the 'arms' of the distribution, and an orthogonal projection would clearly no longer produce a fully independent representation.

A third, and perhaps the most important, restriction of sphering as a preprocessing step lies in its assumption that the number of independent components is no greater than the dimensionality of the data. Figure 3.6(d) shows an example where this is not the case. The data has again been sphered, but is obviously unable to represent the three independent components orthogonally, since there can only ever be two orthogonal directions in two dimensions! A projection that yields independent coefficients for this example requires an *overcomplete* representation, an issue that will be discussed further in Chapter 5.

3.8.2 Independent component analysis

The process of taking a set of data and extracting components that are not just decorrelated, but fully statistically independent, is generally referred to as *independent component analysis* (ICA). Beyond that, however, the term is not well defined, and is used to refer to a wide variety of techniques that contain different assumptions and limitations. The components are often assumed to have undergone a linear mixing process such as we saw in Section 3.3. In this case in particular, the process is also commonly referred to as *blind source-separation* or *blind deconvolution*.

One system for blind source-separation, described by Jutten and Herault (1991), uses a simple neural architecture with nonlinearities that attempt to capture higher order statistics. The approach appears to be a useful heuristic that works well in some cases, but detailed analysis is difficult, and such analysis as there is shows that there are many cases where the system fails to produce the desired solutions (Comon *et al.*, 1991; Sorouchyari, 1991).

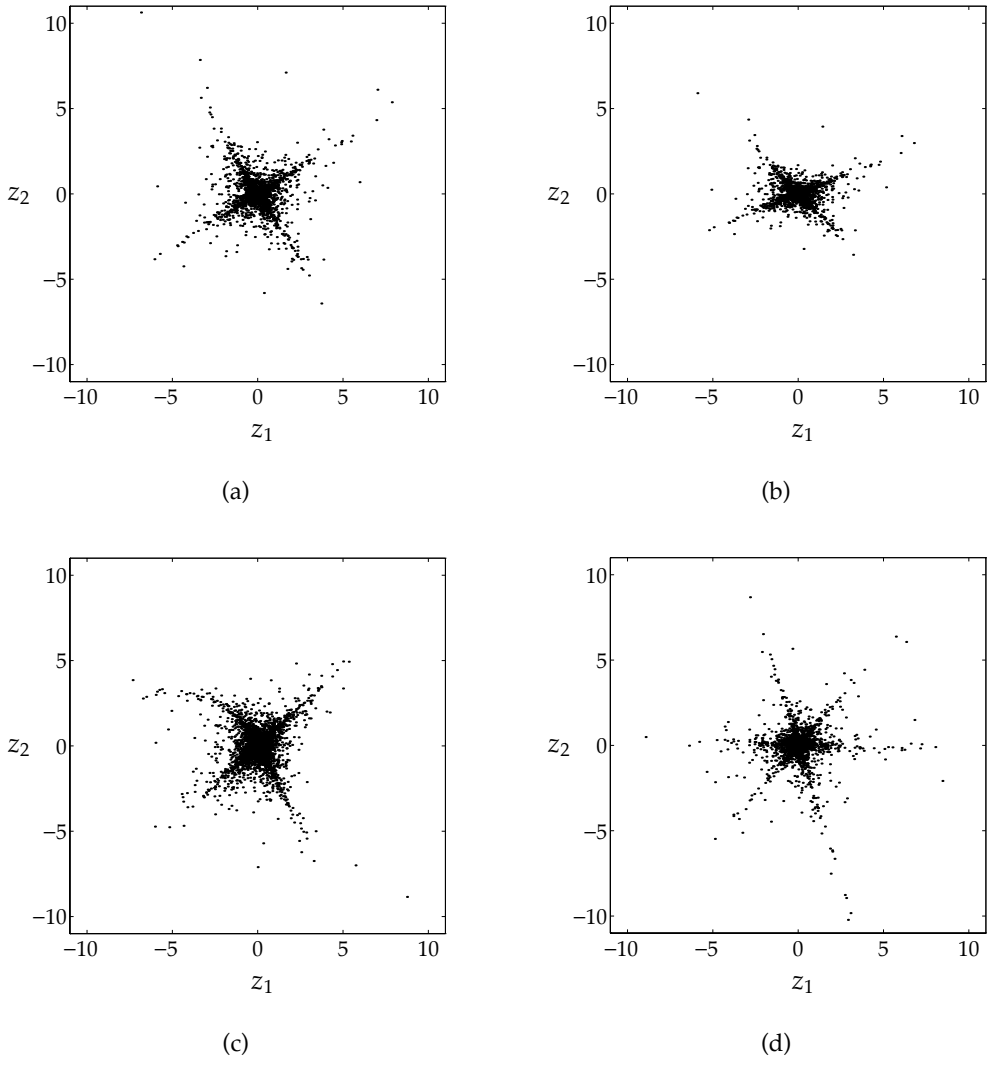


Figure 3.6: Graphs showing the effects of sphering on various distributions. (a) A sphered version of the data previously shown in Figure 3.4(b), illustrating that the independent components are made orthogonal under this transformation. (b) The same data, but with two 'outliers' added (at $x_1 = 0$ and $x_2 = \pm 50$) before sphering. (c) A sphered version of data where the dependency between elements includes nonlinear terms (polynomials up to order three). (d) Sphered data generated from three independent highly peaked distributions in two dimensions.

A much more solid theoretical basis for ICA is provided by Comon (1994). He proposes a technique that is made computationally tractable by only needing to consider components in a pairwise manner. This is sufficient when the underlying components are mixed purely linearly, but is unlikely to produce maximally independent components in other cases, *i.e.* where a linear model can reduce, but not fully remove redundancy.

Good results for source separation have been obtained by Bell and Sejnowski (1995) using an information maximisation ('infomax') technique, and the system has also been applied to audio and visual coding (Bell and Sejnowski, 1996, 1997). The underlying idea was proposed by Laughlin (1981): when a signal is passed through a *sigmoid nonlinearity*, the resulting entropy is maximised when the signal's cumulative distribution function matches the nonlinearity. The system performs this maximisation in an attempt to generate low entropy representations. Improvements to the algorithm, and a maximum likelihood interpretation, are offered by Cardoso (1996) and Mackay (1996).

Data-sphering can improve the performance of the infomax technique by removing second order dependencies. Some other methods simplify matters by performing only orthogonal projections of sphered data (Karhunen *et al.*, 1995; Oja, 1995). The limitations inherent in this approach were discussed in Section 3.8.1.

3.8.3 Nonlinear PCA

Another extension to PCA is to allow projections that involve nonlinear warping of the input space. We saw in Section 3.2.3 that a linear MLP architecture can perform a principal subspace projection, so a seemingly useful extension would be to use nonlinear units, for example by the standard technique of passing activation values through a sigmoid function such as the hyperbolic tangent. However, Bourlard and Kamp (1988) showed that simply introducing such nonlinearities into the two-layer architecture (Figure 3.2) still only yields a principal subspace projection.

An advantage *can* be gained, however, by using a four-layer autoencoder architecture, as shown in Figure 3.7. This performs two nonlinear mappings: a first $\mathbf{F}_A$ on the input vector $\mathbf{x}$ to give a 'compact' (reduced dimensionality) representation $\mathbf{a}$, and a second $\mathbf{F}_B$ that generates the output values $\widetilde{\mathbf{x}}$. The first and third hidden layers use nonlinear units, while the second is linear. Since the aim of training is to reduce the differences between $\mathbf{x}$ and $\widetilde{\mathbf{x}}$, the mapping $\mathbf{F}_B$ becomes the approximate inverse of $\mathbf{F}_A$. If the reconstruction error were zero, then $\mathbf{a}$ would give a complete representation of $\mathbf{x}$ under a nonlinear mapping. The problem in practice is that the nonlinearities, and the need to learn both the mapping and its inverse, tend to produce many local minima in training. Nevertheless, successful applications of this technique have been demonstrated by Kramer (1991) and Burel (1992). Hecht-Nielsen's (1995) 'replicator' neural networks have the same structure, but use a 'stairstep' function in the middle hidden layer, thereby producing a quantised representation.

A method that incorporates general nonlinear mappings *and* attempts to produce independent representations has recently been proposed by Parra *et al.* (1995, 1996).

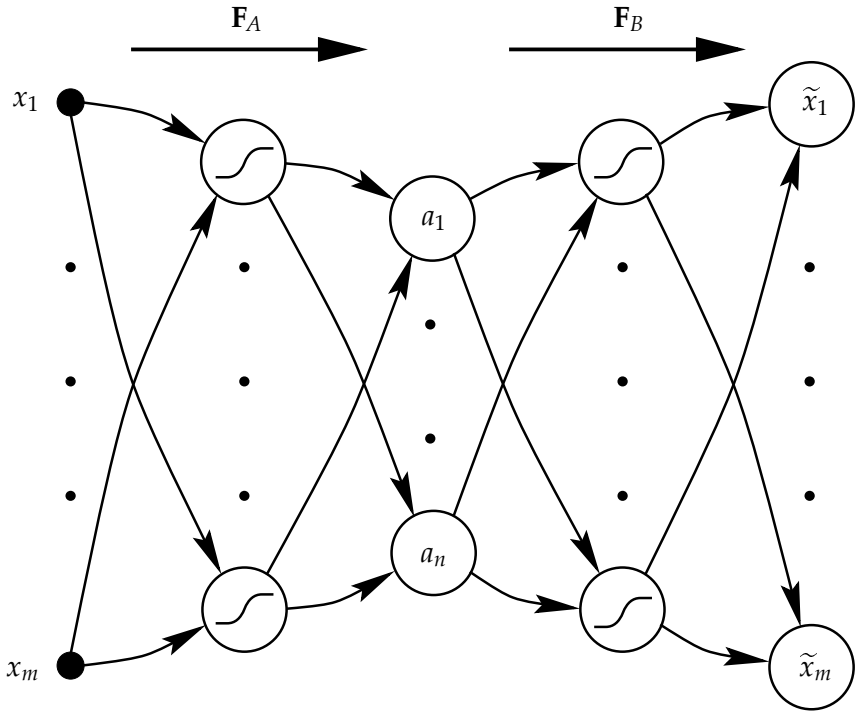


Figure 3.7: A nonlinear autoencoder network. Sigmoid functions are used in the first and third hidden layers to generate the nonlinear mappings $\mathbf{F}_A$ and $\mathbf{F}_B$ respectively.

3.8.4 Projection pursuit

Another class of projection methods is known as *projection pursuit* (PP) (Huber, 1985; Jones and Sibson, 1987). These search for 'interesting' projections of the input space, with the interestingness being defined by a *projection index*. The index is some measure of the distribution of coefficients obtained by projecting the data onto a particular direction. Most indices are based on the distribution's deviation from normality, for example by using entropy or the fourth moment (kurtosis). The contention is that interesting projections result in highly non-normal distributions, since directions picked at random tend to produce distributions that are approximately normal.

Several neural implementations of PP have been explored. The BCM network of Bienenstock, Cooper, and Munro (1982), developed as a computational model of the visual cortex, can be shown to perform a type of PP (Intrator, 1992; Intrator and Cooper, 1992). A feedback network architecture that also performs PP is described by Fyfe and Baddeley (1995a,b).

3.9 Discussion

In this chapter we have looked at a number of unsupervised techniques for modelling data. These can broadly be divided into projection methods, which perform a linear mapping of the input space into a new 'feature space,' and kernel methods, where the input is matched against a number of prototypes. The simplest projection methods perform principal component analysis, and the simplest kernel methods are single-cause or winner-take-all (WTA) models such as vector quantisation. Both approaches have strong limitations. The codes they produce may be pictured as lying at opposite ends of a spectrum that ranges from fully

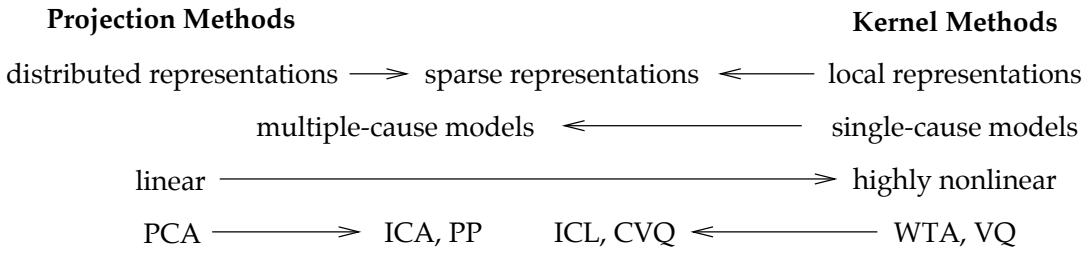


Figure 3.8: Various ways to characterise unsupervised methods. The distinction between projection and kernel methods was made in Section 3.4. These may also be described by the type of representations they produce (Section 2.7), or the nature of the underlying model. Another variable in these methods is the linearity, which can vary from purely linear for PCA to highly nonlinear for WTA models. The final line of the diagram shows the positions on these scales of some of the techniques discussed in this chapter.

distributed to fully localised representations. In most cases, the ideal representation will lie somewhere in between. This idea is depicted in Figure 3.8, which also shows some of the methods lying in the 'middle ground' that have been mentioned in this chapter.

Against the background of such methods, the remainder of this thesis searches for new and more powerful ways to produce codes that find the right balance between distribution and locality. In the next chapter, we shall look at a new type of unsupervised network that has been designed with this aim in mind. Initially it may be viewed as a simple extension to PCA in which the orthogonality constraint has been removed, but later (in Chapter 5) we shall see that this opens up the possibility of constraining the network in new ways that enable us to generate more efficient representations.

Chapter 4

A Recurrent Network Model

In this chapter, we shall look at the unsupervised network model that forms the basis of the remainder of this thesis. The model was developed with the aim of being as simple as possible while at the same time retaining sufficient power for it to be applied generally. Both these factors facilitate the modifications made to it in later chapters. The two-layer structure of the network differs from the majority of neural network architectures in that it incorporates *feedback* from the output to the first layer, and the model is dynamic in the sense that it follows an iterative, or *recurrent*, method of activation.

While this recurrence requires a more complex method of activation than for standard feedforward models, we shall see that this is balanced by much simpler learning rules, and gives the network the flexibility to produce outputs ranging from highly sparse to fully distributed representations. Learning is entirely localised, which, as well as suggesting a good degree of biological plausibility, has implications for possible parallel implementations of the model which will be discussed further in Chapter 7.

The network is introduced, first in terms of its general aims and structure, and then with the specific details of the activation and learning rules. The algorithm's links to information theory and maximum likelihood estimation are discussed, and its operation is demonstrated with a number of examples. We shall begin, for simplicity and ease of analysis, with a purely linear model. It will however quickly be augmented, in this and more especially in subsequent chapters, with nonlinearities suited to the structure of specific environments.

The system will be referred to, because of its structural similarity to the SEC network (Section 3.2.4), as a REC (Recurrent Error Correction) network.

4.1 Inhibitory feedback and recurrence

The problem that the REC network attempts to address is one of automatically producing useful and efficient codes based solely on the properties of the data to be encoded. As discussed in Chapter 2, this can be characterised as an inverse problem (*i.e.* one of trying to infer causes from observations) and such problems are in general hard.

The inverse problem exists on two different levels, and two different time-scales; the system needs firstly to be able to infer the most likely causes for a particular set of inputs (given its current model of the world), and secondly to infer the most likely world-model over a whole series of inputs (under any constraints we might place on the nature of possible

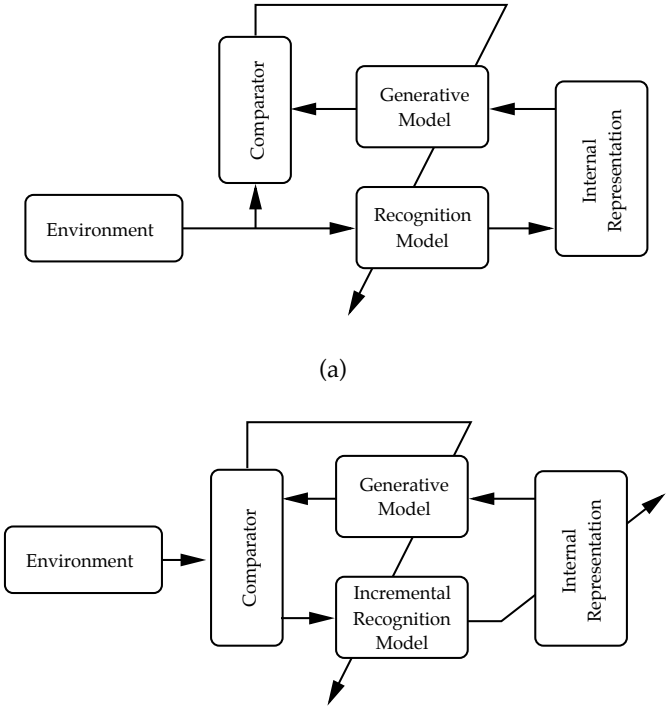


Figure 4.1: (a) A general framework for an unsupervised system, where changes to the models are driven by differences between the original and reconstructed inputs. (b) A variation on the previous model where the internal representation also becomes the subject of iterative adaptation.

models). Many network algorithms address only the second issue directly, and rely on the fact that *in their final state* they will be able to solve the first. A different approach is taken here — the inference problem is tackled for every pattern presented to the network, with the results from this subproblem being used to drive the modelling of the data set as a whole.

A useful concept now commonly used in this area is to think of the system in terms of both inverse, or *recognition*, and forward, or *generative* models (*e.g.* Dayan *et al.*, 1995). A general modelling framework that makes use of these ideas is shown in Figure 4.1(a). An internal representation of the environment is produced by applying a recognition model to the input. A subsequent application of a generative model to this representation attempts to reconstruct the input; differences between the original and reconstructed inputs may be used to improve both recognition and generative models. The exact methods for measuring differences and adapting the models will depend on the relative importance to the system of various properties of the input. The SEC network, discussed in Section 3.2.4, fits exactly into this feedback model, and, as pointed out by Saund (1994), a large number of other error-driven unsupervised learning models may be viewed in this way.

A slightly modified form of this model is shown in Figure 4.1(b). Here, a comparison between the original and reconstructed inputs is also used (via an incremental form of the recognition model) to iteratively adapt the internal representation (it is usual to assume that this occurs on a much faster time-scale than adaptation of the models). The advantage of this approach is that the role of the recognition model is much reduced. Because it is now

driven by differences between the original and reconstructed inputs, it needs only to be able to move the internal representation *toward* the correct values, although the better it is at this the fewer the iterations required to achieve convergence.

This is useful, firstly because it means that only one of the two models needs to be learnt accurately, and secondly because, as mentioned at the beginning of this section, it is the recognition model (corresponding to the inverse problem) that is the more difficult to find. We tackle the inference problem by guessing, testing the guess, and using the result to improve it, a technique known as 'analysis-by-synthesis.'

Iterative approaches in general are commonly used for solving modelling problems (*e.g.* Ljung and Söderström, 1983). Analysis-by-synthesis in particular is the principle underlying 'Pattern Theory,' introduced by Grenander (1976–81). It was proposed as a role for feedback in the brain by Mackay (1956), an idea that has been supported more recently by Pece (1992), Mumford (1992, 1994), Barlow (1994) and Ullman (1994), and is backed by recent physiological evidence (Murphy and Sillito, 1996, and references therein). Feedback of this type may also be found in a number of other neural network models (*e.g.* Carpenter and Grossberg, 1987; Neal, 1992; Kawato *et al.*, 1993; Wada and Kawato, 1993; Fyfe and Baddeley, 1995b).

4.2 Network architecture and activation rules

The REC network closely follows the model of Figure 4.1(b). As is standard for neural networks, the space of all possible models is restricted to a single, *parameterised*, model, with the parameters being represented by the network's weight values.

The basic network assumes a *linear* generative model. This is a very large assumption, and a major limitation of the approach, since few real-world features (which we wish the network to mimic) behave in this way. Nevertheless, it is a useful starting point since a linear model lends itself readily to mathematical analysis, and some of the limitations will be addressed in Chapter 6. Furthermore, we shall see that the linear assumption allows the generative and incremental recognition models to be very closely linked.

The REC network model is shown in Figure 4.2. The method for activating it is mathematically equivalent to a neural network algorithm proposed by Daugman (1988), but this model used three layers, with the weights rather than output values converging to the desired representation. Pece (1992) reformulated the network as a two layer model very similar to that presented here, and linked it to the physiology of the visual system, but this work was not extended to include learning. Ideas very similar to those presented here were also developed independently and concurrently by Olshausen and Field (1996a,b).

Certain aspects of the following discussion, and the subsequent derivation of the Hebbian learning rule in Section 4.7, closely follow the presentation of principal component analysis made in Section 3.1. We shall see in particular that the resulting objective function is the same, and that the basic network model thus finds the principal subspace of its data.

The network has m inputs and n outputs. The input vector $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}}$ is fed to a first layer of units which form the sum of the input and the feedback from the second layer, to give a vector $\mathbf{r} = (r_1, \ldots, r_m)^{\mathrm{T}}$. The feedforward weights to the ith unit in the second layer are denoted by the vector $\mathbf{w}_i = (w_{i1}, \ldots, w_{im})^{\mathrm{T}}$. The feedback weights *from* the same unit

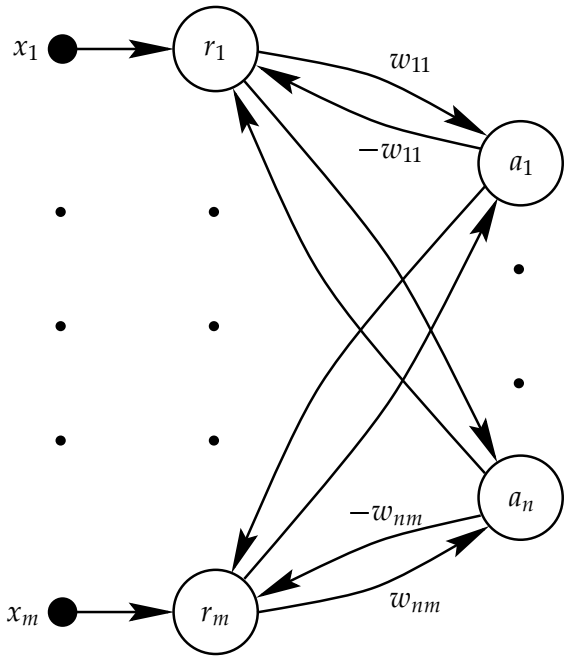


Figure 4.2: The REC network model used to find the optimal (in terms of Euclidean distance) re-representation of the input pattern within the constraints of the current weight values. The symbols within the circles represent the units' outputs, as given by Equations (4.1) and (4.2).

have equal magnitude but the opposite sign, and so the output from the first layer is given by

$$\mathbf{r} = \mathbf{x} - \sum_{k=1}^{n} a_k \mathbf{w}_k. \tag{4.1}$$

The network's output vector is denoted by $\mathbf{a} = (a_1, \ldots, a_n)^{\mathrm{T}}$. Following the recurrent approach described in Section 4.1, the components of $\mathbf{a}$ are not calculated directly. Instead their values, initially zero, are updated using the following rule:

$$\Delta a_i = \mu \mathbf{r}^{\mathrm{T}} \mathbf{w}_i \tag{4.2}$$

where μ is an adjustable rate ($0 < \mu \leq 1$). After each update of the a_i under (4.2), $\mathbf{r}$ is recalculated using (4.1), and so, substituting (4.1) into (4.2), we have that

$$\Delta a_i = \mu(\mathbf{x} - \sum_{k=1}^{n} a_k \mathbf{w}_k)^{\mathrm{T}} \mathbf{w}_i. \tag{4.3}$$

Having summarised, in (4.3), the dynamics of the activation process, we are now in a position to ask what it achieves. The network's aim is to map the input vector $\mathbf{x}$ into a new vector space described by the weight vectors $\mathbf{w}_i$. In the case where the weight vectors fail to span the input space fully, or in other words to form a *complete basis* for the transformation (typically because $n < m$), we can apply the linear generative model to attempt to reconstruct the original input from the representation in the reduced space. The reconstructed input is given by

$$\widetilde{\mathbf{x}} = \sum_{i=0}^{n} a_i \mathbf{w}_i. \tag{4.4}$$

In order to measure how successful this has been, we may use, as for PCA in Section 3.1, an error function given by the squared norm of the difference between the original and the reconstructed inputs:

$$E_r = ||\mathbf{x} - \widetilde{\mathbf{x}}||^2. \tag{4.5}$$

The partial derivatives of this objective function with respect to the outputs a_i are given by

$$\begin{aligned} \frac{\partial E_r}{\partial a_i} &= -2\,(\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}}\mathbf{w}_i \\ &= -2\,(\mathbf{x} - \sum_{k=1}^{n} a_k \mathbf{w}_k)^{\mathrm{T}}\mathbf{w}_i. \end{aligned} \tag{4.6}$$

By comparing the derivatives (4.6) with the output update rule (4.3) we see that

$$\Delta a_i = -\frac{\mu}{2}\frac{\partial E_r}{\partial a_i}$$

or in vector form

$$\Delta \mathbf{a} = -\frac{\mu}{2}\nabla_{\mathbf{a}} E_r$$

where ∇ is the gradient operator. We find therefore that the output update rule (4.3) performs gradient descent on the error function (4.5). The function is quadratic in each element of $\mathbf{a}$, and so we are guaranteed a single, global minimum of E_r, providing that the weight vectors are linearly independent.[1] In the network's equilibrium state ($\Delta\mathbf{a} = \mathbf{0}$), the partial derivatives of E_r with respect to all the outputs will be zero, and thus the global minimum for E_r will have been reached. We may note in passing that in this state, we find from (4.6) and (4.4) that

$$\forall i, \quad \mathbf{r}^{\mathrm{T}}\mathbf{w}_i = 0$$

or, in words, that the residual vector is orthogonal to all the weight vectors. A directly analogous result is found in the design of optimal linear filters, where it is known as the *principle of orthogonality* (Haykin, 1996, p. 197).

4.3 Feedback as a competitive mechanism

In the special case where the weight vectors are orthonormal, the transformation outlined above could simply be done by computing the projection of the input pattern onto each weight vector by forming the inner product $\mathbf{x}^{\mathrm{T}}\mathbf{w}_i$. This can be achieved by a single forward pass of the REC network with $\mu = 1$, and is equivalent to activating a simple feedforward linear network.

The purely feedforward Hebbian networks discussed in Chapter 3 *relied* on the orthonormality (or at least orthogonality) of their weight vectors after learning in order to produce outputs that minimised the sum-squared error function. Without orthogonality, the outputs

[1] If this is not the case then the problem is under-constrained. One method for resolving the ambiguity is mentioned in Section 4.5, and others, directly tailored to producing efficient codes, are explored in Chapter 5.

must interact in some way to produce the desired mapping. This interaction has typically been achieved by lateral ('competitive') connections between the outputs (Section 3.5), but requires an additional learning rule to adapt these extra weights, and means that the mapping will again only be 'correct' (in the least-squares sense) *after* learning.

The REC network may also be viewed as a competitive model, but with the advantages that no extra weights are needed and that the reconstruction error (4.5) is always minimised. The competition is achieved not by lateral connections, but by negative feedback: when an output unit becomes active, it 'claims' that part of the input to which it responded by subtracting it from the outputs of the first layer, and thus preventing any other unit from responding to the same feature. In this way, exactly the right amount of competition is produced: if a single unit is capable of claiming the entire input pattern, then it will prevent any other units from responding at all; at the other extreme, if all the active units are responding to separate (mutually orthogonal) features, then there will be no competition between them, and the responses will be the same as if there were no feedback connections.

Although the REC network has been presented as a feedback model here, it is also possible to view its operation in terms of direct lateral inhibition. Olshausen and Field (1996b) expressed mathematically equivalent network dynamics using a recurrent inhibitory term between two output units i and j equal to the scalar product of their respective weight vectors ($\mathbf{w}_i^{\mathrm{T}}\mathbf{w}_j$). The concept of a feedback model is retained throughout this thesis, however, and will prove particularly useful when generalising to various nonlinear mixture models in Chapter 6.

4.4 Static and dynamic network units

A close look at the algorithm set out in Section 4.2 for activating the REC network reveals a crucial difference between the behaviour of first- and second-layer units. While the output of units in the first layer is determined in a static manner based solely on their current input (4.1), the second-layer units, by contrast, behave more dynamically — they use (some fraction of) their input to *update* their current state (4.2).

To enable both types of unit to exist in a common framework, we can say that a general discrete-time dynamic model of a linear unit is given by

$$a(t+1) = \gamma a(t) + \mu I(t)$$

where $I(t)$ is shorthand for the sum of all external inputs to the unit at time t. From this we see that γ represents a 'forgetting' or 'leakiness' factor and μ is the rate at which external inputs are incorporated into the unit's activity. In the REC model, we have that $\gamma = 0$ and $\mu = 1$ for the first-layer units, and $\gamma = 1$ for the output units with μ an adjustable parameter, as introduced in Equation (4.2). The values for γ mean that units in the first layer have no state (memory), while those in the second have perfect memory. Pece (1992) noted that these are unrealistic assumptions for a model of a physical system, but showed that a physiologically plausible dynamic model could nevertheless closely approximate the behaviour of the 'ideal' network discussed here.

It is important to understand these differences between the characteristics of the different layers of units, because it makes clear the distinction between the REC network and some

other similarly structured network models. Plumbley (1991, Chap. 5), for example, introduces a number of network models for which it is assumed that *all* units are memoryless, so that while, in particular, his 'skew-symmetric network' has an apparently identical structure to the REC model, the mapping it produces is wholly different.

4.5 Practical minimisation techniques

Before attempting to use the REC network in any experiments, we need to address the practical issue of how it is activated. The algorithm set out in Section 4.2 performs gradient descent on the reconstruction error, with the rate controlled by the parameter μ. However, simple gradient descent with a fixed rate μ is a notoriously bad minimisation technique, as it tends to be unstable if μ is too large, and slow if μ is too small (see, for example, Bishop, 1995, p. 264).

So while it is interesting that a simple dynamic model as set out above minimises the reconstruction error, it may be useful for practical reasons to seek more efficient means to achieve the same goal.

We should note first that we are dealing at the moment with a purely linear model, and looking for a *least squares* solution (*i.e.* the minimum of $||\mathbf{W}^{\mathrm{T}}\mathbf{a} - \mathbf{x}||^2$) — this is something that linear algebra is well qualified to provide us with. In the simplest case, where the number of outputs n is equal to the number of inputs m, and all weight vectors (rows of $\mathbf{W}$) are linearly independent, we can solve the problem

$$\mathbf{x} = \mathbf{W}^{\mathrm{T}}\mathbf{a}$$

for $\mathbf{a}$ exactly by computing

$$\mathbf{a} = (\mathbf{W}^{\mathrm{T}})^{-1}\mathbf{x}. \tag{4.7}$$

When the rows of $\mathbf{W}$ are still linearly independent, but $n < m$, the least squares solution is given by use of the so-called 'normal equations' (Strang, 1988, p. 156), which yield

$$\mathbf{a} = (\mathbf{W}\mathbf{W}^{\mathrm{T}})^{-1}\mathbf{W}\mathbf{x}. \tag{4.8}$$

In all other cases, *i.e.* when the rows of $\mathbf{W}$ are linearly dependent (as must be the case when $n > m$), a useful solution may still be obtained using the *pseudoinverse* of $\mathbf{W}^{\mathrm{T}}$, denoted as $(\mathbf{W}^{\mathrm{T}})^{+}$:

$$\mathbf{a} = (\mathbf{W}^{\mathrm{T}})^{+}\mathbf{x}. \tag{4.9}$$

The pseudoinverse can be calculated directly from the singular value decomposition of a matrix (Strang, 1988, p. 449), and resolves the underdetermined nature of the problem in this case by finding the *minimum length* vector $\mathbf{a}$ that fulfils the least squares criterion.

Equations (4.7), (4.8) and (4.9) between them produce least squares solutions for all possible forms of the weight matrix $\mathbf{W}$. In fact the pseudoinverse alone covers all three eventualities, and since it can be obtained using a robust and efficient algorithm, it might seem that we need look no further for a suitable optimisation method.

Unfortunately, however, almost all of the networks used in this and subsequent chapters are not entirely linear, with constraints being applied either by hard limits on the output values, or in the form of penalty functions. In these circumstances, more general minimisation techniques are required. The linear solutions were introduced here, however, in order to note that if there *is* an opportunity to use them, they will give a highly efficient method for determining the network outputs. In particular we may note that for a fixed set of weights, the pseudoinverse need be calculated only once, with subsequent network activation being no more costly than a matrix multiplication. Problems may arise, however, when the weight matrix is *ill-conditioned*, a point discussed further in Chapter 7.

Most general-purpose minimisers, such as may be found in books on numerical analysis, should be applicable to the network activation process. Except where otherwise noted, all of the simulations described in this thesis used an efficient conjugate gradient method due to David Mackay,[2] or, where hard limits on output values were required, a constrained variable metric (or *quasi-Newton*) method due to Gill and Murray (1976) and available as routine E04KA in the NAG Fortran library (Numerical Algorithms Group, 1996). The dynamic network model itself will not be completely abandoned, however, and we shall see in Chapter 7 that in certain cases the approximations obtained by just a few iterations of gradient descent are sufficient to produce good results.

4.6 A simple activation experiment

We looked in Section 3.7.3 at the 'EXIN' network developed by Marshall (1995) in order to tackle the problems of context, uncertainty, multiplicity and scale in a perceptual environment. As the basis for a number of experiments to demonstrate the network's behaviour in these respects, Marshall used a simple set of binary-coded inputs. The code has six bits, and can conveniently be notated by assigning a letter (*a* to *f*) to each of the bits and denoting the presence of each '1' by the corresponding letter, so that, for example, the pattern '111000' is labelled '*abc*.' In the simplest example, there are just six distinct patterns — *a*, *ab*, *abc*, *cd*, *de*, and *def*.

The aim of this experiment was to see how well the REC network performs in comparison with the method for activating the EXIN network, and with regard to the requirements for a self-organising system set out by Marshall.

A network of six inputs and six outputs was used here, and in order to concentrate solely on the activation, the weights were fixed in advance, with each output unit's weight vector $\mathbf{w}_i$ set equal to a different one of the six patterns listed above. In this configuration, the network was presented in turn with each of the sixty-four possible binary input patterns. Crucially, the network's outputs were constrained to be non-negative. This constraint will be discussed in more detail in Section 5.2, but for now we can justify its use by saying that, in the framework that the problem is posed, there is no natural interpretation for negative coefficients of the underlying features.

The sixty-four different patterns, and the corresponding network outputs, are shown in Figure 4.3. The results demonstrate that the network behaves sensibly in respect of Mar-

[2]Code available via the World-Wide Web from `http://wol.ra.phy.cam.ac.uk/mackay/`

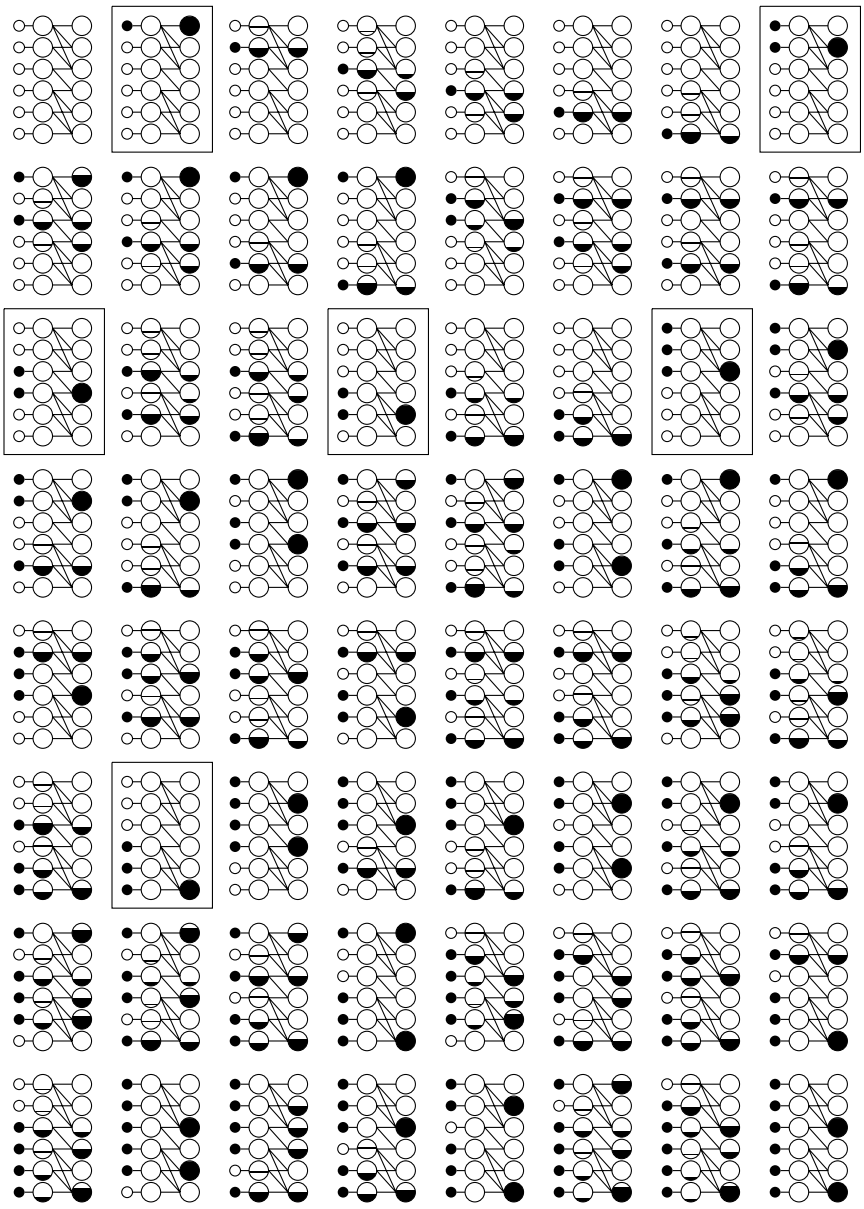


Figure 4.3: The responses of a 'pre-wired' REC network to all possible six-digit binary input patterns. Each of the 64 subfigures is a representation of the network in its stable state for the corresponding input pattern. The six small circles on the left of each subfigure represent the inputs **x**, the larger ones in the middle the first-layer outputs **r**, and those on the right the network outputs **a**. An empty circle denotes a zero value. Positive values (in the range $(0, 1]$) are represented by the fractional area of each circle that is filled. The magnitudes of negative values are similarly represented, but with a horizontal line indicating the extent of the 'filled' area below it. The lines connecting the larger circles represent (forward) weight values of 1. Where there is no line, the corresponding weight value is 0. The boxed subfigures denote the six patterns that are the pre-wired 'features.'

shall's four requirements:

1. Context. The network's outputs are context-sensitive, as required, in the sense that an output may not respond at all, even if its preferred feature appears in the input, if another unit produces a better match. The pattern *ab* (row 1, column 8 of Figure 4.3), for example, results in an output from the '*ab*-detector' alone. Neither the *a* nor the *abc* units produce any response.

2. Uncertainty. When no exact match is possible, the network produces partial responses from related units, so that, for example, the input pattern *c* (row 1, column 4) results in a partial response from the *abc* and *cd* units.

3. Multiplicity. Where the input is the (linear) superposition of two features, both of the corresponding units become active, resulting, for example, in full responses from both the *a* and *cd* units for the input pattern *acd* (row 4, column 3). Note, however, that in a binary environment such as this, we might expect the superposition of two *overlapping* features, such as *cd* and *de* to result in their logical OR (*cde*). This issue is tackled in Chapter 6.

4. Scale. The response of output units is independent of the scale of the features they represent, so that, for example, the 'small' feature *a* produces exactly the same response from the *a*-detector (row 1, column 2) as the 'large' feature *abc* does from *its* detector (row 3, column 7).

Recalling that the outputs from the first layer of processing units represent the 'residual' (difference between input and reconstruction), we can see for each pattern how close the network was to representing it fully. Without the non-negative constraint on the outputs, the network would have been able to reconstruct every pattern exactly, since the six weight vectors are linearly independent, and therefore span the entire six-dimensional input space. Furthermore, as noted in Section 4.5, we could have achieved such a mapping with an entirely feedforward linear network with a weight matrix the inverse of that used here. While this would result in the same output values for any pattern that resulted in a zero residual in Figure 4.3, every other pattern would produce one or more negative output values. The input *b*, for example, is interpreted by the constrained network (row 1, column 2 of Figure 4.3) as a partial *ab* (plus noise), but as *ab* minus *a* when negative coefficients are allowed.

The activation of the REC network, therefore, based on a relatively simple model, and equivalent to a constrained SSE minimisation, appears to fulfil Marshall's requirements for a flexible and general-purpose self-organising system. As yet, however, we have no means of achieving self-organising behaviour by updating the weight values — this issue is addressed in the next section.

4.7 Learning rules

The activation process finds the output values that minimise the reconstruction error E_r for a fixed set of weights and a particular input vector. A reasonable goal for learning is to

minimise the same measure, but now averaged over all input patterns, *i.e.*

$$E_R = \mathcal{E}[||\mathbf{x} - \widetilde{\mathbf{x}}||^2]. \tag{4.10}$$

Differentiating E_R with respect to the individual weight values w_{ij}, we find that

$$\begin{aligned} \frac{\partial E_R}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \mathcal{E}[||\mathbf{x} - \widetilde{\mathbf{x}}||^2] \\ &= -2\,\mathcal{E}\left[(\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}} \frac{\partial \widetilde{\mathbf{x}}}{\partial w_{ij}}\right]. \end{aligned} \tag{4.11}$$

Differentiating each element of $\widetilde{\mathbf{x}}$ in (4.4) gives

$$\frac{\partial \widetilde{x}_k}{\partial w_{ij}} = \frac{\partial a_i}{\partial w_{ij}} w_{ik} + \delta_{jk} a_i. \tag{4.12}$$

Substituting (4.12) into (4.11), we find that

$$\frac{\partial E_R}{\partial w_{ij}} = -2\,\mathcal{E}\left[\frac{\partial a_i}{\partial w_{ij}} (\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}} \mathbf{w}_i + (x_j - \widetilde{x}_j).a_i\right]. \tag{4.13}$$

But we know from (4.6) that $(\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}} \mathbf{w}_i = 0$ as a result of the activation process,[3] so that (4.13) reduces to

$$\begin{aligned} \frac{\partial E_R}{\partial w_{ij}} &= -2\,\mathcal{E}[(x_j - \widetilde{x}_j).a_i] \\ &= -2\,\mathcal{E}[r_j a_i]. \end{aligned} \tag{4.14}$$

Converting this gradient into an 'online' learning rule in the usual way (as discussed in Section 3.2.1) gives:

$$\Delta \mathbf{w}_i = \eta\, a_i \mathbf{r}, \tag{4.15}$$

where η is the learning rate. This is the Hebbian learning rule in its simplest form, based on correlations between the outputs of the first- and second-layer units (*c.f.* Equation 3.12). Conceptually, this means that during learning, any active unit will attempt to take on responsibility for any 'unexplained' part of the input (as represented by the residual vector $\mathbf{r}$) in proportion to its activity. Mathematically, we see from (4.14) that it performs stochastic gradient descent on the overall reconstruction error.

We have assumed in this derivation that, for each input pattern, the outputs have reached their stable values before weight updates are calculated. This approach is justifiable when considering the network as a single dynamic system since, as mentioned in Section 4.1, it is usual to assume that the learning process occurs at a much slower rate than the activation. Implicit in this whole approach, however, is the assumption that input is in the form of a series of discrete patterns, each held stable for long enough for the activation and weight update processes to complete. Some of the issues related to extending these techniques to continuous temporal patterns are discussed in Section 8.3.

[3]The steps leading to this simplification have been given in full here for our particular choice of error function (4.10). Appendix A shows that this is one case of a property that holds more generally, a result that will be useful when considering modifications to the REC network in Chapters 5 and 6.

Hebbian learning, as given by (4.15), is not the only possible form of learning rule for the REC network. A slight variant may be more appropriate, for example, in environments that are primarily *single-cause*, *i.e.* where the majority of patterns are assumed to be produced by the presence of a single feature. This is the assumption made by winner-take-all (WTA) networks (Section 3.5) and most *clustering* algorithms. In this case we may use a 'greedy' learning rule where an active unit attempts to claim the entire input pattern:

$$\Delta \mathbf{w}_i = \eta \, a_i (\mathbf{x} - \mathbf{w}_i). \tag{4.16}$$

The relationship to learning in WTA networks is easily seen because (4.16) is equivalent to (4.15) exactly in the case where just one second-layer unit is active with an output of one. Although the learning rule in this case is similar to that of WTA networks, in using it we still retain the flexibility of the REC network fully to represent multiple or ambiguous patterns should they appear.

Some simple experiments with a rule of this type appear in Harpur and Prager (1994). Since, however, the primary aim of this work is to find *multiple*-cause models, the learning rule (4.16) will not be explored further here.

4.8 Novelty detection

Some neural network models, such as Kohonen's (1989) 'novelty filter,' are intended primarily to signal the presence of patterns that are significantly different from those the system has previously learnt. In the REC network, the magnitude of the residual $\mathbf{r}$, as determined by the stable-state responses of the first-layer units, may be used as a measure of 'novelty,' since a large response indicates that the network's current weight vectors are not well-matched to the features present in the input. Simple examples of this may be seen in Figure 4.3. The magnitude of $\mathbf{r}$ is given directly by the reconstruction error E_r (4.1), a value that will normally be available 'for free' if a general-purpose optimiser has been used to activate the network.

If the error E_r exceeds some threshold ϵ, then we could say that the network has *failed* to represent the input pattern. A simple application of this could be that if $E_r > \epsilon$, then instead of the making the normal small adjustments to weight values via the learning rule, the weight vector for some *uncommitted* output unit (*i.e.* one that has shown very little activity in the past) is set equal to the entire current residual $\mathbf{r}$, or in the case of 'greedy' learning (4.16) to the entire input pattern $\mathbf{x}$. This scheme could potentially speed up the learning process by having weight vectors 'jump' directly to plausible values, and could also help where learning has become stuck in a local minimum. The value $1/\epsilon$ plays a very similar role to the *vigilance* parameter in Adaptive Resonance Theory (ART) networks (Carpenter and Grossberg, 1987).

Although this technique has been used successfully in several experiments with the REC model to speed up the learning process, it is not explored further here because it is essentially an *ad hoc* method which, though potentially useful, does not help with our understanding of the underlying algorithm. However we shall return briefly to this issue in Section 8.3.

4.9 Relationship to some previous models

We looked in the previous chapter at a number of self-organising models, many of which have been the subject of a considerable amount of research. It is reasonable therefore to ask how the model presented here contrasts with those developed previously. To this end, we may note a number of comparisons between the REC network and those discussed in Chapter 3:

Single Hebbian units. We saw in Section 3.2.1 that Hebbian learning gives a simple, local means of adjusting weights. We have been able to retain this form of learning with the REC model. Furthermore, while in the purely feedforward case Hebbian learning has the problem of the weights increasing without bound, the REC network avoids this because the weight updates (4.15) are based on an error signal rather than the raw input.

Oja's rule. In order to deal with the problem of unbounded weights, Oja's rule introduces an extra term whose sole purpose is to produce normalised weight vectors. As already noted, the REC network does not in general need to bound the weights in this way, but if in particular cases such constraints *are* required, there is no reason why Oja's rule (or some other method for weight normalisation) should not be applied.

PCA networks. A number of networks that find the principal components of their inputs were discussed in Section 3.2.2. We saw in Section 3.1 that the principal components are orthogonal, so these networks correspondingly ensure (by various means) that their weight vectors are orthogonal at convergence. For a more general feature-detecting network, it is important that this constraint be removed, and we have seen that the REC network is able to minimise reconstruction error *without* the requirement for orthogonal weight vectors.

Principal subspace networks. There are a number of self-organising networks, such as those proposed by Plumbley (1991), that find a (potentially non-orthogonal) principal subspace of their inputs. The REC network, by virtue of its error function, *is* such a network, but the crucial difference to previous models is that the REC network attempts to minimise error at both the activation and learning stages. This has no great benefit in the unconstrained linear case, but will become important when constraints are applied in the experiments of Section 4.12 and in Chapter 5.

MLP auto-encoders. The linear MLP auto-encoder, as described in Section 3.2.3, finds the principal subspace of its inputs. The non-linear generalisation of this (Section 3.2.3) performs an analogous function, but no longer assumes a linear generative model for the data. In both of these models, however, the learning procedure is made more complex by the fact that the network is required to learn weights for both the generative model and its inverse (the recognition model) simultaneously. The REC network, by contrast, only needs to learn generative weights, which is in general the easier problem, with the recognition model being supplied 'on the fly' by the activation process. A useful side-effect of this is that in the REC model, *all* of the weights relate directly

to the features represented by the network, and furthermore each feature is localised as the weight vector of a single output unit. In contrast to the auto-encoder model, therefore, a new feature may be added without altering any of the existing weights.

The SEC network. The Symmetric Error Correction network (Section 3.2.4) has an almost identical structure to the REC architecture (as shown in Figure 4.2). The crucial difference is the extra generality of the REC model resulting from the use of dynamic output units and a recurrent activation procedure.

WTA networks. A 'winner-take-all' procedure forms the basis of several self-organising systems (Section 3.5). As compared with this form of activation, the REC network is able to represent patterns where *multiple causes* have combined to produce the data. This ability is likely to be important, particularly in the early stages of processing complex data, to avoid the loss of significant amounts of information without the need for very large numbers of units.

Lateral inhibition networks. One method of making more flexible the very strong competition inherent in WTA networks is to introduce (adjustable) lateral inhibition between units, as discussed in Section 3.5. Commonly the weight values for the lateral connections are determined by an anti-Hebbian rule. It is clear that the need to learn an extra set of weight values increases the complexity of training. In Section 4.3 it was argued that the REC model obtains (via feedback) just the right amount of inhibition without the need for these extra parameters. It was also mentioned however that it is possible to think of the REC network's operation in terms of lateral inhibition rather than feedback if so desired.

EXIN networks. A comparison of the REC network to various aspects of the EXIN model (Section 3.7.3) was made in Section 4.6, because several of the principles embodied by the model are highly desirable for self-organising networks. The properties of the two in respect of activation were found to be very similar, and indeed the REC model was originally developed by this author as an attempt to mimic EXIN networks (Harpur and Prager, 1994). The principal advantages of the REC network are firstly that there is a clear objective function, and secondly that activation is less computationally expensive.

'ICA' networks. Various models that make use of higher order statistics were discussed in Section 3.8.2. Several of these require their data to be sphered because, as extensions to PCA networks, they retain the constraint that their weight vectors must be orthogonal at convergence. Some limitations of this approach were pointed out in Section 3.8.1. The REC network does not have the orthogonal constraint, but nor does it yet have any ability to deal with higher order statistics. This issue will be tackled in Chapter 5, at which point it will be possible to make a fuller comparison of the REC model with these and other ICA techniques.

4.10 Minimum reconstruction error as maximum likelihood

We have seen above that activation of the REC network finds a coding that, when used with the current weight values to linearly reconstruct the input, minimises the sum of the squares of differences between the original and reconstructed input values (4.5). Equivalently, it finds the reconstruction that is the *closest* (in terms of Euclidean distance in the input space) to the original input.

This is clearly useful, but what in theoretical terms does it achieve? Let us assume that the network is a 'correct' model of the environment, *i.e.* that both the weight values and the linear model itself exactly fit the way that patterns are produced. Under these (somewhat fanciful) assumptions, the network will be able to reconstruct *exactly* every pattern presented to it. In a slightly more realistic model, we may assume that the input has been additively corrupted by a random 'noise' vector $\boldsymbol{\nu}$, giving

$$\mathbf{x} = \hat{\mathbf{x}} + \boldsymbol{\nu} \tag{4.17}$$

with the signal $\hat{\mathbf{x}}$ and noise $\boldsymbol{\nu}$ independent. Since the network models $\hat{\mathbf{x}}$ but not $\boldsymbol{\nu}$, it can no longer be guaranteed to produce an exact reconstruction of the input, but what we can ask is that the reconstruction $\widetilde{\mathbf{x}}$ be set to the *most likely* value of $\hat{\mathbf{x}}$ that could have resulted in input $\mathbf{x}$, *i.e.*

$$\widetilde{\mathbf{x}} = \arg\max_{\hat{\mathbf{x}}} p(\mathbf{x}|\hat{\mathbf{x}}). \tag{4.18}$$

This is known as a *maximum likelihood* approach. Using the relationship (4.17) along with its independence assumption, it is not too hard to see that $p(\mathbf{x}|\hat{\mathbf{x}})$ is in fact given by $p(\boldsymbol{\nu})$. In other words, given a point $\hat{\mathbf{x}}$, we can place the probability density function of $\boldsymbol{\nu}$ at that point to find the distribution of input signals $\mathbf{x}$ that might result.

In the absence of any other information, the usual assumption to make about $\boldsymbol{\nu}$ is that it has a zero-mean multivariate Gaussian distribution which is *spherical*, *i.e.* its variance is the same in any direction. The p.d.f. of such a distribution is shown for the two-dimensional case in Figure 4.4, where we see that the spherical property results in concentric circular contours and the p.d.f. may therefore be viewed purely as a function of Euclidean distance from the origin.[4]

Placing this density at points $\hat{\mathbf{x}}$, we see that the value of $\hat{\mathbf{x}}$ that maximises $p(\mathbf{x}|\hat{\mathbf{x}})$ is simply $\mathbf{x}$, but *only* if $\mathbf{x}$ is in the subspace of points generated by the noise-free environment (model). Failing this it is clear from the form of Figure 4.4 that we should find the reconstruction $\widetilde{\mathbf{x}}$ *closest* to $\mathbf{x}$ within the constraints of the model. This, as set out at the beginning of this section, is precisely what is achieved by the activation algorithm, so we can say that the network does indeed (under our various assumptions) fulfil the maximum likelihood criterion (4.18).

Since the network's outputs $\mathbf{a}$ and reconstruction $\widetilde{\mathbf{x}}$ are deterministically related (we are assuming a noise-free system), we can equally say that the network finds the outputs $\mathbf{a}$ that

[4] Although we are specifically assuming a Gaussian noise distribution, any multi-dimensional distribution whose p.d.f. is a monotonically decreasing function of distance from the origin would fulfil the property needed here. If, however, we also wish to assume that the noise in each input dimension is independent and identically distributed, then Gaussian noise is the only choice that fulfils both criteria.

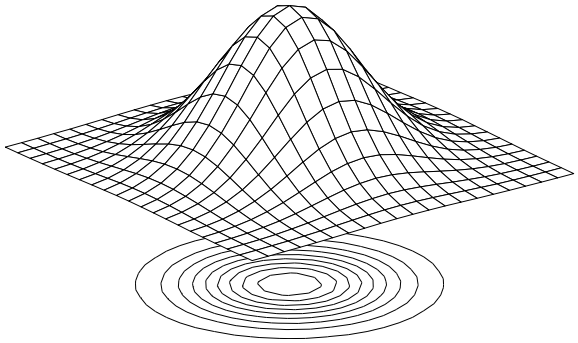


Figure 4.4: A combined surface and contour plot of the spherical Gaussian function in two dimensions.

maximise $p(\mathbf{x}|\mathbf{a})$. This is also equivalent to maximising $p(\mathbf{a}|\mathbf{x})$, but *only* under the assumption that all output values $\mathbf{a}$ are equally likely. We shall return to this matter with a slightly more formal treatment in Section 5.4.

4.11 Reconstruction error and information transfer

In the information-theoretic framework set out in Chapter 2, we saw that maximising information transfer is a desirable property for an unsupervised network. Indeed, it is also a *necessary* property if we are to try to produce low entropy codes, since minimising output entropy without simultaneously maximising information transfer is a pointless task (Figure 2.3).

It has been known for some time that it is possible to interpret a least-squares error function in terms of maximum entropy (van den Bos, 1971). Plumbley (1993) examines this area in some detail, and shows in particular that minimising squared reconstruction error in an unsupervised system is equivalent to minimising an upper bound on the *information loss* with respect to the 'true' signal $\hat{\mathbf{x}}$. In our case, the loss may be written as $\Delta I_{\hat{\mathbf{x}}}$, where

$$\Delta I_{\hat{\mathbf{x}}}(\mathbf{x}, \mathbf{a}) \equiv I(\hat{\mathbf{x}}; \mathbf{x}) - I(\hat{\mathbf{x}}; \mathbf{a}).$$

for input $\mathbf{x}$ and output $\mathbf{a}$. The bound is only tight when signal and noise are Gaussian. Although the signals under consideration will in general be highly *non*-Gaussian, this need not worry us unduly, since we shall be interested primarily in the case where the expected reconstruction error E_R is zero. If this is the case, then the mapping from input to output is *invertible*, and it is clear that the information loss is zero whatever the input distribution may be.

To be slightly more general, providing that the useful input information $I(\hat{\mathbf{x}}; \mathbf{x})$ is contained within an n-dimensional principal subspace of the input, then a REC network with n outputs (or more) will minimise information loss. Since we are interested primarily in feature extraction rather than dimensionality reduction, the number of outputs in examples described here will be made sufficiently large for this to be at least a reasonable assumption.

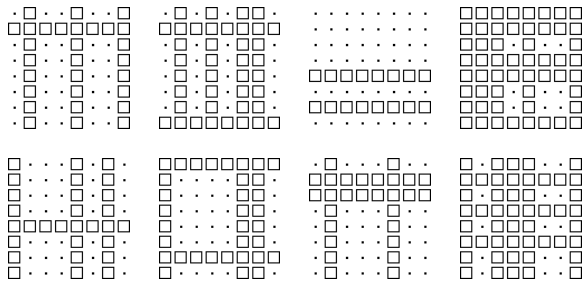


Figure 4.5: A random sample from the lines data set for $p = 0.3$. Values of one are represented by white squares, and zeroes by dots.

4.12 Learning experiments

In the following sections, we shall look at the results of a number of experiments using the basic REC network as outlined in this chapter. The data sets used here are all artificial, and designed to illustrate the network's abilities and limitations with respect to certain simple problems. This will pave the way for enhancing the techniques in subsequent chapters, and, in Chapter 7, applying them to real-world data.

Certain properties are common to the majority of experiments described in this thesis, and so are set out here to avoid the need for repetition. Firstly, except where otherwise noted, the initial weight values for each experiment were produced randomly from a uniform distribution over the permitted range of values, or where the values are unconstrained, from the range $[-0.5, 0.5]$. Secondly, changes to the weight values were in all cases made using the online Hebbian learning rule (4.15), typically, and again unless otherwise noted, with any changes being applied immediately after the presentation of each pattern. Finally, where a data set of finite size was used, each input pattern was generated from an independent random sample from the set (with replacement).

4.12.1 The lines problem

An experiment involving horizontal and vertical lines was first introduced by Rumelhart and Zipser (1985). This was extended by Földiák (1990) to the case where multiple lines may appear simultaneously, and this version has subsequently been used by Zemel (1993), Saund (1995) and Ghahramani (1995), and is thus established as a simple benchmark for a system claiming to produce factorial codes. Input consists of an 8×8 grid, on which are placed combinations of the 16 possible horizontal and vertical lines, each one appearing with an independent probability p. At a point on the grid where there are one or more lines, the input is one — elsewhere it is zero. There are 65 026 distinct patterns in the data set, but for any single value of p the majority of these will appear only extremely rarely.

In this experiment, the line probability p was set at 0.3, giving a mean of 4.8 superimposed lines per image. A random sample from the data set for this value of p is shown in Figure 4.5. Figure 4.6 shows a typical set of weights developed by a network with 16 outputs when presented with randomly generated samples from this data set. The outputs were constrained to be non-negative during activation, and weight values were constrained to lie within the range $[0, 1]$ during learning. Results such as those shown here were typically ob-

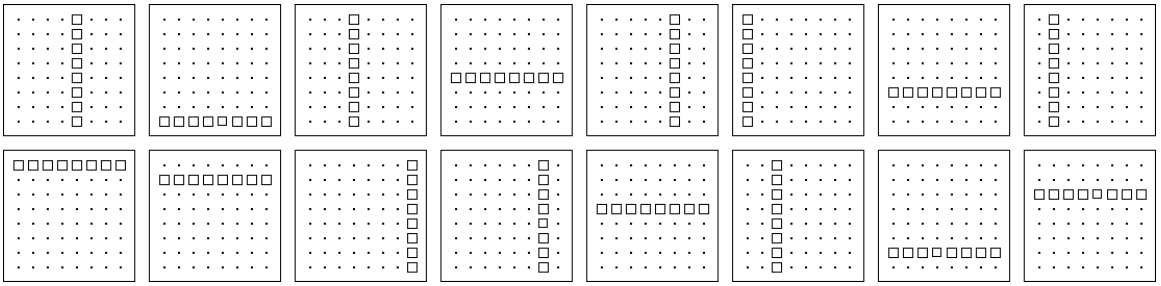


Figure 4.6: An example of the final weight values produced by a REC network of sixteen outputs trained on the lines data. In this and subsequent diagrams of this type, positive values are represented by white squares, with the length of each side proportional to the magnitude of the value. Negative values are similarly represented by black squares, and zeroes by dots.

tained after about 2000 pattern presentations with a learning rate η of 0.2, but the value for η is not critical, and results of this form can be obtained with any rate in the range $(0, 0.9]$.

It may be argued that using just 16 network outputs in an experiment like this is not a reasonable test, since it makes use of the foreknowledge that there are exactly 16 independent components to the input — we are unlikely to have such knowledge when tackling real-world problems. The experiment was repeated with different numbers of outputs, ranging from 8 to 64. With insufficient (< 16) outputs, it was found that each unit typically continued primarily to represent a single horizontal or vertical line, but the weight values also contained traces (typically with weight values of less than 0.5) of the 'missing' lines that were not properly represented. With excess (> 16) outputs, it was found in all cases that 16 of the units continued correctly to discover each of the horizontal and vertical lines. Some of the remainder represented combinations of two or more lines, or 'noisy' versions of this, with the rest remaining essentially unused with weights close to their initial random values.

It is worth noting that the linear model of the network is not correct for this data, since the input value where two lines intersect is one, and not two as the network's generative model predicts. Providing that p is kept reasonably small (less than about 0.4) the basic network is nevertheless able to find the correct solution because the reconstruction errors produced at the intersections are small compared with the overall variance of the input. In Chapter 6 modifications are introduced that enable the network to continue to find solutions for much higher values of p.

One effect of the incorrect model being used here is that weight values do not fully converge due to the residual errors produced at the intersection of lines. However the fluctuations may easily be smoothed out by decreasing the learning rate during the latter stages of training.

4.12.2 The shapes problem

This experiment is based on a data set similar to that used by Ghahramani (1995). Input patterns are composed of three shapes — an empty square, a cross, and a diagonal line — each of which fits into a 3×3 grid and is represented by ones on a background of zeroes. An input pattern is produced by combining one of each of the shapes, each at one of sixteen

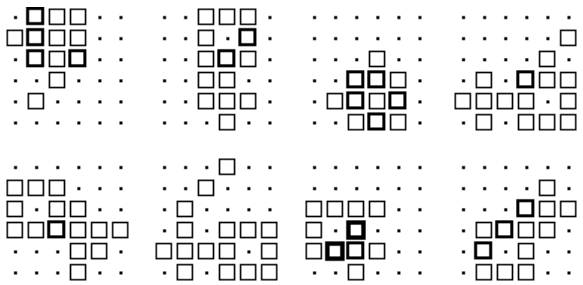


Figure 4.7: A random sample from the shapes data set. Zero inputs are shown as a dot, as previously. Input values of 1, 2 and 3 are shown by boxes with single, double and treble thickness lines respectively.

possible locations on a 6×6 grid. Unlike the previous experiment (and Ghahramini's data set), the shapes are combined linearly, so that a superposition of two shapes produces an input of two, and of all three an input of three. Patterns were generated at random from the 4096 possible combinations. A sample of the input patterns is shown in Figure 4.7. The network outputs were constrained to be non-negative, and weight values to lie in the range $[0, 1]$. The learning rate η was fixed at 0.8.

A typical set of weight vectors produced by a network of 64 outputs after approximately 1000 pattern presentations is shown in Figure 4.8. Note that the weights have been reordered manually for clarity, the original order being essentially random since there is no enforced structure to the output layer. A study of these weight values shows that the network has correctly identified each of the three shapes in each of the sixteen locations. The remaining sixteen units were unused, and remained inactive for any pattern from the data set once the stable state shown here was reached, leaving them free to represent any additional features that might be introduced to the data at a later stage. Their weights are in some instances very close to the random starting values. In others, the unit was used at some stage during learning, but then abandoned when some other unit or units began to represent the same feature or features more accurately.

While similar to the lines experiment in a number of ways, there are also several important differences here. Unlike the lines data set, the features are of different sizes (three pixels for a line, five for a cross, seven for a square). A learning rule based on SSE will tend to favour features of greater magnitude, since they will induce a greater error if not properly represented. In accordance with this, it was observed that the weight vectors representing squares tended to be the first to appear during learning, followed by the crosses and finally the lines. The issue of most importance, however, as discussed in Section 3.7.3, is that the network be able to learn to represent features with a variety of scales — this experiment provided a simple test of that ability.

Secondly, we may note that in this experiment, the network has 64 outputs but only 36 inputs. Given this, there are many possible sets of weights that could represent every input pattern with zero error (even with the non-negative restriction on both output and weight values). A situation where any 36 of the 64 weight vectors represent a different input pixel would achieve this, for example, so why was it that in all the trials the learning procedure nevertheless yielded what we would interpret as the 'correct' representation? It seems likely

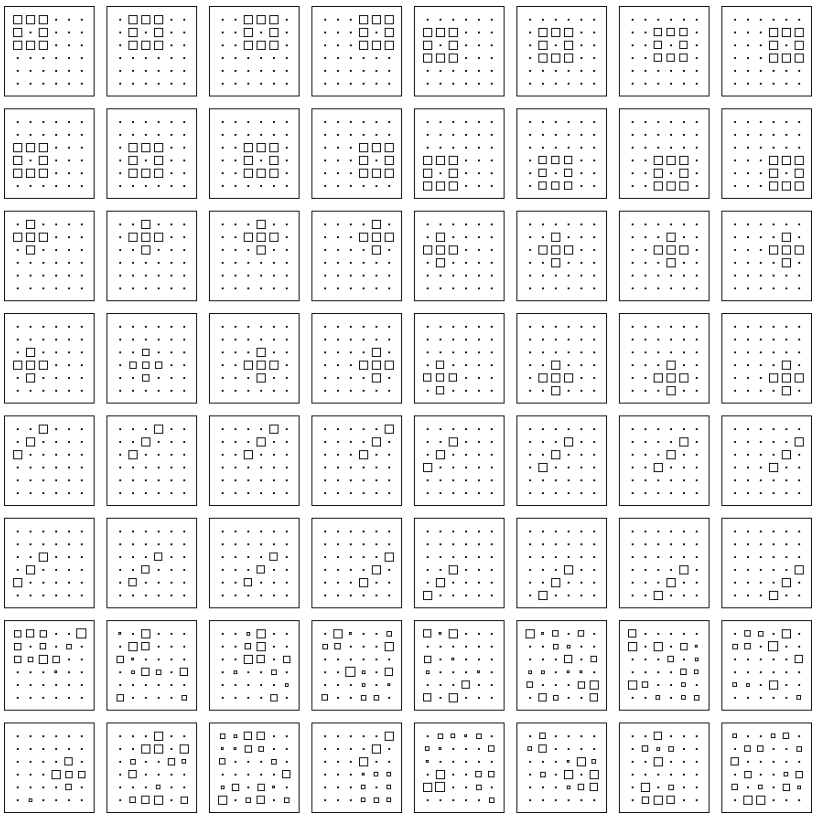


Figure 4.8: The final weight values from the shapes experiment. The order of the weight vectors has been changed by hand to show more clearly the divisions into squares, crosses, lines and unused units.

that this is due to the dynamics of the learning rule: during learning, as noted in Section 4.7, every active unit will receive a change in weight values that attempts to account for any part of the input that has not been properly represented, as given by the residual vector **r**. Referring back to Figure 4.3 on page 49, we see that in almost every case where a pattern is not represented fully, and because of the non-negative output constraint, an active input value x_i has a corresponding positive residual r_i. This results in a pressure for each feature to become as large as possible unless there is evidence to the contrary. This is a highly desirable property, and is analogous to the idea in PCA that the high-variance components are of the most interest. It also explains why the final non-zero weight values tended to be close to their upper limit of one even though in theory any value would suffice. Nevertheless the fact that the network might fail to find the correct solution if it happened to be initialised in or near to an unwanted minimum is a cause for concern — attempts to remove such minima will be made in Chapter 5.

Finally, it was important for this experiment that the data were formed from a linear rather than a binary combination of the features. Unlike the case of the previous lines experiment (Section 4.12.1), the degree of overlap between features in a binary-valued version of the data was sufficient to prevent the correct solution from being found. In experiments with these data, the network converged to a state where most of the squares and some of the crosses were found correctly, but where the majority of line features were only of length two.

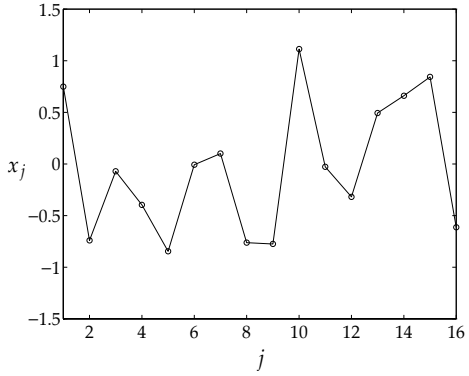
4.12.3 Learning continuous-valued features

In the problems presented so far, the features have been binary in nature. The REC network architecture has no particular predisposition towards binary problems, so in some further experiments described here it was tested on patterns from a continuous domain.

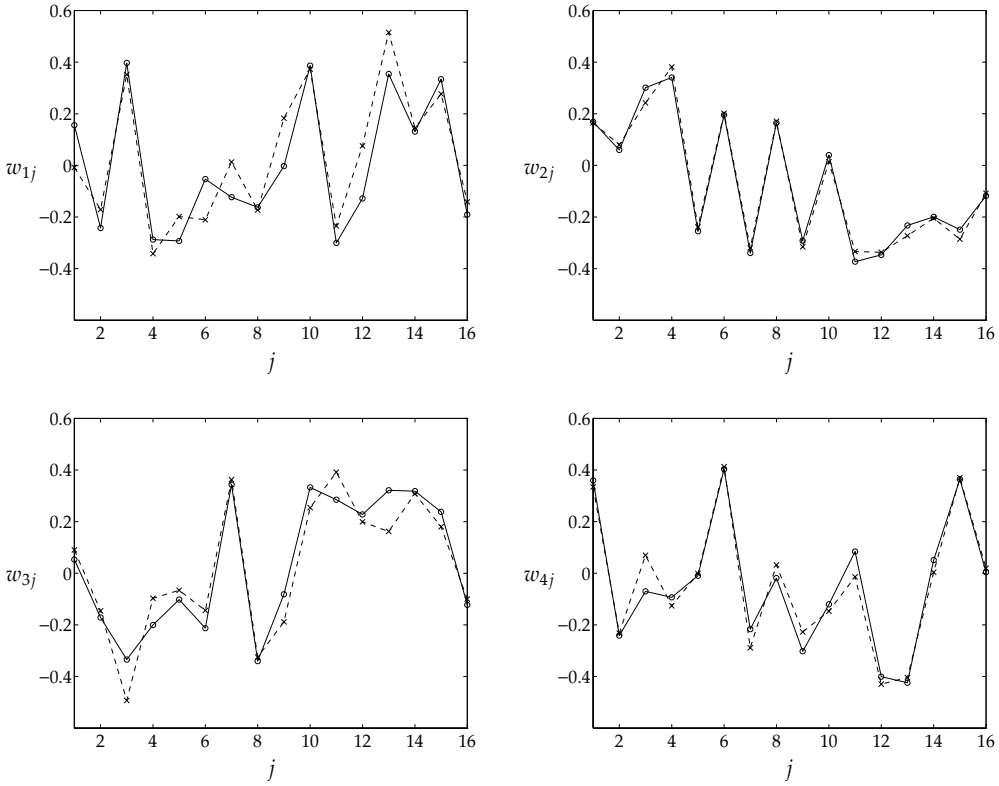
The particular problem used was based on finding four features in sixteen dimensions. Noting that in the previous 'toy' problems the choice of features was made by hand and may thus have introduced regularities that simplified the problem, the features here were chosen at random. Each was composed of sixteen values taken from a uniform distribution over the interval $[-1, 1)$. Training patterns were generated using linear combinations of four such features, produced by multiplying each by uniformly distributed random numbers over the interval $[0, 1)$ and summing the results.

A network with four output units was trained on these patterns. In order to apply the prior knowledge that the features appear only in a non-negative form, the network outputs were similarly constrained to be non-negative. The weights were initialised to random values and updated after each pattern with a learning rate η of 1. After an update, each output unit's weight vector was normalised to unit length. A reasonably stable solution was typically reached after about 1000 pattern presentations, although small changes to the weights continued after this. Figure 4.9(a) shows a sample pattern from the data set, and (b) shows the four randomly generated features, superimposed with the approximations to them found by a network after 2000 pattern presentations.

There are several points to note from this experiment. Firstly, if the network outputs are not forced to be non-negative, then the problem is vastly under-constrained because *any* four



(a)



(b)

Figure 4.9: (a) A sample pattern from the data set described in Section 4.12.3, with the values x_j plotted for each input j. (b) The four randomly generated 'features' used to generate this data set, and the approximations to them produced by the network on a sample run. The circles (connected by solid lines) denote the underlying features, and the crosses (connected by dashed lines) the network weights approximating them. The feature vectors have been normalised to unit length, and ordered so that each appears on the same axes as the most closely matching weight vector. The order of the sixteen inputs has no particular relevance — the lines connecting the points in these graphs are included merely for illustrative purposes.

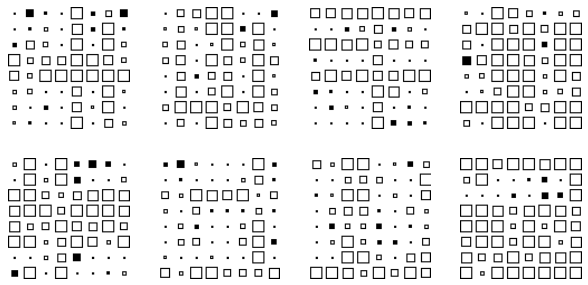


Figure 4.10: A random sample from the noisy lines data set for $p = 0.3$ and $\sigma_\nu^2 = 0.1$. Values are represented as described in the caption for Figure 4.6.

vectors spanning the same space as that spanned by the four 'true' features will encode all possible samples from the data set with zero error. In practice this results in the network quickly converging to a stable state where the weight vectors have no easily recognisable relationship to the original features.

Secondly, we note from Figure 4.9(b) that the weight vectors' match to the underlying features is fairly good, but certainly not exact, even though the values shown represent a stable solution and both data and network are 'noise-free' (down to the tolerances of the floating point representation and minimisation procedure). This suggests that the problem is still under-constrained.

Thirdly, it was noticed that training with relatively large learning rates, which produce large fluctuations in weight values at an early stage of training, tend to produce better stable-state solutions than smaller ones, for which the path to a stable solution is more direct. This also suggests the presence of undesired solutions, and hence that the choice of initial weights may be important.

A more detailed look at the form of problem being solved here will be made in Section 5.2, where we shall see that it is indeed an under-constrained problem, but that a suitable choice of initial weight values and/or the application of additional constraints can help us to find exact solutions.

4.12.4 Learning in the presence of noise

In all of the above experiments, the data sets have somewhat unrealistically been entirely noise-free. To test the performance of the REC network in noisy environments, two of the above experiments were repeated, but with noise-corrupted inputs.

In the first, the lines data set as described in Section 4.12.1 was used, but with independent Gaussian noise of variance $\sigma_\nu^2 = 0.1$ added to each element of each input pattern. Some examples of this noise-corrupted version of the data are shown in Figure 4.10.

The second used a noisy version of the continuous-valued data set of Section 4.12.3, also with additive Gaussian noise of variance 0.1. The noise variance was at a similar level to the expected variance of each element of the signal vector, which is $\frac{1}{9} \approx 0.11$.

In both cases, the networks were identical to those used for the noiseless data. Unsurprisingly one effect of the noise was found to be that learning was less stable, with the result that the learning rate needed to be gradually decreased towards zero for the weights to converge fully, a procedure formalised as the Robbins-Monro stochastic approximation

algorithm (Robbins and Monro, 1951). An alternative means of dealing with the destabilising effect of noise is to accumulate weight changes and apply them only after a number of patterns, thus taking advantage of the fact that the noise has zero mean.

Aside from this effect, the representations produced from the noisy lines data were found to be essentially identical to those in the noise-free case. A slight degradation of results was observed however for the continuous-valued features, and became worse with increasing noise variance. The reasons behind these observations will be discussed in Section 5.2.

4.13 Discussion

We have seen in this chapter the operation of the REC (Recurrent Error Correction) model, its relation to a number of other models and techniques, and the results of its application to a few relatively simple tasks.

The most significant characteristics are firstly that the network attempts to minimise its error function for any set of weights and every input pattern, and not just as the end-result of a learning process; and secondly that in contrast to PCA and a number of ICA techniques, there is no requirement for the features that are found to be orthogonal.

Having removed the orthogonality constraint, it proved necessary in the experiments described here to introduce a new one, in the form of the non-negativity of output values, to prevent the network from producing uninteresting or undesired solutions. The results from Section 4.12.3 suggested however that this will not be sufficient in all cases (nor is non-negativity justifiable for all problems), and in the next chapter we shall examine further the issue of constraints, and how they might be used to produce the most efficient codes possible within a broadly linear framework.

Chapter 5

Constraints

The previous chapter introduced a network model whose objective is to reconstruct its input from its output state. In terms of the information theoretic framework set out in Chapter 2, it aims to maximise transfer of information from input to output. What we do not yet have is a means to ensure the information at the output is represented *efficiently*, or, more specifically, a way to minimise the mutual information between separate outputs.

Entropy itself is a difficult quantity to work with, so the way in which we shall attempt to achieve this second objective is by application of *constraints* to the network. These may broadly be classified as 'hard' or 'soft,' where a hard constraint is one that cannot be violated, and a soft constraint one that merely incurs a *penalty* (introduced as an extra term in an objective function) when it cannot be met.

Any constraint will impose some limitation on the modelling capabilities of the system, and hence reflect a prior assumption about the nature of the data. So, to be of value such assumptions must represent justifiable restrictions of the model. It was argued in Chapter 3 that orthogonality is *not* a valid assumption for a feature-detecting system, and so the REC model was developed so as not to be bound by this constraint. In this chapter we shall examine various alternative means of helping the network to find the 'right' model of its environment which, it is hoped, are better justified.

5.1 Bottlenecks

The number of inputs to an unsupervised system is determined by the input data. The number of outputs, however, is an adjustable parameter of the model, but a parameter that must be fixed in advance of using most such systems,[1] including the REC network. By limiting the number of outputs, we impose a hard constraint in the form of a *bottleneck* on the output of the system; we need to be clear about the effect that this constraint will have on the codes produced.

[1] Exceptions to this are networks that are able to grow or contract according to the demands of the system. The possibility of applying such techniques to the REC network is discussed in Chapter 8.

Tight bottlenecks

If the REC network has at least as many outputs as inputs ($n \geq m$), then any set of n linearly independent weight vectors (of which there are infinitely many) will completely span the input space, giving a complete reconstruction for all possible input patterns. Since the learning rule (4.15) is driven by the residual, there is nothing in the basic REC model that favours one such representation over another.

One crude means of restricting the solutions is to impose a 'tight' bottleneck at the network output by having fewer outputs than inputs ($n < m$). The mapping *reduces the dimensionality* of the data. In such circumstances, the overall objective function (4.10) will be minimised when the weight vectors span the principal subspace of the input. Dimensionality reduction may be justified if we have reason to suspect there are fewer degrees of freedom in the input than there are separate inputs, or at least that a significant proportion of the input information is contained in a subspace of the input space.

There are still many sets of vectors that span the principal subspace however, so this restriction alone is insufficient to ensure the weights bear any close relation to the 'true' features of the input. More serious though is the fact that a restriction of this sort is in many cases unjustifiable. There may well be a large number ($> m$) of independent (or approximately independent) features in the input (the shapes data of Section 4.12.2 is a simple example). If we restrict ourselves to finding fewer than m features we shall discover only those that contribute most to the overall variance of the input, while ignoring 'smaller' or less frequently occurring components.

Loose bottlenecks

A second type of bottleneck, corresponding to a much looser constraint, occurs when there are fewer output units than distinct input patterns. This will almost always apply in complex environments, since even for purely binary inputs, there can be as many as 2^m distinct patterns. A loose bottleneck is an implicit constraint in many unsupervised systems. However, the importance of making it explicit will become clear when techniques to encourage *sparse* representations are introduced in Section 5.6. The sparsest possible information-preserving code uses a *single* active output to convey all of the input information for any particular pattern. This is exactly the type of representation produced by winner-take-all (WTA) networks, whose limitations were discussed in Section 3.5. In particular, a WTA representation is most unlikely to provide a minimum entropy solution. Bounding the maximisation of sparseness with a loose bottleneck, however, *may* enable us to find such a code.

In the light of this, choosing the number of outputs seems hard because we need enough to capture most or all of the input features, but not so many that the maximisation of sparseness leads to a representation involving combinations of independent features. In practice we may well find, however, that the gap between a plausible number of features and the number of distinct patterns is sufficiently large that an informed guess at this number will be adequate. A more principled approach might be to repeat training with successively larger networks until no significant fall (or even a rise) in the sum of individual output entropies is observed.

5.2 The non-negative constraint

We saw in Chapter 4 that forcing the outputs of the REC network to be non-negative was a useful way to constrain the space of possible solutions in a number of simple modelling problems. Constraining the outputs of a feature-detecting system in this way is justifiable in situations where there is no natural interpretation for negative quantities. This is particularly common where the input's features have a direct relationship to physical objects. To use a simply analogy, it is possible to have 3 apples, 0 apples, even 0.4 apples, but not -2 apples, so we should expect the output of an 'apple-detector' to reflect this constraint.

It is also of note that, under this constraint, the system's generative model remains linear (the reconstructed input is just a linear combination of the non-negative outputs), but the recognition model becomes nonlinear (the forward mapping can no longer be expressed as a matrix multiplication). This means that the system is able to produce nonlinear mappings whilst retaining some of the advantages of linearity, such as ease of analysis and the absence of local minima during activation.

In terms of the REC network model, we may think of the nonlinearity as being provided by a threshold-linear output function Θ of the outputs a_i, where

$$\Theta(a) = \begin{cases} 0 & \text{if } a < 0, \\ a & \text{otherwise.} \end{cases}$$

The function Θ is sometimes known as a *rectification nonlinearity*. It may be viewed as a special case of Breiman's (1993) 'hinge functions.' This nonlinearity, although simple, is sufficient to produce a universal function approximator in a multi-layer feedforward system (Diaconis and Shahshahani, 1984; Ripley, 1996, p. 173). A multi-stage REC network could not quite fulfil these criteria because the output units lack a *bias* term, *i.e.* a means to adapt the threshold of the function Θ. Addition of such a term might prove to be a useful future enhancement to the REC model.

What form of data is generated by a non-negative linear model? We shall consider first the noise-free case, *i.e.* where the data $\mathbf{x}$ are equal to $\hat{\mathbf{W}}\hat{\mathbf{a}}^+$ for some vector $\hat{\mathbf{a}}^+$ with non-negative elements and some mixing matrix $\hat{\mathbf{W}}$. An example of a cloud of points generated in this way is shown in Figure 5.1(a). Since there is no restriction on the elements of $\hat{\mathbf{W}}$ to be non-negative, the points need not lie in the positive quadrant as they do in this example. The important feature of Figure 5.1 is that all points lie within a sector (or 'wedge'), whose edges are determined by the rows of $\hat{\mathbf{W}}$ ($\hat{\mathbf{w}}_1$ and $\hat{\mathbf{w}}_2$). This property will hold for *any* non-negative distributions on the elements of $\hat{\mathbf{a}}^+$, in any number of dimensions m, and for any values of $\hat{\mathbf{W}}$, providing that $\text{rank}(\hat{\mathbf{W}}) \leq m$.

This observation sheds some light on the results obtained in Section 4.12.3 from training a REC network on data generated by a non-negative linear model. A consideration of Figure 5.1(a) shows that *any* two weight vectors defining the edges of a sector that contains all of the points will minimise the network's reconstruction error over the data set (to zero). Figure 5.1(b) shows an example of one such set of weights. So while the use of non-negative outputs reduces the space of solutions (both $\hat{\mathbf{w}}_1$ and $\hat{\mathbf{w}}_2$ must lie 'between' $\mathbf{w}_1$ and $\mathbf{w}_2$), it still does not guarantee the 'correct' answer.

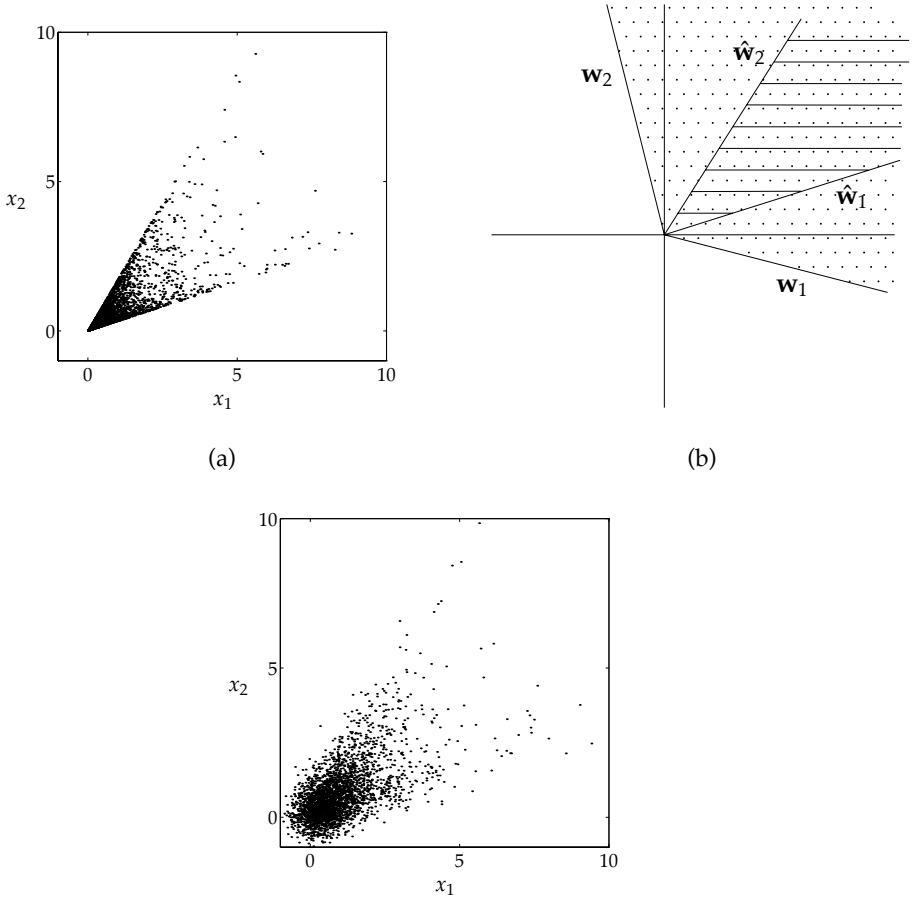


Figure 5.1: (a) A cloud of points generated exactly as for Figure 3.4(b) (page 30), but using single-tailed (non-negative) distributions. (b) A diagrammatic representation of why the REC network, even when using non-negative outputs, is still under-constrained when trained on such data. The sector of points that could be generated by two weight vectors $\mathbf{w}_1$ and $\mathbf{w}_2$ (shown dotted) *contains* those generated by the 'true' features $\hat{\mathbf{w}}_1$ and $\hat{\mathbf{w}}_2$ (shown striped). (c) The data in (a) after corruption by spherical Gaussian noise of variance 0.4.

There is an obvious means to overcome this problem: if the initial weight vectors could be guaranteed to lie *inside* the 'wedge' of data, then during training they would gradually move towards the edges, driven by the residual errors produced by inputs lying outside the sector defined by the current weight values. Once the weight vectors reached the edges of the data, the network would be able to represent all the points exactly and learning would cease. This method requires a suitably small learning rate, because any 'overshoot' of the boundaries of the data cannot be reversed. The problem of initialising the weight vectors is easily solved by setting each to a (normalised) random sample from the data set. Repeating the experiments of Section 4.12.3 using this method confirmed that it was possible to produce very close matches to the true features, but learning was slow because of the small learning rate (≤ 0.01) needed to guarantee good results.

The properties shown in Figure 5.1 could also explain why the original experiments were found to produce better results with high learning rates. Such rates are likely to cause weight vectors to 'jump' into and around the subspace occupied by the data, until a stable solution is found when all weight vectors are on, or just outside the boundaries of this space. With a small learning rate, the solution found is far more dependent on the initial weight values: learning will follow a steady trajectory in weight-space until the reconstruction criterion is fulfilled. As demonstrated in Figure 5.1(b), the weight values $\mathbf{W}$ in this state need not bear any close resemblance to the matrix $\hat{\mathbf{W}}$.

We note in passing that the wedge property of Figure 5.1(a) allows a simple algorithm to discover the model parameters $\hat{\mathbf{W}}$. By testing whether all the data points lie on one side of a straight line, we could imagine rotating lines from the origin around the plane of Figure 5.1(a) until the edges of the sector of points are found, giving the rows of $\hat{\mathbf{W}}$ directly. This is in fact a special case of the *convex hull* problem, for which various techniques exist in two- or three-dimensional space (*e.g.* O'Rourke, 1994), and which can also be extended into higher dimensions (Edelsbrunner, 1987).

The reason we shall not consider such techniques further here is demonstrated in Figure 5.1(c), where we see that the presence of noise causes the data cloud to lose its wedge property, and would prevent algorithms of this type from producing good results. We noted in Section 4.12.4 that for the REC network too, noise degraded the quality of results for continuous-valued features, and can see from Figure 5.1(c) that this is because the data boundaries are pushed apart in comparison to the noise-free case (a). In the experiment with noise-corrupted 'lines' data from the same section, however, the results were essentially unchanged, because for binary features the solutions lie on the boundaries of a $[0, 1]$-hypercube. Since the weight vectors were constrained to lie within this cube, the noise was unable to push the weights further apart, and so did not have a detrimental effect on the final results.

Even in the presence of noise the underlying structure imposed by $\hat{\mathbf{W}}$ is still clearly visible in Figure 5.1(c) and it is this sort of structure that the REC network is intended to discover. We have seen that the hard non-negative constraint, while useful for reducing the space of possible solutions, is not by itself able to extract the underlying components, but in the following sections we shall look at softer forms of constraint, which in many cases offer a significant improvement in this respect.

5.3 Imposing soft constraints with penalty terms

Hard constraints on a network model — imposed for example by limiting the number of outputs, or by forcing output or weight values to lie within fixed ranges — are useful when we can be sure they are justified, that is when we are certain that the optimal solution will be found within these constraints. However, in many cases we know only the form we would *like* the solution to take, but cannot be sure in advance to what extent the model will be able to comply with this wish.

A common approach in such cases is to *penalise* the system for violating the constraint, by defining a measure of the extent to which the model is deviating from the ideal situation. This measure, Ω, may be added in to the system's error function E to give a penalised error:

$$E_p = E + \lambda \Omega \tag{5.1}$$

where the parameter λ controls the extent to which the penalty Ω influences the form of the solution, or equivalently determines the relative importance of the terms E and Ω.

This technique is often known as *regularisation*, first studied in detail by Tikhonov and Arsenin (1977) as a means of finding unique solutions to otherwise under-constrained, or *ill-posed*, problems. Regularisation plays an important role in neural networks (Bishop, 1995, Chap. 9), and in computational vision (Ballard *et al.*, 1983; Poggio *et al.*, 1985). In the neural network literature, regularisation is usually applied to weight, rather than output values, although the method has also been used to constrain the hidden-unit activities of MLPs to encourage efficient representations (Chauvin, 1989a). The term 'regularisation' is avoided here because it is often used specifically to refer to the idea of *smoothing* a function approximator, particularly in the context of neural networks. The penalties introduced below will primarily be functions of the outputs alone, although weight penalties are considered in Section 5.5.

Penalty terms are also used as a method for nonlinear programming (*e.g.*, Fletcher, 1987, Chap. 12), where the penalty parameter is increased towards infinity in successive approximations. This gives a means of finding the approximate region of a solution (on a multimodal error surface) by applying the constraint Ω only loosely at first, but with increased insistence at each iteration.

For the purposes of the REC network, we shall add penalty terms to the network's activation objective function E_r (4.5), giving a penalised error

$$\begin{aligned} E_p &= E_r + \lambda \Omega \\ &= ||\mathbf{x} - \widetilde{\mathbf{x}}||^2 + \lambda \Omega \end{aligned} \tag{5.2}$$

and an overall objective function of

$$E_P = \mathcal{E}[E_p]. \tag{5.3}$$

Recall that the network aims firstly to infer the optimal outputs for a particular input given its current weights, and secondly, and more importantly, to infer the best set of weights over a series of inputs (Section 4.1); the penalty term in (5.2) takes on two corresponding roles.

First, where an objective function based on reconstruction error alone has multiple minima, a penalty may constrain the activation process to a single solution. This is required only

when weight vectors are linearly dependent (an issue covered in more detail in Section 5.7). Second, the penalties help constrain the *learning* process in cases where there are many sets of weights that minimise reconstruction error (as we have seen in several previous examples).

Conceptually, the penalties induce additional residuals in the REC network by shifting the minimum of E_p (5.2) slightly away from that given by reconstruction error alone, towards the 'ideal' values of the outputs. These residuals help to drive weight values towards the desired solution. Mathematically, the penalty term Ω, although a function of the output values, is indirectly a function of the weights too (since the outputs are dependent on the weights), so minimising E_P (5.3) requires the discovery of weights that jointly minimise reconstruction error *and* the penalties incurred by the output values they produce. We shall see in Section 5.3.3 that the REC network's Hebbian learning rule (4.15) performs stochastic gradient descent on the penalised error (5.3), confirming that the penalty term, as well as directly influencing activation, also has the desired indirect effect on learning.

What form of penalty Ω should be used? Recalling from Chapter 2 that we wish output values to be as independent as possible, we might consider some function based on the joint statistics of the output values a_i. There are two main problems associated with this approach:

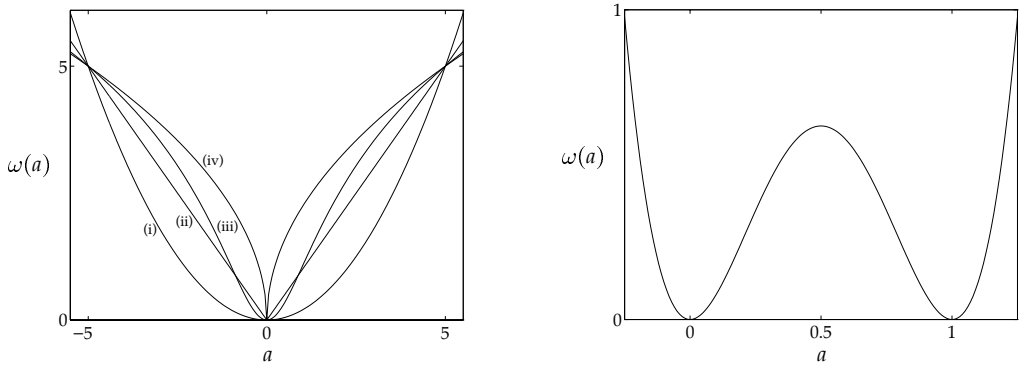
1. Accurate estimation of sample statistics requires a large number of data points. In practical terms, this means the system requires an extra mechanism to collect statistics over a period of time.

2. The problem is compounded when considering *joint* statistics. A full test of independence requires estimation of both joint and marginal densities, demanding a much greater quantity of data to give accurate results. In terms of the REC model, this would also require each output to have knowledge of all other output values, for example by means of lateral connections, which would incur a significant additional computational cost.

An approach of this type is used in Schmidhuber's (1992) 'predictability minimisation' network. The two problems noted above necessitate an extra level of structure in this architecture, where each output uses its own prediction network to 'listen' to all of the other outputs, and to try to make its own responses as *unpredictable* as possible (given the other outputs).

Problem 2 may be avoided, however, by making use of the observation made in Section 2.5, namely: providing that the system maximises information throughput (and we saw in Section 4.11 that minimising reconstruction error achieves this), then simultaneous minimisation of the sum of the individual output entropies will make the outputs as independent as possible. This gives an approach that allows penalties to be calculated by considering the statistics of each output in isolation. It is interesting to note that this is in some sense the opposite of Schmidhuber's approach, because by minimising individual entropies, we are trying to make each of the output values as *predictable* (in isolation) as possible. Because the outputs can be considered separately, we may decompose Ω into $\sum_{i=1}^{n} \omega_i(a_i)$ where the *penalty functions* ω_i attempt to impose a *low entropy* constraint on each output.

A full solution to this problem would be to relate the $\omega_i(a_i)$ to the expected increase in entropy that a particular value a_i would bring to the distribution as a whole. This requires that each penalty ω_i be adaptive, based on the distribution of past values of the outputs



(a) (b)

Figure 5.2: (a) Four candidates for a sparseness-promoting penalty. Each function ω has been multiplied by a constant so that $\omega(5) = 5$. The underlying functions are (i) $\omega(a) = a^2$, (ii) $\omega(a) = |a|$, (iii) $\omega(a) = \log(1 + a^2)$ and (iv) $\omega(a) = \sqrt{|a|}$. (b) A simple function $\omega(a) = a^2(a-1)^2$ that attempts to produce binary outputs by penalising output values other than zero and one.

(problem 1). Furthermore, while we know that low entropy distributions tend to be highly peaked at one or more values, entropy itself gives no clues as to the location, or even number, of peaks to expect in the final output distributions, particularly at the outset of learning. An entropy-based penalty is therefore unable initially to put global pressure on output values toward particular values, and is likely to result in a learning process characterised by large numbers of local minima.

We may avoid these problems, but only by making some strong assumptions. If we have sufficient prior knowledge to know where the peaks (or modes) of the final output distributions will occur, a penalty ω_i need only be a function of the deviation of a_i from the modes. Furthermore, if they occur at the same locations for each output value, then we may use the same penalty function ω for each output, giving as the penalised error

$$E_p = \|\mathbf{x} - \widetilde{\mathbf{x}}\|^2 + \lambda \sum_{i=1}^{n} \omega(a_i). \tag{5.4}$$

Figure 5.2 shows (a) a number of possible penalty functions for a distribution peaked at zero, and (b) a penalty function, namely

$$\omega(a) = a^2(a-1)^2, \tag{5.5}$$

for distributions peaked at zero and one. Equation (5.5) can be used as a soft binary constraint, to encourage output values to take on binary values, without requiring the network to be a purely binary system. We shall, however, concentrate primarily on penalties of the type shown in Figure 5.2(a).

5.3.1 Sparseness-promoting penalties

A sparse code is characterised by most of the elements being at or close to zero most of the time (Section 2.7). Thus, a penalty that increases as an output value moves away from zero is an intuitively appealing means to encourage sparseness. To help us decide which of the

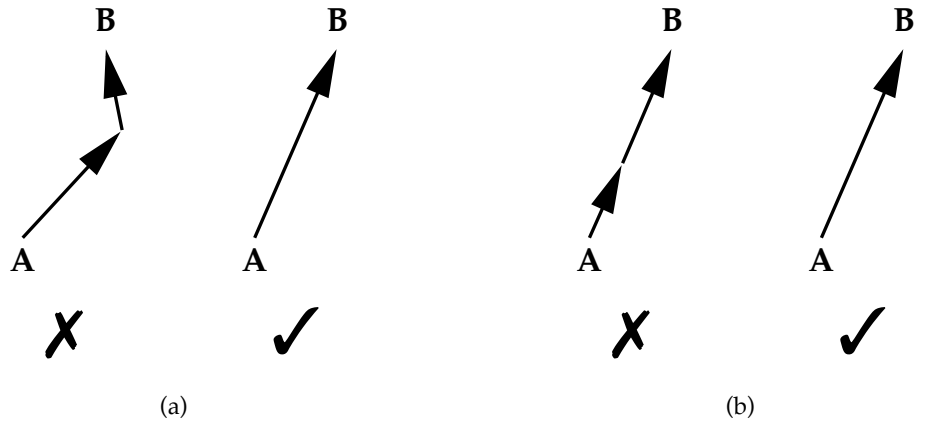


Figure 5.3: 'Getting from A to B with a sparse linear code.' (a) Given a choice between a two- and a one-stage route, a sparse code favours the latter because it is shorter. (b) When the distances are the same, the 'sparser' route is the one with fewer stages.

penalties of Figure 5.2(a) (or other similarly shaped functions) is most useful, we need to make a further observation about sparseness. Where possible, a sparse code will represent a pattern in terms of a few large-valued feature coefficients, rather than many smaller ones. For the case of a linear generative model, as used by the REC network, this idea can be made more concrete with a simple analogy.

Figure 5.3(a) shows two possible routes from A to B. With A the origin, and B a point to be encoded, it is the second, direct, route that corresponds to the sparse code because it only requires the use of a single 'direction' (feature vector) to reach B. Using this analogy, a sparse code is one that discovers short routes to its data points. The shortest possible distance is of course always given by the straight-line route. The limiting case is shown in Figure 5.3(b). Here the choice is between a two-stage journey from A to B, and a journey taking the same route, but in a single stage. Although the distances involved are the same, a sparse code will again favour the latter because it involves only a single feature.

Applying this idea to the REC network, we see that if the weight vectors have unit length then the 'distances' involved for a particular representation are merely the magnitudes of the output values a_i. This makes

$$\omega(a) = |a| \tag{5.6}$$

as shown by line (ii) in Figure 5.2(a) a natural choice of penalty function, because it results in the overall penalty Ω being equal to this distance. It deals with all but the limiting case, as depicted in Figure 5.3(b). To ensure in this example that the choice of a single large feature is penalised less than two small ones, we require ω to be *sublinear*, *i.e.* to satisfy the condition that

$$\forall(a,b), \quad \omega(|a| + |b|) < \omega(a) + \omega(b). \tag{5.7}$$

To achieve this, the absolute value of the gradient must be at its maximum at $a = 0$. This immediately allows us to reject the function $\omega(a) = a^2$, as shown by Figure 5.2(a)(i), for which this condition clearly does not hold.

A problem arises because we must also have a *minimum* at $a = 0$. Both conditions cannot be fulfilled simultaneously without a discontinuity in the gradient (a cusp) at this point. An example of a function that *does* fulfil our criteria is given by

$$\omega(a) = \sqrt{|a|}$$

which is plotted as line (iv) in Figure 5.2(a), or more generally

$$\omega(a) = |a|^{1/r}, \quad r > 1. \tag{5.8}$$

In practical terms, the discontinuity in gradients of both (5.6) and (5.8) at $a = 0$ may prove to be a problem, particularly if we wish to use a gradient-based optimiser to minimise E_p (5.4). The problem is compounded for (5.8) by the fact that the gradient tends to $\pm\infty$ around $a = 0$. For these reasons, we might wish to relax the sublinear condition (5.7) to apply only to values of a and b whose magnitude exceeds a certain threshold. This allows us to apply a penalty such as

$$\omega(a) = \log(1 + a^2)$$

as shown by Figure 5.2(a)(iii), which has a continuous gradient, and fulfils condition (5.7) for $a, b \geq 1$. The points of inflection can, by scaling a, be moved as close to zero as we wish, giving us a means of balancing the requirements of the minimiser with those of the sparseness constraint. It is also worth noting that if the above penalties are used in conjunction with the non-negative output constraint (Section 5.2) then the gradient discontinuity at zero ceases to be an issue anyway.

5.3.2 Penalties and network activation

It is not difficult to incorporate the penalty functions introduced above into the dynamics of the REC model. Differentiating (5.4) with respect to the outputs a_i gives

$$\frac{\partial E_p}{\partial a_i} = -2\,(\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}}\mathbf{w}_i + \lambda\,\omega'(a_i)$$

where ω' represents the derivative of the function ω with respect to its argument. The extra term in this equation, as compared with Equation (4.6), is easily modelled as a self-inhibitory connection on each output unit, as shown in Figure 5.4. Making the corresponding change to the output update rule (4.2) gives

$$\Delta a_i = \mu\,[\mathbf{r}^{\mathrm{T}}\mathbf{w}_i - \lambda\,\omega'(a_i)]. \tag{5.9}$$

Rather than using self-inhibitory connections, we could also think of the penalties as being applied by some cost function implemented *within* the output units themselves.

From a biological perspective, this could take the form of the metabolic cost of maintaining a particular output value (Baddeley, 1996). The linear penalty (5.6) is particularly interesting in this respect. If we assume that the output of a biological neuron is encoded by its firing rate, then a linear penalty on the output is consistent with a constant metabolic cost

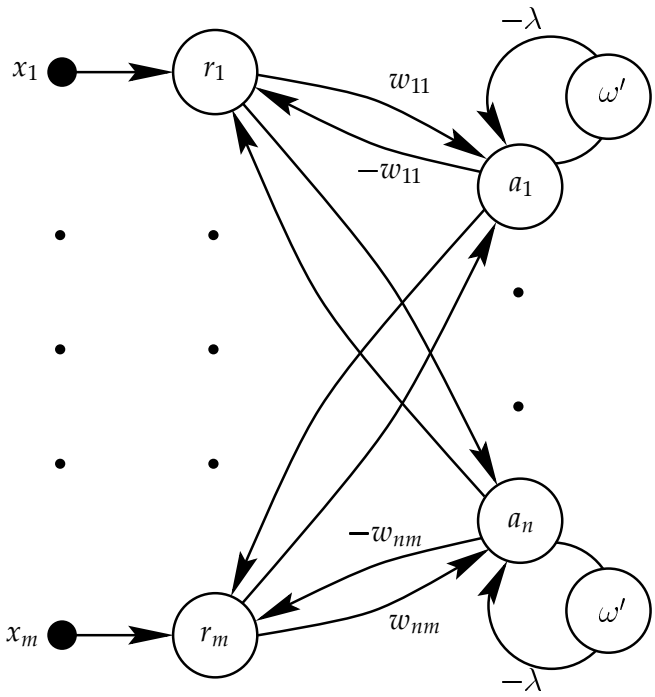


Figure 5.4: The REC network architecture, incorporating self-inhibitory connections. These form some function ω' of the output value, weight it by $-\lambda$, and feed this value back to form part of the unit's input, used when calculating the next update to its activation value. In this way, the revised network implements the 'penalised' update rule (5.9).

for each neural firing event ('spike'). The idea of the brain trying to represent the world effectively while attempting to minimise the amount of energy it needs to do so, is an appealing one.

The absolute-linear penalty function (5.6) will be used extensively as a means to promote sparseness in the experiments in the remainder of this thesis. Although it fails to resolve the case shown in Figure 5.3(b) where two or more feature vectors are parallel, this is not likely to be a major problem in practice, because the network's objective of minimising reconstruction error (with a limited number of output units) applies pressure to prevent outputs being 'wasted' by representing the same feature in this way. In addition, the penalty $|a|$ has the practical advantage of being extremely easy to calculate and apply. Its derivative is just ± 1, according to the sign of a, so in the recurrent model it may be applied merely by moving each output a constant amount λ towards zero at each iteration of the activation process.

5.3.3 Penalties and learning

We saw in Section 4.7 that, in the absence of penalties, Hebbian learning in the REC model performs stochastic gradient descent on the expected reconstruction error E_R. It is not immediately obvious how to extend this to the penalised case.

We may make use, however, of the fact that the gradient of the objective function (in weight-space) may be calculated with the outputs constant at their activation values (Appendix A). This means that the addition of a term that is a function of the outputs alone has no effect on the gradient. We therefore find that the partial derivatives of E_P with respect to

the weights are identical to those of E_R (4.14), namely:

$$\frac{\partial E_P}{\partial w_{ij}} = -2\,\mathcal{E}[r_j a_i]. \tag{5.10}$$

The penalty does of course have an effect on the gradient, but what (5.10) tells us is that this effect is contained entirely in the difference to the output values a_i induced by the presence of the penalty term. Thus the Hebbian learning rule (4.15) continues to provide us with a means of minimising the objective function using stochastic gradient descent.

5.4 Penalty terms and prior probabilities

The rationale behind the choice of penalty functions (page 71) was that their minima should be made to correspond to the expected peaks of the output distributions. In this section, by building on the maximum-likelihood approach of Section 4.10, we shall see that the link is stronger than that, and that a connection can be made between a particular penalty function and the application of a corresponding prior probability distribution on the REC network's output values. This gives a better understanding of the exact role of penalties, further justification for particular choices of function, and an insight into how penalties might be made adaptive to match online estimations of probability distributions. The link in this context was first made by Harpur and Prager (1995, 1996), and has since been explored further by Olshausen (1996).

We saw in Section 4.10 that the basic REC network, based on minimisation of reconstruction error, may be interpreted as finding outputs $\mathbf{a}$ that maximise the *likelihood* $p(\mathbf{x}|\mathbf{a})$. This seems to be the wrong way round, because it is the data $\mathbf{x}$ that is given. What we would really like is the outputs that maximise $p(\mathbf{a}|\mathbf{x})$. We noted that the two estimates would be the same if all values of $\mathbf{a}$ were equally likely, but in general the relationship is given by Bayes' rule:

$$p(\mathbf{a}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{a})\; p(\mathbf{a})}{p(\mathbf{x})}. \tag{5.11}$$

The density $p(\mathbf{a}|\mathbf{x})$ is known as the *posterior* probability, $p(\mathbf{a})$ is the *prior* distribution of $\mathbf{a}$, and $p(\mathbf{x})$ is often termed the *evidence*. The value of $\mathbf{a}$ that maximises $p(\mathbf{a}|\mathbf{x})$ is known as the *maximum a posteriori* (MAP) estimate.

It is important to note that we are dealing solely with the determination of output values here, and that there are several assumptions implicit in the above expressions. We are assuming that the network's world-model is *correct*, *i.e.* that the data really are generated by linear superposition, and subsequently corrupted by Gaussian noise, and that the network's weight values exactly match the true features in the environment. Strictly speaking, we should write the densities $p(\cdot)$ as $p(\cdot\,|\,\mathbf{W}, \mathcal{H})$, where $\mathbf{W}$ are the current weights and $\mathcal{H}$ represents the other 'background assumptions,' or *hypotheses*, of the REC model. Having noted these assumptions, we shall allow them to remain implicit.

Maximising Equation (5.11) is equivalent to *minimising* its negative logarithm (since the logarithm is a monotonically increasing function), giving an objective function E of

$$E = -\log p(\mathbf{x}|\mathbf{a}) - \log p(\mathbf{a}) \tag{5.12}$$

where the term in $p(\mathbf{x})$ has been dropped since it is does not depend on $\mathbf{a}$. Using our previous assumption of independent spherical Gaussian noise on the inputs, we have

$$p(\mathbf{x}|\mathbf{a}) = \frac{1}{Z(\sigma_\nu)} \exp\left(-\frac{||\mathbf{x} - \mathbf{W}\mathbf{a}||^2}{\sigma_\nu^2}\right) \tag{5.13}$$

where σ_ν^2 is the noise variance, and Z a normalising function. Taking the negative logarithm of (5.13) gives

$$-\log p(\mathbf{x}|\mathbf{a}) = \frac{1}{\sigma_\nu^2}||\mathbf{x} - \widetilde{\mathbf{x}}||^2 + C$$

for some constant C.

We shall assume that all elements of $\mathbf{a}$ are independent (our whole approach of looking for low entropy codes assumes that this is at least a reasonable approximation). In this case we may write the second term in (5.12) as

$$-\log p(\mathbf{a}) = -\sum_{i=1}^{n} \log p(a_i).$$

Combining these terms and ignoring constants, we may rewrite the objective function (5.12) as

$$E = \frac{1}{\sigma_\nu^2}||\mathbf{x} - \widetilde{\mathbf{x}}||^2 - \sum_{i=1}^{n} \log p(a_i). \tag{5.14}$$

Comparing (5.14) with the penalised error E_p (5.4), we see that the two are identical (to within a constant positive factor) if we set

$$\lambda = \sigma_\nu^2 \tag{5.15}$$

$$\text{and} \quad \omega(a) = -\log p(a). \tag{5.16}$$

The equivalence (5.15) shows that we may interpret the penalty parameter λ as being the assumed noise variance — the larger the value, the less important the reconstruction error becomes in the determination of the MAP estimate, as we would expect.

More importantly though, (5.16) shows us that there is a direct relationship between the penalty functions, as applied in (5.4), and assumed prior probabilities for MAP estimation. We shall look at a number of penalty functions in this light in Section 5.4.2 below.

5.4.1 Weight estimation and maximum likelihood

We saw in Section 5.3.2 that Hebbian learning for the penalised REC network performs gradient descent on E_P (5.3). In light of the above analysis we see that minimising E_P is equivalent to maximising an error function

$$\begin{aligned} E &= \mathcal{E}[\log\{p(\mathbf{x}|\mathbf{a}, \mathbf{W})p(\mathbf{a}|\mathbf{W})\}] \\ &= \mathcal{E}[\log p(\mathbf{x}, \mathbf{a}|\mathbf{W})] \\ &= \frac{1}{N} \log \prod_{i=1}^{N} p(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}|\mathbf{W}) \end{aligned}$$

where the dependence on the weight values has now been made explicit, and the final line assumes a data set of finite size N, with $\mathbf{x}^{(i)}$ representing elements of the data set, and $\mathbf{a}^{(i)}$ the corresponding outputs. The product in this line is the joint *likelihood* of the inputs $\mathbf{x}$ and outputs $\mathbf{a}$. The REC network therefore attempts to find a set of weights $\hat{\mathbf{W}}$ that maximise this likelihood:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} \prod_{i=1}^{N} p(\mathbf{x}^{(i)}, \mathbf{a}^{(i)} | \mathbf{W}).$$

What we would really like is to find a set of weights that maximise the likelihood of the data alone, *i.e.*

$$\begin{aligned} \hat{\mathbf{W}} &= \arg\max_{\mathbf{W}} \prod_{i=1}^{N} p(\mathbf{x}^{(i)} | \mathbf{W}) \\ &= \arg\max_{\mathbf{W}} p(\{\mathbf{x}\} | \mathbf{W}) \end{aligned} \tag{5.17}$$

where $\{\mathbf{x}\}$ denotes the complete data set. The two estimates are guaranteed equivalent only if the random vectors $\mathbf{x}$ and $\mathbf{a}$ are *fully dependent*. By this we mean that for each $\mathbf{x}$ there is only a single value of $\mathbf{a}$ with non-zero probability, or in other words the density $p(\mathbf{a}|\mathbf{x})$ is a delta function for all values of $\mathbf{x}$. This a reasonable approximation when the noise variance σ_ν^2 is small. A fuller discussion of this issue is given by Olshausen (1996).

A technique known as the *EM algorithm* (Dempster *et al.*, 1977) is also applicable here. In this context, the network outputs $\mathbf{a}$ are known as *hidden variables*, whose *true* values, if they could be known, would make the maximisation of likelihood (5.17) significantly easier. The algorithm, as applied to the task here, would use some initial guess $\mathbf{W}^{(0)}$ for the weight values and generate successive estimates $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, ... by an iterative application of the following two steps, for $t = 1, 2, \ldots$

E Step: Compute the distribution $\tilde{p}^{(t)}$ over the range of $\mathbf{a}$, where $\tilde{p}^{(t)}(\mathbf{a}) = p(\mathbf{a} \,|\, \{\mathbf{x}\}, \mathbf{W}^{(t-1)})$.

M Step: Set $\mathbf{W}^{(t)}$ to the values $\mathbf{W}$ that maximise $\mathcal{E}_{\tilde{p}^{(t)}}[\log p(\mathbf{a}, \{\mathbf{x}\} \,|\, \mathbf{W})]$.

In the M step, $\mathcal{E}_{\tilde{p}^{(t)}}[\cdot]$ denotes the expectation over the current guess $\tilde{p}^{(t)}$ at the true distribution of the hidden variables. Each iteration of the algorithm increases the likelihood of the data (5.17) as desired, unless a local maximum has already been reached. Furthermore, this useful result continues to hold where the M step is only partial, *i.e.* where the likelihood with respect to the estimated distribution $\tilde{p}$ is increased but not maximised.

The REC network algorithm is certainly not an implementation of the EM algorithm, but there are some similarities. Activation is a crude approximation to the E step, substituting a full evaluation of $\tilde{p}^{(t)}$ with a delta function peaked at a value of $\mathbf{a}$ determined by a single sample from the data. A weight update may be viewed as a partial M step.

It might be beneficial to adopt the full EM algorithm for training the REC network, or at least to use a closer approximation to the E step. In doing so, however, much of the simplicity of the network model would be lost, and it is not clear that it would bring drastic improvements to its operation. The experiments in Chapter 7 give empirical evidence that the network is able to provide useful results in its simple form. Furthermore, as noted by

Penalty $\omega(a)$	Prior distribution			
	Density $p(a)$	Name	Entropy (nats)	Kurtosis
a^2	$\frac{1}{\beta\sqrt{2\pi}}\exp\left(-\frac{a^2}{2\beta^2}\right)$	Normal	1.419	0
$\lvert a\rvert$	$\frac{1}{2\beta}\exp\left(-\frac{\lvert x\rvert}{\beta}\right)$	Laplace	1.347	3.0
$\sqrt{\lvert a\rvert}$	$\frac{1}{4\beta}\exp\left(-\sqrt{\frac{\lvert x\rvert}{\beta}}\right)$	–	0.993	22.2
$\log(1+a^2)$	$\frac{\beta}{\pi}\frac{1}{\beta^2+a^2}$	Cauchy	(2.531)	–
$\begin{cases}0 & \text{if } 0\leq a\leq 1,\\ \infty & \text{otherwise}\end{cases}$	$\frac{1}{\beta},\quad 0<a<\beta$	Uniform	1.2425	-1.2

Table 5.1: Some penalty functions, and the corresponding prior distributions. Scaling factors are omitted from the penalties for simplicity. The probability density functions are given, parameterised by the value β that in each case is related to the distribution's variance. The fourth column gives the differential entropies, as defined in Section 2.1.4. To allow meaningful comparison, these values were calculated for distributions of unit variance, except in the case of the Cauchy distribution, which has infinite variance, where β was set (arbitrarily) to one. The final column shows the distributions' kurtoses, as described in the main text.

Neal and Hinton (1993), similar approximate forms of EM are used by the 'K-means' algorithm, and as a short-cut to the full Baum-Welch algorithm in the training of hidden Markov models (HMMs) (*e.g.* Merhav and Ephraim, 1991).

5.4.2 Choice of priors

We have now seen a direct relationship (Equation 5.16) between penalty functions and the assumption of priors for MAP estimation. Table 5.1 shows a number of penalty functions, including all of the sparseness-promoting penalties discussed in Section 5.3.1, together with some details of the corresponding prior distributions.

We saw in Section 2.1.4 that the normal (Gaussian) distribution *maximises* the entropy over all distributions with the same variance. Furthermore the technique of projection pursuit (Section 3.8.4) defines 'interesting' projections as ones where the coefficients are as *non-Gaussian* as possible. It does not therefore appear to be a good choice of prior for our purposes, giving further justification for our rejection of the squared penalty function (a^2) in Section 5.3.1.

The absolute-linear penalty ($|a|$) corresponds to a Laplacian (double-tailed exponential)

distribution. Given the observations in Section 5.3.1 about the relationship of this penalty to sparseness, we note that for a linear model the Laplacian provides a useful dividing line between sparse and non-sparse distributions.

We see that the penalty introduced as a compromise between sparseness and continuity of gradient ($\log[1+a^2]$) in fact corresponds to a Cauchy prior, notable for having no defined moments above the first. The entropy figure in Table 5.1 for this distribution is bracketed because the inability to normalise to unit variance makes a direct comparison meaningless; the figure is nevertheless low for a distribution whose variance is effectively infinite!

The final row of Table 5.1 demonstrates a way to implement a uniform prior (over a bounded interval) using a penalty function. In practice this is equivalent to constraining the output value to lie within this interval, and therefore shows that *hard* constraints on output values (such as the non-negative constraint of Section 5.2) have a simple interpretation as regions of *zero* prior probability on the outputs. We also note that while the uniform distribution maximises entropy for a *discrete* variable, this is not the case for a (fixed variance) continuous random variable.

5.4.3 Kurtosis as an indication of sparseness and entropy

The last column of Table 5.1 gives the *kurtosis* of the distributions. Kurtosis is a normalised form of the *fourth central moment* of a distribution. It is commonly defined as

$$K = \frac{1}{\sigma^4} \int (a - \overline{a})^4 p(a)\, da - 3$$

for a density $p(a)$, where $\overline{a}$ is the mean and σ^2 the variance of the distribution. The offset of -3 is not always used, but is often included because it means that the baseline Gaussian distribution has zero kurtosis.

Kurtosis is introduced here because it has been used in recent literature as a measure of sparseness (Barlow, 1994; Field, 1994; Baddeley, 1996).[2] Amongst unimodal distributions, it is a means of quantifying their 'peakiness' or, equivalently, the proportion of 'weight' (variance) in the tails, with a large positive value indicating a highly peaked and heavy-tailed distribution. Indeed, we observe that the first three distributions featured in Table 5.1, all peaked at zero, have increasing kurtoses, corresponding to increasing peakiness (and decreasing entropy). Conversely, the uniform distribution, which is less peaked than a Gaussian, has negative kurtosis.

This property extends to other unimodal distributions, including those that are non-symmetric (skewed), suggesting that high kurtosis is a good indicator of both high sparseness and low entropy *in the unimodal case*. Its usefulness in more general cases is less clear.

Kurtosis has been used as a projection index for exploratory projection pursuit (Huber, 1985; Jones and Sibson, 1987), and in an extension of this to a neural network architecture (Fyfe and Baddeley, 1995b). It is also a key measure in some recent 'nonlinear PCA' networks (Oja, 1995; Karhunen *et al.*, 1995).

It is not however clear that kurtosis is providing these techniques with a useful quantity. For multimodal distributions, it is quite possible for the kurtosis to be positive, negative or

[2] Other measures of sparseness have also been used. Treves and Rolls (1991) and Rolls and Tovee (1995) give an alternative definition for use with neural firing rates, but this does not appear to be generally applicable.

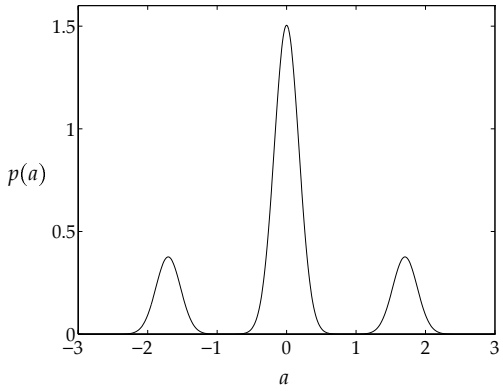


Figure 5.5: A multimodal distribution with zero kurtosis, generated by superimposing three equally-spaced Gaussians with the same width, but with the central one having four times the weight of the other two. Any distribution generated in this way will have zero kurtosis. This particular one also has unit variance. It has a differential entropy of approximately 0.554 nats.

even zero, the last of these cases being illustrated by Figure 5.5. This shows a multimodal distribution that has *zero* kurtosis, but very low entropy (0.554 nats) and could arguably be described as sparse (it is certainly highly peaked at zero).

In the light of this, we may observe that kurtosis by itself is not a reliable indicator of sparseness, entropy, or even (since Figure 5.5 is highly non-Gaussian but has identical kurtosis) of deviation from a normal distribution. In this thesis, subsequent mention of degrees of sparseness may be equated with kurtosis, but *only* amongst distributions that are unimodal and peaked at zero — the sparseness of other distributions is undefined.

It should also be reiterated that in Chapter 2 we identified entropy as the *theoretical* basis for determining the interest of output distributions. From a *practical* viewpoint however, it is not appropriate for a simple neural network model to use entropy directly, as it is both difficult to estimate and, as noted in Section 5.3, difficult to optimise.

Consequently we have introduced, by the use of fixed low-entropy priors, a means of reducing the entropy of the output distributions. The precise form of the assumed priors is not crucial, since there is no guarantee that the true output distributions will ever match them exactly. They nevertheless incorporate strong assumptions, particularly about the location of peaks in the final distributions. The contention is that to make the assumptions clear in this way is a more principled approach than to use kurtosis (or other higher order statistics of individual outputs), for which the underlying assumptions are less easily understood.

5.5 Weight constraints

The penalties thus far have all been functions of the network's output values. It is quite possible to penalise weight values in a similar way. Techniques of this type are in common use for supervised neural networks (*e.g.* Bishop, 1995, Chap. 9), so we shall not go into detail here, except to note that they are also applicable to the REC network. In its simplest form, a weight penalty term is made proportional to the sum of the squares of the weight values. This is easily implemented for gradient descent by incorporating a linear 'decay' term in

each weight update, giving

$$\Delta \mathbf{w}_i = \eta(a_i \mathbf{r} - \lambda_w \mathbf{w}_i) \tag{5.18}$$

as the modified Hebbian learning rule for the REC network, where λ_w is the rate of decay. It is quite possible to interpret such penalties in terms of priors (Mackay, 1995; Ripley, 1996, Chap. 5) as we did for the output penalties, although the independence assumption is less easily justified in this case. Standard weight decay (5.18) corresponds to a zero-mean Gaussian prior distribution on the weight values. It can also be useful to make the degree of weight decay dependent on the magnitude of the corresponding output value, for example using

$$\Delta \mathbf{w}_i = \eta(a_i \mathbf{r} - \lambda_w a_i^2 \mathbf{w}_i). \tag{5.19}$$

This prevents the weights of 'unused' output units from becoming too small.

Although weight decay in this form (5.19) is used in Section 5.9.3 below, the majority of experiments described here use only hard constraints on the weights, typically in the form of upper and lower bounds on values and, in continuous domains, normalisation of the weight values. The latter constraint is most easily applied 'manually' by normalising each weight vector to a fixed length after a set of updates, but, if a more 'online' approach were needed, could equally be achieved using a modification of Oja's rule (3.13):

$$\Delta \mathbf{w}_i = \eta a_i(\mathbf{r} - a_i \mathbf{w}_i).$$

This turns out to be the same as the weight decay rule (5.19) when λ_w is equal to one.

5.6 The entropy of sparse codes

We have set out to discover low entropy codes, and have identified sparseness as one means of characterising output distributions that may help us to achieve this. We need however to be clear about the assumptions inherent in this approach, and the situations under which a sparseness criterion alone will fail to minimise overall output entropy. For convenience, because it avoids complications with entropic quantities, the discussion in this section is based on discrete-valued codes, but the ideas can equally be applied to the continuous case.

Let us consider what happens if we take the sparseness assumption to its limit, while still assuming an information-preserving code. In this case, each pattern will be represented by (at most) a single active output (we may safely reject the case where all outputs are always inactive, which, though maximally sparse, is unable to convey any information). This takes us back to a winner-take-all style of activation, which we previously rejected in Section 3.5. Such a code does not *minimise* entropy as the outputs are not statistically independent (the fact that one output is active implies that all others are inactive), but since this is the direction in which any sparseness-promoting constraint will take us, it is worth considering whether this does at least give us a *low* entropy code.

If we consider the binary case, the bit entropy H_b (sum of the individual output entropies) for N distinct input patterns (and therefore N used outputs) is given by

$$H_b(\mathbf{x}) = \sum_{i=1}^{N} p_i \log p_i + \sum_{i=1}^{N} (1 - p_i) \log(1 - p_i)^{-1}$$

where p_i is the probability of the ith input pattern. Noting that the first term in this expression is simply the input entropy $H(\mathbf{x})$, we obtain the following expression for the redundancy in this code:

$$H_b(\mathbf{x}) - H(\mathbf{x}) = \sum_{i=1}^{N} (1 - p_i) \log(1 - p_i)^{-1}.$$

This has its maximum value when all patterns are equally probable ($p_i = 1/N$), giving $N(1 - 1/N) \log(1 - 1/N)^{-1}$ as the upper bound on the redundancy. This expression approaches a maximum of approximately 1.44 bits as N becomes large, and so a single-bit code of this type will have *at most* that amount of redundancy. This tells us that a maximally sparse binary information-preserving code is guaranteed to have low entropy, and gives some additional justification for pursuing sparseness.

The reason that this does not yield a *minimum* entropy code is that minimising terms independently is not the same as minimising their sum. The overall aim is to find the minimum of $\sum_{i=1}^{n} H(a_i)$. While the minimum may be at a point where each of the terms in the summation is small, it might equally be where one term is large and the remainder zero, or somewhere in between. In other words, a minimum entropy code could require entropy to be spread thinly between many elements, or concentrated on just a few.

This gives us a feel for why finding minimum entropy codes is a combinatorially hard problem: for each element in isolation we do not know whether we should be attempting to increase or decrease the entropy in order to work towards the global minimum. It also suggests that algorithms that claim to perform general-purpose ICA by gradient descent are likely to be making strong hidden assumptions, or be beset by problems of local minima.

The assumption we are making by attempting to minimise the sparseness of all outputs individually is that the entropy is indeed spread reasonably evenly between the outputs at the minimum entropy solution. This technique can be prevented from degenerating to the WTA solution (which is unlikely to represent the true structure of the data) by the loose bottleneck at the output, as discussed in Section 5.1. Put another way, sparseness-promoting penalties tend to spread entropy thinly between outputs, but we can prevent this from becoming *too* thin by limiting the number of outputs available.

The shapes problem (Section 4.12.2, page 58) provides a concrete example of some of these points. Firstly we note that the sparsest code (in terms of minimising the expected sum of output values) for this data set is one that represents each of the 4096 distinct patterns separately. Limiting the number of output units enables us to prevent this unwanted solution.

Secondly, however, the solution presented in Figure 4.8 does not give a minimum entropy code: each pattern contains only one example of each shape, so the fact that one 'square-detector' is active, for example, means that all of the others will be inactive. The minimum entropy code in this case has only three elements, each representing one of the three shapes, and each taking on one of sixteen discrete values indicating the position of the shape on the grid. A system that discovers such a code has 'realised' that, in this example, the concepts of 'what' and 'where' are independent. The linear model of the REC network has no means of generating such a code, however, so we have to be content with a solution that has higher

entropy, and greater sparseness, than the optimum, and perhaps argue that the extraction of position is a job for a higher-level process.

5.7 Overcomplete representations and linear programming

Penalties were introduced to the REC network in Section 5.3 to address two problems: first, to resolve ambiguity in *activation* where reconstruction error alone yields multiple solutions, and second, to resolve ambiguity in *learning* by 'pushing' outputs toward their desired values. We shall look here at a way to resolve the first ambiguity alone, *i.e.* to guarantee unique (and useful) solutions in activation, but without considering the effects on learning. If efficient, such a technique could be useful as a fast means of activating a REC network *after* the weights have reached their optimal values.

As mentioned earlier, the ambiguity in activation only occurs when there is some form of linear dependency in the weight vectors, resulting in a line (or plane, *etc.*) of solutions. We shall concentrate primarily on the case where there are more weight vectors than input dimensions ($n > m$), because the weights *must* be linearly dependent in this case. We shall say that such a set of weights provides an *overcomplete* basis for representing input patterns.[3]

Overcomplete representations are of interest to us, because it is quite possible for a set of data to have more independent features than dimensions (Section 5.1), a fact often overlooked in the development of ICA algorithms. They are also of biological significance, because it appears that such representations are used in the mammalian brain. There is, for example, a vast increase in numbers of neurons between the retina and the primary visual cortex, an issue discussed in this context by Olshausen and Field (1997).

One method of resolving the ambiguity inherent in an overcomplete basis was presented in Section 4.5: using the pseudoinverse of the weight matrix will result in the *shortest* output vector $\mathbf{a}$ that fulfils the least-squares criterion, *i.e.* the solution for which $\sum_{i=1}^{n} a_i^2$ is minimised. This measure is often termed the L_2-norm. More generally, the L_r-norm of an n-element vector $\mathbf{a}$ is defined as

$$||\mathbf{a}||_r \equiv \sum_{i=1}^{n} |a_i|^r.$$

For the case where $r = 1$, this equation becomes identical to the penalty term Ω (Section 5.3) when each of the individual penalty functions is absolute-linear ($\omega(a) = |a|$). Since this constraint was adopted to find sparse codes, we see that minimising the L_1-norm, subject to achieving zero reconstruction error, finds the *sparsest* way to represent a point in an overcomplete basis. Similarly, we may reject the L_2-norm minimisation given by the pseudoinverse for the same reason we rejected the a^2 penalty function as a sparseness-promoting penalty. The difference between the two is illustrated in Figure 5.6 by a simple example.

L_1-norm minimisation for the non-negative case may be expressed as a linear programming (LP) problem. The *standard form* for linear programming (*e.g.* Luenberger, 1984) may

[3]It is also of course possible to have linearly dependent weights when $n \leq m$, corresponding to the situation where the model can only represent a subspace of the input space but produces an overcomplete representation *within* this subspace.

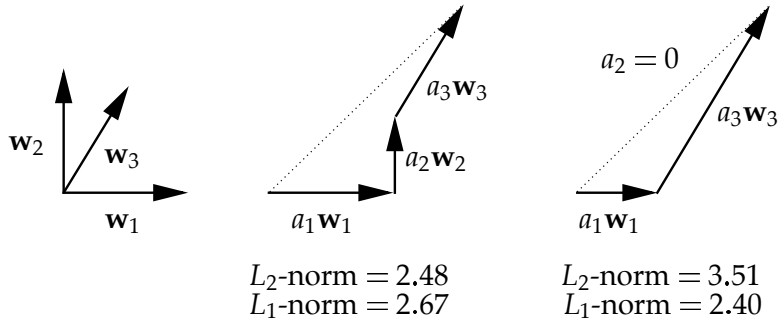


Figure 5.6: A demonstration of the difference between the representations in an overcomplete system resulting from minimisation of the L_2- and L_1-norms. The three 2-D basis vectors, $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$, each of unit length, are shown on the left. The representations of a 'data point' (shown as a dotted line) are shown for a minimisation of the L_2-norm (centre) and the L_1-norm (right) of the vector $\mathbf{a} = (a_1, a_2, a_3)^{\mathrm{T}}$. Note that for the representation with minimum L_1-norm, the vector $\mathbf{w}_2$, whose direction is furthest from the data point, is not used at all.

be written as

$$\begin{aligned} &\text{Minimise} \quad \mathbf{c}^{\mathrm{T}}\mathbf{a} \\ &\text{subject to} \quad \mathbf{W}\mathbf{a} = \mathbf{x} \quad \text{and} \quad \mathbf{a} \geq \mathbf{0} \end{aligned} \tag{5.20}$$

where the vector inequality $\mathbf{a} \geq \mathbf{0}$ means that all elements of $\mathbf{a}$ are non-negative. The standard notation associated with LP has been adjusted here to fit in with the network model. The problem of finding the representation with minimum L_1-norm in an overcomplete system now simply involves setting $\mathbf{c} = \mathbf{1}$, *i.e.* a column vector with all elements equal to one (Chen, 1995).

Note that the non-negative constraint on the elements a_i is required here because setting $\mathbf{c} = \mathbf{1}$ creates an objective function of $\sum_{i=1}^{n} a_i$, which is the L_1-norm but without the absolute value operator. However, removing this constraint requires no more than expanding the matrix $\mathbf{W}$ with a negated version, *i.e.* replacing $\mathbf{W}$ in (5.20) with $\mathbf{W}' = (\mathbf{W}, -\mathbf{W})$ and $\mathbf{a}$ with $\mathbf{a}' = (\mathbf{u}; \mathbf{v})$, where $\mathbf{u}$ and $\mathbf{v}$ are both n-element column vectors. Having performed the minimisation, a set of coefficients *without* the non-negative constraint is given by $\mathbf{a} = \mathbf{u} - \mathbf{v}$.

In coping with overcomplete representations in this way, the minimisation has been switched entirely to the L_1-norm of the output vector $\mathbf{a}$, and zero reconstruction error has become a hard constraint. The assumption here is that an exact reconstruction will always be possible, *i.e.* that the matrix $\mathbf{W}$ has rank m. If this were not the case, a sensible representation could still be found using a two-stage process: first, project the input down into the subspace spanned by $\mathbf{W}$, subject to the least-squares criterion, then use this projected version as the input to the LP problem (5.20). The only problem with this approach is that by ordering the two stages it gives fixed priorities to the two criteria of reconstruction and sparseness, with minimisation of reconstruction error taking absolute precedence. By contrast, the use of penalty terms gives a way to *adjust* the relative priorities through the parameter λ.

The advantage however of casting the task as an LP problem is that it makes available a range of techniques tailored specifically to linear programming (*e.g.* Gill *et al.*, 1991). These methods have been the subject of much research in recent years, and are likely to be significantly more efficient than more general-purpose optimisers. As noted above, the absence of

residuals means that the technique is not directly applicable to the REC network during the learning phase, but the next section looks at a method that could overcome this limitation.

5.8 Promoting sparseness by under-activation

The primary role of introducing a penalty term into the activation error function for the REC network is to induce an extra reconstruction error to help drive the learning process toward a desired solution. For 'sparseness penalties,' *i.e.* all monotonically increasing functions of $|a_i|$, this will in all cases result in an output vector $\mathbf{a}$ that is closer to the origin than would have been the case without the penalty. In the light of this, it is worth asking whether a more simple-minded approach could achieve the same objective. In this section we shall examine what happens when we remove the penalty from the error function, and simply move each output value by a fixed amount toward zero after activation.

We shall assume for simplicity that $n = m$ and that the weight matrix $\mathbf{W}$ has full rank (but the ideas remain applicable in the more general case). Under this assumption, every input pattern $\mathbf{x}$ may be represented without error in the unpenalised case. Moving the resulting output value a_i by a fixed amount λ toward zero artificially induces a reconstruction error of $\lambda \operatorname{sgn}(a_i)\mathbf{w}_i$, where

$$\operatorname{sgn}(a) = \begin{cases} -1 & \text{if } a < 0, \\ 0 & \text{if } a = 0, \\ +1 & \text{if } a > 0. \end{cases}$$

We are assuming that the change in the output value is determined solely by its original sign, so that if the value was originally close to zero, the sign may be different after the adjustment, *i.e.* the change in value may 'overshoot' zero. This is mathematically convenient, but will fail in some cases to achieve the stated aim of making all the coefficients a_i closer to zero. Nevertheless, overshoot may always be prevented in a majority of cases by making λ suitably small.

Applying these changes to all of the outputs results in a total reconstruction error $\mathbf{r}$ of

$$\mathbf{r} = \lambda \sum_{i=1}^{n} \operatorname{sgn}(a_i)\mathbf{w}_i. \tag{5.21}$$

Using the Hebbian learning rule as previously (4.15), the weight updates are given by

$$\Delta\mathbf{w}_i = \eta\,\lambda(a_i - \lambda \operatorname{sgn}(a_i)) \sum_{k=1}^{n} \operatorname{sgn}(a_k)\mathbf{w}_k \tag{5.22}$$

where a_i represents the output values *before* applying the changes toward zero. It appears difficult to associate a Lyapunov (objective) function with this update rule, but it is informative to examine the situation in a stable state (*i.e.* where $\forall i, \Delta\mathbf{w}_i = \mathbf{0}$), assuming such a state can be reached. If we assume λ to be small in comparison with the expected absolute values of the outputs a_i, then for purposes of analysis, we may approximate (5.22) as

$$\Delta\mathbf{w}_i \approx \eta\,\lambda\,a_i \sum_{k=1}^{n} \operatorname{sgn}(a_k)\mathbf{w}_k.$$

To achieve a stable state under this assumption therefore requires that

$$\mathcal{E}\left[a_i \sum_{k=1}^{n} \operatorname{sgn}(a_k)\mathbf{w}_k\right] = \mathbf{0}. \tag{5.23}$$

In fact, in the absence of bounds on the weights such a state will never be reached, because the residuals introduced to promote sparseness cause the magnitude of the weight values to grow without bound. If we assume that the weight vectors are constrained by normalisation, then any changes that occur along the direction of the weight vector will have no effect. Subtracting these out results in a stability requirement of

$$\mathcal{E}[a_i\{\mathbf{r} - (\mathbf{r}^{\mathrm{T}}\mathbf{w}_i)\mathbf{w}_i\}] = \mathbf{0} \tag{5.24}$$

where $\mathbf{r}$ is the residual, as defined in (5.21). In particular, we note that the term in the summation where $k = i$ in (5.23) may be ignored because it represents a change solely along the direction of the weight vector. Using this fact, and substituting (5.21) into (5.24) for the case where $n = 2$, the twin requirements for a stable state are found to be

$$\begin{aligned}
&\mathcal{E}[a_1 \operatorname{sgn}(a_2)\{\mathbf{w}_2 - (\mathbf{w}_2^{\mathrm{T}}\mathbf{w}_1)\mathbf{w}_1\}] = \mathbf{0} \quad \text{and} \\
&\mathcal{E}[a_2 \operatorname{sgn}(a_1)\{\mathbf{w}_1 - (\mathbf{w}_1^{\mathrm{T}}\mathbf{w}_2)\mathbf{w}_2\}] = \mathbf{0}
\end{aligned}$$

which, since we assume that $\mathbf{w}_1$ and $\mathbf{w}_2$ are not parallel, implies that

$$\begin{aligned}
&\mathcal{E}[a_1 \operatorname{sgn}(a_2)] = 0 \quad \text{and} \\
&\mathcal{E}[a_2 \operatorname{sgn}(a_1)] = 0.
\end{aligned} \tag{5.25}$$

We may compare these conditions with the requirement for two decorrelated zero-mean variables, *i.e.* the requirement that $\mathcal{E}[a_1 a_2] = 0$. It is not so easy to make a statistical interpretation of (5.25), although some insight may be gained by attempting a polynomial expansion of $\operatorname{sgn}(a)$. This is not possible directly because of the discontinuity at $a = 0$, but may be approximated (close to zero) by a hyperbolic tangent with sharp cutoff, or by the Fourier series for a square wave. In either case, the Taylor expansion results, as we would expect for an odd function, in a series of terms in all of the odd powers of a.

Broadly, therefore, the conditions (5.25) are requiring some function of the higher order joint statistics of the output values to be zero. This is potentially useful because if two random variables a_1 and a_2 are statistically independent, then we know that

$$\mathcal{E}[f(a_1)g(a_2)] = \mathcal{E}[f(a_1)]\,\mathcal{E}[g(a_2)] \tag{5.26}$$

for any functions f and g. The right-hand side of (5.26) will be zero when a_1 and a_2 are symmetrically distributed and f and g are odd, so we may interpret the fulfilment of (5.25) as evidence (although certainly not proof) of the statistical independence of a_1 and a_2 assuming they are symmetrically distributed about zero. The functions f and g are often known as *contrast functions* (Comon, 1994).

We see that our technique of under-activation has, with two outputs, produced the functions $f(a) = a$ and $g(a) = sgn(a)$. These are among the possible functions suggested by Jutten and Herault (1991) for their 'blind separation' algorithm. Their effect is illustrated in Figure 5.7. The solid lines show the weight values where the two conditions in (5.25) are met

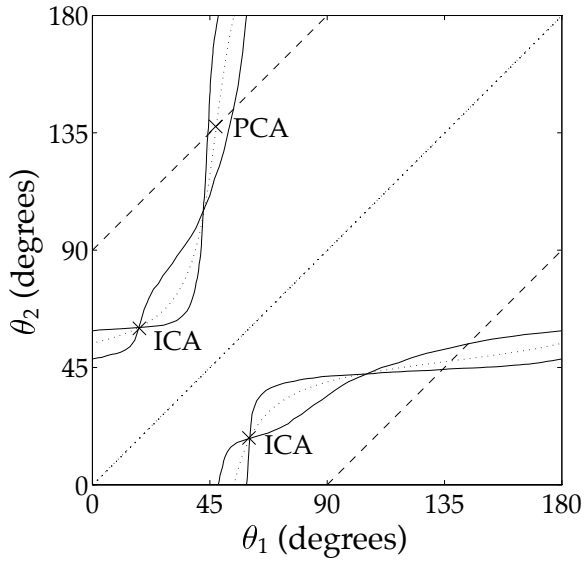


Figure 5.7: Three zero-contours from the same sparse data as was used to plot Figure 3.5 (31, *q.v.*). The values θ_1 and θ_2 are the angles to the horizontal of two weight vectors, $\mathbf{w}_1$ and $\mathbf{w}_2$, with the (curved) lines showing the values for which $\mathcal{E}[f(a_1)g(a_2)] = 0$, for the output values a_1 and a_2, and various functions f and g. The dotted line shows the zero contour where f and g are identity functions, and hence where a_1 and a_2 are uncorrelated, as in the earlier diagram. The solid lines show the cases where $f(a) = a$ and $g(a) = \text{sgn}(a)$, and *vice versa*.

for an example using the two-dimensional sparse data shown previously in Figure 3.4(b). The conditions are only satisfied simultaneously at two points (excluding permutations of the weight vectors), one of which is where the weights represent the underlying independent components of the data, marked 'ICA.' Further empirical investigation shows the other point to be an unstable solution, in that even if the network in started in this state, the weight vectors will, under the algorithm set out above, migrate to the independent component solution.

The technique that has been presented here is somewhat *ad hoc*, and indeed has only been analysed in detail for the case $n = 2$, but is nevertheless interesting for a number of reasons. Firstly it suggests that there may be a sparseness assumption implicit in the Jutten and Herault approach to ICA, at least for certain contrast functions, and this may indicate why the algorithm has been shown to fail in more general cases (Comon *et al.*, 1991; Sorouchyari, 1991). Secondly, this technique could well be useful as a fast means of applying sparseness constraints to the REC network. While activation with penalties requires a potentially costly minimisation, this method allows the use of faster techniques such as the pseudoinverse (Section 7.4.1) or linear programming (Section 5.7) for the activation of the network. Experiments demonstrating its use are described in Sections 5.9.1 and 7.3.2. Finally, even where penalties *are* applied as part of the activation process, we have an indication that we may be able to get away without doing a very good job of the minimisation, for example by only using a limited number of iterations. An experiment taking advantage of this idea is presented in Section 7.1.

5.9 Experiments

A number of ways to constrain the solutions produced by the REC network have been presented and discussed in this chapter. In this section we look at some simple experiments that use these ideas, and for which the unconstrained network of Chapter 4 would have been unable to provide correct results.

5.9.1 Independent components from linear data

In this series of experiments, the REC network was tested on a number of examples where data were generated by linear superposition of univariate distributions, as set out on page 29. In each case, appropriate priors were applied using soft and/or hard constraints. The problems tackled here are often described as *linear ICA* (Deco and Obradovic, 1996, Chap. 4) or *blind deconvolution* (Bell and Sejnowski, 1995).

Two sparse distributions

The first experiment used the data previously shown in Figure 3.4(b) (page 30), generated by the superposition of two (Weibull) distributions peaked at zero. A two-output REC network was used, with a linear penalty (Equation 5.6) applied during activation. The objective function is therefore

$$E_p = ||\mathbf{x} - \widetilde{\mathbf{x}}||^2 + \lambda \sum_{i=1}^{n} |a_i|.$$

The penalty parameter λ was set to 0.1. The network was trained with a learning rate η of 0.005, with weight updates applied after every 500 patterns, and the weight vectors kept normalised to unit length.

The results of a typical run are shown in Figure 5.8(a). The values of the two weight vectors after every five weight updates are plotted as circles and crosses respectively, with the final positions shown as solid lines. The network has correctly discovered the orientations of the two independent distributions (to within $\pm 0.1°$).

The technique of under-activating the network by a fixed amount rather than using penalties (Section 5.8) was also applied to this problem, with qualitatively very similar results.

Two uniform distributions

The second experiment used data generated by two uniform (non-sparse) distributions over the interval $[-1, 1]$. No penalties were used, but the network's outputs were constrained to lie within the same $[-1, 1]$ interval. Results from a run with a high learning rate (5.0) are shown in Figure 5.8(b). The weight vectors were again constrained to have unit length. Updates were applied after each pattern, and the trajectories are plotted at intervals of 5000 updates.

The weight values moved rapidly towards their final values at first, but then increasingly slowly. This is because residual errors are produced only by points in the 'corners' of the distribution, which are unreachable by incorrectly oriented weight vectors. Both the

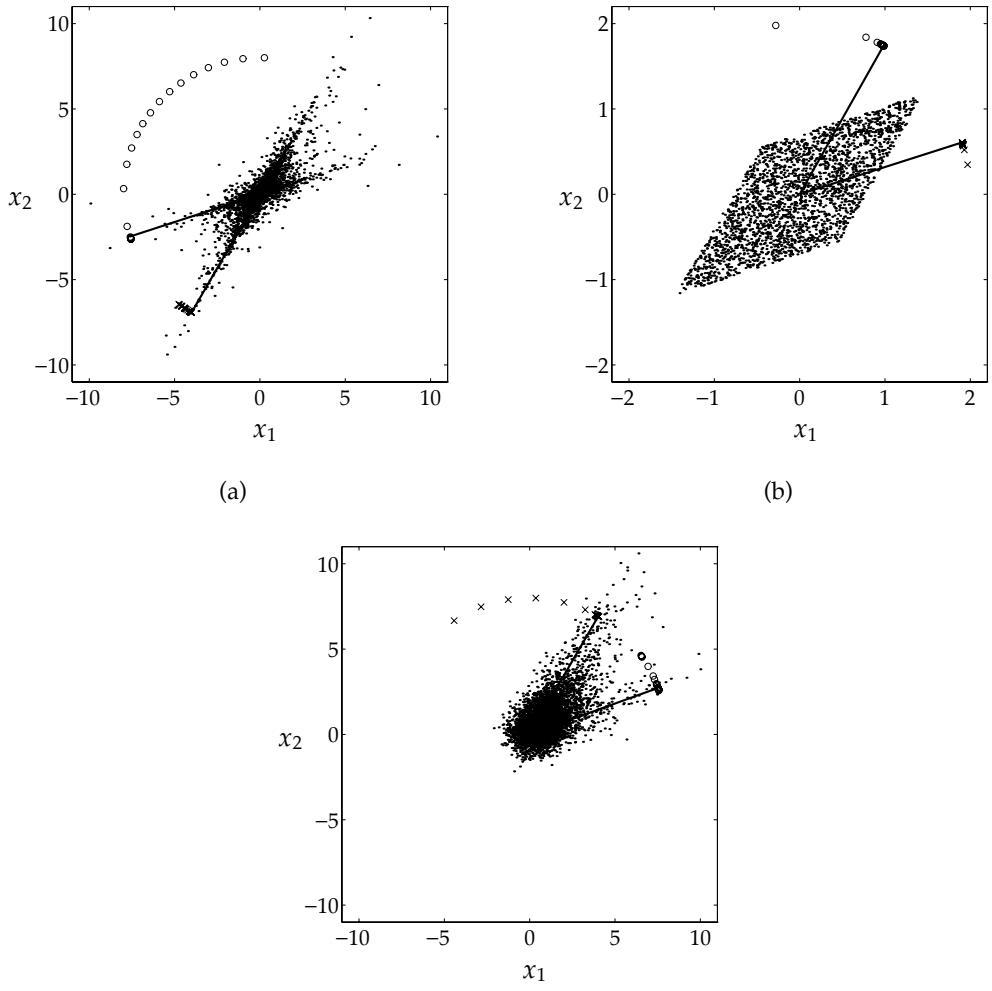


Figure 5.8: Samples of data generated by the linear superposition of two independent univariate random variables in two dimensions. In each case the underlying distributions are oriented at 18° and 60° to the horizontal, but the densities are different, as described in the main text. The sample size is 5000 for (a) and (c) and 2500 for (b). The data sets (a) and (b) are noise-free, but (c) is corrupted by spherical Gaussian noise of variance 0.4. Each diagram also shows the trajectories of the weight vectors for a REC network trained on the data, with initial and intermediate values shown by a series of circles or crosses, and the final values represented with a solid line.

number of unreachable points and the magnitude of the errors decrease as the weights near the solution.

In this case we have used precise prior knowledge of the extents of the underlying distributions. This example is, however, more artificial than most, and is only included here because it is used as to illustrate several ICA techniques (*e.g.* Karhunen *et al.*, 1995; Deco and Obradovic, 1996). We should also note that we are using an extremely inefficient way of finding the underlying features in this case — if a set of data really were distributed in this way, we would be much better off using a convex hull type method (as discussed in Section 5.2) to find the solution.

Two non-negative distributions with noise

A third experiment returned to the example shown in Figure 5.1(c) (page 68), *i.e.* where the data are generated by non-negative distributions, peaked at zero, with added spherical Gaussian noise. The network outputs were constrained to be non-negative, and a linear penalty was applied.

A sample of the data, and some sample results, are shown in Figure 5.8(c). The results obtained are sensitive to the value of the penalty parameter λ in this case: if set too low, the final weight vectors are spread too widely, the problem observed in the absence of penalties in Section 5.2; if set too high the final weight directions tend to be too close together, although this effect is far less pronounced. Although we noted in Section 5.4 that the parameter λ has a direct relation to noise variance, this assumes that we can know in advance the exact form of the 'signal' distributions, including their variance. This is unrealistic for real data, but we should often be able to make a reasonable guess.

The results depicted in Figure 5.8(c) were obtained for $\lambda = 1.5$, and gave final weight values to within 2° of the correct orientations. The learning rate η was 0.005, weight updates were applied after 500 patterns, and the weight trajectories have been plotted after every 10 updates.

The continuous-valued feature problem

In a fourth set of experiments, the network was tested on a higher-dimensional problem using the data first introduced in Section 4.12.3 (page 61). As noted there, and in Section 5.2, the non-negative constraint alone is not sufficient to force the REC network to produce exact solutions.

By additional application of a linear sparseness-promoting penalty, it proved possible to find accurate matches to the underlying features in both noise-free and noise-corrupted cases. For noise-free data, precise results could be obtained rapidly by beginning training with a high value of λ, and subsequently reducing it towards zero. In the noisy case, as noted above, the optimal choice of λ is dependent on the noise variance.

5.9.2 Overcomplete representations

This experiment concentrates on an example that, to the author's knowledge, none of the existing ICA techniques are able to tackle, namely one where the number of independent

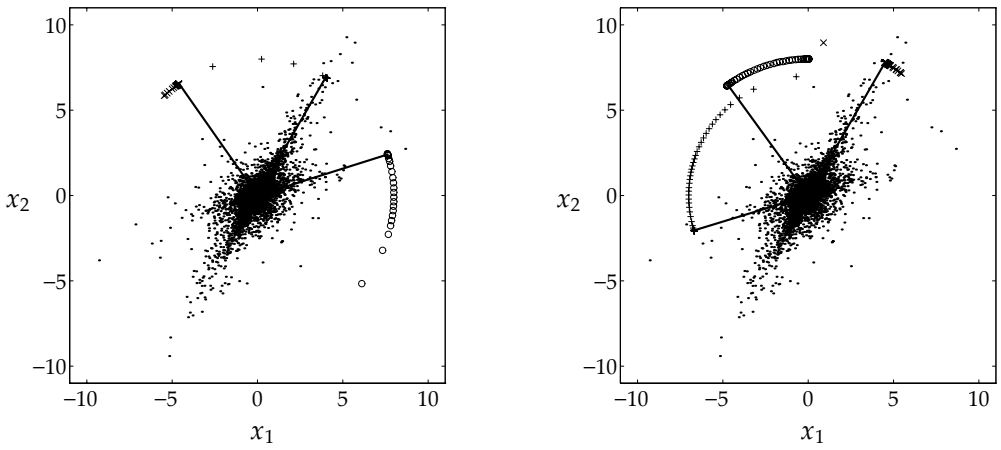


Figure 5.9: Samples generated from three independent univariate distributions in two dimensions, thus requiring an *overcomplete* system to represent correctly the underlying structure. (a) Results obtained from a REC network trained on these data from random initial starting weights. The trajectories of the weight values are represented as previously, using circles, crosses, and pluses. (b) Results from the same network with the task deliberately made harder by setting the initial orientations of the weight vectors close to each other at around 90° to the horizontal.

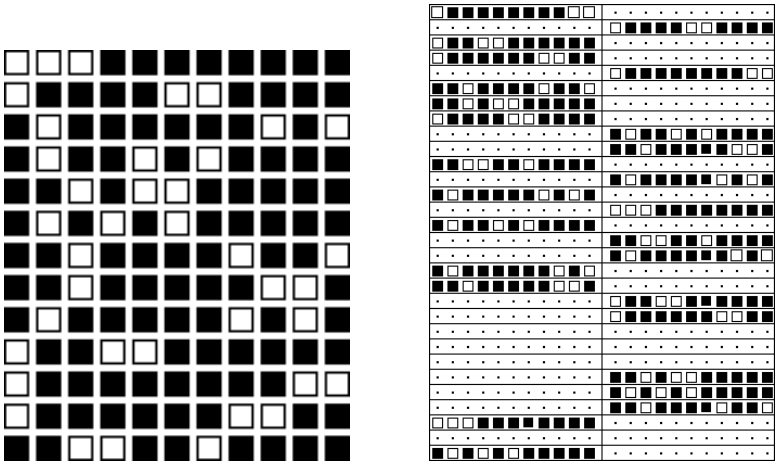
features is greater than the number of input dimensions.

To find these features with a REC network, we require that the number of outputs exceeds the number of inputs ($n > m$). The penalty functions now take on two simultaneous roles: first to resolve the ambiguity in activation caused by an overcomplete linear representation, and second, as in the previous experiments, to induce errors that push the weight vectors toward the underlying features.

In this example, a set of data points was generated by the superposition of three univariate distributions in two dimensions. Double-tailed Weibull distributions (Equation 3.17) were used, with parameters $\alpha = 1$ and $\beta = \frac{2}{3}$, as previously. They were oriented at angles of 18°, 60° and 125° to the horizontal, with variances of 0.7, 2.0 and 0.2 respectively.

A three-output REC network was used, and a linear penalty applied with a parameter λ of 0.01. The network was trained with a learning rate η of 0.005, with weight updates applied after 500 patterns, and the weights normalised to unit length after each update. The results of a typical run with random initial weights are shown in Figure 5.9(a). The paths of the weight values during learning (plotted after every 10 updates) show that the high-variance component (at 60°) has been picked out very quickly, with the network being slower to discover accurately those with lower variance.

As an empirical test for the presence of local minima, the same network was retrained with the initial orientations of the weights set very close to each other (at around 90° to the horizontal). There is a danger in this case that only two of the features will be found, with one of the weight vectors 'stuck' in between. The results are plotted in Figure 5.9(b). This shows that the high-variance feature was again picked out very rapidly, but the corresponding weight vector 'overshot' slightly to compensate for the fact that nothing, as yet, was properly



(a) (b)

Figure 5.10: (a) The 'dictionary' of binary patterns used by both the left and the right processes to generate the dual-component data set. Each row represents a different one of the thirteen possible patterns, with a white square representing one and a black square minus one, as previously. (b) An example of the weight vectors produced by a thirty-output REC network trained on the dual-component data. Each row represents a different weight vector. The left and right halves are divided with a line for illustrative purposes, but this structure was in no way enforced by the network architecture.

representing the distribution oriented at 18°. The progress of a second weight vector towards this orientation was slowed, but not halted, by having to pass over the 125° feature, with the third remaining largely static until this had happened. The network took approximately three times as long to reach the solution as in the previous example.

In a final test, the weights were initialised at around 45°, between the two higher-variance features. Even this did not prevent the correct solution from being found, although the time taken was slightly longer again. In each of these experiments the final orientations of the three weight vectors matched those of the underlying features to within half a degree.

5.9.3 Dual-component data set

The 'dual-component' data set was introduced by Zemel (1993) as an example requiring a competitive system that is nevertheless able to represent two pattern elements simultaneously. There are twenty-two inputs. Patterns are created by two independent processes, one producing the first eleven inputs, and the other the remaining eleven. Each process generates one of thirteen possible binary patterns (each occurring with equal probability). The 'dictionaries' of possible patterns are the same for each process, and are shown in Figure 5.10(a). A good solution to the problem will identify the two halves of the input as coming from independent sources. Unlike previous examples, a binary zero is here represented by a minus one input value, to allow a zero weight to represent a new 'don't care' value, an issue that will be discussed further in Section 6.3.2.

An example of the weights produced by a REC network with thirty outputs is shown in Figure 5.10(b). With these weights, the network will encode every input pattern as desired

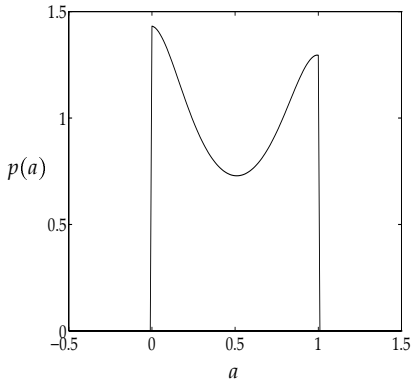


Figure 5.11: The prior probability density function imposed upon output values of the REC network used to tackle the dual-component problem, produced by a combination of hard constraints and both sparseness-promoting and 'binary' penalties.

with two active outputs, each representing one half of the input. Learning this representation is, however, not an easy task for what is essentially a continuous-valued projection-based network, for a number of reasons.

Firstly, we may note that for a purely linear model, the problem requires an overcomplete representation, since there are only 22 inputs, but a minimum of 26 outputs are needed for the correct solution. Secondly the solution requires a very sparse code, because we expect to find only two active units for each input pattern. Thirdly, even when the inputs are correctly separated into two halves, the desired representation is still overcomplete, since we require a set of 11-dimensional values to be represented by 13 output values.

To obtain solutions of the type shown in Figure 5.10(b), therefore, a number of constraints were applied to the network:

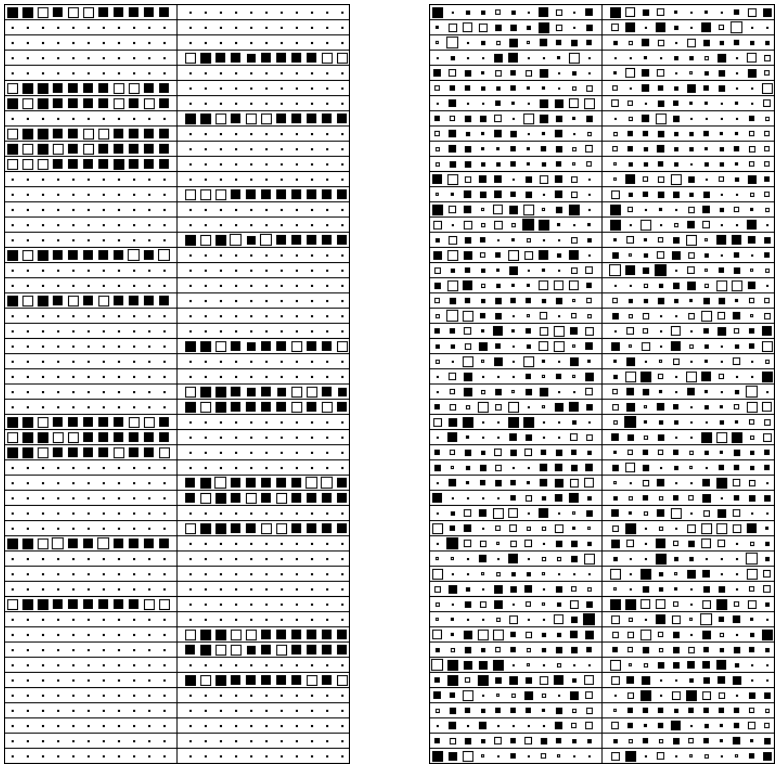
1. The output values were constrained to be non-negative (Section 5.2).
2. A linear sparseness-promoting penalty (5.6) was applied during activation.
3. A 'binary' penalty (5.5) was also applied, and an upper bound of one placed on the output values.
4. Weight values were constrained to lie in the range $[-1, 1]$.
5. Weight decay (Section 5.5) was applied during learning.

The exact form of the output penalty term was

$$\omega(a) = 100.[a^2 + (a-1)^2] + a. \tag{5.27}$$

This was combined with the reconstruction error as in (5.4) with the penalty parameter λ set to 0.3. A graph of the prior placed on the output values by using this penalty function, combined with the 'hard' constraint that the values must lie in the range $[0, 1]$ is shown in Figure 5.11. This function was calculated from (5.27) using $c.\exp[-\omega(a)]$, where c is a normalising constant.

Weight values were randomly initialised in the range $[-0.05, 0.05)$, with these small values being used merely so that the weight values of 'unused' units do not clutter Figure 5.10(b). The learning rate η was fixed at 0.5. Weight decay was applied using (5.19)



(a) (b)

Figure 5.12: (a) An example of the weight values produced by a fifty-output REC network trained on the dual-component data, showing that the penalties cause a sparse representation to be produced even when the output bottleneck is loosened. (b) An example of the final weight values produced by the same network, but with the penalty terms omitted from the error function.

with a parameter $\lambda_w = 0.01$. A stable state of the form shown in Figure 5.10(b) was typically reached after about 1000 pattern presentations. Constraints 3 and 5 in the above list were not essential to obtaining good solutions, but their inclusion in this instance was found to speed convergence by a factor of two or more.

The use of the linear penalty (constraint 2) *was* however found to be essential. Even when the network had many more outputs than required, its use prevented unwanted solutions, such as cases where units represent only a few bits of a pattern, from being found. An example of the representation produced by a network with fifty outputs, but otherwise unchanged, is shown in Figure 5.12(a). Without any form of penalty ($\lambda = 0$), a network of this size will quickly converge to a seemingly random representation of the input, an example of which is shown in Figure 5.12(b).

A fair degree of domain-specific knowledge has been used in this experiment in order to achieve the desired solution. Although it might appear unreasonable to expect to have this amount of prior knowledge for real-world problems, the system is nevertheless arguably more flexible than Zemel's (1993) original solution to this problem, which requires the dual nature of the data to be 'wired in' to the model to enable it to generate the desired code.

5.10 Discussion

This chapter has looked at various means of constraining the REC model to enable it to find *efficient* and not just *complete* codes for its input data. It is accepted that this approach requires various prior assumptions about the nature of the data. The contention is, however, that similar assumptions are being made by most other feature-detection techniques. Various projection pursuit and ICA methods, for example, use projection indices and contrast functions that only partially capture higher order statistics. This implicitly introduces assumptions, but in a way that prevents us from understanding their exact nature. It is argued that where, as here, the constraints are explicit and easily understood, we are in a much better position to evaluate their applicability to particular problems, and modify them where necessary.

The next chapter takes a sideways step to explore modifications to the REC network's linear generative model, but we shall return in Chapter 7 to the techniques developed here, in order to look at their application to real-world problems.

Chapter 6

Mixture Models

The modelling framework used in this thesis incorporates the idea that single data points may be produced by multiple causes, *i.e.* that the influences of several processes may combine to produce the observed data. The way in which this combination is achieved is often termed the *mixture model*,[1] and we shall refer to the function underlying this as the *mixing function*. Up to this point we have used linear summation as the mixing function. While this may be valid (or at least a reasonable approximation) in many cases, in this chapter we shall look at examples where more sophisticated mixture models are required, and at ways in which they can be incorporated into the REC network architecture.

6.1 Related work

Several of the issues relating to the use of mixing functions are examined by Saund (1995). He also discusses the closely related ideas of imaging models and voting rules. The term 'imaging model' is used in the fields of computer graphics and typography, where it refers to the way in which different layers of colour are combined on a surface. A simple example would be a number of processes laying down white on a black surface, where one or more layers of white are sufficient to give a white result at any point — this is known as a 'write-white' imaging model, and corresponds to a logical OR mixing function.

'Voting rules' are methods of combining different signals, each representing some 'belief,' in order to achieve a consensus. In this terminology, the logical OR mixing function becomes a scheme whereby each individual can either vote 'on' or can abstain: a single 'on' vote is sufficient for the consensus to be 'on.'

Similar ideas are also found in the supervised neural network literature with mixtures of experts (Jacobs *et al.*, 1991) and ensembles or 'committees' of networks (Perrone and Cooper, 1993). The mixture of experts model has a network divided into separate subnets, with their outputs mixed under the control of a gating network. The mixing function is potentially quite complex, but in practice it tends to be used to select a single output from one of the subnets. Such an architecture can therefore be viewed as mixing at a slightly higher level, with each subnet becoming an 'expert' in a particular region of the input space, as

[1] The term 'mixture model' is also sometimes used to describe a linear combination of densities in probability density estimation — it is used here in a more general sense.

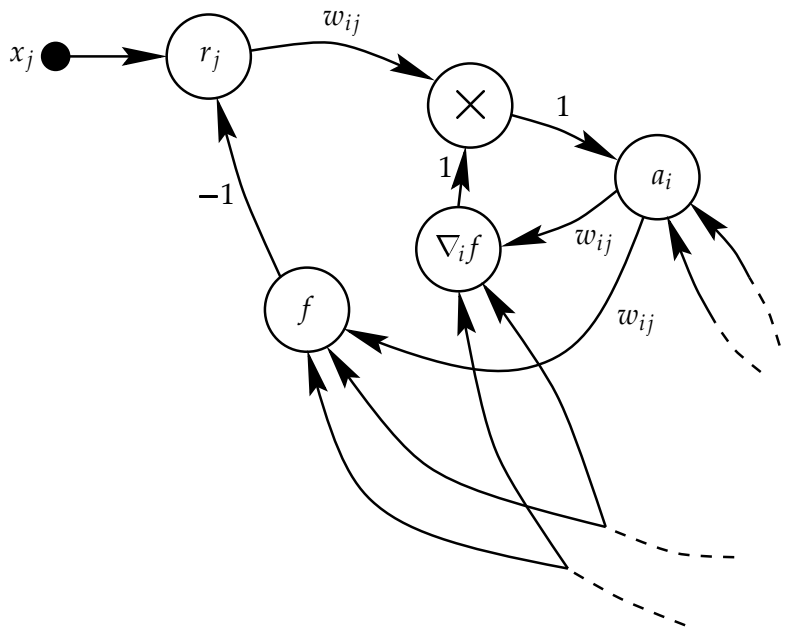


Figure 6.1: Part of a network implementation for a general mixing function f. The units labelled 'f' and '$\nabla_i f$' compute the corresponding function of their inputs. The unit labelled '$\times$' computes a product. The remaining two units are linear and labelled with their output values. The dashed lines represent feedforward and feedback connections from other first-layer and output units respectively.

opposed to combining multiple signals for a single input pattern as we shall consider here. At a higher level still, entire networks may be trained separately and their predictions subsequently combined. The outputs may be combined by averaging (for regression), majority voting (for classification), or a more complex weighted combination (Krogh and Vedelsby, 1995).

6.2 Incorporating mixing functions into a network

Before examining examples of mixture models for particular problem domains, we shall look in this section at how a general form of mixing function can be incorporated into the REC network model.

Consider a general mixing function f, which takes as arguments the values of all outputs weighted by the feedback connection weights to a particular input. This gives

$$\widetilde{x}_j = f(w_{1j}a_1, \ldots, w_{nj}a_n)$$

as the jth element of the reconstructed input, and

$$\frac{\partial E_r}{\partial a_i} = -2(\nabla_i f)(w_{1j}a_1, \ldots, w_{nj}a_n)\,[(\mathbf{x} - \widetilde{\mathbf{x}})^{\mathrm{T}}\mathbf{w}_i] \tag{6.1}$$

as the derivative of the reconstruction error, where $\nabla_i f$ is the partial derivative of f with respect to its ith argument. In the following equations the arguments to $\nabla_i f$ are omitted for brevity.

Figure 6.1 shows how this mixing function can be incorporated into the network. This is in effect a graphical representation of (6.1). The original REC model (Figure 4.2) was much simpler since f was linear, making $\nabla_i f = 1$, and removing the need for the extra units shown here.

Despite the extra complexity in activating the network for the general case, learning remains simple. The result from Appendix A tells us that the gradient of E_r with respect to the weights may be evaluated assuming the outputs **a** are fixed, so that

$$\frac{\partial E_r}{\partial w_{ij}} = -2(x_j - \widetilde{x}_j)(\nabla_i f)a_i. \tag{6.2}$$

Referring back to Figure 6.1, we see that (6.2) corresponds still to local Hebbian learning, since the input to the ith output unit along its jth link is given by $(x_j - \widetilde{x}_j)(\nabla_i f)$.

6.3 Models for binary patterns

Problems based on binary data give an example where the linear model is incorrect, since under this model two superimposed 'one's give a 'two,' which is outside the binary domain. We saw in the lines experiment of Section 4.12.1 that even with this incorrect model it is possible to get good results provided that the degree of overlap between features is reasonably small in comparison with the overall magnitude of the feature vectors. With substantial overlap, however, the inaccuracies of the mixture model prevent the correct solution from being found.

6.3.1 Write-white imaging models

The lines data set is formed according to a write-white imaging model.[2] As mentioned previously, this corresponds to a logical OR mixing function. To allow gradient descent activation and learning this can be 'softened' by linear interpolation to form a 'soft OR' function (Saund, 1995), which gives

$$\widetilde{x}_j = 1 - \prod_{k=1}^{n}(1 - a_k w_{kj}) \tag{6.3}$$

as the reconstructed input. A plot of this function (for $n = 2$ and $w_{1j} = w_{2j} = 1$) is shown in Figure 6.2(a).

In Saund's experiments, the objective function used was based on log-likelihood in the binary domain:

$$E_b = \sum_{i=1}^{m} \log[x_i \widetilde{x}_i + (1 - x_i)(1 - \widetilde{x}_i)]. \tag{6.4}$$

The experiments below (Section 6.5.1) explore the use of the soft OR mixing function in conjunction with the squared residual (Equation 4.5) as the error metric.

[2] Of course a write-black model (as used by Saund) is exactly equivalent if the binary representations for white and black are swapped. Write-white is used as the 'default' here simply because the majority of diagrams in this thesis use white to represent positive values.

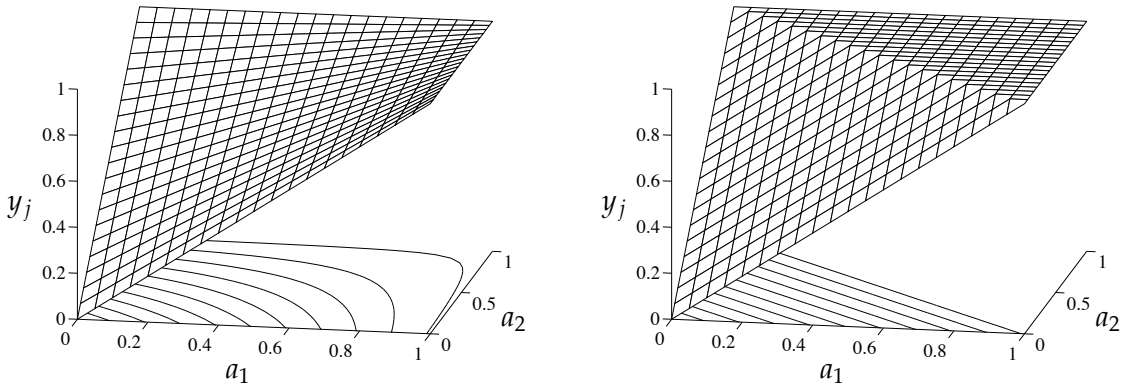


Figure 6.2: Mesh and contour plots of two possible mixing functions for use with a write-white imaging model. (a) The soft OR function (6.3), produced by linear interpolation between a zero at the origin and a one at all the other vertices of a $[0, 1]$-hypercube, shown here for $n = 2$ and $w_{1j} = w_{2j} = 1$. (b) A saturated linear function that, while not as theoretically attractive, lends itself more readily to a network implementation; the parameters are the same as for (a).

A network implementation of the soft OR function requires the full complexity of Figure 6.1. This complexity is undesirable, since it requires more computation to activate the network than previously: the evaluation of the error function and its gradient is more costly, and a minimiser will typically require more gradient calculations because of the non-quadratic nature of the error surface. It is worth considering therefore whether it is possible to produce a similar but simpler mixing function with a scheme closer to the original network model.

The principal requirements of the mixing function are that a combination of any number of ones should give a one, and that a complete set of zeroes should combine to give zero. A secondary requirement is that there be some form of interpolation between these binary limits, ideally with two or more values that are less than one combining to give an overall reconstruction that is also less than one. All of these requirements were met by the soft OR function of Figure 6.2(a), but if we relax the final requirement and allow multiple values below one to give a value of one when mixed, we can use the much simpler function shown in Figure 6.2(b). This is just a 'saturated' linear summation with a maximum value of one. Using this mixing function, we have that

$$\frac{\partial \widetilde{x}_j}{\partial a_i} = \begin{cases} 0 & \text{if } \widetilde{x}_j = 1, \\ w_{ij} & \text{otherwise.} \end{cases} \tag{6.5}$$

An extra modification makes life even easier. It is only essential to apply the saturation limit on $\widetilde{x}_j$ (in order to achieve a zero reconstruction error) when the input value x_j is one. When this is done the gradient of the reconstruction error with respect to the outputs is simply

$$\frac{\partial E_r}{\partial a_i} = -2\,(\mathbf{x} - \widetilde{\mathbf{x}}).\mathbf{w}_i$$

exactly as for the original network (4.6) — the case where $\widetilde{x}_j = 1$ in (6.5) no longer needs to be

considered separately, because $x_j - \widetilde{x}_j$ will be zero anyway. Another advantage is that this removes the discontinuity in the error gradient which causes problems for some minimisers.

The method for activating the output units therefore remains unchanged from the linear model (4.2), and all the modifications required to give the new mixture model can be incorporated into the first-layer units, which now compute

$$r_j = \begin{cases} x_j - \max(\sum_k a_k w_{kj}, 1) & \text{if } x_j > 1, \\ x_j - \sum_k a_k w_{kj} & \text{otherwise} \end{cases} \tag{6.6}$$

where max() is a function that returns the greater of its two arguments. Weight adjustment is achieved by standard Hebbian learning (4.15), but a slight modification can improve performance. Because patterns in this environment are produced using an OR operation, an input of zero means that *no* output unit should write to that location (*i.e. all* units should have a corresponding weight of zero), while an input of one means only that *some* unit or units should have a corresponding weight of one. As a result of this, negative error signals (resulting in downward pressure on weight values) are more significant than positive ones. This imbalance can be reflected by using a larger learning rate for negative weight changes than for positive ones, giving as the modified learning rule

$$\Delta w_{ij} = \begin{cases} \eta_- a_i r_j & \text{if } r_j < 0, \\ \eta_+ a_i r_j & \text{otherwise} \end{cases} \tag{6.7}$$

with $\eta_- > \eta_+$.

In Section 6.5.1, the method of mixing discussed here is applied to the lines problem, and its effectiveness compared with both the unmodified network and the soft OR mixing function. Before that we shall look at an extension of these ideas to slightly more complex models.

6.3.2 Write-white-and-black imaging models

It is also possible for binary patterns to be formed by processes that explicitly produce zeroes as well as ones. In terms of an imaging model, this means that they may write both white *and* black; in terms of voting rules, they can vote 'yes' *or* 'no,' or can abstain ('don't know/care').[3] We have already seen a problem of this type with the dual-component data set (Section 5.9.3). The solution used there was simply to encode white using $+1$, black using -1 and 'no colour' using 0, while continuing to use the linear mixing function. The limitations of this approach are analogous to those of the network used to tackle the lines problem in Section 4.12.1, namely that (for example) multiple ones will combine to give a value greater than one. This was not a problem when using the dual-component data set, since each input value was produced by exactly one process. In more complex environments, one would need to determine the appropriate voting rule — this could be a majority vote, or a softer variant whereby any process voting against the majority introduces some degree of doubt into the outcome, represented for example by values with a magnitude less than one.

[3]Such patterns might more properly be seen as ternary rather than binary since an input could take on one of three values: white, black or 'no colour.' In the case where at least one process writes white or black to each input value, however, the patterns *will* be binary.

Saund (1995) argues that a linear mixing function is not appropriate, because opposing predictions by different output units will tend to cancel out, resulting in 'little global pressure for [units] to adopt don't know values when they are not quite confident about their predictions.' He therefore suggests a mixing function with boundary values given by:

$$\widetilde{x}_j = \begin{cases} 0 & \text{if } \sum_k a_k ||w_{kj}|| = 0, \\ \frac{\sum_k a_k w_{kj}}{\sum_k a_k ||w_{kj}||} & \text{if, } \forall k, a_k w_{kj} \in \{-1, 0, 1\}. \end{cases}$$

The use of a normalised sum means that two opposing votes will not fully cancel out, but will produce a degree of uncertainty in the combined prediction, causing a finite reconstruction error and driving any prediction that conflicts with the consensus toward zero. The remaining values for $\widetilde{x}_j$ should ideally be determined by linear interpolation, but this has a cost that is exponential in n, so Saund suggests a tractable approximation.

It is, however, possible to avoid this considerable complexity. We saw in Section 5.9.3 how the required 'global pressure' to adopt don't know values could be provided instead by priors on the output values and weights (penalties and weight decay). In environments where, in addition, multiple processes potentially write to a single input, a scheme directly analogous to the saturated linear mixing function presented in Section 6.3.1 could be used.

6.4 Modelling occlusion

Occlusion in a visual scene occurs when one object is partially or fully obscured by another.[4] A number of approaches have been taken to discovering occlusion in machine vision (see Chiu, 1996, Chap. 1, for a review). These include the use of T-junctions in line drawings, first detailed by (Huffman, 1971), stereopsis (Malik, 1996), lighting effects (Yang and Kak, 1989), or motion of the viewer (Chiu, 1996) or of objects within the scene (Marshall *et al.*, 1996).

Here we shall look at a different means of determining occlusion, one that does not utilise knowledge about junctions, or any of the depth cues listed above. We shall also assume there is no perspective, meaning that all objects will appear the same size whatever their relative depth. A good real-world analogy is therefore one of painting objects onto a planar surface. A scene is produced by painting a selection of the objects in a particular order. The system's task, having been presented with a number of such scenes, is to determine the (unoccluded) appearance of each object. Under these conditions, it would be impossible to infer any depth information without some form of constraint on the visual environment: the constraint here is that there are a limited number of distinct objects[5] and we can therefore use bottlenecks and sparseness constraints (Chapter 5) to discover them.

The write-white imaging model could be considered a simple case of an occlusion model: one in which there is no need to consider the order in which objects are written, since the fact that they are all of the same colour makes it impossible to determine relative depth even if we wish to.

[4] It is also of course possible to have a concave object partially obscure itself, but for simplicity consideration here is limited to planar objects perpendicular to the line of sight

[5] As in all previous examples, the system has no translation invariance, so congruent objects at different locations are treated as distinct.

To add occlusion to the write-white-and-black imaging model however does require a notion of depth. In terms of voting rules, all processes vote in depth order, closest first: the first non-abstention is decisive. In other words, for a particular point in the scene, the imaging model considers *only* the closest object whose boundary includes that point.

This can be achieved with a mixing function of the following form:

$$\widetilde{x}_j = \sum_{i=1}^{n} b_{ij} a_i w_{ij}, \tag{6.8}$$

where b_{ij} is a weighting between the ith process and the jth input location based on the relative depth of the objects within the scene and i's 'vote.'

The boundary conditions for this model are that $a_i \in \{-1, 0, 1\}$ and $w_{ij} \in \{-1, 0, 1\}$ $(\forall i, j)$. The possible values for a_i include -1 here to allow for the case where the output values are allowed to become negative to represent the inverted form of their weight vectors (black becomes white, white becomes black). The depth of each object will be represented by d_i, with larger values used for objects closer to the observer. Under these conditions the mixing function should select, for each position j within the scene, the colour of the object i with the largest d_i from all those with non-zero a_i and non-zero w_{ij}. More formally,

$$b_{ij} = \begin{cases} 1 & \text{if } i \in S \text{ and } d_i = \max_{k \in S} d_k, \text{ where } S = \{k : ||a_k|| = 1 \text{ and } ||w_{kj}|| = 1\}, \\ 0 & \text{otherwise.} \end{cases} \tag{6.9}$$

As before, some form of 'softening' is required to deal with intermediate values. Firstly, we need a function that interpolates between the boundary values for the a_i and w_{ij} given by (6.9), such as:

$$b_{ij} = \begin{cases} 1 & \text{if } i = \arg\max_k (d_k ||a_k w_{kj}||) \\ 0 & \text{otherwise.} \end{cases} \tag{6.10}$$

Secondly, we need to soften the maximisation, to allow alternative possible sources for a particular point to compete during training. This can be achieved with a normalising function of the form

$$b_{ij} = \frac{g(d_i ||a_i w_{ij}||)}{\sum_{k=1}^{n} g(d_k ||a_k w_{kj}||)} \tag{6.11}$$

where $g()$ is a monotonically increasing function. When $g(x) = e^x$, this becomes the well-known 'softmax' function (Bridle, 1990a,b). In practice it was found that the use of softmax could cause problems for the minimiser, and better results were obtained using $g(x) = x^3$. A possible explanation for this is shown in Figure 6.3, where both versions of the maximisation function are plotted for $n = 2$. As the value of one of the arguments to the softmax function is changed, the threshold moves while the shape remains otherwise constant, as shown by the parallel contours in Figure 6.3(a). The cut-off for the x^3-normalisation, by contrast, is very sharp when one of the values is low, but the zero-one transition becomes more gradual as that value increases, producing contours that radiate from $(0, 0)$ in Figure 6.3(b). This results in the elimination of the plateau regions of the softmax function, which could explain the improved performance of gradient-based optimisers with this form of the function.

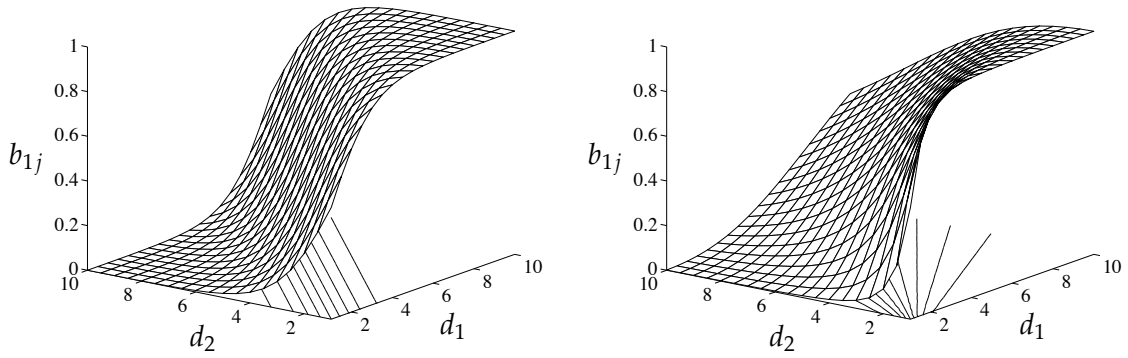


Figure 6.3: Mesh and contour plots of two softened maximisation functions. (a) The softmax function, produced by setting $g(x) = e^x$ in (6.11), and shown here for $n = 2$ and $a_1 = a_2 = w_{1j} = w_{2j} = \pm 1$. (b) An alternative function, with $g(x) = x^3$, and the parameters otherwise as in (a).

The error gradients with respect to both output and weight values are far from straightforward for this model, because the mixture weightings b_{ij} used to calculate the reconstructed input (6.8) are themselves functions of the outputs and weights, as are the depths d_i. For this reason, the output values are best calculated by a minimisation technique that does not require direct evaluations of the gradient (such as the simplex method). Learning poses a problem, but if we make the assumption that any residual error requires action from all active units, whatever their relative depth (since an error means that the depths may well be wrong anyway), then Hebbian learning (4.15) is again a natural choice, and has been found to work reasonably well in practice.

Experiments using this occlusion mixture model are presented below in Sections 6.5.2 and 6.5.3. Unfortunately there is no simple network interpretation of the calculations required by this model. One approach that might lend itself more readily to a network implementation would be to include a temporal mechanism, and 'play back' the objects in depth order, most distant first, in a method directly analogous to the use of a z-buffer in computer graphics. Softening could be incorporated as necessary by giving units performing the reconstruction a partial memory of previously written values. Such an approach is not discussed further here, but would merit further investigation.

6.5 Experiments

6.5.1 The lines problem revisited

Section 4.12.1 gave results produced by training the basic REC network on the lines data set, and it was noted at the time that the network's mixture model was incorrect for this data. In this section, we shall compare the performance of the original network on this task with both the soft OR and the saturated linear models of Section 6.3.1.

In the experiments each network was trained on a number of variants of the data set, each consisting of 10 000 randomly generated patterns with a different value of p (the probability

p	Linear	Soft OR	Saturated
0.2	524	165	123
0.3	1105	252	121
0.4	2002	478	137
0.5	7945	1225	169
0.6	—	8010	275
0.7	—	—	655
0.8	—	—	2728
0.9	—	—	39080

Table 6.1: The results from training three networks on the lines data set. The average number of patterns required to reach a stable solution is shown for each network (linear, soft OR, saturated linear), and for a range of values for the line probability p. A blank in the table indicates that the network showed no signs of reaching a solution within 50 000 patterns.

of a line appearing in each of the sixteen locations), the values varying from 0.2 up to 0.9 in steps of 0.1.

Each of the three networks (linear, soft OR, saturated-linear) had 20 output units, and weight changes were made after each pattern presentation. The linear network was given a 'binary' penalty (5.5) with parameter $\lambda = 1.0$, and trained with a learning rate η of 0.4. The soft OR network did not use penalised activation, and was trained with $\eta = 1.0$. The saturated linear network used a linear penalty (5.6) with $\lambda = 0.1$, and weight updates were calculated using (6.7) with $\eta_- = 2.4$ and $\eta_+ = 0.4$. The parameter values given here are the ones found to be best for a particular network, but the results obtained were broadly similar for a wide range of values.

The results for all three networks are shown in Table 6.1. For each type of network and value of p, the approximate number of pattern presentations required to reach a stable solution is shown. The numbers were produced by averaging the results from five runs of the network with different random initial weights. A network was judged to have reached a solution when all sixteen lines had been found, and when the weights for the units representing these lines were all either less than 0.15 or greater than 0.85.

As found previously, the linear network was able to find solutions reasonably rapidly for small values of p, but failed for values greater than 0.6. As expected, the soft OR network showed some improvement on this, finding results after fewer patterns, and for values of p up to 0.6. This concurs roughly with Saund (1995), who reports a positive result using the same mixing function but a different objective function (6.4) for $p = 0.625$.

Dramatic improvements were obtained however with the saturated linear network, which was able to produce results extremely rapidly for values of p up to 0.7. It should also be noted that the saturated linear network required well under half the CPU time per pattern compared to the soft OR network. The network continued to produce solutions for values of p up to 0.9, albeit requiring nearly 40 000 patterns to achieve this. It is worth bearing in mind though that for $p = 0.9$, 68% of the input patterns are all ones, and therefore convey no useful information about the structure of the data, and a further 15% have only one zero. A random sample from this data set is shown in Figure 6.4.

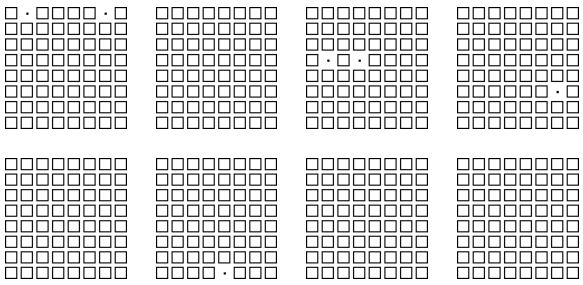


Figure 6.4: A random sample from the lines data set for $p = 0.9$.

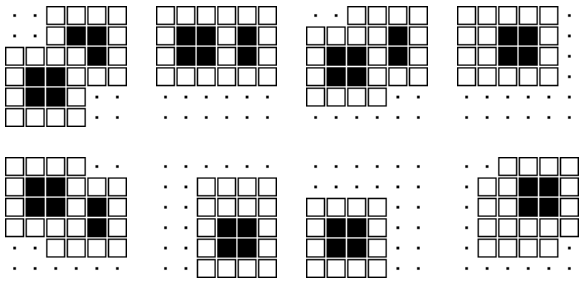


Figure 6.5: Some example input patterns from the occluded squares data set.

6.5.2 Occluded squares

In this experiment, the data set was composed of combinations of 4×4 squares on a 6×6 grid. Squares had a white border (denoted with input values of 1) and were filled black (denoted by -1). Each input pattern was composed of two such squares, each at one of the nine possible positions, picked at random. The squares were superimposed such that one partially occluded the other (or fully occluded if they happened to occupy the same position), with an input value of 0 used for points occupied by neither square (the background). Four example input patterns are shown in Figure 6.5.

A model[6] with nine outputs was used. The output values were constrained within the range $[0, 1]$, and the weights in the range $[-1, 1]$. The depths d_i were allowed to vary in the range $[1, 50]$. The soft maximum function (6.11) was used with $g(x) = x^3$. A linear penalty (5.6) with parameter $\lambda = 0.3$ was applied and Hebbian learning (4.15) with rate $\eta = 0.4$ was used to update the weights. A stable solution, such as the one shown in Figure 6.6, was typically reached after about 300 pattern presentations. In this configuration, the presence and relative depth of the nine possible squares was correctly determined for each pattern.

6.5.3 Illusory contour formation

Illusory contours are regions of an image where an edge is perceived even though there is no corresponding luminance gradient. Much has been written about this phenomenon (*e.g.* Lesher and Mingolla, 1995; Kumaran *et al.*, 1996) with analysis ranging from low-level neural

[6]The systems used to tackle the problems in this and the following section are referred to simply as 'models' since, as mentioned in Section 6.4, the methods used do not fit readily into a network model. For convenience though, much of the language associated with networks (weights, units, *etc.*) is retained.

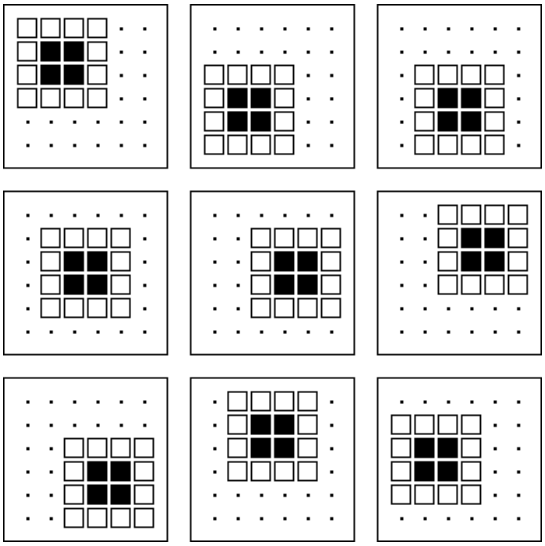


Figure 6.6: The final weight values from the occluded squares experiment.

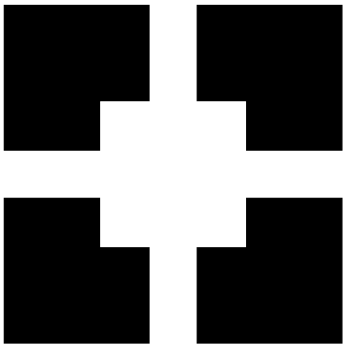


Figure 6.7: An example of a figure exhibiting illusory contours, after the manner of Kanizsa (1979).

mechanisms to high-level cognitive theories. Here we shall examine whether the occlusion mixture model put forward above is consistent with the ideas of illusory contour formation.

Most people, on staring at Figure 6.7 for a short while, will perceive a white square 'floating' above four black ones. Associated with this, and particularly when the figure is viewed monocularly, they will often perceive the central white square to be brighter than the surrounding white areas, with the boundaries of this bright area producing the so-called illusory contours. Something about our visual system causes us to 'see' five squares and assign relative depth to them, rather than just the four black hexagonal shapes that are on the page. In a simple experiment to reproduce this effect, a model similar to that used above (Section 6.5.2) was trained on combinations of 3×3 solid black or white squares on black or white backgrounds. The squares could appear in one of nine positions equally spaced over a 7×7 grid. Each training pattern consisted of a random number (1, 2 or 3) of superimposed squares, on a background that was either black or white with equal probability. The squares themselves were also black or white with equal probability, except that the one furthest from the viewpoint was never the same colour as the background. As previously, white was represented by 1 and black by -1. Examples of the training patterns are shown in Figure 6.8.

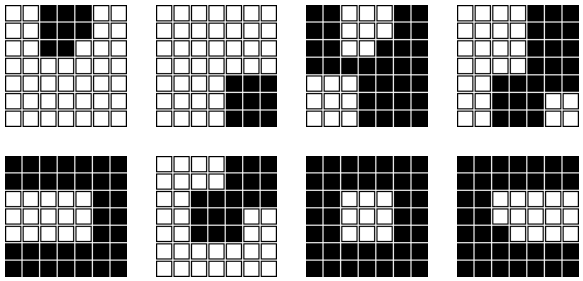


Figure 6.8: Some examples of the input patterns used to train the 'illusory contour' model.

Ten outputs were used, with one of the units 'hard-wired' with a weight vector of all ones to represent the backgrounds. In addition, it was found necessary to 'tell' the model about the background by fixing the value of that unit at either 1 or -1, according to the background colour, and fixing its depth at 1. The outputs a_i of the other units were allowed to move in the range $[-1, 1]$, and their depth values d_i in the range $[1, 50]$. During learning the weights were constrained to lie in the range $[0, 1]$, so that a unit's weights themselves only represented the presence or absence of an object, its colour being determined by the sign of the output value.

After an average of about 150 pattern presentations, with a learning rate η of 0.4 and a linear penalty with parameter $\lambda = 0.5$, the model converged to a state where each weight vector represented a square in one of the nine possible positions, as desired. In this configuration, the model was tested on the pattern of Figure 6.7. The resulting pattern of activation was as follows: the 'background' unit had its activation and depth at the fixed values of 1.0; the four units representing squares in the corners of the grid all had activations of -1.0 and depths in the range 4.4 to 5.1; the unit representing the central square had an activation of 1.0 and a depth of 13.8. Thus the representation produced by the model was of a white background, four black squares of roughly equal depth, and a central white square positioned above them.

There is no suggestion that either the visual environment or the model used here are akin to those of humans. What this experiment does demonstrate, however, is that in a simple environment consisting solely of opaque black and white squares, the modified REC model is able to learn that squares underly the patterns, and subsequently to interpret Figure 6.7 as containing four black squares with one white one partially occluding them. In the visual environment of humans, there are many non-square objects, but it is a reasonable assumption that square shapes have a significantly higher prior probability than squares with missing corners. It is quite possible that the 'illusion' produced by Figure 6.7 is similarly the brain's 'maximum *a posteriori*' interpretation of that figure. The illusory contours could then be explained by feedback from the high-level interpretation of shapes to the earlier levels of processing responsible for edge-detection, which 'fill in' details that make the overall representation more consistent.

6.6 Discussion

The assumption that separate causes combine linearly can often be found in models of real-world signals, such as speech and vision. At a low level of analysis (at the level of frequency components, for example), or for simple signals (where features are non-overlapping, for example), the linear assumption may well be valid. At a higher level, or for more complex signals, the assumption ceases to hold: objects that overlap in a visual scene do not sum linearly, they occlude one another; a speech model based on phonemes should take into account the nonlinear transition effects when they are joined.

This chapter has looked at ways to extend the linear model to account for more complex modes of interaction between causes. We saw in Section 6.2 how a general mixing function could be incorporated into the network model, and that while the activation became more complex, learning remained local. Interestingly, the extra complexity appeared primarily in the feedback direction — this may give a further clue as to why, in biological systems, we observe many more feedback than feedforward connections.

The OR-based models of Section 6.3.1 provide a simple example of nonlinear mixing. A possible practical application would be to the processing of hand-written characters. To a first approximation, the independent components of such characters are the different strokes of the pen. The write-white model deals correctly with the overlap of these strokes, and could give an important advantage over a linear model, particularly if the pen were relatively wide. An experiment exploring this idea is reported in Section 7.2.

The write-white-and-black model of Section 6.3.2 gives a useful extension to standard binary models by allowing the inclusion of 'don't care' states. It could be used, for example, in a visual environment composed of multiple non-overlapping black-and-white objects, or to detect independent processes that each produce a different part of a bit pattern.

Finally, and most importantly, if the ideas presented in this work are to be extended to an object-level analysis of visual scenes, it is essential that the notions of depth and occlusion be included in the models. Section 6.4 looked at one means of achieving this, but of course many other thorny problems, such as those of scale, translation and rotation invariance, remain.

Chapter 7

Applications and Practical Issues

The operation of the REC model has so far been demonstrated only on artificial data. In this chapter, several experiments are described where the network is applied to *real* data. The tasks are all in the fields of vision and speech, two areas where low-level feature detection is highly applicable.

The results are highly promising, and these techniques could provide a useful preprocessing step in future pattern-recognition systems. In particular we shall see that the network is able automatically to develop *wavelet codes*, a subject of considerable recent interest in the signal-processing community.

For these techniques to be used in real applications, it is important that the computational burden is not too high. At the end of this chapter we shall look at some methods that can be used to make large networks run more efficiently.

7.1 Natural image coding

In the past, unsupervised networks have been shown to extract the principal components of image patches (*e.g.* Sanger, 1989). For images with stationary statistics[1] the principal components correspond to a Fourier transform (Field, 1994) and as such do not directly convey any positional information — a single edge, for example, will typically be represented as the superposition of a number of the principal components. Therefore, although they are decorrelated, we should not expect to find principal components as the underlying *independent* features of natural images. These issues are discussed further by Field (1994) and Olshausen and Field (1996b).

Several authors have proposed an image code based on 2-D Gabor wavelets (Watson, 1983; Daugman, 1985; Field, 1987), which achieves a minimum in the joint uncertainty between spatial and frequency resolution (Daugman, 1985), and uses primitives that bear a close resemblance to receptive fields found in the mammalian visual cortex (Field and Tolhurst, 1986; Jones and Palmer, 1987). Most importantly from our point of view, a wavelet code of this type is highly sparse and has very low entropy (Daugman, 1989), and therefore gives a very efficient way to represent images. This section explores the ability of the techniques presented in this work to generate such codes.

[1] *i.e.* where the statistics at each point in the image are identical. This is a reasonable assumption for natural scenes.

Figure 7.1: One of the 512×512 grey-scale images of natural scenes used to generate training data for an image-coding network.

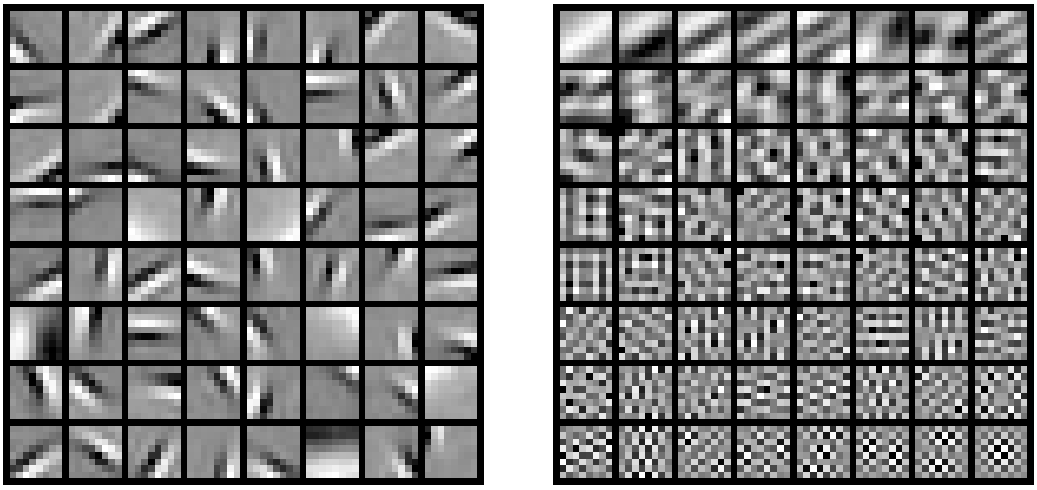
In these experiments, data were obtained by taking square patches from ten 512×512 grey-scale images of natural scenes from the American northwest.[2] The images were preprocessed using a filter with frequency profile $R(f) = fe^{-(f/f_0)^4}$ where $f_0 = 200$ cycles/image, as described in detail by Olshausen and Field (1997). The filter's effect is to 'whiten' the images by (approximately) flattening their power spectra. This prevents low-frequency components from dominating the REC network's squared error term, and hence the codes it generates. The filter also has a low-pass element to put a uniform upper bound on spatial frequencies at all orientations, and thus counteract the side-effects of sampling with a rectangular grid. An example of one of the images (before filtering) is shown in Figure 7.1.

An initial experiment used randomly sampled 8×8 patches from the images as the input to a REC network. The aim was to simplify the network's operation as far as possible, so that it could be scaled to larger problems. Rather than performing a full minimisation at the activation stage, the gradient-descent dynamics of the network (Section 4.2) were used directly with a limited number of iterations.

For each pattern, the network was given a single feedforward pass with the activation rate μ set to 1.0, equivalent to the activation of a simple linear network. Then, six iterations of the network were made according to (5.9) with $\mu = 0.3$ and $\lambda = 0.4$. The penalty function used was $\omega(a) = |a|$ (5.6), so in practice this could be implemented simply by moving each output a fixed value $\mu\lambda$ towards zero at each iteration. A final iteration was made without a penalty, and with $\mu = 0.05$.

Weight updates were calculated with an initial rate η of 0.01, and applied after every 100 patterns. After approximately 20 000 pattern presentations, η was reduced to 0.002. The weight values were stable after about 30 000 patterns, and are shown in Figure 7.2(a).

[2] Thanks are due to Bruno Olshausen and David Field for providing these images.



(a) (b)

Figure 7.2: (a) The weight vectors produced by training a REC network on 8×8 patches of natural images. Each of the 64 squares represents one of the weight vectors, and each of the 64 grey pixels within a square represents an individual weight value. (b) The principal components of the same data, ordered left-to-right and top-to-bottom. In both (a) and (b), zero is represented by mid-grey, with positive values shown with progressively lighter shades, and negative values darker. The contrast for each feature vector has been scaled so that the element with greatest magnitude is either white or black.

Almost all the features extracted by the REC network have a consistent 'wavelet-like' form. Four features represent bright or dark regions in each of the four corners of the 8×8 samples (two in row 4 of Figure 7.2(a), one in row 6, and one in row 7); the remainder may all be characterised as *edge-detectors* at various scales, spatial frequencies, orientations and positions. Each has a line about which it is approximately *anti-symmetric*, *i.e.* each light region is mirrored by a dark region of similar shape and size.

The features produced by the REC network are in stark contrast to the principal components of the same data, obtained by singular value decomposition, and shown in Figure 7.2(b). Because the whitening filter does not fully flatten the power spectrum, we see that the principal components are ordered in terms of increasing spatial frequency. Unlike the features produced by the network, none of the principal components are *localised* in the 8×8 grid, and since all components are constrained to be orthogonal, the orientations, where discernable, are all much the same as that of the first component.

7.1.1 Coding efficiency

To give a quantitative feel for the degree to which the two different codes reduce redundancy, entropy measurements were made for each component of the raw data, the principal-component code, and the network code. The data used for this test were 4096 non-overlapping 8×8 patches from the image shown in Figure 7.1.

In order to calculate the entropies of the outputs from the REC network with the weights shown in Figure 7.2(a), the weight matrix was first scaled to have a determinant of magnitude one. This means that both generative and recognition mappings are in effect *volume-*

Code	Sum of entropies (nats)	Reconstruction error
Raw data	199.0	–
PCA	163.8	0.0%
REC network	135.0	0.7%

Table 7.1: Entropy calculations for 8×8 patches of whitened image data, and for the same data represented by two (approximately) volume-conserving codes. Entropies are the sum of the individual entropies of each of the 64 code elements, calculated as described in the main text. The reconstruction error is the MSE for a reconstructed image as a percentage of the original image variance.

conserving (Section 2.1.4), and a comparison of differential entropies becomes meaningful. To generate the network-coded versions of the images, the reconstruction error alone (without penalties) was minimised.

In theory this could have been done simply by premultiplying each data sample by the inverse of the weight matrix. In practice the non-orthogonality of the weights means that the matrix is *poorly conditioned* (Press *et al.*, 1986, Chap. 2), and use of the inverse leads to many large output values, destroying the sparse properties of the code. Instead, therefore, code values were calculated by conjugate gradient minimisation, ending when changes in all outputs were less than 0.001. So although no explicit penalties were used, there is an *implicit* sparseness constraint in this technique. Because the optimiser is started at the origin and has limited accuracy, it will tend to favour results where the outputs are close to zero.

As a result of this approach, reconstruction errors were small but not zero. The MSE was approximately 0.7% of the image variance. In terms of loss of input information, this is a fairly small price to pay for an efficient code. In practice, when the reconstructed image was quantised to 256 grey-levels, it was found that the error resulted only in differences of one grey-level and that in less than 0.1% of the image pixels.

The PCA codes are easier to calculate. The principal components form a well-conditioned matrix whose determinant is automatically one, the codes may be obtained by premultiplying the data by the inverse (transpose) of the matrix, and the resulting reconstruction errors are zero to within machine precision. However, we expect the PCA codes to have significantly greater redundancy.

Entropies were calculated by discretising the distributions with a fixed bin size of 0.05, resulting typically in about 80 non-empty bins. The results are listed in Table 7.1. These show that the redundancy in the raw data is much reduced by rotating it to the principal component axes, as we would expect, since we know that the coefficients so obtained are uncorrelated. However, the redundancy is reduced by approximately the same amount again using the features obtained from the REC network.

We may note that the maximum possible sum of entropies here, obtained by assuming a normal distribution of variance equal to the image variance, and calculating entropy as above, is 204.1 nats. It is much more difficult however to say what the true total entropy of the input is, and hence obtain the *lower* bound for the sum of the output entropies. Calculating the entropy of a 64-dimensional distribution would require far more data than we have here to obtain any sort of meaningful figure.

For the particular test image used here, the most efficient information-preserving code (assuming, as is reasonable, that all 4096 patches are distinct) would require only $\log_2 4096 = 12$ bits or $\log_e 4096 = 8.3$ nats to represent any sample of the data, but this is not a useful comparison, since the code's ability to generalise to other images would be extremely poor!

The ability of the REC network (or any other system) to generate efficient image codes from these data is severely limited in this example by its being restricted to consider only 8×8 patches in isolation. Much of the redundancy in these and other images occurs at distances of far greater than 8 pixels, but the network has no means of discovering and removing these. Evidence of the great efficiency (and inherent sparseness) of codes based on wavelets with much larger extents is given by Daugman (1988).

7.1.2 A larger image-coding network

The results of Figure 7.2(a) have also been extended to larger image samples. A REC network with 256 inputs and 192 outputs was trained on 16×16 patches of the same data described above. Beside its increased size, the network was otherwise identical to that used previously. Stable weight values were obtained after approximately 45 000 pattern presentations with a learning rate η of 0.01 and a further 15 000 with a rate of 0.002.

These are shown in Figure 7.3(a), and are used in the following section to perform a closer examination of the nature of the wavelet codes being produced.

7.1.3 Analysis of the wavelets

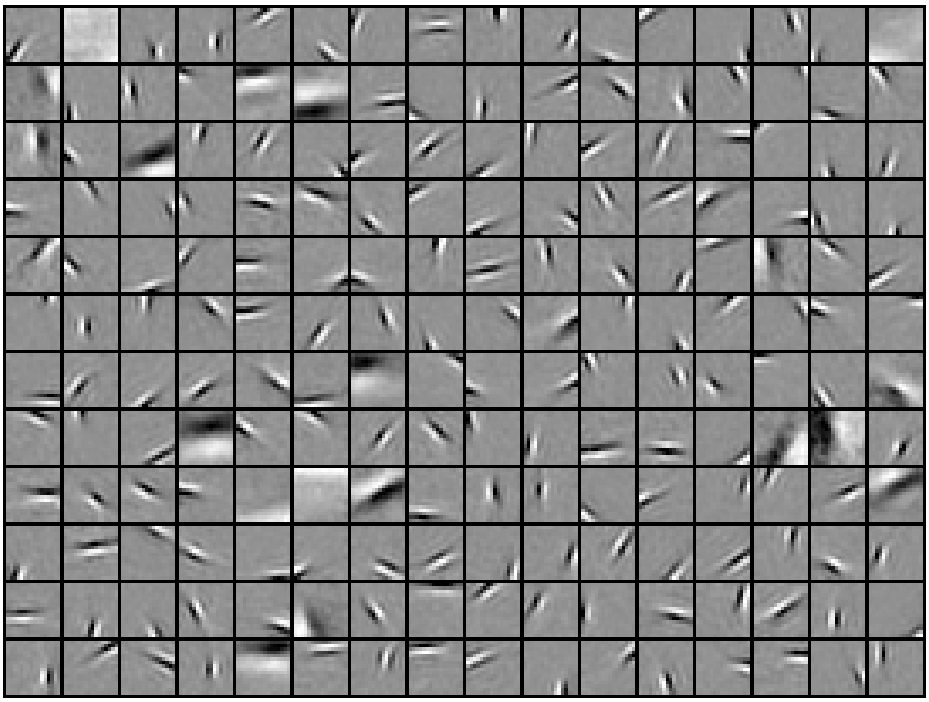
The fact that a REC network with a very simple sparseness constraint is able automatically to generate wavelet codes is certainly of theoretical interest, both because they are known to be efficient, and because of the links to biology. An immediate *practical* application of such a network however is in analysing the wavelets produced by data from a particular visual environment, and using this information to create efficient wavelet codes tailored specifically to the statistics of that environment. This section describes how such an analysis could be carried out, and looks at the results obtained through this method. The technique is similar to that used by Jones and Palmer (1987) for characterising the receptive fields of cats.

The weight vectors shown in Figure 7.3(a) were fitted, using a least squares criterion, to *2-D Gabor wavelets* (Daugman, 1985). The wavelets were modelled using the following formula:

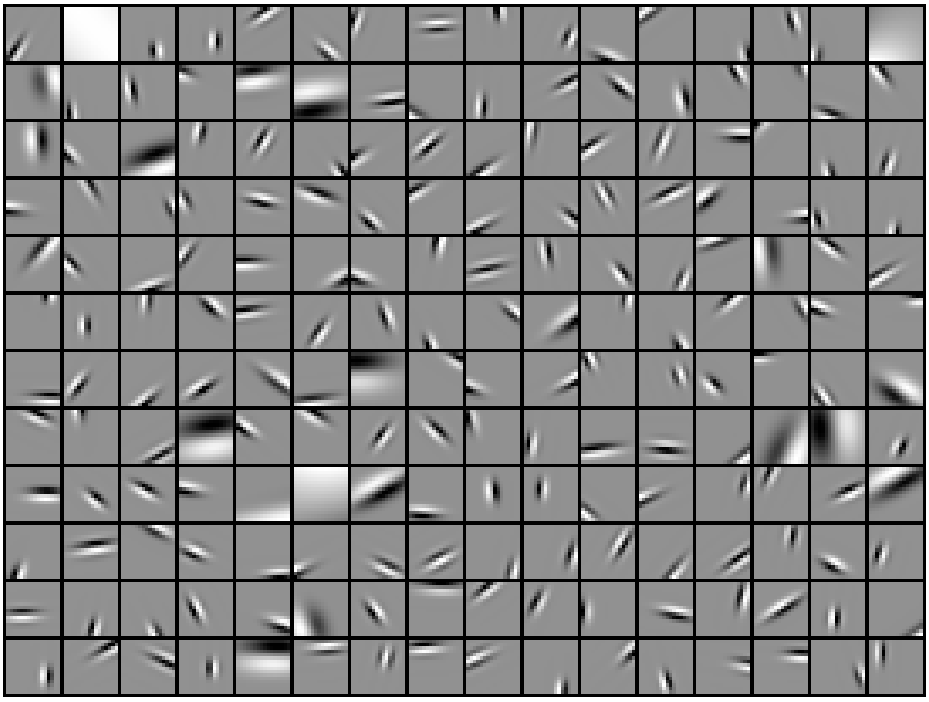
$$G(x, y) = A \exp[-\pi(\check{x}^2\alpha^2 + \check{y}^2\beta^2)].\cos[-2\pi(\check{x}u_0 + \check{y}v_0) - \phi]$$

where the coordinates $(\check{x}, \check{y})$ are the result of rotating (x, y) by an angle $-\theta$ and then translating by $(-x_0, -y_0)$. The cosine term is a plane wave, and the exponential term an elliptical Gaussian envelope. There are a number of parameters: A gives the amplitude, x_0 and y_0 denote the centre of the wavelet, θ its orientation (relative to the x-axis), α and β relate to the spatial extents perpendicular and parallel to the orientation (width and length) respectively, $(u_0^2 + v_0^2)^{1/2}$ gives the spatial frequency, $\arctan(v_0/u_0)$ the frequency orientation (relative to θ), and ϕ the phase.

The fit was performed by minimising the SSE between the weight values and a Gabor wavelet (sampled on a 16×16 grid of notional side-length one) with respect to the nine

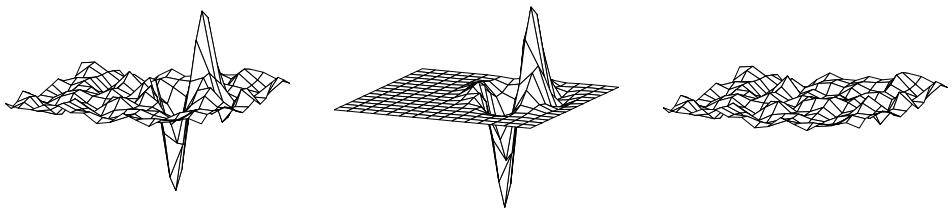


(a)



(b)

Figure 7.3: (a) The weight vectors produced by training a 192-output REC network on 16×16 patches of natural images, depicted as in Figure 7.2. (b) A set of 2-D Gabor wavelets fitted to the values shown in (a). The fitting method is described in the main text.



(a) (b) (c)

Figure 7.4: (a) One of the weight vectors from Figure 7.3(a) (row 6, column 8) depicted using a mesh plot. (b) A Gabor wavelet fitted to these weight values by the method described in the main text. (c) The difference between (a) and (b).

parameters $A, \alpha, \beta, u_0, v_0, \phi, x_0, y_0$ and θ. The minimisation was performed using a simplex search method (*e.g.* Press *et al.*, 1986, p. 289). For this to be successful, it was important to have reasonable initial estimates for the parameter values. These were made by finding, for each network output, the coordinates of the maximum and minimum weight values and using their orientation, position and separation to estimate values for $\theta, (x_0, y_0)$ and u_0 respectively. The remaining parameters were initialised with fixed estimates ($A = 0.3$, $a = b = 0.6$, $\phi = \pi/2$, $v_0 = 0$).

This method of initialisation enabled the minimiser to find a reasonable match in all 192 cases. The wavelets obtained by matching in this way are shown in Figure 7.3(b). The mismatches between the weights and the fitted wavelets had a variance of $6.2\%(\pm 3.8\%)$ of the image variance. Figure 7.4 shows (a) one of the weight profiles, (b) the fitted Gabor wavelet and (c) the difference between the two.

While differences between the weights and the fits could generally be accounted for by random 'noise' resulting from incomplete convergence of the weight values, systematic deviations were apparent in some instances, and curvature in normal probability plots indicated that the residues were not always normally distributed. This might indicate that Gabor wavelets do not properly describe the features in all cases, or could simply be the result of local minima in the fitting process, or the lack of robustness to outliers inherent in the squared error metric.

In any case, the parameter values given by the fitting process allowed several useful statistics of the wavelet-like features to be deduced. Several of these are graphed in Figures 7.5 and 7.6. Figure 7.5(a) shows the ratio of length to the width (*i.e.* the aspect ratio) of the Gaussian envelopes in the spatial domain (given by α/β). These values are in close correspondence, both in mean and range, to physiological results from the cat striate cortex reported by Jones and Palmer (1987). In (b) the relative area was calculated using the area of the ellipse defined by the half-height contour of the Gaussian envelope. There are a number of outliers represented in the last bin of the histogram. Figure 7.5(c) shows that there are a greater number of wavelets with high spatial frequencies than low, as one would expect in order to achieve a 'tiling' of frequency space (Field, 1994).

Figure 7.5(d) shows the number of cycles of the plane wave lying within the width of the Gaussian envelope, with the boundary taken to be at 5% of its peak value. There is a clear upper bound in this measure at approximately 1.75. The histogram (e) shows the degree of

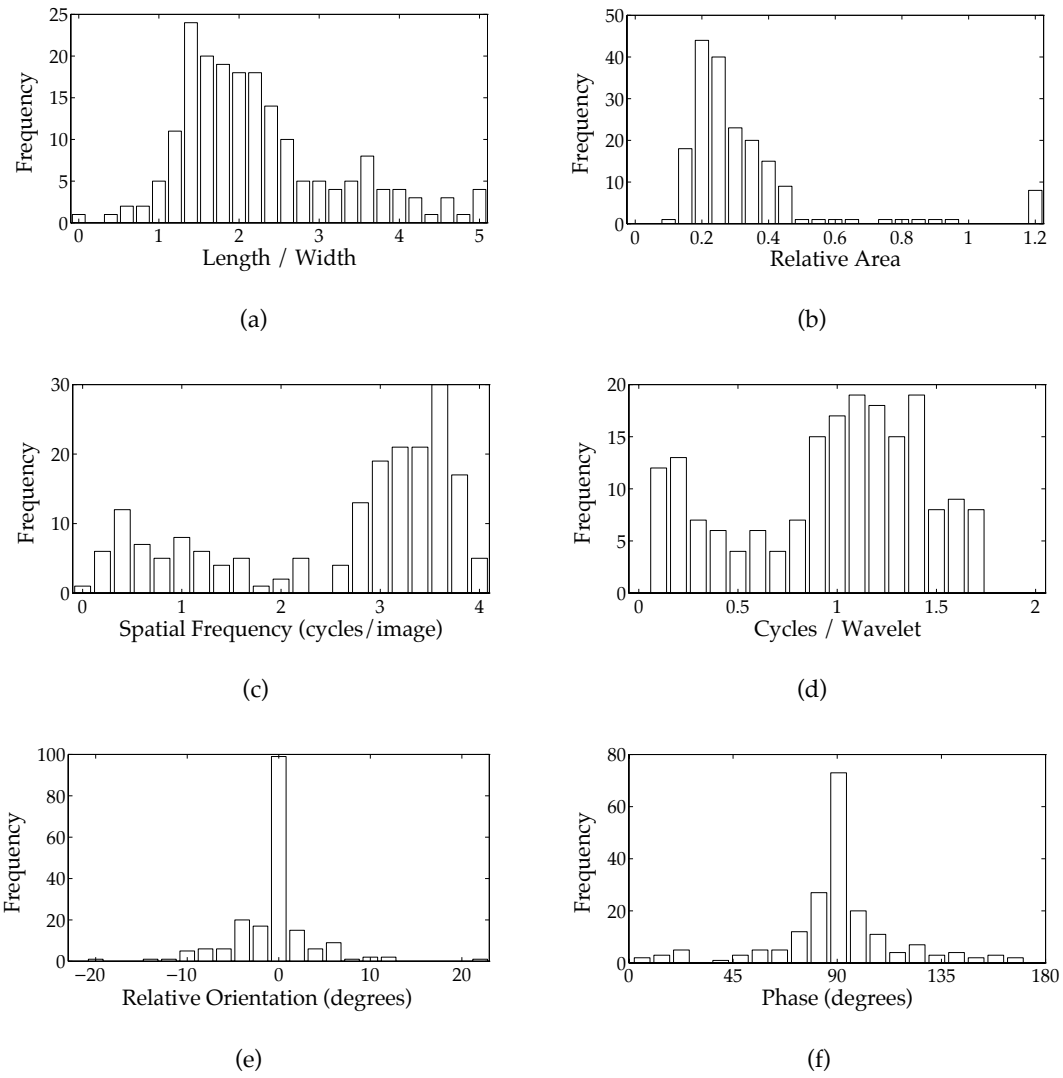


Figure 7.5: Histograms for various measures of the Gabor wavelets shown in Figure 7.3(b). Further details of the calculation of these quantities are given in the main text. (a) The ratio of length to width for the Gaussian envelopes. (b) The ratio between the effective area occupied by the wavelets and the full area of the 16×16 image. (c) The phase of the plane wave (at the centre of the Gaussian envelope). (d) The spatial frequency of the plane wave, measured as the number of cycles over the width of the 16×16 image. (e) The difference between the orientations of the plane wave and the envelope. (f) The number of cycles of the wave within the width of the envelope.

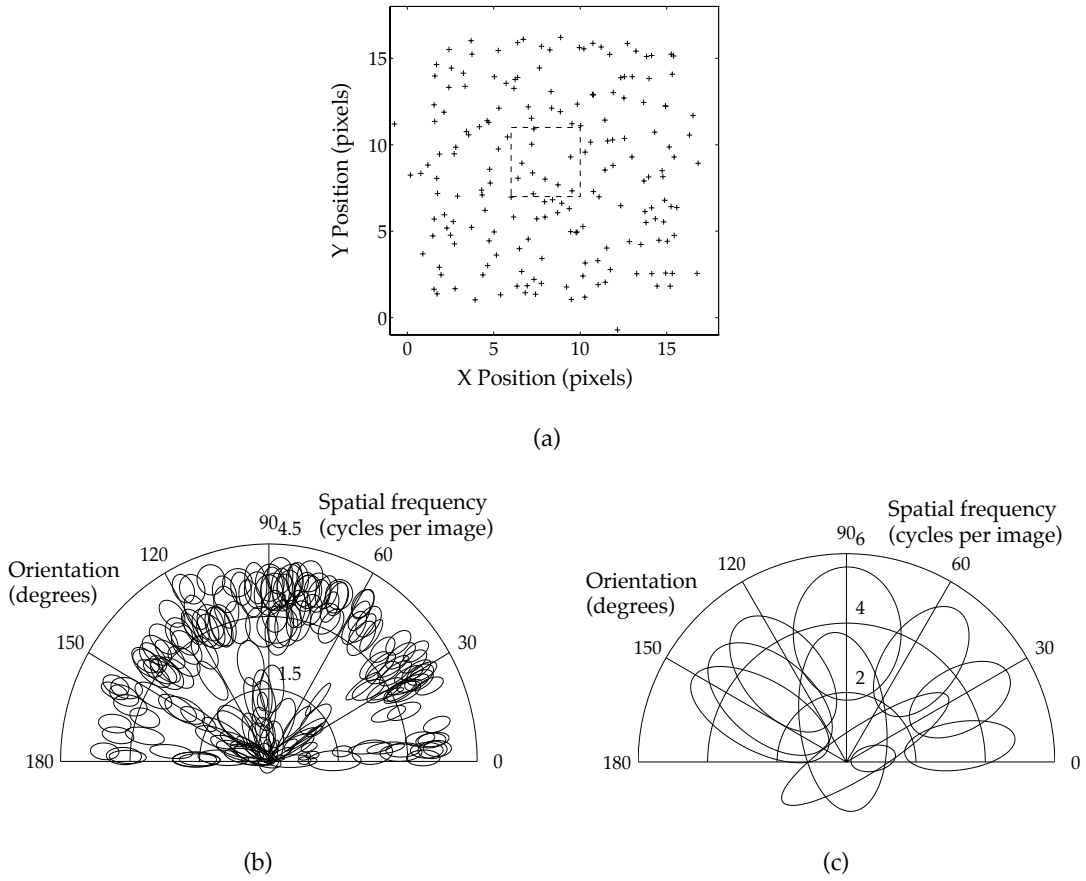


Figure 7.6: Some indicators of the coverage given by the Gabor wavelets of Figure 7.3(b) in the space and frequency domains. (a) A scatter-plot of the centres of the wavelets on the 16×16 grid of pixels. The centres do not in all cases lie within the grid, and four points are outside the axes used here. (b) The 'footprints' of all the wavelets in frequency space. The ellipses are isoamplitude contours of the spectral response profiles. These give an idea of the *relative* bandwidths, but their sizes have been scaled by a constant factor to allow the individual elements to be picked out in the diagram. (c) Footprints from a set of ten wavelets whose centres occur in a limited area of the image patches, shown dashed in (a). The ellipses here show the *half-heights* of the spectral response profiles, indicating the degree to which frequency-space is covered by the wavelets.

mismatch between the orientations of the spatial envelope and the plane wave. In the great majority of cases they are closely aligned. Finally, Figure 7.5(f) shows the phase of the cosine wave at the centre of the wavelets. There is a very marked peak at 90°, indicating that the majority of wavelets have odd symmetry. This is an interesting result, as physiological data typically show a much higher proportion of profiles with approximately *even* symmetry. We can characterise this by saying that the network appears here to favour edge-detectors over line-detectors.

Figure 7.6(a) shows that the wavelets are fairly evenly spread in the spatial domain, and (b) indicates both that the frequency orientations are similarly well spread, and that the magnitudes appear to come in two bands centred very approximately at 0.8 and 3.3 cycles per image, as also indicated by Figure 7.5(c). The degree to which frequency space is covered by the wavelet representation is indicated by Figure 7.6(c), which shows the frequency response profiles for ten wavelets centred in the middle of the 16×16 grid. Almost the entire space up to 4.2 cycles per image is contained within these profiles. Other examples of such 'footprint' diagrams have been produced by Jones and Palmer (1987). The means of calculating the response profiles from the wavelet parameters is discussed in detail by Daugman (1985).

There are many further issues relating to the interpretation of the wavelet codes, and their relation to physiological data. Olshausen and Field (1997), for example, address the issue that the 'features' taken directly from weight values here are related to, but not directly equivalent to, classical receptive field profiles. Such issues are however outside the scope of this work. The purpose of this section is rather to show how a wavelet analysis may be done, and to give the results of such an analysis on the weights produced by a REC network.

Gabor wavelets form a very large family, and until now the choice of a set of wavelets for image coding or analysis has primarily been directed by a mixture of theoretical considerations, physiological data, and guesswork. There is, however, no single set that will yield optimal results in all environments. The optimal aspect ratios in particular are likely to depend on the frequency with which straight edges occur in the images. The automatic generation of wavelet codes by the REC network, combined with an analysis of the results, provides a new technique that has the potential to determine wavelet parameters for highly efficient codes that are tailored to particular visual environments.

7.1.4 A comparison with Olshausen and Field's sparse coding

Similar codes to those presented here have been reported by Olshausen and Field (1996a,b, 1997). The methods they used are also very similar, but neither the techniques nor the results are exactly the same, so some of the differences are highlighted here:

- Olshausen and Field use (up to) ten iterations of conjugate gradient minimisation for activating their model. The simulations described here use a smaller number of iterations of simple gradient descent, corresponding directly to a network model. Eight iterations were used in total, but very similar results have been obtained using as few as four iterations of the recurrent model. A comparison of this method with a full, and more complex, minimisation showed that this approximation, besides reducing simulation times dramatically, actually *improves* the quality of the results.

- Olshausen and Field use adaptive penalty terms, based on moving averages and variances of the output values. This approach was found to be unnecessary to produce the results given here. Indeed, if not applied carefully, there was a tendency for learning with this technique to become unstable, with variances for a few units becoming very high, and the rest close to zero.

- The wavelets that result from the two methods are similar in most respects, but Olshausen and Field's features show a much higher proportion of 'sandwich'-like features, *i.e.* where there are one and a half or more cycles of the oscillatory component within the wavelet's envelope. As we saw in Figure 7.5(d), the number of such cycles for the results in Figure 7.3(a) cuts off quite sharply at about 1.75. This allows the wavelets to be interpreted more directly as edge- and line-detectors, and bears a closer resemblance to physiological results.

In addition, a number of new extensions to the results have been made, described in the following two sections.

7.1.5 Coding with non-negative features

We saw in Section 5.2 that where a system is detecting 'real-world' features, there is a case for making the outputs non-negative, because negative coefficients have no natural interpretation. By permitting negative network outputs, we appear to have ignored this argument in the image-coding experiments.

It may be argued, however, that this approach is just a short-cut made possible by the fact that the features that are produced have a symmetry such that, if negated, they would still represent 'valid' wavelets. In other words, we could produce the same codes by doubling up the features with their negated versions and applying the non-negative constraint, but doing so would significantly increase simulation times with no real gain.

It has been demonstrated that this argument holds in practice by repeating the initial coding experiment with 8×8 patches, but using a network with non-negative outputs. The results were very similar in form to those of Figure 7.2(a), and in particular the features were found to maintain their symmetry and wavelet-like shapes.

7.1.6 Coding without pre-whitening

The use of a whitening filter to preprocess the image data, as mentioned earlier, prevents the reconstruction error from becoming swamped with residuals from low frequency components in the images, and thus from failing to represent high-frequency features. From a biological perspective, there is good evidence that this kind of filtering is performed in the retina (Atick and Redlich, 1992).

From a more general point of view, however, we appear to be cheating slightly by applying a filter that puts the data into a form more amenable to the REC network. A truly unsupervised system cannot always expect this luxury.

In an experiment to try to tackle this issue, a network was trained on raw grey-scale images, without any form of pre-filtering. A single simple modification was made to the

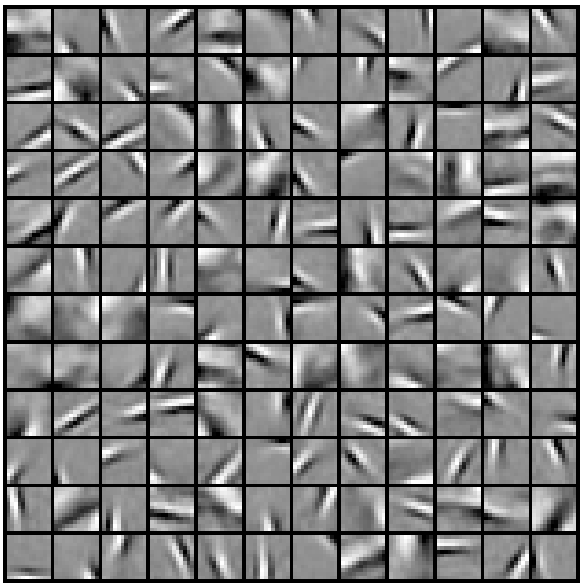


Figure 7.7: The weight values from a 144-output REC network trained on 12×12 patches of unwhitened images.

image-coding network described previously, in the form of an extra term in the Hebbian learning rule:

$$\Delta \mathbf{w}_i = \eta a_i |a_i| \mathbf{r}. \tag{7.1}$$

This is just the standard rule (4.15) with a factor of $|a_i|$ added. Broadly speaking, the justification for its inclusion is as follows. Lower frequencies in the raw images have higher variance. During learning, therefore, the residuals, and hence the weight changes to any active unit, will tend to be dominated by low-frequency components too. If we could arrange, however, for units that were tending towards representing low-variance high-frequency components to have a lower learning rate, then they would have more of a chance to average out the low-frequency 'noise' in their weight changes. A very simple means of adjusting learning rate based on variance (about zero) is simply to include an extra term relating to the magnitude of the unit's response.

A 144-output network was trained on unfiltered 12×12 image patches using the modified learning rule (7.1). The initial learning rate was set higher than previously (to 0.2), but the network's parameters and method of activation were otherwise the same as before. After approximately 40 000 patterns, the last 10 000 with a reduced learning rate, the results shown in Figure 7.7 were obtained. A good proportion of the features (over 80%) are wavelet-like. The remainder are low frequency components that are less easily characterised. Using the standard learning rule, almost all of the features tended to be 'uninteresting' ones of this type.

To the author's knowledge, this is the first demonstration of the automatic development of a large number of localised wavelet-like features from unwhitened image data. While we have a practical demonstration of its effectiveness, there remains scope for developing a better theoretical understanding of the effect that the modified learning rule has on the learning process.

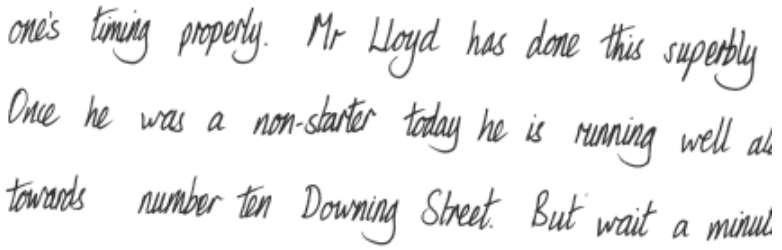


Figure 7.8: A sample of the data used in the script-coding experiment.

7.2 Coding of handwritten script

In this experiment, the REC network was applied to images very different to the natural scenes used previously. The data were based on twenty-five pages of handwritten script written by a single author, as described by Senior (1994).[3] A sample of the data is shown in Figure 7.8. Each page was subsampled to a 512×720 8-bit image, and the pixel values then negated and scaled to lie in the range $[0, 1]$, with zero indicating white and one black. Randomly sampled 12×12 patches of the images were used as the input to the network. A network with 108 outputs was activated using the recurrent model, as in Section 7.1, but using 12 rather than 8 iterations of gradient descent. The penalties and other parameters were the same as in the previous examples. Network outputs were constrained to be non-negative, and weight values to lie in the range $[0, 1]$.

The network used an OR-type mixture model using 'saturated' first-layer units as introduced in Section 6.3.1. This corresponds to a write-black imaging model in this case, and reflects the fact that separate pen strokes mix in an OR-like way. The means of calculating the first-layer activations was a version of Equation (6.6), modified slightly to reflect the fact that the inputs are not fully binary:

$$r_j = \begin{cases} x_j - \max(\sum_k a_k w_{kj}, \theta) & \text{if } x_j > \theta, \\ x_j - \sum_k a_k w_{kj} & \text{otherwise,} \end{cases}$$

where the threshold θ, above which an input is considered to be 'on,' was set to 0.8.

The stable weight values are shown in Figure 7.9. Despite the random initial values, we see that the weights have all (with the slight exception of those at row 6, column 8) converged to values where they represent features that are connected and localised within the 12×12 grid. The predominant orientation of the features is just off the vertical, corresponding to the slant of the script.

The experiment was repeated with the normal (*i.e.* purely linear) imaging model, and gave similar results. However the features produced by this model, with an average length of approximately 3.0 pixels, were not as extended as those in Figure 7.9, whose average length is about 4.8 pixels. This provides evidence that the use of an OR-type mixture model, where superimposed features do not incur a reconstruction error, is beneficial in this case for extracting extended features.

Nevertheless, we might expect to find dependencies in handwritten characters extending over a greater spatial extent that those produced here, corresponding, for example, to

[3]Thanks are due to Andrew Senior for making these data available.

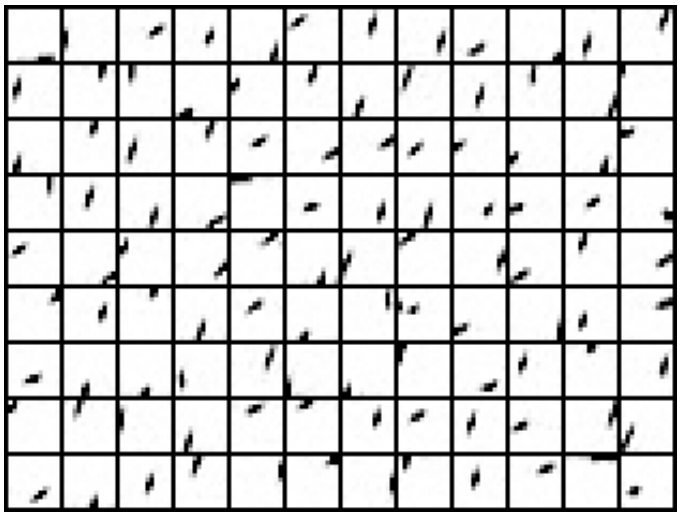


Figure 7.9: The weight vectors from a 108-output REC network trained on 12×12 patches from images of handwritten script. The weights are displayed with shades of grey, as previously, but scaled so that zero is white, and one is black.

full strokes and loops of the pen. The network's ability to discover such features is limited here by the need to reconstruct accurately with only a constrained number of outputs. Furthermore, no effort was made to segment or align individual letters within the patches, and thus to limit the degree of variation within the data.

A greater number of output units could help to give a larger and sparser representation, but in the configuration described here it was found that this approach led largely to the introduction of 'junk' features. The increased use of penalties might overcome this, but an alternative method would be to feed the network outputs into a second level of processing, and thus produce a higher-level representation based on combinations of the features shown in Figure 7.9.

7.3 Speech coding

The coding of speech is another area where an unsupervised feature-detection system is highly applicable. Speech signals have great complexity, and the high-level causes responsible for them (such as words, or the identity of a particular speaker) are sufficiently removed from the raw signals that it is difficult to apply a supervised system at a low level without losing important information or making false assumptions.

Particular difficulty is posed by the wide range of time-scales over which speech features can occur. Vowel sounds can be characterised by fairly constant waveforms for up to 400 ms, while *stop consonants* (such as 't' and 'k') can result in complex transients on the scale of just a few milliseconds (O'Shaughnessy, 1987). Speech is typically preprocessed into a series of *frames* over time, but the size of the frames is a trade-off between the need to capture short-time features and the desire to reduce the redundancy inherent in features that are constant for long periods.

This section describes some experiments where the REC model has been used to extract features from speech signals, with data obtained by preprocessing the raw waveforms in

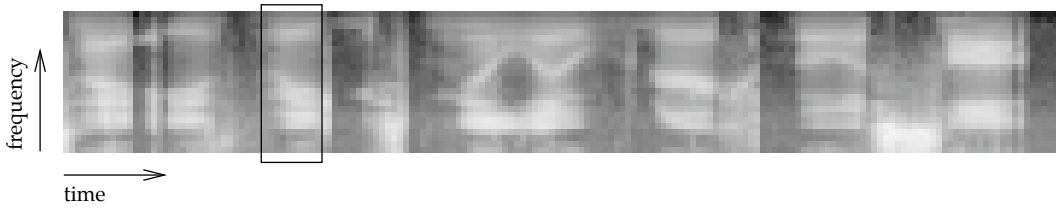


Figure 7.10: A sample of 170 frames of speech data, corresponding to 1.7 seconds of speech. Each row of the image shows the values over time for one of the 24 coefficients derived from the filterbank outputs, and each column represents one frame of data. Lighter grey-levels indicate larger values. The rectangle, with a width of 10 frames, indicates the size of the samples of this data used as single input patterns to the REC network. The image consists largely of white bands of high energy, known as the *formants* of voiced speech. These are often horizontal, corresponding to steady-state vowel sounds, but also in places move up or down, where the formants are undergoing *transitions*. We can also see abrupt changes in the image, corresponding to various *stops*. Less visible at this level of contrast, since they tend to have lower energy than the formants, are various bursts of high frequency, produced by *fricatives*.

two different ways.

7.3.1 Filterbank representation

It is almost universal amongst speech processing systems to adopt a frequency-based representation at an early stage of processing. Indeed, it appears that the basilar membrane performs such a transformation in the peripheral auditory system of humans (see, for example, Buser and Imbert, 1992). One means of acquiring such a representation is by using a *filterbank*, a series of (partially overlapping) bandpass filters whose centres are spread over a range of frequencies.

The filters' centre-frequencies are not usually spaced evenly, because certain frequencies require more precise discrimination than others for the understanding of speech. A commonly used means of warping the space is the *mel-scale* defined as:

$$Mel(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right).$$

This scale broadly mimics the response properties of the basilar membrane.

In this experiment a sample of 200 sentences from the Wall Street Journal (WSJ) continuous speech database (Paul and Baker, 1992) were used as the raw data. These were preprocessed into a notional filterbank representation, modelled by performing a short-time Fourier transform and convolving with 24 overlapping triangular filters covering the range from 0 Hz to 8 kHz (the Nyquist limit) equally spaced on the mel-scale. The Fourier transform used a Hamming window of width 25 ms, and the frame rate was 10 ms.

The logarithms of the filter outputs were taken, and the resulting coefficients shifted and scaled to have approximately zero mean and unit variance. A sample of the data is shown in Figure 7.10. The 24 coefficients for each frame were used in 'segments' of 10 consecutive frames as the input to a REC network, giving a total of 240 inputs. The total time period covered by each input pattern was therefore (a little over) 100 ms. The network had 200

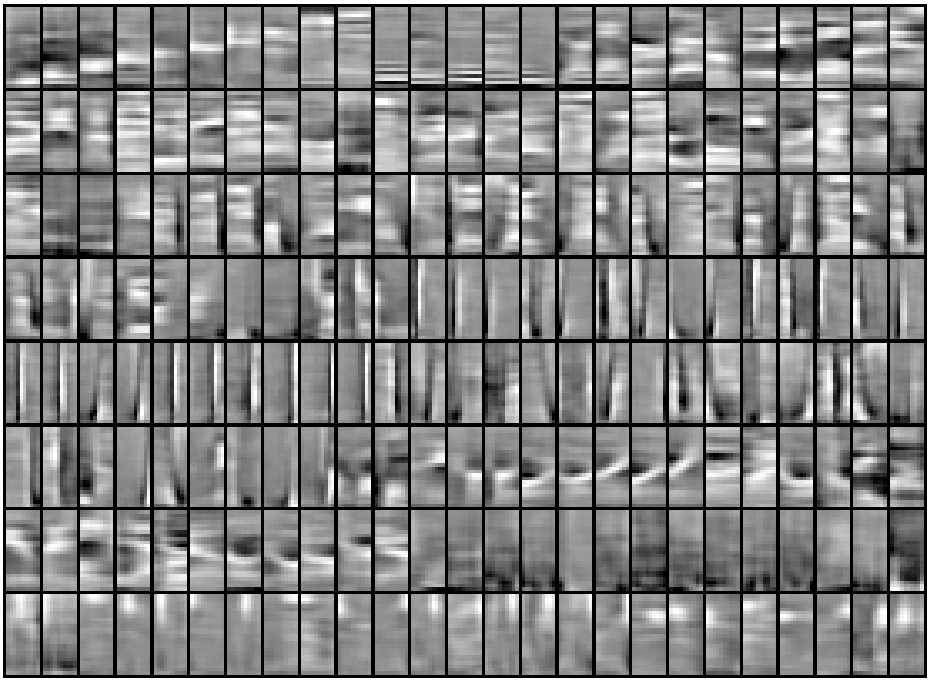


Figure 7.11: The weight values from a 200-output REC network trained on speech data represented by filterbank outputs. Each rectangle corresponds to one of the weight vectors. Values are represented by grey-levels, as previously (see Figure 7.2).

output units. Activation was much the same as for the image-coding networks, using the same sparseness penalty, and 12 iterations of gradient descent. Because there was no reason to expect the features to show the symmetry of the wavelets found in the image-coding case (see Section 7.1.5), the network's outputs were constrained to be non-negative.

The weight values obtained after approximately 80 000 training patterns are shown in Figure 7.11. The weight vectors has been reordered so that they appear in groups that broadly correspond to similar types of features. The network has identified many of the 'standard' features of speech. Level formants (horizontal white bands) are visible singly at a range of frequencies (row 1 of Figure 7.11). We also find a number of low frequency voicing patterns (row 1), and combinations of formants (rows 1–2). In some cases the formants start or end within the 10-frame segments (rows 3–4). There are a number of 'vertical' features at a range of positions in time (rows 4–6). The majority of these correspond to bursts of noise over a wide range of frequencies at the onset of speech (a black line followed by a white line). Several vectors representing rising and/or falling formants are clearly visible (rows 6–7). There are also a number of 'silence' features (row 7), and finally (row 8) a number of vectors corresponding to short high-frequency bursts, such as those generated by fricatives.

We note, therefore, that the network has isolated a number of components that are 'interesting' characteristics of a frequency-based representation of speech. A fuller discussion of such features is beyond the scope of this work, but further details may be found, for example, in O'Shaughnessy (1987).

In order to test the practical merit of a representation based on these features, a simple phone-recognition task was devised. It was based on data from the DARPA TIMIT speech database (Zue *et al.*, 1990). The data were initially preprocessed with a filterbank as pre-

viously. Two supervised single-layer neural networks (with sigmoid output units) were trained on 6400 different samples of phones (basic speech sounds) from the database. Each training pattern was generated by using five frames either side of the 'middle' of a particular phone, as designated by the segmentation data in the database. The number of distinct phones in the database was reduced from 61 to 40 using a mapping described by Robinson and Fallside (1992).

One of the networks (with 240 inputs) was trained on 10-frame segments of the (filterbank) data, and the other (with 200 inputs) on the outputs of the REC network, now without a penalty term, when presented with the same data. The networks were tested on a set of 350 patterns separate from the training data. After a number of runs, the network using the raw data was found to average a recognition rate of 58.8%, while that using the REC network's representation achieved an average success rate of 70.4%.

What this suggests is that the representation created by the REC network, despite having a lower number of dimensions, gives a significant advantage to subsequent processing by supervised systems. In particular, since the supervised networks used were single-layer, we may conclude that the preprocessed representation has a higher degree of *linear separability* than the original data. This experiment is slightly artificial, however, and further work will be needed to assess whether such representations can offer a real advantage to state-of-the-art phone recognition systems.

7.3.2 Mel-frequency cepstral coefficients

It is common to perform another preprocessing step for speech data beyond that of generating the filterbank representation outlined above. This aims to deconvolve the excitation of the vocal tract from its response characteristics at any particular time. It can be achieved with a further Fourier transform from the log-spectral domain, or more speedily approximated with a discrete cosine transform, and results in *mel-frequency cepstral coefficients*, or MFCCs (see, for example, Gales, 1995).

Such representations often form the basic input to HMM-based speech-recognition systems. Inherent in these systems are the assumptions that one frame of data is independent from the next (Rabiner, 1989), and that individual coefficients within each frame may be well-modelled by Gaussian distributions, or mixtures of a small number of such distributions. Neither of these assumptions is correct. There are dependencies over many frames of speech at standard rates, and while MFCCs are approximately decorrelated, they may be multimodal, and contain complex higher-order dependencies, as illustrated by a plot of two such coefficients shown in Figure 7.12(a).

In a second speech-coding experiment, a REC network was trained on MFCCs of speech data from the Resource Management (RM) database (Price *et al.*, 1988). The inputs were created from three consecutive frames of 12 MFCCs, giving a total of 36 inputs. The same number of outputs were used in order to allow a complete representation to be formed. The method of promoting sparseness by under-activation (Section 5.8) was used in this instance to constrain the network. While this method is less theoretically justified than the penalty-based approach, the computational simplicity of activating the network with a pseudoinverse (Section 4.5) reduced learning times (on an HP C160 workstation) from five or more

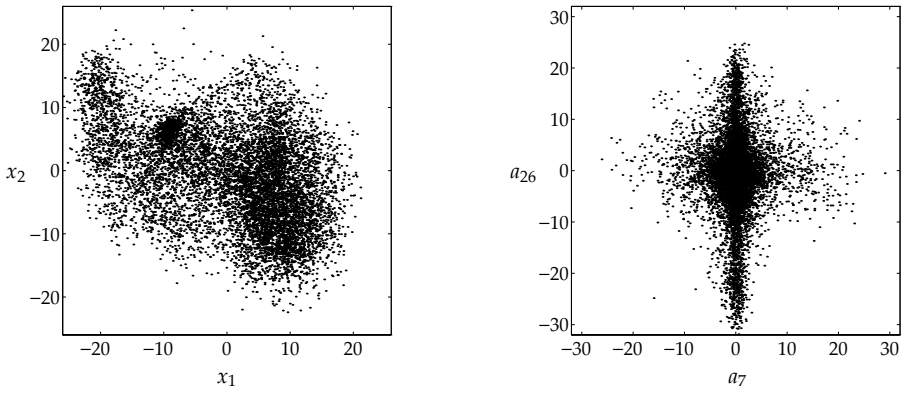


Figure 7.12: (a) A scatter plot for the first two mel-frequency cepstral coefficients from 30 000 frames of speech data, demonstrating the multimodality of the distributions and strong statistical dependency. (b) Two of the outputs of the REC network described in the main text when presented with the same data. The cross-shaped nature of the plot is indicative of low statistical dependence between the coefficients, and the strong vertical bar shows that the a_7 value is highly peaked at zero.

hours for the previous examples in this chapter to just a few minutes.

Output values were unconstrained, weight updates applied after 100 patterns, and weight vectors normalised to unit length. The results after approximately 200 000 pattern presentations are shown in Figure 7.13. The experiment was repeated several times with different initial weights, including one run with the weight matrix initially set to the identity matrix, in order to check for the presence of local minima, but the form of the results was very similar in each case.

There are some interesting similarities between the features of Figure 7.13 and the 'dynamic' features (Furui, 1986) commonly used as a preprocessing step for HMM systems. The first two rows of Figure 7.13 represent components that are *static* over the three frames. Most of the next two represent features where the coefficients of the centre frame are close to zero, while those on either side have equal magnitudes but opposite signs. This is just the difference approximation to a first derivative. There are also a number of features (second half of row 4, and row 5), where the central frame coefficients are large, and the side coefficients both have smaller magnitudes and opposite sign. This approximates a second derivative. These features are similar in form therefore to *delta* and *delta-delta* (or *acceleration*) coefficients, which are a popular means of capturing the between-frame dynamics of cepstral coefficients. The differences here are firstly that each feature is based on *combinations* of the cepstral coefficients within a frame, rather than each in isolation, and secondly that they have been produced directly from the statistics of the data, rather than through theoretical considerations.

Calculations of mutual information were made both between inputs to the network and between outputs. The mutual information between each pair of values was estimated by normalising each coefficient to have unit variance, approximating both joint and marginal densities by 'binning' with bin-widths of 0.3, and calculating $I(x_i; x_j)$ according to Equa-

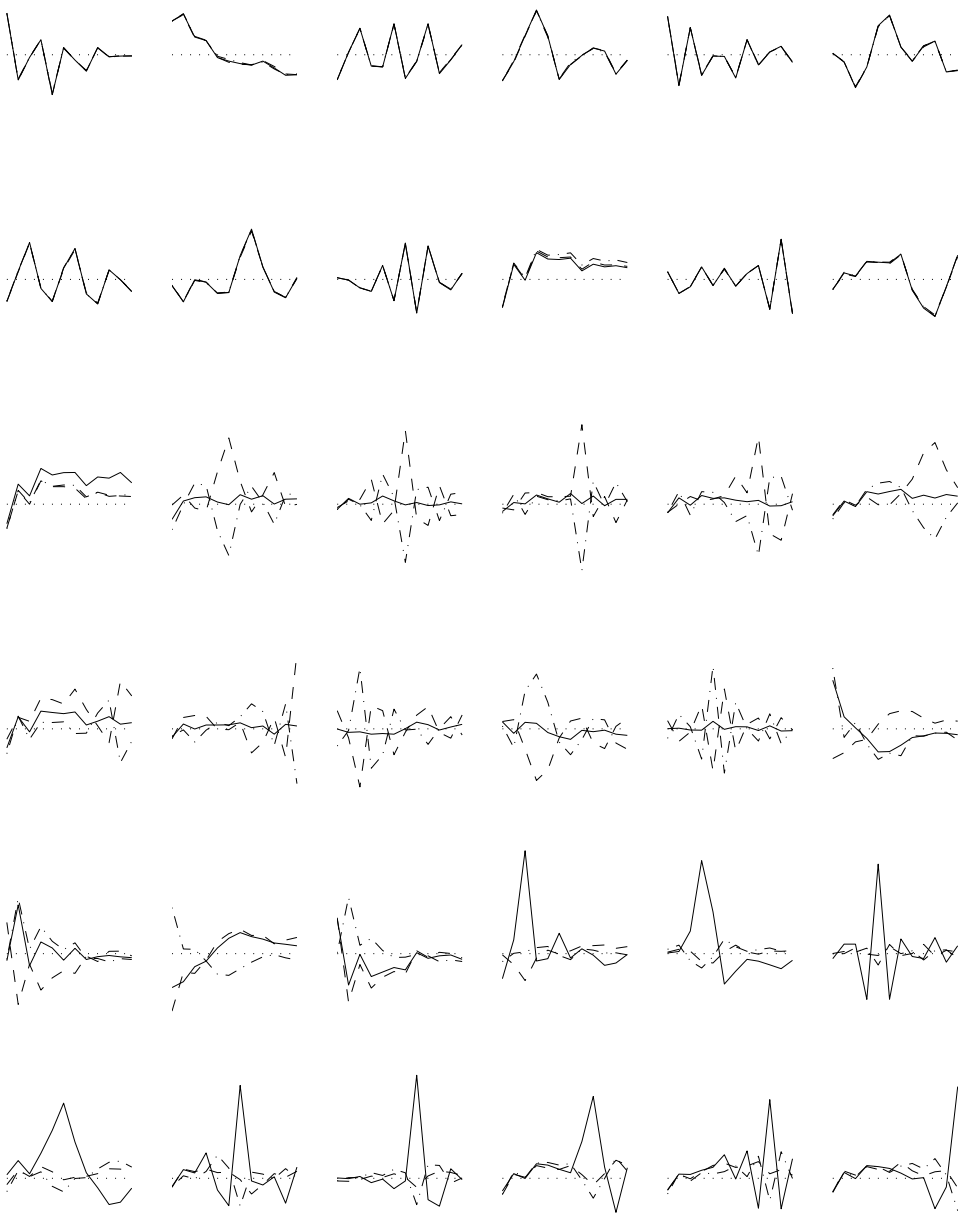


Figure 7.13: The weight values from a REC network trained on speech data in patterns consisting of three consecutive frames of twelve MFCCs. Each of the 36 weight vectors is plotted using three lines. Each line has twelve points, corresponding to the twelve coefficients in a frame. The different lines correspond to the three different frames. The first of the three is plotted as a dashed line, the second solid, and the third with alternate dashes and dots. The zero level is shown as a dotted line. All the lines are plotted to the same scale, but the vertical axis has been omitted for clarity since the scaling is arbitrary.

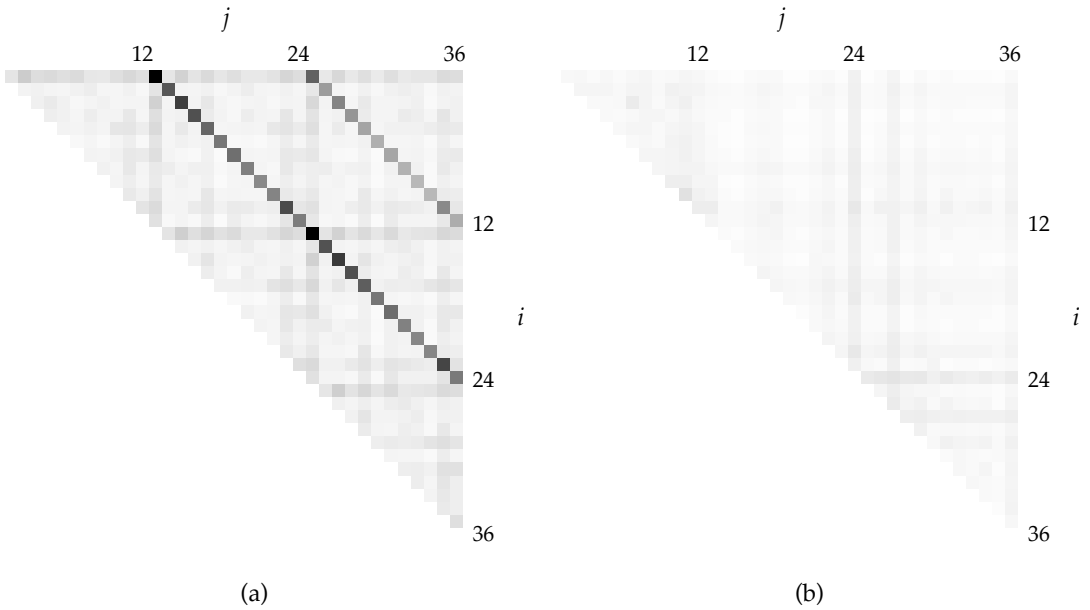


Figure 7.14: Grey-scale representations of the upper triangles of the pairwise mutual information matrices for (a) the inputs (x_i, x_j) to a REC network consisting of three frames of MFCCs from speech data and (b) the corresponding outputs (a_i, a_j) produced with the weight values shown in Figure 7.13. The grey-levels were set on a linear scale with white representing zero and black the maximum value from both matrices (2.3 nats). Each pixel, therefore, gives an indication of the mutual information between two of the code coefficients, corresponding to its row and column. Particularly prominent, unsurprisingly, are the strong dependencies between one MFCC and the same coefficient in adjacent frames. (b) shows that the transformation made by the REC network has greatly reduced mutual information between coefficients.

tion (2.4). 10 000 data samples were used in the estimate. This crude approximation was needed because of the computational intensity of calculating such measures. The results are depicted in Figure 7.14, and show that the mapping produced by the network creates a substantial decrease in mutual information between coefficients, as we would hope. It seems likely that feature-detection of this sort has the potential to provide a valuable preprocessing step in preparing data for HMM (and other) speech-processing systems.

7.4 Reducing computational complexity

In this section we look at some of the practical issues associated with REC networks, and in particular at ways of reducing the times taken to activate and train networks so that they may feasibly used on larger problems, or in systems with strong time constraints.

7.4.1 Activation techniques

The computational cost of the REC network is in large part determined by the method used to activate it, namely some form of minimisation of the objective function with respect to the output values. A good general-purpose minimiser, such as one using a conjugate gradient or variable metric method (Press *et al.*, 1986, Chap. 10) requires $\mathcal{O}(n)$ line minimisations for

n unknowns, with each line minimisation requiring an approximately constant number of evaluations of the objective function and its gradient vector. In our case, the time taken to compute both the basic objective function (4.5) and its gradient (4.6) is $\mathcal{O}(mn)$ (for m inputs and n outputs), giving an overall cost for the minimisation of $\mathcal{O}(mn^2)$.

It may be possible in some cases, however, to use less general but correspondingly less complex techniques. Recall that if all weight vectors are mutually orthogonal, the REC network may be correctly activated in a single forward pass, simply requiring each of n output units to multiply and add m pairs of input and weight values, and giving a complexity of $\mathcal{O}(mn)$. A backward pass of the recurrent model has the same cost, so the overall complexity remains at $\mathcal{O}(mn)$ if the output values can be obtained by a fixed number of iterations. Such a scheme is possible if the degree of non-orthogonality in the final weight vectors is small. Wavelet codes are an example where this is the case — while not all the weights of Figure 7.3(a) for example are orthogonal, many are because they are localised in different regions of the 16×16 grid. The fact that these, and many of the other results in this chapter, were produced with a fixed number of iterations of gradient descent demonstrates empirically that using the network model directly can give good results with real-world data.[4]

In order to represent the computational complexity solely in terms of the number of input dimensions, we may further note that for an m-dimensional input, the network will require $\mathcal{O}(m)$ outputs to capture all input information, giving a complexity of $\mathcal{O}(m^3)$ for a general minimisation technique, and $\mathcal{O}(m^2)$ for the network model.

We have already seen (Section 4.5) that where constraints on activation are not required (as may be the case *after* the learning process), the pseudoinverse of the weight matrix may be used as a cheap means of network activation ($\mathcal{O}(m^2)$ when $n = m$). A practical problem occurs, however, as noted in Section 7.1.1, when the weight matrix is ill-conditioned. It may nevertheless be applicable when the dimensionality of the problem is not too large, and may also be used *during* learning if we approximate sparseness constraints using the under-activation technique of Section 5.8. The effectiveness of this method in a practical example was demonstrated in Section 7.3.2.

7.4.2 Pruning

In most of the experiments we have seen, many of the final weight values are very close to zero. This sparseness in the weight vectors may be exploited by ignoring weights that are close to zero when activating the network. This technique is commonly used in neural network models (*e.g.* Bishop, 1995, Chap. 9) and is known as *pruning*. Here we are talking about pruning connections only, but the possibility of pruning units is discussed in Section 8.3.

All that is needed is for weight values whose magnitudes are below a certain small threshold to be removed from output calculations. If, for example, all features have only c non-zero elements (where $c \ll m$), the cost of evaluating errors and/or error gradients would be reduced to $\mathcal{O}(cn)$. In practical terms, the amount of pruning possible would need to be sufficient to compensate for the computational overhead of implementing sparse weight ma-

[4]This is not to say that near-orthogonality cannot be exploited when using techniques other than simple gradient descent. The results produced by Olshausen and Field (1996a,b, 1997), for example, demonstrate the effectiveness of using a conjugate gradient method with a limited number of iterations.

trices.

Weight pruning can also increase the speed of learning, since fewer weights means fewer weight updates to be calculated and applied. Indeed it may also speed the convergence of learning by removing the 'noise' of small weight values oscillating around zero. The one major limitation of this approach, however, is that the pruning must not be applied at too early a stage in learning, *i.e.* before we can know for sure that particular connections will not be significant in the features represented by each of the output units.

7.4.3 Faster learning

We noted in Section 4.5 that gradient descent tends to be a slow minimisation method, and therefore looked at other techniques for finding optimal outputs for the REC network. In determining *weight* updates, however, we have been using gradient descent throughout in the form of the Hebbian learning rule.

When using MLP networks, a variety of faster methods for minimising error with respect to the weights are commonly applied. These include conjugate gradient and other second-order methods, as well as various heuristic techniques (see, for example, Jervis and Fitzgerald, 1993).

Such methods have not been used in this work, because the emphasis has been on developing and understanding a simple network model. There is no reason in principle, however, why they should not be applied to the REC network if training times are important, or if particular tasks are beset by problems of local minima.

7.5 Discussion

This chapter has attempted to demonstrate that the ideas put forward previously may be applied to real data with interesting results. The REC network is most applicable in complex environments, where patterns are typically composed of many independent and non-orthogonal features, and at a low level of processing, where those features may be assumed to mix at least approximately linearly. Visual and auditory environments fulfil these criteria, and hence the experiments have concentrated on these. The results are promising, and will, it is hoped, give grounds for developing the work in this area.

We set out a theoretical framework for an information-processing system in Chapter 2, and argued that information theory can be used to define the 'true' goals of an unsupervised system, even if a model does not use the theory directly. In this chapter, we have returned to some of these ideas, and seen that the REC model does indeed reduce the entropy of its outputs while simultaneously preserving input information, and does indeed reduce the mutual information between them.

In the next, and final, chapter, we shall review the principal results of this thesis and their relation to the work of other researchers, and look at ways in which they might be extended in the future.

Chapter 8

Conclusions and Future Work

This thesis has set out a framework for the automatic generation of codes based on the twin goals of maximising information transfer and reducing mutual information between the code elements. It has described the development of a recurrent network model (the REC network) that aims to achieve the first goal by minimising reconstruction error and the second by the application of various constraints, in particular through the use of low-entropy priors.

The cost of using recurrent activation is relatively high compared to the more common feedforward methods, but can be justified for a number of reasons: it results in a very simple learning rule; it removes the constraint of orthogonality found in several other unsupervised models; it means that no changes to existing weights are required to accommodate new patterns and that all weight values relate directly to the features they encode; and finally it allows the network to operate without an explicit recognition model, relying instead on the generative model, which is likely to be simpler and more robust.

The simplicity and locality of the basic network are attractive features that remain even when it is adapted to include penalties and nonlinear mixture models. The network also appears to be robust in the sense that simplifications such as limiting the number of iterations used in activation and 'under-activating' do not dramatically degrade, and in some cases even improve, the quality of the results.

The network has been shown in a variety of cases to generate 'low entropy codes,' this term being favoured over the more commonly cited 'minimum entropy,' 'independent' or 'factorial' codes to reflect the fact that it is virtually impossible to generate complete, fully independent, codes in real perceptual environments. The aim instead has been to reduce code redundancy as far as possible within a simple framework.

The approach of using penalties has assumed that it is possible to predict in advance the approximate probability distributions of the code coefficients in the optimal solutions. The recent work of Bell and Sejnowski (1995, 1996, 1997) on blind deconvolution uses exactly the same assumption. There are a variety of other ICA techniques that do not consider distributions directly, but use various nonlinearities to capture higher order statistics (Jutten and Herault, 1991; Karhunen *et al.*, 1995; Oja, 1995). The problem with these approaches is that it is not easy to determine what assumptions are implicit in the nonlinearities that are used.

Other approaches, most notably that of Comon (1994), approximate the dependencies

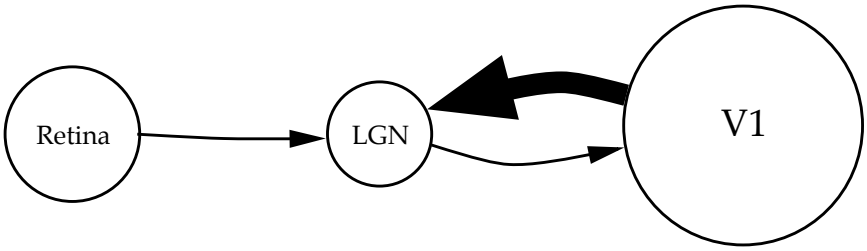


Figure 8.1: A hugely simplified depiction of the early visual system in mammals, representing only one hemisphere, and the input from only one eye. LGN is the lateral geniculate nucleus, and V1 the primary visual cortex, with arrows representing connecting fibres. The sizes of circles and arrows give a very rough indication of the relative numbers of cells and connections, but are not drawn to any kind of scale. See, for example, Kandel *et al.* (1991) for a more detailed description.

between coefficients directly using cumulants and polyspectra. To make the technique tractable, Comon proposes an algorithm that considers only a pair of coefficients at a time, but this requires the assumption that the independent components *are* linearly separable. This is a reasonable assumption for blind deconvolution problems, but not in general for feature detection. In particular, the problems we considered in Chapter 7 are not amenable to algorithms of this type, since none of the features are fully independent of each other, and so extracting them in a stepwise manner is unlikely to minimise the overall redundancy.

A common feature of all the alternative techniques mentioned here is that they are unable to produce representations of higher dimensionality than their input. The REC network, by contrast, *is* able to develop overcomplete representations, a property that could prove useful as a preprocessing step for recognition systems.

8.1 The biological perspective

The REC network was developed primarily from practical considerations rather than a desire to model neurophysiology. Nevertheless, the overall structure was inspired by the pattern of connections in the brain, and there are some interesting links to be made, both as a basis for future modifications to the REC model, and as a means of shedding some light on the brain's operation.

Feedback is undoubtedly an extremely important part of the brain's structure, and one that is overlooked by the majority of neural network models. In the visual pathway of mammals, for example, the signals from the retina are fed to an area of the thalamus known as the lateral geniculate nucleus (LGN), from where they are in turn fed to the primary visual cortex, usually termed V1. This arrangement is depicted in Figure 8.1. There are however somewhere between 10 and 100 times as many connections back from V1 to the LGN as there are in the forward direction, suggesting that feedback plays a key role in this stage of processing.[1]

This fits nicely with the structure of the REC model: not only does the network incor-

[1]It is inconceivable that, in an organ as costly (in metabolic terms) as the brain, evolution would produce this disparity in numbers of connections if the function of feedback were not of great importance.

porate feedback, but it is also far more important for the generative model (in the reverse direction) to be accurate than the recognition model. Indeed, using recognition weights that are equal to the generative weights is simply a very crude approximation to a recognition model that happens to work. Pece (1992) describes in more detail the modelling of the visual pathway with a feedback model of this type. The discussion in Section 4.1 gave further references to related work.

A second area of interest is that V1 has a much larger number of cells than the primary source of its input, the LGN. This raises the possibility that the cortex may be forming an *overcomplete* representation. We have seen in this work that the REC network is capable of generating such representations, and indeed that they may be required in order to produce minimum entropy codes. This issue is also discussed by Olshausen and Field (1997).

A third issue is the REC model's use of iterative processing. Is it possible that the brain uses a similar method? In discussing this issue, Mumford (1992) cites Gray and Singer (1989) as evidence for the existence of cortical oscillations with a period of 20–30 ms (corresponding to frequencies of 35–50 Hz). It is possible that these could be the result of the synchronising mechanism that would be needed for iterative processing. Based on measurements of neural response times, this would give time for five to six iterations in which to perform recognition, but other considerations such as the need for additional processing and damping could easily reduce this to two or three.

Whatever the exact figures, it seems that the slow hardware of the brain does not have time for many iterations of a recurrent activation process. This observation need not conflict with the REC model, since experiments have shown that in some cases the results can be *improved* by reducing the number of iterations. This number could be reduced further by using a recognition model that is able to get close to the optimal solution in a single forward pass. In the brain, it is possible that the feedback connections are used for 'offline training' of the forward connections during sleep, an idea exploited by the *Helmholtz machine* (Dayan *et al.*, 1995; Hinton *et al.*, 1995), and discussed further in Section 8.3.

8.2 Limitations

Inherent in the REC model are several key assumptions that limit the scope of this work. Some of these are discussed here.

We began by assuming a linear mixture model. In many environments this is not a good approximation to the way in which independent components mix. Redundancy-reducing linear models may be a useful first step in a processing system, but there is still likely to be a need for nonlinear models at later stages.

This issue was partially addressed in Chapter 6, but still limited us to *fixed* mixture models tailored to specific problems. An ideal system would adapt its mixture model as well as the parameters within it. However, having an unsupervised system explore the entire space of nonlinear functions in an unconstrained way is unlikely to produce useful results. A more profitable approach could be to have multiple levels, each with the aim of reducing redundancy, and each with greater nonlinearity than the previous one. This work has only tackled the low levels of such a system.

The technique of penalising outputs that was introduced in Chapter 5 assumes that we have some idea of the true distributions of the data's underlying causes. An ideal system would not require such prior assumptions, and would determine the distributions automatically, but once again the removal of all constraints is likely to produce an intractable problem. There may, however, be scope for making the penalties adaptive to a limited degree, an issue discussed further in the next section.

We have assumed throughout this work that it is always desirable to *reduce* redundancy. This is not necessarily true: in a noisy system, there may be a need to *introduce* redundancy to limit the impact of noise-induced errors. However, it is a theorem of information theory that the reduction of redundancy to achieve optimal compression and the introduction of redundancy to reduce errors on a noisy channel may be treated as two separate problems (known as *source coding* and *channel coding* respectively). In other words, although we have only addressed the first of these in this work, a subsequent consideration of ways to introduce redundancy need not conflict with the techniques presented here.

A further assumption in this thesis is that patterns presented to the REC networks are *static*. We have also assumed that each pattern is generated independently from the last, an approach that was inherent from the initial definition of a memoryless source (Section 2.1.1). In real environments, it is common for patterns to be both dynamic and strongly correlated over time. Except for the simple approach of building a single input pattern from several consecutive time-slices (used in the speech-coding experiments of Section 7.3), temporal patterns have not been explored in this work. The advantages that could come from doing so are discussed in the following section.

8.3 Directions for future work

There are many ways in which this work could be extended, some of which have already been mentioned in earlier chapters. Some further suggestions are given here:

Temporal processing. There is scope for building a temporal aspect into the REC network, to allow it to discover dependencies not just between inputs at any one instant, but also over time. This could be achieved with some form of memory mechanism in the first and/or second layer of units. This would not only allow the network to generate more efficient codes in dynamic environments, but would also provide a natural and flexible way to learn *invariances*. In a visual environment, for example, the same object can appear at a variety of positions, scales and viewing angles, making it difficult for recognition systems to identify it as the same in all cases. However, an object that is present at one instant in usually also present at the next, albeit possibly transformed in some way. Finding temporal dependencies for moving objects therefore allows a system to automatically *learn* the nature of these transformations. This issue has been discussed by Földiák (1991) and Edelman and Weinshall (1991), and demonstrated in recent work by Wallis and Rolls (1996) — it would be interesting to apply these ideas to the REC model.

Adaptive penalties. The penalty functions introduced in Chapter 5 were fixed in advance of learning. Making these adaptive to a limited degree could prove useful in some

cases, such as where there is a wide variation in features' variances. The distributions underlying the penalties could, for example, be adapted to reflect the mean (or perhaps mode) and variance of the individual output values. Indeed such an approach has been used by Olshausen and Field (1996a,b), but as mentioned in Section 7.1.4, there can be problems with stability when using this technique.

Multiple priors. In a similar vein, there are situations where it could be useful to assume different prior distributions for different output values, rather than penalising each in the same way. It is possible, for example, for a set of data to be clustered, but for there to be additional structure within each cluster: in such a case it could be useful to place 'binary' priors on the outputs representing cluster centres, reflecting the idea that a point is either in or out of a particular cluster, and unimodal priors on the remainder. Initial experiments using this idea reveal problems with local minima in both activation and learning, but it might be possible to overcome these by *prioritising* outputs according to their role, or using a multi-layer system where the residuals from one stage are fed to the next for further 'explanation.'

Growing and pruning. The experiments in this work have all used fixed size networks. There could be advantages, particularly for very large problems, in allowing the networks to expand or contract according to need as learning progresses. The addition of extra units to a network could be incorporated as an extension of the 'vigilance' mechanism discussed in Section 4.8: if reconstruction error exceeds some threshold but there are no 'unused' units, then a new output unit could be added to the network and its weights adjusted to represent the residual error. Similarly, output units could be 'pruned' if particular output values are found to have very low variance, indicating that the units are not being used. The ease with which outputs can be added to or removed from the REC network comes from the fact that the representation of each feature is entirely localised within the connections to the corresponding unit.

Lateral inhibition. All competition in the REC network is achieved via feedback connections. In competitive networks it is more common for competition to come from *lateral inhibition*, *i.e.* direct inhibitory connections between outputs. Olshausen and Field (1997) show that it is quite possible to view the mathematics of the REC model in terms of lateral connections, but we have avoided doing so in this work because of the additional advantages of feedback (such as the explicit representation of the residual and the ability to consider alternative mixture models). Nevertheless, there could be advantages to building a model that has *both* types of connection. Suitable lateral connections, for example, could provide a fast means of finding a 'winner' from two strongly competing output units, thus reducing the number of iterations required to fully activate the network.

Topographic mapping. In the brain, cells responding to similar features tend to be located near to each other. Network models such as the self-organising map (Kohonen, 1982, 1990) are able to mimic this behaviour. In the REC model there is no such ordering, since there is no concept of 'distance' from one output to another. However, by distributing the outputs over, for example, a plane and making connections according to

the distance between them, it should prove possible to introduce these ideas to the REC network. Short-range excitatory connections at an early stage of learning, for example, could be used to encourage neighbouring units to respond to similar patterns.

A separate recognition model. The REC network is based on the model of Figure 4.1(b) (page 42), where there is no explicit recognition model, and recognition is instead achieved through a number of iterations of the generative model. We have argued that this approach gives several useful properties. Nevertheless, there is scope, particularly in highly nonlinear systems, for having a recognition model that is separate from the generative model. This is the approach used in the *Helmholtz machine* (Dayan *et al.*, 1995), where the two sets of weights can be trained in distinct 'wake' and 'sleep' phases (Hinton *et al.*, 1995; Neal and Dayan, 1996). Without abandoning the recurrent model, it could be useful to incorporate some of these ideas into the REC network. Separate recognition weights, for example, could allow activation to be achieved with fewer iterations, while the error-driven approach would still allow good representations to be found without requiring an accurate recognition model.

Stochasticity. We noted in Section 5.4.1 that, by using only the MAP estimate of the output values in learning, the REC network fails to *guarantee* a maximum likelihood solution. If instead it carried a *distribution* of output values from the activation stage to the learning stage, it would achieve a full implementation of the EM algorithm. One means of performing this without abandoning the network model is to make the outputs *stochastic*, that is for them to take on values according to the full posterior distribution for a particular input pattern, rather than just the value with maximum probability. By this *Monte Carlo* method, the output values would, over a number of patterns, come to *represent* the output distribution. This is the approach used, in the binary case, by the Helmholtz machine and its predecessor the *Boltzmann machine* (Ackley *et al.*, 1985). In the REC network such an approach could, at the expense of greater complexity in activation, yield better results from the learning process. It is not clear that it would give much benefit where the output priors are unimodal, but for more complex multimodal cases the improvement is likely to be significant.

Bibliography

Ackley, D. H., G. J. Hinton, and T. J. Sejnowski (1985). A learning algorithm for Boltzmann machines. *Cognitive Science 9*(1), 147–169.

Ash, R. B. (1965). *Information Theory*. New York: Interscience.

Atick, J. J. (1992). Could information theory provide an ecological theory of sensory processing? *Network: Computation in Neural Systems 3*(2), 213–251.

Atick, J. J. and A. N. Redlich (1992). What does the retina know about natural scenes? *Neural Computation 4*(2), 196–210.

Attneave, F. (1954). Some informational aspects of visual perception. *Psychological Review 61*, 183–193.

Baddeley, R. J. (1996). Visual perception: An efficient code in V1. *Nature 381*(6583), 560–561.

Baldi, P. (1989). Linear learning: Landscapes and algorithms. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1*, pp. 65–72. San Mateo, CA: Morgan Kaufmann.

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

Baldi, P. F. and K. Hornik (1995). Learning in linear neural networks: A survey. *IEEE Transactions on Neural Networks 6*(4), 837–858.

Ballard, D. H., G. E. Hinton, and T. J. Sejnowski (1983). Parallel visual computation. *Nature 306*(5938), 21–26.

Barlow, H. B. (1959). Sensory mechanisms, the reduction of redundancy, and intelligence. In *The Mechanisation of Thought Processes*, pp. 535–539. London: Her Majesty's Stationery Office.

Barlow, H. B. (1961). Possible principles underlying the transformations of sensory messages. In W. A. Rosenblith (Ed.), *Sensory Communication*. New York: MIT Press.

Barlow, H. B. (1983). Understanding natural vision. In O. J. Braddick and A. C. Sleigh (Eds.), *Physical and Biological Processing of Images*, pp. 2–14. Berlin: Springer-Verlag.

Barlow, H. B. (1989). Unsupervised learning. *Neural Computation 1*(3), 295–311.

Barlow, H. B. (1994). What is the computational goal of the neocortex? In C. Koch and J. L. Davis (Eds.), *Large-Scale Neuronal Theories of the Brain*, pp. 1–22. Cambridge, MA: MIT Press.

Barlow, H. B., T. P. Kaushal, and G. J. Mitchison (1989). Finding minimum entropy codes. *Neural Computation 1*(3), 412–423.

Becker, S. (1989). Unsupervised learning procedures for neural networks. *International Journal of Neural Systems* 2(1), 17–33.

Becker, S. and G. E. Hinton (1992). A self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature 355*(6356), 161–163.

Bell, A. J. and T. J. Sejnowski (1995). An information maximization approach to blind separation and blind deconvolution. *Neural Computation 7*(6), 1129–1159.

Bell, A. J. and T. J. Sejnowski (1996). Learning the higher-order structure of a natural sound. *Network: Computation in Neural Systems 7*(2), 261–267.

Bell, A. J. and T. J. Sejnowski (1997). The independent components of natural scenes. To appear in *Vision Reseach*.

Bellman, R. (1961). *Adaptive Control Processes: A Guided Tour*. Princeton, NJ: Princeton University Press.

Bertero, M., T. A. Poggio, and V. Torre (1988). Ill-posed problems in early vision. *Proceedings of the IEEE 76*(8), 869–889.

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

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford: Clarendon.

van den Bos, A. (1971). Alternative interpretation of maximum entropy spectral analysis. *IEEE Transactions on Information Theory 17*(4), 493–494.

Bourlard, H. and Y. Kamp (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics 59*(4–5), 291–294.

Breiman, L. (1993). Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory 39*(3), 999–1013.

Bridle, J. S. (1990a). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman-Soulié and J. Herault (Eds.), *Neurocomputing: Algorithms, Architectures and Applications*, pp. 227–236. Berlin: Springer-Verlag.

Bridle, J. S. (1990b). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*, pp. 211–217. San Mateo, CA: Morgan Kaufmann.

Brown, T. H., E. W. Kairiss, and C. L. Keenan (1990). Hebbian synapses: Biophysical mechanisms and algorithms. *Annual Review of Neuroscience 13*, 475–511.

Burel, G. (1992). Blind separation of sources: A nonlinear neural algorithm. *Neural Networks 5*(6), 937–947.

Buser, P. A. and M. Imbert (1992). *Audition*. Cambridge, MA: MIT Press.

Cardoso, J.-F. (1996). Infomax and maximum likelihood for blind source separation. To appear in *IEEE Signal Processing Letters*.

Carpenter, G. A. and S. Grossberg (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics and Image Processing 37*(1), 54–115.

Chauvin, Y. (1989a). A back-propagation algorithm with optimal use of hidden units. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1*, pp. 519–526. San Mateo, CA: Morgan Kaufmann.

Chauvin, Y. (1989b). Principal component analysis by gradient descent on a constrained linear Hebbian cell. In *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, Volume I, pp. 373–380. New York: IEEE.

Chen, S. S. (1995). *Basis Pursuit*. Ph. D. thesis, Stanford University.

Chiu, C. C. (1996). *Detection of Occluding Boundaries in Spatiotemporal Images*. Ph. D. thesis, Cambridge University Engineering Department.

Cohen, M. A. and S. Grossberg (1986). Neural dynamics of speech and language coding: Developmental programs, perceptual grouping, and competition for short term memory. *Human Neurobiology 5*(1), 1–22.

Comon, P. (1994). Independent component analysis, a new concept? *Signal Processing 36*(3), 287–314.

Comon, P., C. Jutten, and J. Herault (1991). Blind separation of sources, part II: Problems statement. *Signal Processing 24*(1), 11–20.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, Shaker Heights, OH, pp. 151–158. New York: Association for Computing Machinery.

Cover, T. M. and J. A. Thomas (1991). *Elements of Information Theory*. New York: Wiley.

Dagpunar, J. (1988). *Principles of Random Variate Generation*. Oxford: Clarendon.

Daugman, J. G. (1985). Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *Journal of the Optical Society of America A 2*(7), 1160–1169.

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

Daugman, J. G. (1989). Entropy reduction and decorrelation in visual coding by oriented neural receptive fields. *IEEE Transactions on Biomedical Engineering 36*(1), 107–114.

Daugman, J. G. (1993). High confidence visual recognition of persons by a test of statistical independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*(11), 1148–1161.

Dayan, P., G. E. Hinton, R. M. Neal, and R. S. Zemel (1995). The Helmholtz machine. *Neural Computation 7*(5), 889–904.

Deco, G. and D. Obradovic (1995). Linear redundancy reduction learning. *Neural Networks 8*(5), 751–755.

Deco, G. and D. Obradovic (1996). *An Information-Theoretic Approach to Neural Computing*. New York: Springer-Verlag.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B 39*(1), 1–38.

Diaconis, P. and M. Shahshahani (1984). On nonlinear functions of linear combinations. *SIAM Journal on Scientific and Statistical Computing 5*(1), 175–191.

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. New York: Wiley.

Edelman, S. and D. Weinshall (1991). A self-organizing multiple-view representation of 3D objects. *Biological Cybernetics 64*(3), 209–219.

Edelsbrunner, H. (1987). *Algorithms in Combinatorial Geometry*. Berlin: Springer-Verlag.

Field, D. J. (1987). Relations between the statistics of natural images and the response properties of cortical cells. *Journal of the Optical Society of America A 4*(12), 2379–2394.

Field, D. J. (1994). What is the goal of sensory coding? *Neural Computation 6*(4), 559–601.

Field, D. J. and D. J. Tolhurst (1986). The structure and symmetry of simple-cell receptive field profiles in the cat's visual cortex. *Proceedings of the Royal Society of London Series B 228*(1253), 379–400.

Fletcher, R. (1987). *Practical Methods of Optimization* (2nd ed.). Chichester, UK: Wiley.

Földiák, P. (1989). Adaptive network for optimal linear feature extraction. In *Proceedings of the International Joint Conference on Neural Networks*, Washington, DC, Volume I, pp. 401–405. New York: IEEE.

Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics 64*(2), 165–170.

Földiák, P. (1991). Learning invariance from transformation sequences. *Neural Computation 3*(2), 194–200.

Földiák, P. (1992). Models of sensory coding. Technical Report CUED/F-INFENG/TR 91, Cambridge University Engineering Department.

Földiák, P. and M. P. Young (1995). Sparse coding in the primate cortex. In M. A. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks*, pp. 895–898. Cambridge, MA: MIT Press.

Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition* (2nd ed.). Boston, MA: Academic Press.

Furui, S. (1986). Speaker-independent isolated word recognition using dynamic features of speech spectrum. *IEEE Transactions on Acoustics Speech and Signal Processing 34*(1), 52–59.

Fyfe, C. and R. J. Baddeley (1995a). Finding compact and sparse distributed representations of visual images. *Network: Computation in Neural Systems 6*(3), 334–344.

Fyfe, C. and R. J. Baddeley (1995b). Nonlinear data structure extraction using simple Hebbian networks. *Biological Cybernetics 72*(6), 533–541.

Gales, M. J. F. (1995). *Model-Based Techniques for Noise Robust Speech Recognition*. Ph. D. thesis, Cambridge University Engineering Department.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: W. H. Freeman.

Ghahramani, Z. (1995). Factorial learning and the *EM* algorithm. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in Neural Information Processing Systems 7*, pp. 617–624. Cambridge, MA: MIT Press.

Gill, P. E. and W. Murray (1976). Minimization subject to bounds on the variables. Technical Report NAC 72, National Physical Laboratory.

Gill, P. E., W. Murray, and M. H. Wright (1991). *Numerical Linear Algebra and Optimization*. Redwood City, CA: Addison-Wesley.

González, G., R. E. Badra, R. Medina, and J. Regidor (1995). Period estimation using minimum entropy deconvolution (MED). *Signal Processing 41*(1), 91–100.

Gray, C. M. and W. Singer (1989). Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *Proceedings of the National Academy of Sciences of the United States of America 86*(5), 1698–1702.

Gray, R. M. (1984). Vector quantization. *IEEE ASSP Magazine 1*(2), 4–29.

Grenander, U. (1976–81). *Lectures in Pattern Theory I, II and III: Pattern Analysis, Pattern Synthesis and Regular Structures*. New York: Springer-Verlag.

Hadamard, J. (1923). *Lectures on the Cauchy Problem in Linear Partial Differential Equations*. New Haven, CT: Yale University Press.

Halman, H. H. (1976). *Modern Factor Analysis*. Chicago: University of Chicago Press.

Harpur, G. F. and R. W. Prager (1994). A fast method for activating competitive self-organising neural networks. In *Proceedings of the 1994 International Symposium on Artificial Neural Networks*, Tainan, pp. 412–418.

Harpur, G. F. and R. W. Prager (1995). Techniques in low entropy coding with neural networks. Technical Report CUED/F-INFENG/TR 197, Cambridge University Engineering Department.

Harpur, G. F. and R. W. Prager (1996). Development of low entropy coding in a recurrent network. *Network: Computation in Neural Systems 7*(2), 277–284.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. New York: Macmillan.

Haykin, S. (1996). *Adaptive Filter Theory* (3rd ed.). Upper Saddle River, NJ: Prentice-Hall.

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

Hecht-Nielsen, R. (1995). Replicator neural networks for universal optimal source coding. *Science 269*(5232), 1860–1863.

Hertz, J., A. Krogh, and R. G. Palmer (1991). *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley.

Hinton, G. E., P. Dayan, B. J. Frey, and R. M. Neal (1995). The wake-sleep algorithm for unsupervised neural networks. *Science 268*(5214), 1158–1161.

Hinton, G. E. and J. L. McClelland (1988). Learning representations by recirculation. In D. Z. Anderson (Ed.), *Neural Information Processing Systems*, pp. 358–366. New York: American Institute of Physics.

Hinton, G. E. and R. S. Zemel (1994). Autoencoders, minimum description length and Helmholtz free energy. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6*, pp. 3–10. San Mateo, CA: Morgan Kaufmann.

Hornik, K. and C.-M. Kuan (1992). Convergence analysis of local feature extraction algorithms. *Neural Networks 5*(2), 229–240.

Huber, P. J. (1985). Projection pursuit (with discussion). *Annals of Statistics 13*(2), 435–525.

Huffman, D. A. (1971). Impossible objects as nonsense sentences. In B. Meltzer and D. Michie (Eds.), *Machine Intelligence*, Volume 6, pp. 295–323. Edinburgh: Elsevier.

Intrator, N. (1992). Feature-extraction using an unsupervised neural network. *Neural Computation 4*(1), 98–107.

Intrator, N. and L. N. Cooper (1992). Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions. *Neural Networks 5*(1), 3–17.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton (1991). Adaptive mixtures of local experts. *Neural Computation 3*(1), 79–87.

Jervis, T. T. and W. J. Fitzgerald (1993). Optimization schemes for neural networks. Technical Report CUED/F-INFENG/TR 144, Cambridge University Engineering Department.

Jolliffe, I. T. (1986). *Principal Component Analysis*. New York: Springer-Verlag.

Jones, J. P. and L. A. Palmer (1987). An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. *Journal of Neurophysiology 58*(6), 1233–1258.

Jones, M. C. and R. Sibson (1987). What is projection pursuit? (with discussion). *Journal of the Royal Statistical Society Series A 150*(P1), 1–36.

Jutten, C. and J. Herault (1991). Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing 24*(1), 1–10.

Kandel, E. R., J. H. Schwartz, and T. M. Jessell (Eds.) (1991). *Principles of Neural Science* (3rd ed.). New York: Elsevier.

Kanizsa, G. (1979). *Organization in Vision*. New York: Praeger.

Karhunen, J., L. Y. Wang, and R. Vigario (1995). Nonlinear PCA type approaches for source separation and independent component analysis. In *Proceedings of the 1995 IEEE International Conference on Neural Networks (ICNN'95)*, Perth, Australia, pp. 995–1000.

Kawato, M., H. Hayakawa, and T. Inui (1993). A forward-inverse optics model of reciprocal connections between visual cortical areas. *Network: Computation in Neural Systems 4*(4), 415–422.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43*(1), 59–69.

Kohonen, T. (1989). *Self-Organization and Associative Memory* (3rd ed.). Berlin: Springer-Verlag.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE 78*(9), 1464–1480.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal 37*(2), 233–243.

Krogh, A. and J. Vedelsby (1995). Neural network ensembles, cross validation and active learning. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in Neural Information Processing Systems 7*, pp. 231–238. Cambridge, MA: MIT Press.

Kuan, C.-M. and K. Hornik (1991). Convergence of learning algorithms with constant learning rates. *IEEE Transactions on Neural Networks 2*(5), 484–489.

Kumaran, K., D. Geiger, and L. Gurvits (1996). Illusory surface perception and visual organization. *Network: Computation in Neural Systems 7*(1), 33–60.

Kung, S. Y. and K. I. Diamantaras (1990). A neural network learning algorithm for adaptive principal component extraction (APEX). In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing*, Albuquerque, NM, pp. 861–864.

Laughlin, S. (1981). A simple coding procedure enhances a neuron's information capacity. *Zeitschrift Für Naturforschung Section C: Biosciences 36*(9–10), 910–912.

Leen, T. K. (1991). Dynamics of learning in linear feature-discovery networks. *Network: Computation in Neural Systems 2*(1), 85–105.

Lesher, G. W. and E. Mingolla (1995). Illusory contour formation. In M. A. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks*, pp. 481–483. Cambridge, MA: MIT Press.

Linde, Y., A. Buzo, and R. M. Gray (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications 28*(1), 84–95.

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

Linsker, R. (1990). Self-organization in a perceptual system: How network models and information theory may shed light on neural organization. In S. J. Hanson and C. R. Olson (Eds.), *Connectionist Modeling and Brain Function: The Developing Interface*, pp. 351–392. Cambridge, MA: MIT Press.

Linsker, R. (1992). Local synaptic learning rules suffice to maximize mutual information in a linear network. *Neural Computation 4*(5), 691–702.

Ljung, L. and T. Söderström (1983). *Theory and Practice of Recursive Identification*. Cambridge, MA: MIT Press.

Luenberger, D. G. (1984). *Introduction to Linear and Nonlinear Programming* (2nd ed.). Reading, MA: Addison-Wesley.

Mackay, D. J. C. (1995). Probable networks and plausible predictions: A review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems 6*(3), 469–505.

Mackay, D. J. C. (1996). Maximum likelihood and covariant algorithms for independent component analysis. Draft paper.

Mackay, D. M. (1956). The epistemological problem for automata. In C. E. Shannon and J. McCarthy (Eds.), *Automata Studies*, pp. 235–251. Princeton, NJ: Princeton University Press.

Malik, J. (1996). On binocularly viewed occlusion junctions. In B. Buxton and R. Cipolla (Eds.), *Lecture Notes in Computer Science*, Volume 1064, pp. 167–174. Berlin: Springer-Verlag.

von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in striate cortex. *Kybernetik 14*, 85–100.

Marshall, J. A. (1995). Adaptive perceptual pattern recognition by self-organizing neural networks: Context, uncertainty, multiplicity, and scale. *Neural Networks 8*(3), 335–362.

Marshall, J. A., R. K. Alley, and R. S. Hubbard (1996). Learning to predict visibility and invisibility from occlusion events. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems 8*, pp. 816–822. Cambridge, MA: MIT Press.

Merhav, N. and Y. Ephraim (1991). Maximum-likelihood hidden Markov modeling using a dominant sequence of states. *IEEE Transactions on Signal Processing 39*(9), 2111–2115.

Mozer, M. C. (1991). Discovering discrete distributed representations with iterative competitive learning. In R. P. Lippman, J. E. Moody, and D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems 3*, pp. 627–634. San Mateo, CA: Morgan Kaufmann.

Mumford, D. (1992). On the computational architecture of the neocortex, 2: The role of cortico-cortical loops. *Biological Cybernetics 66*(3), 241–251.

Mumford, D. (1994). Neuronal architectures for pattern-theoretic problems. In C. Koch and J. L. Davis (Eds.), *Large-Scale Neuronal Theories of the Brain*, pp. 125–152. Cambridge, MA: MIT Press.

Murphy, P. C. and A. M. Sillito (1996). Functional morphology of the feedback pathway from area 17 of the cat visual cortex to the lateral geniculate nucleus. *Journal of Neuroscience 16*(3), 1180–1192.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence 56*(1), 71–113.

Neal, R. M. and P. Dayan (1996). Factor analysis using delta-rule wake-sleep learning. Technical Report 9607, Department of Statistics, University of Toronto.

Neal, R. M. and G. E. Hinton (1993). A new view of the EM algorithm that justifies incremental and other variants. Submitted to *Biometrika*.

Numerical Algorithms Group (1996). *NAG Fortran Library Manual Mark 17*. Oxford: Numerical Algorithms Group.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology 15*(3), 267–273.

Oja, E. (1989). Neural networks, principal components, and subspaces. *International Journal of Neural Systems 1*(1), 61–68.

Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Networks 5*(6), 927–935.

Oja, E. (1995). PCA, ICA, and nonlinear Hebbian learning. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN-95)*, Paris, France, pp. 89–94.

Oja, E. and J. Karhunen (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications 106*(1), 69–84.

Olshausen, B. A. (1996). Learning linear, sparse, factorial codes. Memo AIM-1580, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Olshausen, B. A. and D. J. Field (1996a). Emergence of simple-cell receptive-field properties by learning a sparse code for natural images. *Nature 381*(6583), 607–609.

Olshausen, B. A. and D. J. Field (1996b). Natural image statistics and efficient coding. *Network: Computation in Neural Systems 7*(2), 333–339.

Olshausen, B. A. and D. J. Field (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? To appear in *Vision Research*.

O'Rourke, J. (1994). *Computational Geometry in C*. Cambridge: Cambridge University Press.

O'Shaughnessy, D. (1987). *Speech Communication: Human and Machine*. Reading, MA: Addison-Wesley.

Parra, L., G. Deco, and S. Miesbach (1995). Redundancy reduction with information-preserving nonlinear maps. *Network: Computation in Neural Systems 6*(1), 61–72.

Parra, L., G. Deco, and S. Miesbach (1996). Statistical independence and novelty detection with information preserving nonlinear maps. *Neural Computation 8*(2), 260–269.

Paul, D. B. and J. M. Baker (1992). The design for the Wall Street Journal-based CSR corpus. In *Proceedings of the Fifth DARPA Speech and Natural Language Workshop*, pp. 357–362. San Francisco, CA: Morgan Kaufmann.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine 2*, 559–572.

Pece, A. E. C. (1992). Redundancy reduction of a Gabor representation: A possible computational role for feedback from primary visual cortex to lateral geniculate nucleus. In I. Aleksander and J. Taylor (Eds.), *Artificial Neural Networks 2*, pp. 865–868. Amsterdam: Elsevier.

Perrone, M. P. and L. N. Cooper (1993). When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone (Ed.), *Artificial Neural Networks for Speech and Vision*, pp. 126–142. London: Chapman & Hall.

Plumbley, M. D. (1991). On information theory and unsupervised neural networks. Technical Report CUED/F-INFENG/TR 78, Cambridge University Engineering Department.

Plumbley, M. D. (1993). Efficient information transfer and anti-Hebbian neural networks. *Neural Networks 6*(6), 823–833.

Poggio, T. A., V. Torre, and C. Koch (1985). Computational vision and regularization theory. *Nature 317*(6035), 314–319.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling (1986). *Numerical Recipes: The Art of Scientific Computing*. Cambridge: Cambridge University Press.

Price, P., W. M. Fisher, J. Bernstein, and D. S. Pallett (1988). The DARPA 1000-word Resource Management database for continuous speech recognition. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP 88)*, New York, Volume I, pp. 651–654. New York: IEEE.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE 77*(2), 257–286.

Redlich, A. N. (1993a). Redundancy reduction as a strategy for unsupervised learning. *Neural Computation 5*(2), 289–304.

Redlich, A. N. (1993b). Supervised factorial learning. *Neural Computation 5*(5), 750–766.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press.

Rissanen, J. (1978). Modeling by shortest data description. *Automatica 14*(5), 465–471.

Rissanen, J. (1985). Minimum description length principle. In S. Kotz and N. L. Johnson (Eds.), *Encyclopedia of Statistical Sciences*, Volume 5, pp. 523–527. New York: Wiley.

Robbins, H. and S. Monro (1951). A stochastic approximation method. *Annals of Mathematical Statistics 22*, 400–407.

Robinson, A. J. and F. Fallside (1992). Phoneme recognition from the TIMIT database using recurrent error propagation networks. Technical Report CUED/F-INFENG/TR 42, Cambridge University Engineering Department.

Rolls, E. T. and M. J. Tovee (1995). Sparseness of the neuronal representation of stimuli in the primate temporal visual cortex. *Journal of Neurophysiology 73*(2), 713–726.

Rubner, J. and P. Tavan (1989). A self-organizing network for principal component analysis. *Europhysics Letters 10*(7), 693–698.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Volume 1: Foundations, pp. 318–362. Cambridge, MA: MIT Press.

Rumelhart, D. E. and D. Zipser (1985). Feature discovery by competitive learning. *Cognitive Science 9*(1), 75–112.

Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks 2*(6), 459–473.

Satorius, E. H. and J. J. Mulligan (1992). Minimum entropy deconvolution and blind equalisation. *Electronics Letters 28*(16), 1534–1535.

Saund, E. (1994). Unsupervised learning of mixtures of multiple causes in binary data. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6*, pp. 27–34. San Mateo, CA: Morgan Kaufmann.

Saund, E. (1995). A multiple cause mixture model for unsupervised learning. *Neural Computation 7*(1), 51–71.

Schmidhuber, J. (1992). Learning factorial codes by predictability minimization. *Neural Computation 4*(6), 863–879.

Senior, A. W. (1994). *Off-Line Cursive Handwriting Recognition Using Recurrent Neural Networks*. Ph. D. thesis, Cambridge University Engineering Department.

Shannon, C. E. and W. Weaver (1949). *The Mathematical Theory of Communication*. Urbana, IL: University of Illinois Press.

Sorouchyari, E. (1991). Blind separation of sources, part III: Stability analysis. *Signal Processing 24*(1), 21–29.

Stockmeyer, L. J. (1975). The set basis problem is NP-complete. Technical Report RC-5431, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Strang, G. (1988). *Linear Algebra and Its Applications* (3rd ed.). San Diego: Harcourt Brace Jovanovich.

Thorpe, S. (1995). Localized versus distributed representations. In M. A. Arbib (Ed.), *The Handbook of Brain Theory and Neural Networks*, pp. 549–552. Cambridge, MA: MIT Press.

Tikhonov, A. N. and V. Y. Arsenin (1977). *Solutions of Ill-Posed Problems*. Washington, DC: V. H. Winston.

Treves, A. and E. T. Rolls (1991). What determines the capacity of autoassociative memories in the brain? *Network: Computation in Neural Systems 2*(4), 371–397.

Tukey, P. A. and J. W. Tukey (1981). Preparation; prechosen sequences of views. In V. Barnett (Ed.), *Interpreting Multivariate Data*. Chichester: Wiley.

Ullman, S. (1994). Sequence seeking and counterstreams: A model for bidirectional information flow in the cortex. In C. Koch and J. L. Davis (Eds.), *Large-Scale Neuronal Theories of the Brain*, pp. 257–270. Cambridge, MA: MIT Press.

Wada, Y. and M. Kawato (1993). A neural network model for arm trajectory formation using forward and inverse dynamics models. *Neural Networks 6*(7), 919–932.

Wallis, G. and E. T. Rolls (1996). A model of invariant object recognition. To appear in *Progress in Neurobiology*.

Wandell, B. A. (1995). *Foundations of Vision*. Sunderland, MA: Sinauer Associates.

Watanabe, S. (1985). *Pattern Recognition: Human and Mechanical*. New York: Wiley.

Watson, A. B. (1983). Detection and recognition of simple spatial forms. In O. J. Braddick and A. C. Sleigh (Eds.), *Physical and Biological Processing of Images*. Berlin: Springer-Verlag.

Weigend, A. S., D. E. Rumelhart, and B. A. Huberman (1991). Generalization by weight-elimination with application to forecasting. In R. P. Lippman, J. E. Moody, and D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems 3*. San Mateo, CA: Morgan Kaufmann.

Wiggins, R. A. (1978). Minimum entropy deconvolution. *Geoexploration 16*(1), 21–35.

Williams, R. J. (1985). Feature discovery through error-correction learning. Technical Report 8501, Institute for Cognitive Science, University of California, San Diego.

Xu, L. and A. L. Yuille (1993). Self-extracting rules for robust PCA. In S. J. Hanson, J. D. Cowan, and C. L. Giles (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 467–474. San Mateo, CA: Morgan Kaufmann.

Xu, L. and A. L. Yuille (1995). Robust principal component analysis by self-organizing rules based on statistical physics approach. *IEEE Transactions on Neural Networks 6*(1), 131–143.

Yang, H. S. and A. C. Kak (1989). Edge extraction and labelling from structured light 3D vision data. In S. Haykin (Ed.), *Selected Topics in Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall.

Zemel, R. S. (1993). *A Minimum Description Length Framework for Unsupervised Learning*. Ph. D. thesis, Graduate Department of Computer Science, University of Toronto.

Zue, V., S. Seneff, and J. Glass (1990). Speech database development at MIT: TIMIT and beyond. *Speech Communication 9*(4), 351–356.

Appendix A

Error function gradients for the REC network

Error functions for the REC network are minimised with respect to both outputs and weights. There is nothing complicated in the mathematics associated with this, but it is not easy to make the associated vector calculus clear in a concise notation without some explanation. This appendix sets this out, and helps to show why, at the expense of recurrent activation in the REC model, learning is straightforward.

A single-pattern error function E for the REC network is, very generally, some function of the inputs, the network weights, and the output values. The aim of activation is to minimise E with respect to the outputs $\mathbf{a}$, and of learning to minimise $\mathcal{E}[E]$ with respect to the weights $\mathbf{W}$. For ease of notation, we shall reshape the weight matrix $\mathbf{W}$ into one large column vector $\mathbf{w}$. The input data are fixed, so we may think of the error for these purposes as being a function of the weights and outputs only, where the outputs themselves are also functions of the weights. We can make this explicit by writing E as

$$E\Big(\mathbf{w}, \mathbf{a}(\mathbf{w})\Big).$$

Activation minimises E w.r.t. $\mathbf{a}$ for a fixed value of $\mathbf{w}$. Assuming $\mathbf{a}$ is unconstrained, the minimum occurs at a stationary point, so the gradient of E w.r.t. $\mathbf{a}$ is zero,[1] which we shall write as

$$\nabla_{\mathbf{a}} E\,\big|_{\mathbf{w}} = \mathbf{0}. \tag{A.1}$$

In general, for a scalar function $f(x, y)$ of dependent scalar variables, we may write the gradient as

$$\frac{df}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx}.$$

Extending this relation to the vector case, and applying it to the error function E, we may write

$$\nabla_{\mathbf{w}} E = \nabla_{\mathbf{w}} E\,\big|_{\mathbf{a}} + \mathbf{J} \,.\, \nabla_{\mathbf{a}} E\,\big|_{\mathbf{w}} \tag{A.2}$$

[1] When the values of $\mathbf{a}$ *are* constrained, and the minimum occurs on the boundary of the permitted values, it is quite possible that (A.1) will not be satisfied. We should note that the results of this analysis cannot be justified fully in this case.

where **J** is the *Jacobian* matrix with elements

$$J_{ij} = \frac{\partial a_j}{\partial w_i}.$$

Equation (A.2) gives the gradient of the error function w.r.t. the weights and hence, after negation, yields a learning rule for stochastic gradient descent on $\mathcal{E}[E]$. This expression is potentially complex, but for the fact that the condition (A.1) resulting from the activation stage allows us to simplify (A.2) to

$$\nabla_{\mathbf{w}} E = \nabla_{\mathbf{w}} E \big|_{\mathbf{a}}$$

for the stable output values **a**. This tells us that the full gradient in weight-space is equal to the partial gradient with the outputs **a** fixed. It is this simplification, which has come about as a result of the minimisation at the activation stage, that allows us to use local Hebbian learning rules in all variants of the REC network presented in this work. This contrasts, for example, with feedforward MLPs, where activation is simple, but learning is non-local and more complex.