# Neural Networks, Information Theory and Knowledge Representation

**Julian Smith**

**Ph.D.**

**The University of Edinburgh**

**1995**

# Contents

## Introduction

This thesis is concerned with investigating the modelling of cognition using neural networks, with emphasis on making the network models biologically plausible.

It is impossible to build neural network models of the brain from precise mathematical models of individual neurons, for two reasons: first, the exact behaviour of real neurons is not known and, second, even if we knew the precise behaviour of real neurons, present day computers would not be powerful enough to model large numbers of them.

Despite these problems, using networks of units as cognitive models is still attractive because it is known that the brain is constructed in this way. For example, one hope is that the precise behaviour of neurons is not important to brain function, and that the fact that the brain is made up of many relatively simple units connected together *is* important. Hence we can attempt to model cognition using networks of idealised units which approximately correspond to real neurons.

Although knowledge of the details of neuron behaviour is incomplete, there are a number of details which are relatively certain (Kuffler et al [1984], Hodgkin & Huxley [1952], Douglas & Martin [1990]):

- Most neurons have many inputs, and one output which splits up and connects to many other neurons.
- The input/output signals consist of pulses of varying frequency.
- Neuron behaviour is determined at least partly by the synapses, which are the interface between inputs and a neuron's dendrites.

The simplifications which lie behind most neural network models are that the signal frequency can convey a continuous signal, and that the neurons alter their behaviour by adjusting their synapse strengths, so changing how much or how little a particular input affects the neuron.

Much of the enormous amount of neural network research done in the last few years has been prompted by the invention of the back propagation learning algorithm. This enables neural network models to be made which can learn to solve extremely complicated problems, sometimes in ways similar to real cognition. Unfortunately, a back propagation

neural network is a rather extreme distortion of the skeleton neural network described above, as it postulates the existence of error signals which travel in the opposite direction to the main neuron output signals, so as to inform neurons about whether they are having a good or bad influence on neurons further down the network.

While such additional features certainly add to the power of neural networks, it is preferable to make as few assumptions which go beyond known physiological evidence as possible. Otherwise, we could add all sorts of features to make a particular modelling task easier, but which might make the task completely different from the real task which the brain has to cope with.

A common criticism of back propagation is that it is a supervised algorithm. I don't consider this to be the main problem though; at some stage there has to be feedback involved in the brain, which is similar to the idea of supervision - for example semi-autonomous networks supervising each other.

However, back propagation's use of separate error signals which are provided for each unit, is a much more important issue. The main criticism of these individual error signals is that there is no biological evidence that they exist in the brain. They have been invented because they provide a particularly simple and effective way of making networks learn to perform arbitrary tasks: with error signals back propagating from unit to unit, one can effectively optimise the performance of each unit individually, by using the chain rule from differential calculus to find the effect that each unit is indirectly having on the output of the whole network.

Apart from there being little biological evidence for the existence of error signals to individual neurons in the brain, it is clear that much useful processing of data can take place *without* such error signals. A particularly persuasive example of this is the work in Finch & Chater [1992] and Finch et al [1995], where statistical analysis of a large corpus of written text resulted in the creation of word-categories which were similar to those used by linguists, and also seem to have some semantic validity. For examples of more general statistical analysis of data by unsupervised neural networks, see the the work on principal components analysis in Plumbley [1991], Földiák [1991], Földiák [1992], and some of the work described later in this thesis.

If we restrict ourselves to non-error correcting network learning algorithms, we can't

perform arbitrary data transformations - a detailed error signal would be required for this. Instead, we must rely on there being some statistical structure present in the input data set - for example, if you put completely random data into a principal components network, the weight vectors will move around aimlessly and never converge to a stable state. However, real data does indeed have lots of structure, so simple statistical methods are worth considering for at least the early stages of sense-data processing.

For this reason, the work in this thesis is an investigation into how the brain might extract and use statistical structure in input data.

## Thesis summary

A very brief description of each of the chapters in the thesis is included below. The central idea in the thesis is presented in chapter 4. Its consequences are explored in chapters 5 and 6, and ideas for further work based on this idea are explained in chapter 7.

Chapters 1 and 2 discuss unsupervised neural network learning, and the idea that significant statistical information is present in environmental data. Chapter 3 described a (non-network) model of reading aloud.

#### Chapter 1

A description of some common unsupervised neural network learning algorithms.

#### Chapter 2

A discussion of problems with unsupervised network models which have recurrent connections. One can make this sort of network perform Principal Components Analysis on the data, e.g. Plumbley [1991], Földiák [1992]. Unfortunately, I have found significant problems with the practical behaviour of this sort of network, caused by the use of recurrent connections. The lateral connections mean that the network is susceptible to oscillations, and to overcome this, one has to use what I call a 'differential state update' method, in which the state of each unit is increased by an amount proportional to the units inputs, rather than being set directly to a value proportional to the inputs. This method is implicitly used by Földiák [1992]. I have found that even with differential state update, the network can take many iterations to stabilise, which is at variance with known response times in everyday psychological experiments, where there is virtually no time for real neural networks to iteratively approach a stable set of activations.

#### Chapter 3

This is on a slightly different subject matter from the rest of this thesis. A model of reading aloud is described, which was developed using the ideas presented earlier about how the brain could take advantage of structure contained in the statistics of sense data, although the statistical analysis is not done with neural networks.

**Chapter 4**

This chapter presents the central idea in this thesis. It explains how information theory can be used to develop a general definition of a high-level representation. This is a very important idea, as it provides a mathematical framework for research into how the brain transforms sense data into a useful high-level form, and suggests ways in which this could be done using unsupervised neural networks.

High-level cognitive functions like walking, talking, abstract thinking etc., are clearly only possible because we deal with the world at a 'higher level' than simple sense data. Hence the importance of the general definition of a high-level representation presented in this chapter.

**Chapter 5**

This presents an application of the ideas developed in chapter 4, to low-level vision. A very simple statistical model of visual data is described, and the filter which would convert this data into a high-level representation is derived. The filter shows some similarities to the early visual centres in the brain, and provides a striking example of how the rather abstract information-theoretic ideas from chapter 4 can have very interesting and realistic consequences.

**Chapter 6**

A very simple neural network learning rule is described which attempts to make the output of the network have maximum entropy, in accordance with the ideas developed in chapter 4. The rule uses one extra signal in learning, representing the average activation of all units, to provide a crude form of de-correlation. The rule is applied to input patterns which are bit-images of the digits 0-9, and units are shown to approximately develop weights which make them respond to different input patterns.

**Chapter 7**

A summary of the thesis, and some ideas for future work which could extend the work on maximum-entropy representations and neural networks.

**Acknowledgements**

I am very grateful to my supervisor, Nick Chater, who has helped in the development of many of the ideas in this thesis. Of particular influence were many interesting discussions about information theory and maximum-entropy representations, and their relevance to unsupervised neural network models.

I would also like to thank Chris Huckle for reading through and commenting on later versions of the thesis.

**Declaration (University of Edinburgh Postgraduate Study Regulation 3.4.7)**

**(a)** This thesis has been composed by myself.

**(b)** The work described in this thesis is my own.

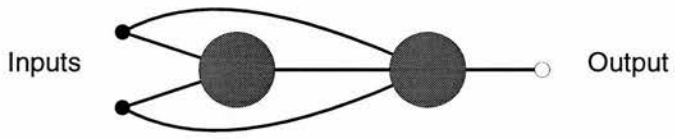
# Chapter 1 Some common unsupervised neural network learning algorithms

This chapter is a description of some common local and non-error-correcting network learning algorithms.

### Some notes on error-correction

A very powerful way of training a neural network is to use a form of back propagation (Werbos [1974], Rumelhart et al [1986], Parker [1985]), where error signals are calculated and sent to individual units. Unfortunately there is no evidence that such error-correcting signals at the level of individual neurons occur in real brains. Instead, a unit's weights have to be changed using the much less specific (from a weight-changing point of view) information contained just in the inputs to that unit. This means that a layer of units in a realistic (i.e. biologically plausible) network cannot be expected to organise themselves in order to perform any given function unless that function can be realised by the action of learning rules which are in some sense localised to each unit's inputs and output.

This is a serious restriction on a neural network. For example, if one wants the following network to solve the Exclusive-OR problem, and the output unit ends up at some stage with weights that approximately OR the two inputs, then it would be very useful for a hidden unit to AND the two inputs, as this would provide the output unit with just the extra information that it requires in order to solve the task.




***A simple network which could solve the XOR problem if the hidden unit is given an error signal.***

However, with no single-unit error signals, there is no particular reason why the hidden unit should suddenly decide to AND its inputs. Thus it would seem that not being able to have detailed error signals is a serious restriction on network learning ability.

Reinforcement learning (Barto and Anandan [1985], Barto and Jordan [1987], Grossman [1990]) is a sort of halfway house between this seemingly completely unsupervised type of

network and the back-propagation algorithm. With reinforcement learning, there is a single error, which is known to the entire system, and which condenses the performance of the whole network into a single number. In the Exclusive OR problem mentioned above, any random fluctuation of the hidden unit's weights which made it perform more like an AND gate (and assuming by chance that the weight from hidden to output was negative) would result in the network performance improving. This could be detected, and the hidden unit's weights could be made to continue changing in this direction.

However, this example illustrates the low power which any reinforcement algorithm has - it has only one number to guide the evolution of hundreds of weights in a reasonably sized network. One could increase the number of reinforcement signals available, but it still seems that individual units in the brain manage to change their weights in useful ways without using many external learning signals.

If a unit is not given any explicit information like "in future, respond more strongly to this input pattern", how is it to learn? The most common solution is to use a variant of the Hebbian learning rule. Often, a non-local controller is also included, for example in competitive learning, in order to make a network's weights evolve in a particular way. This has obvious biological problems, although inhibitory connections can be used to partly mimic the effect of such global controllers. Unfortunately, the use of inhibitory connections often results in the units in a network taking a long time to reach stable activations, and even in oscillatory behaviour. This means that the network is slow, a serious drawback considering the speed with which visual stimuli are recognised by people (Sejnowski [1986], Marr [1982], Morell [1972]).

To sum up, any unsupervised network learning model which doesn't use individual unit error signals can only process input data in a specific built-in way determined by the architecture of the model and the learning rule used by the units.

Neural networks function by being repeatedly presented with data, so the evolution of weights will be determined by the frequency with which the various input patterns occur. Thus one can look at the network as responding to particular statistics of the input dataset, as determined by the architecture and learning rules. The statistics that the network works with may not always be of a clean mathematical type, such as calculation of the principal components of the input distribution, but I think that regarding the processing of a neural network as being statistical ensures that one has a realistic expectation of its capabilities.

## Some well-known unsupervised non error-correcting neural networks

Some of the algorithms mentioned below use a global supervisor to control some of the training. This is obviously a problem, but it is sometimes possible to use lateral inhibitory connections to implement the effect of the global controller, so the behaviour of these algorithms is interesting in the present context.

Most of the following algorithms can be viewed as trying to make units decorrelated and/or maximise the mutual information between the input and output patterns. Chapter 4 of this thesis will give a general theoretical principle which explains why these two processes are useful.

This section doesn't include any original work, except for some of the comments on Linsker's network and its implications for purely statistical methods of data-transformation.

### Competitive learning

In competitive learning (Rosenblatt [1962], von der Malsburg [1973], Fukushima [1975][1980], Kohonen [1982], Rumelhart and Zipser [1985], the unit whose weight vector best matches the input vector is chosen as the winner, and is allowed to fire, and its weight vector is moved closer to the actual input.

A problem with this is that some units can get ignored initially because their initial weights don't match the input pattern sufficiently well, and so the other units' weights continuously get adjusted to fit the input patterns better. This results in the group of units keeping the same weights, and so never winning the competition.

One way of reducing this problem is to give units an adjustable threshold parameter which is subtracted from the weighted input. If a unit has not won the competition to be the best matched input for a long time, the threshold is lowered, making the unit more likely to win a competition in the future. This mechanism also works as a *conscience* (Rumelhart and Zipser [1985]) in units which happen to win too many competitions, by raising their threshold and so making them less likely to win in the future.

Each unit of the network can be looked on as responding to inputs that are in a certain category, so the network performs a form of cluster analysis.

One can attempt to implement competitive learning with inhibitory connections. The

feature-detecting networks described in chapter 2 can be viewed like this.

Clearly, the competitive learning algorithm tries to force units to behave differently from each other. This has important parallels with the information theoretic analysis of unsupervised learning described in chapter 4 of this thesis.

#### Kohonen networks

Kohonen networks are similar to competitive learning networks, except that the units are treated as if they are in a topographic array in (for instance) two dimensions (see Kohonen [1982] and Kohonen [1989] for a more detailed description of Kohonen networks). For each input pattern presentation, the unit with its weight vector closest to the input vector is nominated as the winner, and the weight vector of this winning unit is moved closer to that particular input, as in normal Hebbian learning.

Those units which are close to the winning unit in the topographic array also have their weights updated in the same way, though with a smaller learning rate determined by their distance form the winning unit. This means that units that are close together in the topographic map will eventually have similar weight vectors, so that after training the network will map the high dimensional input onto the much smaller dimensional space that the units are in, in the sense that inputs that are close together in the high dimensional input space will be mapped to neighbouring units in the unit's space.

Unfortunately, Kohonen networks can take many thousands of iterations to reach a stable set of weights, which is not really surprising considering the difficulty of mapping a high dimensional space to a lower dimensional one while attempting to preserve topographic information.

#### Linsker's self-organising network (Linsker [1986][1988])

Linsker's network is completely unsupervised, but when fed with random noise, units develop which perform visually important functions such as *centre-surround* and *orientation-sensitive* detectors. This seemingly miraculous behaviour is a result of the units having limited receptive fields.

The network uses *linear* units. It is well known that any multi-layer linear network is equivalent to a single-layer linear network, because each layer just performs a matrix

multiplication, and the transformation given by successive multiplication by any number of matrices can always be performed by just one matrix multiplication. However, the multi-layer aspect of Linsker's network is important for learning.

The units in a particular layer of the network receive inputs only from those units within their receptive fields in the previous layer. These receptive fields overlap, so the outputs from a layer will each go to more than one unit in the next layer. The learning rule for the weights is a generalised Hebbian rule:

$$\delta w_{ij} = a(B_i A_j) + b(B_i) + c(A_j) + d$$

where $B_i$ is the activation of unit $i$, $A_j$ the $j$-th input to the layer, and $w_{ij}$ is the weight for the $j$-th input to the $i$-th unit. $a$, $b$, $c$, $d$ are constants, which determine the particular patterns of weights that develop.

The weights are also constrained to lie in a certain range $w_{min} \rightarrow w_{max}$.

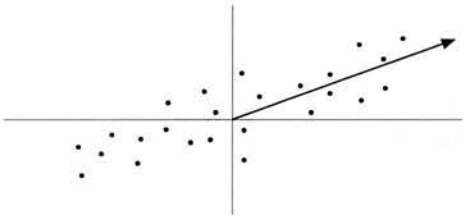
The four parameters in the learning rule have to be carefully selected so that the network would develop the feature detectors described earlier. The addition of locally excitatory connections in the layers that developed orientation-sensitive units made the units in the layer have a smoothly changing orientation preference, similar to those found biologically.

Linsker shows how the network maximises the variance of the output unit activations, subject to constraints on weight size. When the inputs are normally distributed, and there is gaussian noise on all lines, maximising the output variance also maximises the Shannon information transmission from inputs to outputs. Linsker uses this as a general principal (Infomax): units and architectures should be designed to maximise the Shannon information they transmit to subsequent layers. The assumption that inputs have a gaussian distribution is hardly realistic however, so Linsker's actual network learning rule is not a generally useful one. The Infomax principle has links with the Maximum Entropy Filter principle developed in chapter 4.

#### Networks which perform Principal Components analysis

Most of the results presented here are already well-known, but are included here because chapter 2 presents the results of simulations of principal component networks, and the mathematical derivations below can aid the understanding of the results presented there. See Hertz et al [1991] for a similar treatment, and also Sanger [1989] and Oja [1989].

Given many data points in an $n$-dimensional space, one can represent these points as linear combinations of $n$ basis vectors. The basis vectors can be chosen at random, with the proviso that they are linearly independent. Intuitively, the "First Principal Component" of a set of data points is the direction in which the data points vary most widely. If this direction is chosen as a basis vector, then the best way of representing the data using just one parameter is to take components of the data points along this Principal Component. Here, "best way of representing the data" means the method whose reconstruction of the input has the least mean-square error. The principal component is also the eigenvector of the correlation matrix of the data which has largest eigenvalue.



***A set of data points in 2 dimensions, with the arrow representing the principal component of the data***

Adding a second basis vector enables one to represent the data more accurately. Choosing the second basis vector to be along the direction which shows the highest variation, after subtracting the component along the first P.C., defines the second principal component. For the data in the diagram, there can only be 2 principal components because the data itself is only two-dimensional.

It is possible to make an unsupervised neural network find the principal components of input data. In fact the simple Hebbian algorithm with a linear unit will tend to align its weight vector with the first principal component of the input data. Mathematically, the first principal component is the direction of the maximal eigenvector of the correlation matrix of the input data:

The proof of this is straightforward, but rather long-winded. We will use a matrix notation for the various quantities as this simplifies things: We will consider a single linear unit with $n$ inputs, and let the activation of this unit be $\boldsymbol{B}$. The inputs will be represented as an $n$-row column matrix, $\boldsymbol{A}$, and the weights of the unit will be represented as $\boldsymbol{w}$, a row matrix with $n$ columns. Hence the state of the linear unit will be $\boldsymbol{B} = \boldsymbol{wA}$.

We are interested in the development of the weight vector when the unit is exposed to a set

of different input patterns, so for each different input pattern enumerated by $\mu$, we will set $\boldsymbol{A} = \boldsymbol{A}^{\mu}$.

Our unit has a Hebbian learning rule which in terms of components of the weight and input matrices is $\Delta \boldsymbol{w}_b = \eta \boldsymbol{B}\boldsymbol{A}_b$, where $\eta$ is the learnrate. In matrix notation this is $\Delta \boldsymbol{w} = \eta \boldsymbol{B}\boldsymbol{A}^T$, because $\boldsymbol{w}$ is a row matrix while $\boldsymbol{A}$ is a column matrix.

To see what happens to the weight vector when the unit is trained with many input patterns, we need to know what the average weight change $\langle \Delta \boldsymbol{w} \rangle$ is, by averaging $\Delta \boldsymbol{w}$ over all input patterns. We will also substitute in the expression for the state of the unit in terms of the input pattern and the weight vector, $\boldsymbol{B} = \boldsymbol{w}\boldsymbol{A}$, and ignore the learnrate:

$$\begin{aligned} \langle \Delta \boldsymbol{w} \rangle &= \langle \boldsymbol{B}\boldsymbol{A}^T \rangle \\ &= \langle \boldsymbol{w}\boldsymbol{A}\boldsymbol{A}^T \rangle \\ &= \mathbf{w}\mathbf{C}, \end{aligned}$$

where $\mathbf{C}$ is the covariance matrix of the input patterns, $\boldsymbol{C} = \langle \boldsymbol{A}\boldsymbol{A}^T \rangle$, assuming the input patterns have zero mean.

To show that the weight vector will tend to align itself along the first principal component of the input patterns, we will express it as a linear combination of all the eigenvectors of $\boldsymbol{C}$, and look at how the coefficients of this linear combination change, instead of how the whole weight vector changes.

We will let the $n$ eigenvectors of $\boldsymbol{C}$ be $\boldsymbol{c}_\nu$, and the $n$ coefficients of the linear expansion be $\alpha_\nu$, where $\nu$ runs from 1 to $n$ in both cases.

Then the weight vector can be written $\boldsymbol{w} = \sum_\nu \alpha_\nu \boldsymbol{c}_\nu$. Hence we get $\langle \Delta \boldsymbol{w} \rangle = \langle \Delta \sum_\nu \alpha_\nu \boldsymbol{c}_\nu \rangle = \sum_\nu \langle \Delta \alpha_\nu \rangle \boldsymbol{c}_\nu$. Also, $\boldsymbol{w}\boldsymbol{C} = \sum_\nu \alpha_\nu \boldsymbol{c}_\nu \boldsymbol{C}$. Because $\boldsymbol{c}_\nu$ is an eigenvector of $\boldsymbol{C}$, we must have $\boldsymbol{c}_\nu \boldsymbol{C} = \boldsymbol{c}_\nu \lambda_\nu$, where $\lambda_\nu$ is the eigenvalue of the eigenvector $\boldsymbol{c}_\nu$, so $\boldsymbol{w}\boldsymbol{C} = \sum_\nu \alpha_\nu \boldsymbol{c}_\nu \lambda_\nu$.

Hence $\langle \Delta \boldsymbol{w} \rangle = \mathbf{w}\mathbf{C}$ becomes $\sum_\nu \langle \Delta \alpha_\nu \rangle \boldsymbol{c}_\nu = \sum_\nu \alpha_\nu \boldsymbol{c}_\nu \lambda_\nu$. Because the $\boldsymbol{c}_\nu$ are orthogonal, we can equate their coefficients to get $\langle \Delta \alpha_\nu \rangle = \alpha_\nu \lambda_\nu$.

Clearly, if all the $\alpha$'s are initially of roughly equal size, then the $\alpha$ for the maximal eigenvector will increase the fastest, because its $\lambda_\nu$ will be the largest. If some sort of weight limitation procedure is used, then the weight will not increase in length indefinitely, but its direction will still tend towards that of the maximal eigenvector.

There are many ways of keeping the components of the weight vector bounded, such as dividing all components by the length of the vector after each training step which will keep $|w| = 1$, though this is clearly non-local with respect to individual synapses.

A different way of generalising the Hebb rule to keep the weights bounded is the Oja rule (Oja [1982]), which keeps $|\mathbf{w}_i| = 1$ for linear units, and has the attractive property that it doesn't involve any summing over all the weights on a neuron: each synapse update is dependent on only that synapse and the activation of the unit, and not dependent on any other synapse strength. This rule is: $\Delta w_a = BA_a - B^2 w_a$, or in matrix notation, $\Delta w = BA^T - B^2 w$.

This Oja rule with a single unit will always find the first principal component of the input distribution, regardless of the initial weight values, and will ensure that $|w| = 1$:

As before, the state of the unit is $B = wA$. The learning rule is $\Delta w = BA^T - B^2 w$, so to find the average weight change for all input patterns, $\langle \Delta w \rangle$, we need to know $\langle BA^T \rangle$ and $\langle B^2 \rangle$.

For $\langle BA^T \rangle$, we substitute $B = wA$ to get $\langle BA^T \rangle = \langle wAA^T \rangle = w \langle AA^T \rangle = wC$, where $C$ is the correlation matrix for the set of input patterns.

Similarly, for $\langle B^2 w \rangle$ we get:$\langle B^2 w \rangle = \langle (wA)(wA)w \rangle = \langle (\sum_{a\hat{a}} w_a A_a w_{\hat{a}} A_{\hat{a}}) w \rangle$
$= \sum_{a\hat{a}} w_a \langle A_a A_{\hat{a}} \rangle w_{\hat{a}} w = (\sum_{a\hat{a}} w_a C_{a\hat{a}} w_{\hat{a}}) w = (wCw^T) w$.

Hence the average weight change is $\langle \Delta w \rangle = wC - (wCw^T) w$

For the weight vector to be stable, we must have $\langle \Delta w \rangle = 0$, so $wC = (wCw^T) w$ at equilibrium. The R.H.S. is a scalar times the weight vector $w$, so this equation says that any stable weight vector must be an eigenvector of $C$, the covariance matrix of the input distribution.

Given that $\omega$ is an eigenvector of $C$, with some eigenvalue $\lambda$, we must have $wC = \lambda w$ and $wCw^T = \lambda w w^T = \lambda |w^2|$. If we substitute these back into the above expression that says that, at equilibrium, $wC = (wCw^T) w$, we get $\lambda w = \lambda |w^2| w$, so $|w^2| = 1$ (assuming $w \neq \mathbf{0}$). Hence the Oja rule makes the length of the weight vector be unity at equilibrium.

We can also investigate how the weight vector changes in terms of its components along the eigenvectors of $C$, and show that the only stable equilibrium is when the weight vector is the maximal eigenvector: As before, we shall express $w$ as a linear combination of all the unit eigenvectors of $C$, $w = \sum_\alpha a_\alpha c_\alpha$, where the eigenvector $c_\alpha$ has eigenvalue $\lambda_\alpha$. The

expression for the average change in the weight vector becomes:

$$\begin{aligned}\langle \Delta w \rangle &= wC - (wCw^T)w \\ &= w(C - (wCw^T)) \\ &= \sum_\alpha \alpha_\alpha c_\alpha \left( C - \left[ \left( \sum_{\hat{\alpha}} \alpha_{\hat{\alpha}} c_{\hat{\alpha}} \right) C \left( \sum_{\hat{\alpha}} \alpha_{\hat{\alpha}} c_{\hat{\alpha}}^T \right) \right] \right)\end{aligned}$$

The term in square brackets is zero apart from when $\hat{a} = \hat{\hat{a}}$, as eigenvectors are orthogonal. A

$$\begin{aligned}\langle \Delta w \rangle &= \sum_\alpha \alpha_\alpha c_\alpha \left( C - \sum_{\hat{a}} \alpha_{\hat{a}} c_{\hat{a}} C c_{\hat{a}}^T \alpha_{\hat{a}} \right) \\ &= \sum_\alpha \alpha_\alpha c_\alpha \left( \lambda_a - \sum_{\hat{a}} \alpha_{\hat{a}}^2 \lambda_{\hat{a}} \right)\end{aligned}$$

$$\text{or equivalently,} \quad \langle \Delta \alpha_a \rangle = \alpha_\alpha \left( \lambda_a - \sum_{\hat{a}} \alpha_{\hat{a}}^2 \lambda_{\hat{a}} \right)$$

The second term on the R.H.S. is the same for all $\alpha_a$, so the component of $w$ along the eigenvector with greatest $\lambda_a$ will increase fastest.

If the weight vector has already reached an equilibrium, then only one of the $\alpha$'s will be 1, say $\alpha_{\hat{a}}$ and the rest zero because the weight vector will be an eigenvector and will have unit length. Hence the above expression will be $\langle \Delta \alpha_a \rangle = \alpha_a (\lambda_a - \lambda_{\hat{a}})$. This means that any perturbation of the weight vector along an eigenvector which has a higher eigenvalue than $\alpha_{\hat{a}}$ will be unstable. This means that the only stable equilibrium is when the weight vector is along the maximal eigenvector with eigenvalue $\lambda_{max}$, as no other eigenvector has a higher eigenvalue. This also corresponds to the principal component of the input data.

It is possible to make an unsupervised net with two units which respond to the first and second eigenvectors of the input data set. This is done by using two linear units training with a Hebb rule. Normally these would each tend to the maximal eigenvector. To prevent this, we can arrange for a lateral connection $u$ from the output of unit 1 to the input of unit 2, which learns in an *anti*-Hebbian way, i.e. the strength of the weight will *decrease* if the two units are on together. Unit 1 will detect the first principal component as usual, as its inputs are identical to the normal principal component detecting case. To find what unit 2 will do, we just note that the lateral connection $u$ will change until the two units are uncorrelated. Because the normal weights will try to converge to the highest-eigenvalue eigenvector, the unit will eventually find the second principal components. Also, the lateral connection will end up at zero.

A fuller analysis is rather tedious, but is given here for completeness:

The states of the two units are given by: $B_1 = w_1A$, and $B_2 = w_2A + uB_1$. We can expand the expression for $B_2$ to get $B_2 = w_2A + uw_1A$.

If we assume for the moment that the lateral connection $u$ changes much more quickly than the normal weights, we find that $u$ as a stable equilibrium at $u = -\frac{w_1Cw_2}{w_1Cw_1}$: We have $\langle \Delta u \rangle = -\langle B_1B_2 \rangle = -\langle w_1A(w_2A + uw_1A) \rangle$. This is $\langle \Delta u \rangle = -w_1Cw_2^T - uw_1Cw_1^T$. Because $w_1Cw_1^T$ is positive, this equilibrium is stable.

Hence $u$ being at equilibrium means that $w_1Cw_2^T + uw_1Cw_1^T = 0$, or $u = -\frac{w_1Cw_2^T}{w_1Cw_1^T}$.

We can now find the average change in $w_2$: $\langle \Delta w_2 \rangle = \langle B_2A^T - {B_2}^2w_2 \rangle$. We can substitute the expression found earlier, $B_2 = w_2A + uw_1A$, to get:

The expression for $u$ found earlier was $u = -\frac{w_1Cw_2^T}{w_1Cw_1^T}$. Substituting this gives:

$$\begin{aligned}
\langle \Delta w_2 \rangle &= \langle (w_2A + uw_1A)A^T - (w_2A + uw_1A)^2 w_2 \rangle \\
&= w_2 \langle AA^T \rangle + uw_1 \langle AA^T \rangle \\
&\quad - (w_2 \langle AA^T \rangle w_2^T) w_2 - u^2 (w_1 \langle AA^T \rangle w_1^T) w_2 - 2u (w_2 \langle AA^T \rangle w_1^T) w_2
\end{aligned}$$

But $\langle AA^T \rangle = C$, so:

$$\langle \Delta w_2 \rangle = w_2C + uw_1C - (w_2Cw_2^T) w_2 - u^2 (w_1Cw_1^T) w_2 - 2u (w_2Cw_1^T) w_2$$

The expression for $u$ found earlier was $u = -\frac{w_1Cw_2^T}{w_1Cw_1^T}$. Substituting this gives:

$$\begin{aligned}
\langle \Delta w_2 \rangle &= w_2C - (w_2Cw_2^T) w_2 + u \{ w_1C + (w_1Cw_2^T) - 2 (w_2Cw_1^T) \} w_2 \\
&= \langle (w_2A + uw_1A)A^T - (w_2A + uw_1A)^2 w_2 \rangle \\
&= w_2 \langle AA^T \rangle + uw_1 \langle AA^T \rangle \\
&\quad - (w_2 \langle AA^T \rangle w_2^T) w_2 - u^2 (w_1 \langle AA^T \rangle w_1^T) w_2 - 2u (w_2 \langle AA^T \rangle w_1^T) w_2 \\
&= w_2C + uw_1C - (w_2Cw_2^T) w_2 - u^2 (w_1Cw_1^T) w_2 - 2u (w_2Cw_1^T) w_2 \\
&= w_2C - (w_2Cw_2^T) w_2 + u (w_1C - u (w_1Cw_1^T) w_2 - 2 (w_2Cw_1^T) w_2) \\
&= w_2C - (w_2Cw_2^T) w_2 + u [w_1C + (w_1Cw_2^T) w_2 - 2 (w_2Cw_1^T) w_2]
\end{aligned}$$

because $u = -\frac{w_1Cw_2^T}{w_1Cw_1^T} = \frac{w_2Cw_1^T}{w_1Cw_1^T}$

$$= w_2C - (w_2Cw_2^T) w_2 - \frac{w_1Cw_2^T}{w_1Cw_1^T} [w_1C - (w_1Cw_2^T) w_2]$$

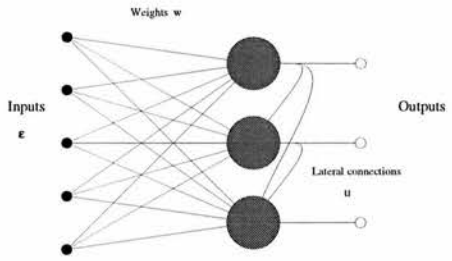
The first two terms are the normal Oja learning rule, while the last is easier to understand if

we substitute in the fact that $\boldsymbol{w}_1$ will converge to the first principal component of $\boldsymbol{C}$, in which case $\boldsymbol{w}_1\boldsymbol{C} = \lambda_{max}\boldsymbol{c}_{max} = \lambda_{max}\boldsymbol{w}_1$, so $\boldsymbol{w}_1\boldsymbol{C}\boldsymbol{w}_2^T = \lambda_{max}\boldsymbol{w}_1\boldsymbol{w}_2^T$ and $\boldsymbol{w}_1\boldsymbol{C}\boldsymbol{w}_1^T = \lambda_{max}$. This makes the third term: $-\boldsymbol{w}_1\boldsymbol{w}_2^T\left[\boldsymbol{w}_1\lambda_{max} - \left(\lambda_{max}\boldsymbol{w}_1\boldsymbol{w}_2^T\right)\boldsymbol{w}_2\right] = -\lambda_{max}\boldsymbol{w}_1\boldsymbol{w}_2^T\left(\boldsymbol{w}_1 - \left(\boldsymbol{w}_1\boldsymbol{w}_2^T\right)\boldsymbol{w}_2\right)$. This is a vector which is perpendicular to the current vector $\boldsymbol{w}_2$ when $\boldsymbol{w}_2$ has reached unit length, and is directed away from the first principal component. This third term is zero when $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are perpendicular.

Also, when the network has stabilised, it turns out that the lateral connections decay to zero. This can be proved mathematically by solving the simultaneous equations for the two units. The lateral connection is still needed however, as it acts to reverse any small random change in the input weights which makes the units become slightly correlated again.

Another way of looking at the network is that the second unit receives an input equivalent to a simple unit with no lateral connections, but with the first principal component removed from its inputs. This is because after the first unit has found the first principal component, its weight vector will be $\boldsymbol{w}_1 = \boldsymbol{c}_{max}$, so the inhibitory weight from unit 1 to unit 2 will be $u = -\frac{\boldsymbol{w}_1\boldsymbol{C}\boldsymbol{w}_2^T}{\boldsymbol{w}_1\boldsymbol{C}\boldsymbol{w}_1^T} = -\frac{\lambda_{max}\boldsymbol{c}_{max}\boldsymbol{w}_2^T}{\lambda_{max}\boldsymbol{c}_{max}\boldsymbol{c}_{max}^T} = -\boldsymbol{c}_{max}\boldsymbol{w}_2^T$, which is minus the component of $\boldsymbol{w}_2$ along the first principal component. The 'input' to this lateral weight will be the activation of unit 1, which is just the magnitude of the input vector along the first principal component, so this will cancel exactly the component of the input vector along the first principal component, hence it will find the second principal component.

The anti-Hebbian lateral connection scheme can be extended to have any number of units, for example (Sanger [1989]):




***A network which detects the first three Principal Components of its input distribution***

From a biological point of view, the structure of these networks is rather contrived because of the unsymmetrical arrangement of the lateral connections. More plausible would be to have anti-Hebbian lateral connections between all pairs of units (Oja [1989]). This arrangement is described and modelled in chapter 2, and the $m$-unit network learns to span

the $m$-dimensional sub-space of the original $n$-dimensional inputs. Unfortunately, real modelling of units with this pattern of lateral connections is complicated because the network is recurrent, so that the network can take many iterations before the activations are stable. Moreover, there is evidence that the output axons from neurons do not form both excitatory and inhibitory connections, so the lateral connections in the principal component model would have to be mediated by extra neurons (e.g. Plumbley [1991]). See chapter 2 for the details of some simulations of this type of network.

Despite these implementational problems, P.C.A. is an attractive algorithm for a network, because it performs a form of *data compression*. The most variable parts of the input data will often be the most informative from an informational-theory point of view[1], so the simple network can be thought of as extracting the most important parts of the input data. This idea will be further developed in chapter 4.

#### Földiák's use of anti-Hebbian learning to provide de-correlation (Földiák [1990])

This is similar to principal component networks, in that it uses simple units with lateral connections. The main differences are that the units are non-linear, and the units have adjustable thresholds. The network is designed so that it learns to represent input data in a particular way:

Földiák describes two extreme ways in which a network can encode information:

- A fully distributed representation can encode many patterns in a few units, but it is hard for a subsequent layer to make use of this information.
- A grandmother representation is very easy to make use of, but is wasteful of resources.

The best way is somewhere between these two extremes - *sparse coding*. Földiák describes a network that uses non-linear units with anti-Hebbian lateral connections. The parameters of the model which control the threshold of the units are arranged so that most units will be off for any input pattern. This makes the network perform sparse coding. The particular compromise between grandmother and fully-distributed representation is determined by the average number of units which are allowed to fire for each pattern.

[1] The precise definition of the most informationally useful part of the input data depends on the sort of noise present and the form of the input data. If both of these are gaussian, then representing the data in terms of the principal components is optimal.

The network units develop into feature-detectors, and the network can develop units which respond to commonly occurring patterns in the input data. Of particular interest is that the output for frequently occurring input patterns is sparser than the output for infrequent input patterns. This would make for easier recognition of commonly occurring patterns by a subsequent layer, and also means that the output of the network is efficient from an information theory point of view.

Földiák uses a differential state update in order to model the network, although doesn't discuss this. Instead of the usual equation for finding the new state of unit $i$ with a non-linear response function $f(x)$:

$$\boldsymbol{B}_i = f(\boldsymbol{w}_i\boldsymbol{A} + \boldsymbol{u}_i\boldsymbol{B} - threshold_i),$$

Földiák uses a differential equation of the form:

$$\Delta\boldsymbol{B}_i = f(\boldsymbol{w}_i\boldsymbol{A} + \boldsymbol{u}_i\boldsymbol{B} - threshold_i) - \boldsymbol{B}_i$$

Note the last term. This is designed so that when stability is reached ($\Delta\boldsymbol{B}_i = 0$), the state of the unit will equal the total input, so that stability corresponds to a network with normal units (i.e. without differential state update) at stability, even though such a network would not always reach this state because of instability problems.

### Networks with connectivities less than 100%

All the networks considered so far, except for the Linsker network, have had 100% connectivity, i.e. each unit receives input from every unit in the previous layer (or from every input to the network for a one-layer network). It is curious that very little work has been done on networks which are not fully connected, as the brain certainly hasn't got 100% connectivity.

It is intuitively clear, and also follows in a particularly direct way from information theory (see chapter 4), that in order to be useful, units should respond differently from each other, otherwise one might as well use fewer units. This is done in principal components networks by the use of lateral connections; back propagation networks solve the same problem by sending error signals to different units which tend to amplify any small initial variations in each unit's weight vectors.

However, if a network has limited connectivity this problem is, at least partially, solved automatically. One idea for further work following on from this thesis is to investigate what effect limited connectivity has on the problem of making networks transform data into useful representations.

From the point of view of the later work in this thesis about processing visual data, Linsker's generation of interesting receptive fields from completely random data is, at first glance, problematic, because the information theory approach only works when the input data has significant structure. However, it turns out that the first layer in Linsker's network is actually adding structure to the (structureless) input data because of the limited receptive fields, and this structure is very similar to that assumed in chapter 6 as a first approximation of real visual data.

This follows fairly clearly: First, the input units receive completely random data and so develop uniform weights. Secondly, the receptive fields of nearby second-layer units overlap. This means that the activations of nearby second-layer units will be correlated. Hence the third layer will be receiving structured data, and so can develop the interesting weight patterns described earlier.

# Chapter 2 The problems of implementing neural networks that use lateral connections

Lateral connections are usually used in neural networks to inhibit units when other units are more strongly activated. Thus they encourage units to develop different weight vectors, and so ensure that units behave in different ways even if, as is common in neural network simulations, they are connected to identical inputs.

Here, I will illustrate some of the problems which arise when lateral inhibition is used in neural network simulations. I will first use principal components networks as examples, because they are one of the simplest unsupervised network types. Then I will give a description of problems which I encountered when trying to make a neural network develop 'feature detectors', of a kind used in my model of reading aloud (described in chapter 3).

### Principal Components networks

Neural networks which find the principal components of their input data were briefly described in an earlier chapter. They are one of the simplest examples of unsupervised networks which use lateral connections. Here, I will give some examples of simulations of such networks in action, to demonstrate some of the problems involved in simulating such networks.

In all of the following diagrams the input patterns, which have varying numbers of dimensions, are represented as dots on a 2D graph. The weight vectors of the units are represented as vectors from the origin of the same graph, and the evolution of the weight vectors is also drawn so that one can see how the weights develop.

The inputs to the networks are identical in each case, and are generated from random numbers in such a way that the first principal component of the data is horizontal, the second vertical, and any others are naturally perpendicular to the first two, so cannot be easily shown. That the first principal component is along the $x$-axis can be seen by noticing that the density of input patterns at the extremes of the $x$-axis is high. The maximum length of the input vectors was 1; in the diagrams, the inputs are shown to a slightly smaller scale than the weight vectors for clarity.

This input method was chosen as it is easy to generate quickly, and has well defined

principal components. In general, the $n^{th}$ principal component of the distribution is a vector whose components, expressed in terms of the $xyz$ axis, are all zero except for the $n^{th}$ component.

Because the inputs were generated using random numbers rather than choosing an input from a training set, each iteration was the presentation of just one pattern.

All units in the networks discussed in this chapter are linear, and their learning rule is the Oja learning rule, described in chapter 1, which is (ignoring the learnrate):

$$\delta w_{ba} = A_a B_b - B_b^2 w_{ba}$$

Here, $A_a$ is input number $a$, $B_b$ is the state of unit $b$, and $w_{ba}$ is the strength of the connection from input number $a$ to unit $b$.

First, a single-unit network. The weight vector can be seen to quickly find the first principal component of the input data:

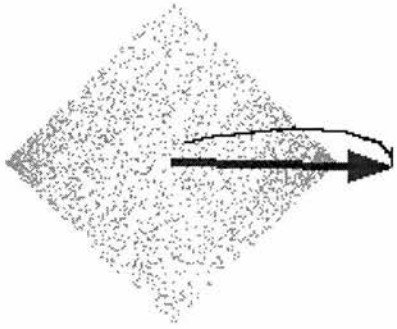
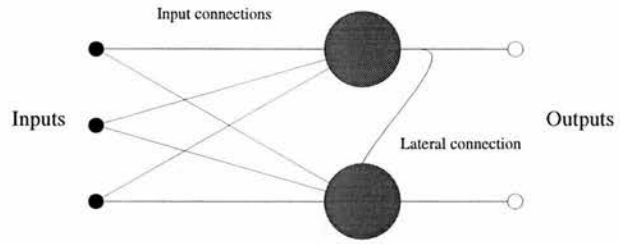


***Figure 1***
***Single-unit P.C.A network. 2500 iterations learnrate=0.01***

The next diagram is for a two-unit network, which has just one lateral connection from unit 1 to unit 2, (i.e. unit 2 is inhibited by unit 1, but unit 1 is not affected by unit 2). This connection learns using an anti-Hebbian rule, as described earlier. This network looks like:




The weights quickly align themselves with the first two principal components, and it turns out that the lateral connection strength ends up at 0 for this network. Hence the two units end up uncorrelated, and the asymmetric connections mean that finding the state of the network doesn't require any iterating, if one calculates the state of unit 1 before that of unit 2.

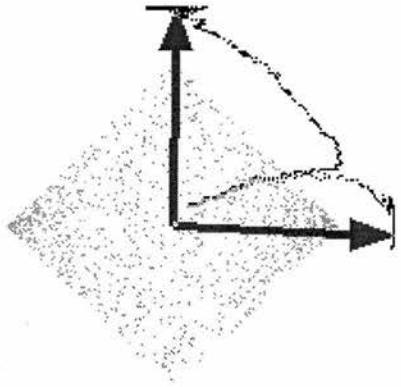
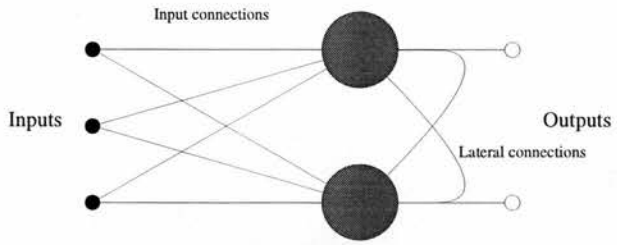


***Figure 2 Feed-forward weights of two-unit asymmetric P.C.A network. 1000 iterations learnrate=0.1 lateral learnrate=0.2***

Hence there is no problem with the time taken for the network activations to stabilise for each input pattern.

However, strictly one-way connections between units are not likely to occur in the brain, and the units above are activated very unequally. For example, the first unit, which responds to the first principal component, has a higher R.M.S. activation than any others, the second unit, which responds to the second principal component, has a higher R.M.S. activation than all but the first unit, etc.

A network with anti-Hebbian connections between all pairs of units can be constructed, which looks like:




The learning process for this network, when presented with the same input distribution, is illustrated next:

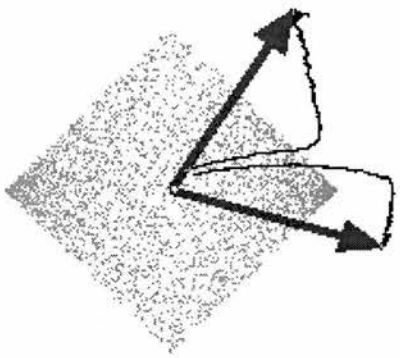


***Figure 3***
***Two-unit symmetric P.C.A network.***
***3300 iterations***
***learnrate=0.01***
***lateral learnrate=0.02***

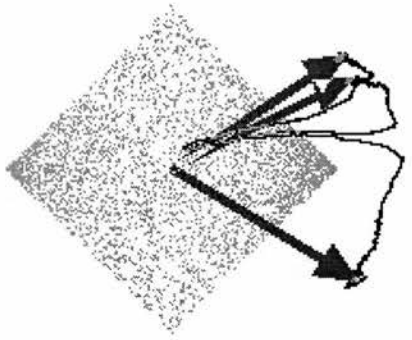
The lateral connections in the above simulation were both –0.314. The variances of the two units were different though: 0.132 and 0.288 for the unit with the upper and lower vector respectively. These variances depend on the starting point of the vectors; in some circumstances it would be useful if the two units would share the variance equally, but this doesn't seem to happen.

N.B. the reason why the lateral connections keep the units uncorrelated is that the anti-Hebbian learning rule for the lateral connection from unit $b$ to unit $\hat{b}$ is $\Delta u_{b\hat{b}} = -B_b B_{\hat{b}}$. Hence the average change in the strength of this lateral connection is $\langle \Delta u_{b\hat{b}} \rangle = -\langle B_b B_{\hat{b}} \rangle$. This means that the lateral connection will be at equilibrium when the two units at either end are uncorrelated. The minus sign makes this a stable equilibrium.

The fact that the lateral connections don't decay to zero in the symmetrical case is rather problematic, as it means that the network is always going to take some time to stabilise. In

the last example, an average of 2.4 iterations were required to stabilise the network, where each iteration updated both units in a random order (The criteria for the network being stable was that the average percentage change in the states was less than 10%). While this actually means that only 0.4 of an extra iteration was needed, things get very much worse when there are more units.

For example, the next diagram is for three units:



***Three-unit symmetric P.C.A network. 3800 iterations learnrate=0.01 lateral learnrate=0.02***

The weights for this network have significant components perpendicular to the page, but they are still very similar, and so the lateral connections needed to make the units uncorrelated are strong. This means that more iterations are required before the network stabilises - an average of 5.4 iterations.

For the record, and to give an idea of the magnitudes involved, the input weights for this network after training were:

| | Unit 1 | Unit 2 | Unit 3 |
|---|---|---|---|
| Input 1 | 0.821 | 0.847 | 0.786 |
| Input 2 | 0.417 | -0.517 | 0.523 |
| Input 3 | 0.38 | -0.0507 | -0.339 |
| Input 4 | -0.00126 | 0.0022 | 0.00698 |

The input data was constructed so that the $n^{th}$ principal components of the input data was that vector which has all components zero except for the $n^{th}$. As can be seen, all units have a very small weight to the fourth input, so the network is only sensitive to the sub-space formed by the first three principal components.

The lateral connection strengths were:

| | Unit 1 | Unit 2 | Unit 3 |
|---|---|---|---|
| Unit 1 | | -0.462 | -0.748 |
| Unit 2 | -0.462 | | -0.42 |
| Unit 3 | -0.748 | -0.42 | |

and the correlations between units were:

| | Unit 1 | Unit 2 | Unit 3 |
|---|---|---|---|
| Unit 1 | 0.13 | -0.0037 | -0.00046 |
| Unit 2 | -0.0037 | 0.22 | -0.00071 |
| Unit 3 | -0.00046 | -0.00071 | 0.114 |

In this last table, the diagonal elements show the variances of the units.

A five-unit network needed approximately 8.3 iterations before it gave a stable output for each input pattern.

Subjects in psychological experiments have reaction times of a few tenths of a second which, given the slow speed of neurons, doesn't allow much time for gradually approaching a stable set of neuron firing levels. This suggests that there is a very real problem with using networks, like these P.C.A. networks, which have strong lateral connections (Marr [1972], Sejnowski [1986]).

### Feature-detecting networks

For the model of reading aloud described in chapter 3 (which, although it uses simple feature-detectors, doesn't make use of *neural-network* feature-detectors), I was interested in making a neural network in which each unit would develop a weight vector which was

tuned to the presence or absence of particular groups of letters in the visual field. The networks tried never worked particularly well, so I ended up looking at the more general problem of how data should be processed, in particular using information theory, as described in chapters 4-6. However, there follows a description of the networks that I tried in order to make feature detectors:

I wanted the outputs of the network to be 'digital' - each unit either on or off depending on whether the unit's chosen group of letters was present. Because of this requirement, I used a variation of the differential state update rule used by Földiák [1990], $\frac{dB_b}{dt} = \sum_a w_{ba} A_a + \sum_{\hat{b}} u_{b\hat{b}} B_{\hat{b}} - B_b$.

The modification used was to omit the last term, resulting in a differential state update rule $\frac{dB_b}{dt} = \sum_a w_{ba} A_a + \sum_{\hat{b}} u_{b\hat{b}} B_{\hat{b}}$.

Also, the response function of the units was chosen so that the outputs were clipped between zero and one, otherwise the states could increase indefinitely. Like Földiák's network, each unit had a threshold which changed so that the unit's average activation was somewhere between a minimum and maximum allowed value (there has to be a range for the average activation so that the units can learn to detect features which have various frequencies).

The network wasn't particularly good at functioning as a feature detector, and certainly performed too poorly to be used in the reading model. Nevertheless, it exposed some of the pitfalls of simulating networks with lateral connections.

Various state update strategies were tried in an attempt to make the network reach a stable set of activations a quickly as possible, but even for a small network with 10 units and 104 inputs, iterating the state update equation took 10-20 iterations, with several 'intelligent' tweaks such as changing the time step to be as large as possible while keeping the state-changes reasonably small. The input data was the first four letters of a list of words, encoded by having 26 input units for each letter position, and turning on the four input units which correspond to the input word.

There were problems with using both batch state update and random state update. Batch update gave oscillations, while random state update, although it reached a stable set of activations, could sometimes make the network respond differently to identical input patterns, because the first unit to be chosen to be updated would inhibit other units, and so stand a better chance of being on after the network had stabilised.

It was hoped that using an adjustable threshold would make units learn to respond to a highly specific input pattern, and so develop a set of weights which could be described as 'digital', i.e. each weight either very strong or very weak. This happened only to a limited extent though. It was also found that even if the input weights were fixed (e.g. zero learnrate), the units could still decorrelate themselves and maintain a reasonable average activation by changing just the lateral connections and the thresholds.

# Chapter 3 Using unsupervised feature-detectors in a model of reading aloud[2]

## Overview

In this chapter, a model of reading is described that uses simple unsupervised feature-detectors (not neural network-based) and a very rudimentary model of eye movements to learn how to convert input words into phonemes. The model will function as a table lookup where possible; otherwise it will piece together a pronunciation for the input word. The learning process has been deliberately chosen to be a statistical processing of the training database (much of which could be implemented as simple unsupervised neural networks) with no explicit rule-based learning. This does mean that the performance of the model is relatively poor compared to back-propagation neural network models of reading aloud (Rumelhart & McClelland [1986], Sejnowski & Rosenberg [1987], Seidenberg & McClelland [1989], Bullinaria [1994][1995][1996]), but it has the advantage that it can deal with any length of word, and it depends only on the simple statistics of the training set of words. Also, with specific teaching it is expected that the performance will improve markedly.

The model is similar to Analogy-based models such as Dedina & Nusbaum [1986][1991], Sullivan & Damper [1990][1991][1992]. Pronouncing by analogy is also discussed in Glushko [1979][1981], while Damper [1995] gives a general review of self-learning models of reading aloud.

The model can also learn to perform the reverse task of converting input sounds into words.

## Introduction

The relationships between letters/words and their pronunciations (usually in terms of phonemes) has been called a 'quasi-regular system' (Seidenberg & McClelland [1989]) because although there are well defined rules, there are also many different exception words, the pronunciations of which are completely unrelated to any other words. An example of a rule which normally applies is the pronunciation of the letter 'a' in words such as 'apple' and 'plan'; this rule is sometimes overridden by the more complicated 'rule of "e" '

[2] This chapter is based on my paper of the same title published in the Irish Journal of Psychology, Smith [1993].

in words such as 'ate' and 'plane'. There are many more examples of these two rules in action (which is why they are given the status of rules in the first place). However, the pronunciation of the first syllable of 'colonel' does not follow rules that are applicable to any other words; similarly for 'yacht'. In between these extremes is a range of rules which are often applicable, but also frequently broken, with the result that reading is a very difficult task for a potential computational model.

Assuming that the brain is some sort of vast neural network, and ignoring the usual problems of not knowing exactly what a neuron actually does etc., the problem is how to build a trainable model of reading which uses neural-network units. It would obviously be desirable for a model of reading to learn how to convert text to phonemes by being presented with examples of correct mappings, as that is how people learn to read (albeit with some specialised instruction); this is just the sort of thing that one would expect a neural network to be capable of. However, the only way to get a neural network to do anything as complicated as reading is to use a training method like back propagation, which has well known (though disputed) problems concerning biological plausibility.

Neural networks training algorithms which don't use back propagation do exist, but are only capable of performing simple data transformations such as principal component analysis or feature detection. Also, Hebbian learning can be viewed as making units function as coincidence detectors. While such transformations on their own couldn't perform a complicated task such as reading, they can be used to make the reading task easier in the same way as having edge detectors makes visual recognition tasks easier. The model described here has feature detectors that are trained to respond to commonly occurring groups of letters such as '`ing`' or '`th`', and similarly for commonly occurring groups of phonemes. It is hypothesized that such feature detectors could be part of the visual and auditory/vocal systems as one would expect the visual system to automatically build up such a list of commonly seen patterns, and the auditory system to learn to recognise commonly heard sounds (this process could perhaps be the result of the information-theoretic generation of compressed representations, discussed in later chapters).

### The input representation in neural network models

The way the input word is presented to a conventional neural network model is important. If the letters are presented all at once (e.g. by having a set of 26 units for each letter position),

the network will have no indication that a particular letter represents the same symbol irrespective of where it is in the word. On the other hand, presenting the letters one by one (maybe in a moving window, as in NetTalk, Sejnowski & Rosenberg [1987]) is problematic because there are usually fewer phonemes than letters in words (an extreme example is 'eight' which has five letters but only 2 phonemes). This would mean that the network would have to learn to wait until it had seen enough letters before outputing a phoneme. NetTalk avoided this issue by having a pre-processor which added blanks to the sound part of the input, so that the text and sound were always aligned. Seidenberg & McClelland [1989] used a different approach, in which the whole word was presented to the network in one go, but using a distributed encoding of the presence or absence of Wickelfeatures (three adjacent letters / phonetic features) in order to encode the largely position-invariant nature of letter pronunciations. Unfortunately, the use of this scheme meant that the output of Seidenberg & McClelland's network was very difficult to convert into a list of phonemes, because it was in such a distributed representation. In fact, to find out what the output of their network actually was, Seidenberg & McClelland were forced to work backwards; they first calculated what the network output would be for all plausible output sounds, and then chose the one that was closest to the actual output.

Another criticism of the Seidenberg & McClelland model is that although it performs very well on the 2998-word data set it was trained on, it would not be easy to extend the model to multi-syllable words, as a Wickelfeature description of a long word is more likely to be ambiguous and difficult to decode.

### Evidence from eye movements

The above models all assume some sort of fixed method of presenting the input to the model, either in a moving window or as Wickelfeatures. However, the input data about the text being read comes originally from the eyes, so it would make sense to look at how the eyes take in the text.

There is evidence that sometimes a word is fixated on more than once, and that two short words are sometimes read with just one fixation. For example R. E. Morrison [1983], in an investigation into fixation positions for different viewing distances, (where he found that fixation points in words were unaffected by viewing distance), gave the following figure with fixation points of a subject together with the text the subject was reading:

```
The crazed boy was locked into a cage
.   .  .      .       .      .   .

The unruly guy was heaved into a cell
.    .  .     .       .      .    .
```

(the two sentences were at different distances from the subject, but that is not relevant here.)

This shows that that the input to the text-to-phoneme part of the brain is sometimes already roughly segmented into usefully sized pieces that are smaller than whole words. Of course, the eye movements that cause this segmentation must be at least partially controlled by the text to phoneme processing module, but this doesn't mean that one should expect a neural network model to do this automatically - such a mechanism would be almost impossible to develop from scratch, so a model would have to develop a different way of solving the problem. The fact that eye movements are potentially capable of segmenting words in a useful way, and appear to be used sometimes within a single word, means that they are important to a model of reading. Hence ignoring the existence of eye movements constrains the reading model in an unsatisfactory way, for example Seidenberg & McClelland's use of distributed Wickelfeatures. NetTalk's moving window is also unsatisfactory, because people's eyes don't progress smoothly from letter to letter of the word they are reading.

If eye movements are used as the main method of splitting up words, this would be good news for neural network models in general because these models on their own are completely unsuited to problems that involve looking at different parts of input data in a serial manner. A system that has available the facility to control (via muscular control) the position of attention is more appropriate to neural networks.

Of course, eye movements don't perform segmentation completely, as the information from the eyes is not cut off at sharp end points. The important point is that they allow the use of position-specific feature detectors (e.g. simple unsupervised neural networks) instead of much more complicated position-independent feature detectors.

It should be pointed out that eye movement data is, of course, very complicated. As a result, the model described below doesn't claim to be a model of eye movements, but rather to incorporate, and make use of, the existence of eye movements in a model of reading.

### General description of the model

Once some method of segmentation has been found for the model, the problem remains of what to do with the parts of words that form the input to the model. The method chosen here is probably the simplest possible: build up a list of known parts of words (referred to from now on as **text features**), and then find the best pronunciation for each of these text features by finding the string of phonemes (referred to from now on as **sound features**) that most often occurs in the same words. The finding of the best pronunciation for each text feature is done in a very simple statistical way; it is described in the next section.

When required to pronounce a word, the model simply parses the word into known text features, and then outputs the pronunciations of these text features in the same order as they are found in the word. The model attempts to parse the input word into the longest known text features possible, so that for instance 'th' would not be split up if it was recognised as a single text feature. Thus the model is effectively a table lookup, but at the sub-word level.

If an input word is matched by a single text feature, then the model will function as a normal table lookup, and so will be able to cope with exception words such as 'yacht'. Conversely, if a word as a whole is unknown, it will be split into known patterns of letters (text features) which the model would hopefully deal with correctly.

The model possesses similarities to that by Coltheart et al. [1992]; in their model, single letter to single phoneme matches are developed, but if a word contains more letters than phonemes, it attempts to map single letters to pairs of phonemes. Other more advanced rules are developed, for example to deal with cases in which a letter is often pronounced in more than one way. While their model certainly performs very well, it uses very high-level and complex decision processes in learning, which is a serious drawback.

### Learning

This is a two-stage process:

- Two lists are developed, one containing common text features, the other containing common sound features. The training data set consists of a list of words together with the phonemes that form their pronunciations (for example the 2998-word Seidenberg & McClelland set). Analysis of the text part of this

data set will result in 'ing' being noted as a commonly occurring text-feature, along with 'the' and 'er' etc. At the same time, common strings of phonemes such as '^mp', and 'pE' will be detected in the sound data (N.B. all phoneme symbols are as used in Seidenberg & McClelland [1989]). Obviously, a three-letter string will always be less common than any two-letter string that it contains, so the model makes sure that longer strings are found by searching for the 200 (say) most common strings of each length; i.e. the 200 most common three-letter strings, together with the 200 most common two-letter strings, and the 200 most common single-letter strings (though it will only find 26 single-letter strings). The methods used for finding text features and sound features are identical.

- Each text feature is matched with a sound feature, which is henceforth used as its pronunciation. This is done by looking for pairs of text/sound features that occur most often in the same word. For instance, the text feature 'ing' and the sound feature 'iN' will nearly always occur in the same word e.g. 'beginning', 'knowing' or 'wing', (though not in 'Leningrad'). For each text feature, a score is given to each of the sound features, the score for each sound feature being the number of words in the database that contain both the sound feature and the text feature. These scores are weighted by the length of the sound feature (i.e. multiplied by the number of phonemes in the sound feature), because otherwise short sound features would always have higher scores.

The particular method used here to find the pronunciations of the text features (i.e. using a list of common sound features) was chosen partly because it is computationly efficient, but it also has the advantage that a lot of the work (the finding of text and sound features) is done without reference to the correlations between text and sounds in the data set, i.e. before explicit learning-to-read-aloud occurs. This would correspond, for example, to a child learning to recognise and pronounce certain sounds before he/she ever comes into contact with text, or to the child learning about text without any concurrent speech input.

### Reading words

When presented with text to be converted into sound, the model looks for the presence of previously learnt text features, starting with the longest known features. The input word is

scanned from its first letter for the presence of a recognised group of letters. When part of the word has been recognised as a feature, its position is noted, and then that part of the word is replaced by blanks, so that the letters can't be noted twice. The word is then re-scanned for the presence of other features, until all the letters have been blanked. (This will always happen, because single letters are recognised text features).

Essentially this means that the word is parsed into non-overlapping known text features. The important part is that longer features have precedence over shorter ones. This is so that words like 'eight' are not pronounced like 'e-i-gu-hu-t', the method provides for the automatic priority of exception pronunciations when needed. Also, priority is given to those text features that occurred frequently in the training set. This parsing method is reminiscent of eye movements in that it is not strictly left-to-right.

Having found the text features, each one is replaced by the sound feature with which it was most commonly associated in the training data, and these sound features are then output in the order that the text features occurred in the original word.

As an example, consider the pronunciation of the word 'training'. The first and longest text feature to be found might be 'rain'. The search is then continued on 't....ing', resulting in 'ing', and then a search in 't.......' resulting in 't'. The matching sounds for these text features are then found, and output in the same order: 't' 'rAn' 'iN'.

### Results

The performance of the model depends on how many separate features are allowed to be recognised. The model will function as a lexicon if it is allowed to detect enough long features, otherwise the piecing-together procedure will be used. Using the Seidenberg & McClelland word list, it was possible to get 92% accuracy by allowing the model to generate so many text features that most words were dealt with in one piece (there were 3000 features allowed of each length from 1 to 6 letters/phonemes). Errors were still made because whole words sometimes occur inside other words (for example the word 'apt' also occurs in 'leapt' and 'rapt'), so the pronunciation chosen is not just dependent on the whole word. A possible way around this problem could be to have beginning-of-word and end-of-word symbols. Also, seven-letter words were not pronounced correctly because, in most cases, the word was split into a six-letter text feature whose corresponding pronunciation was the whole word. See this chapter's appendix for a complete list of the incorrectly-

pronounced words.

When fewer features are allowed, forcing the model to build up the pronunciation of each word from a list of text features, the performance is very dependent on the quality of the mappings from text features to sound features. Most of the time, these mappings are sensible, but often the model makes mistakes, such as matching the text feature 'spe' to the sound feature 'sp' instead of 'spe'. This appears to occur because the sound of vowels is often much more variable than consonants, so that the longer sound feature 'spe' doesn't occur consistently enough with the text feature to have a higher score than the shorter sound feature 'sp'.

The mutual information given by the probability distributions of pairs of text/sound features can also be used as a measure to score the sound features[3]. This gives only slightly better performance.

With 200 features of each length from 1 to 5 letters/phonemes, the model scored only 44% on the Seidenberg & McClelland data set. One shouldn't expect much more than this because the model has no way of coping with the large number of exception words when it has so few recognised features. Also, the only criterion that the model has to go on is how many words contain pairs of features; children are given the simpler matches explicitly when they are taught to read.

A typical error is to map the letters 'pt' to the phonemes 'ept' instead of to 'pt'. This occurs because out of all the words that have the letters 'pt' in them, many contain the letters 'ept' and hence the phonemes 'ept'. For example, the five highest scores for the text feature 'pt' in one run were:

[3] I.e. the score for a particular pair of sound and text features is taken to be: score = $p_t p_s \operatorname{Log}\left(\frac{p_{ts}}{p_t p_s}\right) + p_t p_{\bar{s}} \operatorname{Log}\left(\frac{p_{t\bar{s}}}{p_t p_{\bar{s}}}\right) + p_{\bar{t}} p_s \operatorname{Log}\left(\frac{p_{\bar{t}s}}{p_{\bar{t}} p_s}\right) + p_{\bar{t}} p_{\bar{s}} \operatorname{Log}\left(\frac{p_{\bar{t}\bar{s}}}{p_{\bar{t}} p_{\bar{s}}}\right)$. Here, $p_t$ $p_{\bar{t}}$ and $p_{t\bar{s}}$ are the probabilities of a word containing the text feature, the probability of a word not containing the text feature and the probability of a word containing the text feature but not containing the sound feature etc.

Note that this method doesn't require that the score is weighted by the length of the sound feature - the mutual information measure effectively corrects for rarely-occurring features automatically.

| Sound feature | Score |
|---|---|
| ept | 338 |
| pt | 322 |
| ep | 292 |
| ke | 186 |
| p | 145 |

(These scores were obtained using the mutual-information method)

The reverse error can sometimes occur, for example mapping the letters 'arch' to the phonemes 'rtS' instead of 'ortS'. In general, when an incorrect match is made, the score for the correct sound feature is almost as high as the chosen best match. For example the scores for the text feature 'oll' were:

| Sound feature | Score |
|---|---|
| rOl | 90 |
| Ol | 80 |
| rO | 61 |
| l | 47 |
| O | 35 |

Problems also occur when the sound feature that is the best match for a particular text feature is not part of the sound features list. This can be avoided by scanning all words containing a particular text feature for sound features, and picking the most common one, but this takes too much computer time for regular testing, especially with large training lexicons. When tried, this method results in an improvement from 44% to 52% words correct when trained and tested on the 2998-word Seidenberg & McClelland database, and using 200 features of each length from 1 to 5 letters/phonemes.

The performance of the model on non-words was simulated by using the same text and sound features that gave 92% accuracy on the Seidenberg & McClelland set, but forcing the model to use at least two text features per word. The performance dropped to 54%, but it should be remembered that this is including all the irregular words in the data set.

The model has also been tested on the reverse task of converting input phonemes into text.

The difference between the two operations is that there are usually more letters than phonemes in words, and there are often alternative spellings for the same sound (e.g. leek-leak). Performance was again measured using the Seidenberg & McClelland database, and with 200 features of each length from 1 to 5 letters/phonemes.

Performance was only 31%, but many of the incorrect spellings were phonetically correct, for example: `zink` (zinc), `tril` (trill), `scool` (school), `kee` (key), `fackt` (fact), `croke` (croak), `kare` (care), `ile` (aisle). Counting these cases as correct increased the score to 49%.

### Psychological data and problems

Double dissociations between the reading of non-words and exception words have been observed in patients with reading problems (Humphreys & Evett [1985], Coltheart et al [1980]). These could occur in this model:

An inability to pronounce non-words could be caused by:

- Poor eye-movement control, resulting in missing out letters, or including them twice in different text features.
- Failure to remember which parts of the non-word have been recognised.
- A failure of the feature detectors to recognise features that aren't surrounded by spaces.

Inability to pronounce exception words could be caused by:

- Failure of the lexicon to store long groups of letters/phonemes, thus forcing the person to build all words up from small units.

Thus the model can be thought of as being able to account for double dissociation, even though it doesn't possess separate modules for phonological and lexical reading.

There is evidence (e.g. in Humphreys & Evett [1985]) that the pronunciation of novel words is slowed if the word is homophonic with a real word, compared to a similar novel word that is nonhomophonic; e.g. `BRANE` takes longer to pronounce than `BRAME`. The model described here would not show this effect, or any of the other small variations in response time found for various types of novel word, because it is modelling only the most basic level of reading aloud. There may be some interactions between different ways of parsing words that could account for some variations in response time, and a more detailed model of the

feature-detectors might show variations in response time for different categories of words.

A possible weakness of the model is that only one pronunciation of each text feature is ever considered when reading; some sort of mixing of sound features akin to the operating of some analogy models of reading (such as Sullivan & Damper [1992]) may be needed, but this would greatly complicate the model.

The simplest improvement to the model would be to amend the chosen pronunciations of the text features whose pronunciations (derived from the statistics of the training set) cause errors when the model is tested. While this is contrary to the spirit of an automatically trainable model, it has some justification in the fact that children are taught explicit pronunciations for many single letters, and they certainly receive error-correcting feedback when they make a mistake.

In defence of the apparently poor performance of this model on non-words, it should be pointed out that people don't always give the same pronunciations of novel words. In Coltheart et al. [1992], the criteria for deciding whether a model's output is correct is that it should match a pronunciation made by any one of 20 human subjects.

This general class of model has positive aspects; it is capable of coping with any length of word, and the simple detectors of commonly occurring features could almost be expected to exist in the brain. The algorithm also works in reverse -the model can learn to convert phonemes into text, if the training data is swapped around. Hence the feature detectors developed are useful for transcribing as well as reading aloud.

### Possible improvements and further work

The model is extremely simple, and there are many aspects which could be improved. Ideally, the a feature detector which responds to (say) '`ing`' would be influenced slightly by the letters outside of its three-letter window, so that detection of features would be context dependent.

Some more specific ideas are:

- Add beginning and end-of-word symbols. As noted above, these could improve the performance of the model markedly. They are also a plausible addition to the model, because real words have easily visible white space surrounding them.

- Try some sort of error correcting mechanism. This could be made an automatic part of the learning process by amending text feature pronunciations which consistently cause errors. Alternatively, an external 'teacher' could be used to give the model many repeated examples containing certain text feature pronunciations.
- Test the model explicitly on non-words, instead of using the rather artificial method described earlier.
- Make further investigations into the models performance on the reverse task of converting sound into text.

One experiment which has not yet been conducted is to look at people's eye movements when they are presented with novel words. The model described here depends very heavily on the use of eye movements to segment novel words, so results of such experiments would be very interesting, and could potentially suggest developments to this kind of model.

## Appendix

There follows a list of the incorrectly-pronounced words when the model was allowed to use up to 3000 text and sound features of each length from 1 to 6.

Each incorrectly-pronounced word is shown as two lines in the following form:

*Text* *Sound*
*Text-features* *Sound-features*

Where *Text* and *Sound* are the text and sound of the input word, *Text-features* is a space-separated list of the text features into which the model split the input word, and *Sound-features* is the corresponding list of space-separated sound features.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| ache | Ak | aunt | ant | bough | bW | butt | b^t |
| ache | kaS | aunt | nt | bough | b*t | butt | byUt |
| apt | apt | bas | bo | bow | bW | cat | kat |
| apt | pt | bas | bAs | bow | bO | cat | ka |
| arc | ork | bass | bas | braille | brAl | clan | klan |
| arc | ortS | bass | bAs | b raille | b brAl | clan | kla |
| arch | ortS | bear | bAr | breadth | bredT | clot | klot |
| arch | rtS | bear | bErd | b readth | b bredT | clot | kl |
| are | or | bing | biN | breathe | brED | cloth | kl*T |
| are | Ar | bing | bindZ | breath e | breT r | cloth | kl |
| arm | orm | boot | bUt | brig | brig | con | k*n |
| arm | rm | boot | bU | brig | br | con | kOn |

| | |
|---|---|
| cop | kop |
| cop | kOp |
| crib | krib |
| crib | skrIb |
| crow | krO |
| crow | kr |
| cry | krI |
| cry | kript |
| cub | k^b |
| cub | kyUb |
| cut | k^t |
| cut | kyUt |
| deal | dEl |
| deal | delt |
| dear | dEr |
| dear | derT |
| do | dU |
| do | d |
| doe | dO |
| doe | d^z |
| dove | d^v |
| dove | dOv |
| draught | draft |
| d raught | d draft |
| dream | drEm |
| dream | dremt |
| drought | drWt |
| d rought | d r*t |
| ear | Er |
| ear | r |
| earth | erT |
| earth | rT |
| ease | Ez |
| ease | Es |
| fan | fan |
| fan | fa |
| flu | flU |
| flu | fl |
| fort | fOrt |
| fort | fOr |
| freight | frAt |
| f reight | f frAt |
| gag | gag |
| gag | gAdZ |
| gal | gal |
| gal | g |
| gas | gas |
| gas | ga |
| glad | glad |
| glad | gl |
| glimpse | glimps |
| g limpse | g glimps |
| go | gO |
| go | g |
| grad | grad |
| grad | gr |
| grim | grim |
| grim | gr |
| grip | grip |
| grip | gr |
| hack | hak |
| hack | ak |
| hag | hag |
| hag | ag |
| hair | hAr |
| hair | Ar |
| ham | ham |
| ham | am |
| hang | haN |
| hang | CAndZ |
| hank | hank |
| hank | ank |
| hard | hord |
| hard | ord |
| hare | hAr |
| hare | Ar |
| hark | hork |
| hark | ork |
| harm | horm |
| harm | orm |
| harp | horp |
| harp | orp |
| hart | hort |
| hart | ort |
| has | haz |
| has | As |
| haste | hAst |
| haste | Ast |
| hat | hat |
| hat | at |
| haw | h* |
| haw | * |
| hay | hA |
| hay | A |
| heal | hEl |
| heal | helT |
| heap | hEp |
| heap | Ep |
| hear | hEr |
| hear | her |
| heart | hort |
| heart | hor |
| heat | hEt |
| heat | Et |
| heck | hek |
| heck | ek |
| heel | hEl |
| heel | El |
| hell | hel |
| hell | el |
| help | help |
| help | elp |
| hem | hem |
| hem | Em |
| hen | hen |
| hen | en |
| hence | hens |
| hence | ens |
| here | hEr |
| here | Ar |
| hey | hA |
| hey | A |
| hick | hik |
| hick | ik |
| hid | hid |
| hid | Id |
| hide | hId |
| hide | Id |
| high | hI |
| high | I |
| hill | hil |
| hill | il |
| him | him |
| him | im |
| hip | hip |
| hip | ip |
| his | hiz |
| his | is |
| hit | hit |
| hit | it |
| hive | hIv |
| hive | Iv |
| hoc | hok |
| hoc | ok |
| hoe | hO |
| hoe | SU |
| hone | hOn |
| hone | On |
| hook | huk |
| hook | uk |
| hoop | hUp |
| hoop | Up |
| hoot | hUt |
| hoot | Ut |
| hop | hop |
| hop | op |
| horn | hOrn |
| horn | Orn |
| hose | hOz |
| hose | Oz |
| host | hOst |
| host | Ost |
| hot | hot |
| hot | ot |
| house | hWs |
| house | hW |
| house | hWz |
| house | hW |
| huck | h^k |
| huck | ^k |
| hug | h^g |
| hug | hyUdZ |
| hum | h^m |
| hum | ^m |
| hump | h^mp |
| hump | ^mp |
| hunk | h^nk |
| hunk | ^nk |
| hunt | h^nt |
| hunt | ^nt |
| hush | h^S |
| hush | ^S |
| hut | h^t |
| hut | ^t |
| jut | dZ^t |
| jut | dZ |
| laid | lAd |
| laid | plad |
| lead | led |
| lead | lEd |
| leap | lEp |
| leap | lept |
| lease | lEs |
| lease | lE |
| lid | lid |
| lid | lId |
| limb | lim |
| limb | klIm |
| live | liv |
| live | lIv |
| lose | lUz |
| lose | klOs |
| me | mE |
| me | m |
| mean | mEn |
| mean | ment |
| moot | mUt |
| moot | mU |
| no | nO |
| no | n |
| now | nW |
| now | nO |
| once | w^ns |
| once | ns |
| one | w^n |
| one | On |
| our | Wr |
| our | Or |
| pad | pad |
| pad | spAd |
| pal | pal |
| pal | p |
| past | past |
| past | st |
| pear | pAr |
| pear | perl |
| pie | pI |
| pie | pE |
| plus | pl^s |
| plus | pl^ |
| prim | prim |
| prim | pr |

| | |
|---|---|
| pro | prO |
| pro | pr |
| quit | kwit |
| quit | kw |
| rang | raN |
| rang | rAndZ |
| rat | rat |
| rat | rAt |
| read | rEd |
| read | red |
| real | rEl |
| real | relm |
| rib | rib |
| rib | rIb |
| rind | rInd |
| rind | grind |
| road | rOd |
| road | br*d |
| rob | rob |
| rob | rOb |
| roe | rO |
| roe | TrOz |
| rot | rot |
| rot | r*T |
| rough | r^f |
| rough | r* |
| rouse | rWz |
| rouse | rW |
| route | rWt |
| route | rUt |
| row | rO |
| row | rW |
| sag | sag |
| sag | sAdZ |
| say | sA |
| say | sez |
| scar | skor |
| scar | sk |
| schnook | Snuk |
| s chnook | s Snuk |
| scour | skWr |
| scour | skerdZ |
| scourge | skerdZ |
| s courge | s skerdZ |
| scrap | skrap |
| scrap | skr |
| scratch | skratS |
| s cratch | s skratS |
| screech | skrEtS |
| s creech | s skrEtS |
| sear | sEr |
| sear | sertS |
| sit | sit |
| sit | sIt |
| ski | skE |
| ski | ski |
| slat | slat |
| slat | sl |
| sleight | slAt |
| s leight | s slAt |
| slid | slid |
| slid | sl |
| slim | slim |
| slim | sl |
| slop | slop |
| slop | sl |
| soot | sut |
| soot | sU |
| sooth | sUT |
| sooth | sU |
| sour | sWr |
| sour | sOrs |
| sow | sW |
| sow | sO |
| spa | spo |
| spa | sp |
| spat | spat |
| spat | sp |
| spin | spin |
| spin | sp |
| spit | spit |
| spit | sp |
| splurge | splerdZ |
| s plurge | s splerd |
| squeeze | skwEz |
| s queeze | s skwEz |
| stag | stag |
| stag | stAdZ |
| staunch | st*ntS |
| s taunch | s st*ntS |
| steal | stEl |
| steal | stelT |
| stealth | stelT |
| s tealth | s stelT |
| straight | strAt |
| st raight | st strAt |
| strange | strAndZ |
| s trange | s strAnd |
| strength | streNT |
| st rength | st streNT |
| stretch | stretS |
| s tretch | s stretS |
| strip | strip |
| strip | str |
| sue | sU |
| sue | swAd |
| suit | sUt |
| suit | swEt |
| sun | s^n |
| sun | s^ |
| swam | swam |
| swam | swomp |
| swan | swon |
| swan | swank |
| swat | swot |
| swat | swo |
| teak | tEk |
| teak | stAk |
| tear | tEr |
| tear | tAr |
| tent | tent |
| tent | ten |
| than | Dan |
| than | an |
| the | D^ |
| the | D |
| thin | Tin |
| thin | Ti |
| thou | DW |
| thou | T*t |
| though | DO |
| though | T*t |
| thought | T*t |
| though t | T*t t |
| through | TrU |
| t hrough | t TrU |
| thru | TrU |
| thru | Tr |
| ton | t^n |
| ton | tOn |
| tong | t*N |
| tong | t^N |
| toot | tUt |
| toot | tU |
| trip | trip |
| trip | tr |
| try | trI |
| try | trist |
| twelfth | twelfT |
| t welfth | t twelfT |
| use | yUs |
| use | Ws |
| vie | vI |
| vie | vyU |
| wag | wag |
| wag | wAdZ |
| was | w^z |
| was | w |
| wind | wInd |
| wind | nd |
| wind | wind |
| wind | nd |
| wing | wiN |
| wing | twindZ |
| won | w^n |
| won | wOnt |
| word | werd |
| word | rd |
| wound | wUnd |
| wound | nd |
| wound | wWnd |
| wound | nd |
| writ | rit |
| writ | rI |
| wrought | r*t |
| w rought | w r*t |
| yea | yA |
| yea | y |
| year | yEr |
| year | yern |

## Information theory, and an introduction to chapters 4-6

Unsupervised neural networks can only respond to statistical structure in their input data. What is needed in order to make real progress in understanding how unsupervised networks can be used in models of cognition is a concrete mathematical way of talking about statistical structure. Luckily, there is a branch of mathematics which is devoted to the analysis of statistical structure - information theory. For an introduction to information theory, see Shannon [1948][1963] and Plumbley [1991].

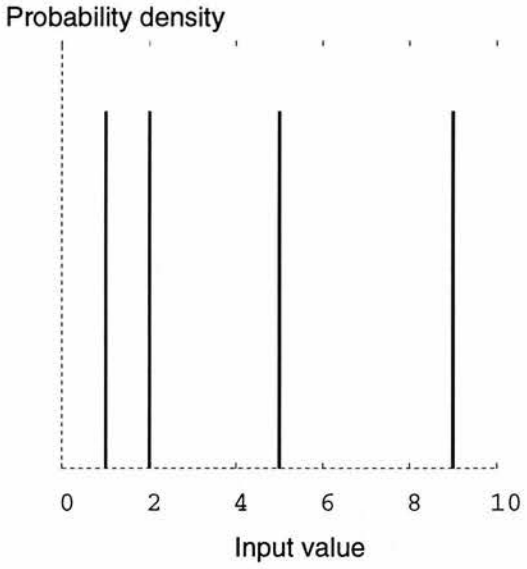
Information theory deals solely in terms of statistical structure - there is no (explicit) concept of data *meaning* anything. As far as an information theoretic analysis is concerned, a set of data is characterised by the set of probabilities of each different pattern occurring (this is much more informative in the continuous case than the discrete one). From this set of values, one can calculate the entropy of the dataset, and the average amount of information it conveys about another dataset and so on. Because of this reliance only on the probabilities of different input patterns, the particular bit-patterns (if the data is encoded in binary form) of the data are irrelevant. For example, an important calculation in information theory is to find the entropy of a set of patterns. The entropy is a measure of how random a set of data is, and is a maximum when each data pattern has the same probability. This is just entropy, $H = -\sum_{i=1}^{n} p_i \operatorname{Log}(p_i)$, where $n$ is the number of patterns and $p_i$ is the probability of pattern $i$ occurring.

For example. if our dataset consists of four input patterns, with probabilities 0.3, 0.1, 0.4, 0.2, we could represent these four patterns in any way we want, for example as $\{0001_{0.3}\ 0010_{0.1}\ 0100_{0.4}\ 1000_{0.2}\}$, or $\{00_{0.3}\ 01_{0.1}\ 10_{0.4}\ 11_{0.2}\}$, where the subscripts denote the probability of each pattern. One could even represent the four patterns as bitmaps of the four characters **abcd**, with the same probabilities as above, i.e. probability of (**a**)=0.3, probability of (**b**)=0.1, probability of (**c**)=0.4 and probability of (**d**)=0.2. Information theory would treat all of these representations identically, with the entropy of this set of data being $-(0.3 \log(0.3) + 0.1 \log(0.1) + 0.4 \log(0.4) + 0.2 \log(0.2)) \approx 1.28$.

The reason why information theory is applicable to neural networks, and knowledge representation in general, is that when there is noise present the simple definition of entropy described above *is* affected, indirectly, by the actual bit patterns used to describe the patterns - usually when the data is represented with continuous values rather then discrete ones. In

this case, there is an infinite number of possible patterns, because the noise will distort each pattern by a random amount.

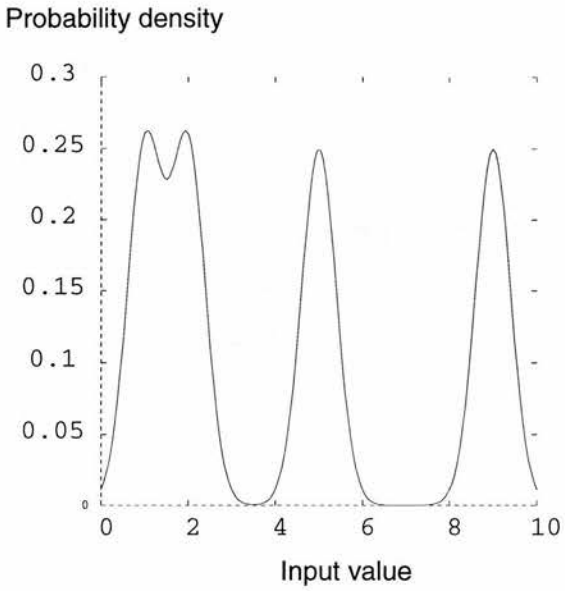
As an example of this, consider a set of 4 one-dimensional patterns, where each pattern is a real number between (say) 0 and 10. Let the set of patterns be $\{1, 2, 5, 9\}$. We can represent this set of patterns as:




***Probability distribution for a set of four discrete one-dimensional patterns.***[4]

If these patterns are distorted by random noise, the probability distribution in the space of patterns will look like:

[4] The probability distribution for these discrete patterns is a set of four delta-functions, so there is no scaling on the $y$-axis.




***Probability distribution for four one-dimensional patterns, after distortion by gaussian noise of width 0.4.***

As can be seen, the probability distribution has changed, and this affects (for example) calculations of the entropy of the data. In particular, if the particular values of the input patterns are changed, while their probabilities are unchanged and the size of the noise is also unchanged, the probability density curve would change and so the entropy of the data would be different.

The reason why I consider information theory to be important to the study of models of cognition is that there is a strong link between the information-theoretic concept of maximising the amount of information transmitted by the output of a black-box about the black-box's input, and the intuitive idea that sense data should be represented in terms of high-level concepts. This will be explained in chapter 4. These ideas also correspond closely with the Minimum Description Length (M.D.L.) principle (Rissanen [1989]).

A consequence of linking information theory with the idea of developing high-level categories is that it provides a concrete mathematical idea of what high-level representations are. It will be shown in chapter 4 that one can actually derive the precise function which should be applied to sense data in order to transform it into a representation which is in terms of high-level categories. This work can be regarded as a generalisation of the ideas of Barlow and Földiák (Barlow [1989]; Földiák [1992]) on reducing redundancy. An example of the technique is described in chapter 5, in which the characteristics of the visual filter best fitted to simple visual data is derived, and shown to consist of elements which find, amongst other things, Fourier transforms of the input data.

These ideas about information theory and the analysis of sense data are extremely general, and as such are expressed most naturally using mathematics rather than, for example, as a neural network learning algorithm. Naturally, the reasons for preferring neural network models of cognition still apply, so I have developed a very crude neural network algorithm which attempts to learn weights that make the network process data in accordance with the theory. This learning algorithm will be described in chapter 6

# Chapter 4 The Maximum-Entropy-Filter (M.E.F.) principle

The raw sense-data that is available to the brain is not completely random, but contains an enormous amount of structure. For example, most visual images have large areas of one colour, and sounds usually have harmonics present.

It is because raw data contains such structure that the brain is able to extract high-level structure and form an internal representation based on high-level structure and work with this rather than the original low-level data (Barlow [1989]).

Hence it is interesting, for example, to view the early visual system[5] as a black box or filter which transforms the retinal image into a high-level representation. However, a 'high-level' representation, though sounding intuitively reasonable, is hardly a precise concept; we need to somehow define what a high-level representation is more rigorously before we can attempt to develop a visual filter which transforms a retinal image into such a high-level representation.

One characteristic of high-level representations is that they are very efficient - instead of describing an image of a car by giving the millions of pixel colours which make up the image, we just say something like 'A red Ferrari'. This works for most common images but, for example, it doesn't work very accurately when we try to describe the picture on an untuned television screen. So it would seem that high-level descriptions will only work accurately for a subset of the set of all possible input pictures, such as those which are seen very often, like people's faces.

This suggests that our internal high-level representation of the images on our retina is determined to a large degree by the statistics of our visual environment, are most detailed for the most common visual images.

Going back to the idea of making the high-level representation as efficient as possible, this requires that it doesn't contain redundancies between the elements of the high-level representation, as these could be removed, resulting in an even more compact representation. For example, we think of a picture of a car in terms of the make of the car, its colour, its orientation on the page and so on. All of these properties are largely independent.

[5] Throughout this chapter, I will concentrate on visual sense data. However, the main ideas presented could, in principle, be applied to hearing, touch, taste etc.

So far, it looks like high-level representations are characterised by being the most compact representation of the data (Rissanen [1989]). This works very well for discrete input patterns, and one could, for example, look at file-compression techniques used on computer systems to see how normal text could be represented. Wolff [1982] discusses how learning language is similar to data compression.

However, it is not obvious how to apply these ideas to the case where the input data is continuous, because in this case there is no way of making a more compact representation than the original input representation unless we are prepared to lose some detail.

The way of looking at the case of continuous input patterns developed here will be the following: we assume that our input data is presented with no distortion, and we have to transform it with some sort of filter into a representation which has limited bandwidth (or, equivalently, has some sort of output noise present) in such a way as to preserve as much of the original data as possible. The motivation for doing this is that, in the continuous case, trying to preserve as much data as possible in a limited bandwidth corresponds to representing the data as efficiently as possible in the discrete case. Hence it will be the continuous equivalent of the discrete case where we equated a compact representation with a high-level representation[6]. Also, we will assume that the output noise is constant additive noise, which is reasonable because it corresponds approximately to the case where the output has a constant finite resolution.

There are two ways of seeing how this would work:

The first is to consider the earlier discussion about compact codes, and to remember that they look random. In the continuous case, this generalises to the internal representation having maximum-entropy.

The second is to use the idea, from information theory, of *mutual information*, and find an output representation which contains, on average, the most information about the input pattern, given that the output is distorted by noise. This calculation is carried out in appendix A at the end of this chapter, which concludes that the output should have maximum entropy.

Importantly, this result doesn't depend on the size of the output noise, as long as the output

[6] Other studies, for example Plumbley [1991], look at a similar problem, but also postulate input noise. We are interested here in developing high-level representations for data though, rather than how to best filter-out any input noise. Ignoring input noise will also allow a more general result to be derived.

noise is constant.

Hence it seems that we can form high-level representations by trying to maximise the entropy of the representation. This will be referred to from now on as the maximum-entropy-filter (M.E.F.) principle.

A maximum-entropy representation might not sound like being equivalent to a high-level representation, and the arguments presented so far about high-level representations have very little concrete evidence to support them. However, the example presented in chapter 5, where the M.E.F. principle is applied to a very simplified model of early visual images gives a striking example of how such a representation could be formed, and of how the characteristics of the required filter seem very interesting and relevant to the study of real cognitive systems. Also, the next section shows how a system which generates a maximum-entropy representation would have to contain conventional constructs such as feature detectors.

Maximising entropy can be thought of as a generalisation of Barlow's principle (Barlow [1989]) that the sensory systems should minimise correlations within a representation. In particular, entropy is sensitive to 'hidden' structure within data which the correlation measure ignores. For example, if a three-bit representation is given the set of 'images' {000, 011, 101, 110}, one can calculate the correlation between any pair of bits, and it turns out to be zero. Hence such a scheme would contain no structure from a correlation point of view. However, the dataset is, in fact, highly redundant - each bit is the XOR of the other two. The entropy of the dataset exposes this redundancy very clearly, because it is less than the maximum entropy for a bit representation.

In general, the entropy measure will be sensitive to *any* redundancy, even a very complicated dependence between many parts of the representation. To see how this works, we need to look at what is involved when we try to form a maximum-entropy representation, and we can do this by looking at the probability distributions in the space of all input/output patterns

### M.E.F. in terms of input/output probability distributions

There is a unique solution for making the output probability distribution have maximum entropy when there is uniform additive noise, which is to make the output probability density constant, i.e. make all output pattern patterns equally probable. This is a standard

result, and appendix B at the end of this chapter contains a proof.

If the input patterns are discrete, and there is a finite number of them, there is no way of making such an output probability distribution, even with output noise, since the output probability distribution will be given by convolving the output patterns with the output noise, resulting (for gaussian output noise) in a a set of gaussians (one for each input pattern) centred at different places in output space. All we can do to increase the maximum entropy of this output distribution is to move the centres of the gaussians (i.e. the outputs with zero noise) around so that the output probability distribution is as flat as possible (where the measure of 'flatness' is the output entropy). The neural network learning rule described in chapter 6 uses this scheme because neural networks are most easily thought of as having discrete input patterns.

If our input patterns are infinite in number though, we are effectively dealing with an input probability distribution instead of individual input patterns. In this case, the M.E.F. principle says that the input patterns should be transformed in such a way that the output probability distribution is flat, and it is theoretically possible to do so.

Considering how the M.E.F. principle would work in terms of input/output probability distributions is particularly useful because the statistical structure in input data which we have been considering manifests itself in the input probability density being non-uniform.

For example, if the input pattern is a patterns of activations on a 100 by 100 array of pixels, giving 10,000 pixels in all, then input space would be 10,000-dimensional, and each point in this 10,000-dimensional space would represent a complete image. A set of input patterns would thus be represented by specifying the probabilities of any possible pattern occurring. If we are told that a set of input patterns for this 100 by 100 retina contains a disproportionate number of faces, this means that the probability of an input pattern being in that small region of the 10,000-dimensional space which corresponds to faces, is greater than normal. The goal of the M.E.F. principle is to find a representation for this input distribution which has a constant probability distribution.

At first sight, this seems almost trivial - a constant probability distribution is very easy to make. However, it is very important to remember that we have to make the flat output probability distribution using *only* the (very un-flat) input distribution. For example, we can't use a random number generator inside the neural network/black box which does the

transformation (this intuitively obvious restriction is also formally required by the maximising-information-transmission calculation in appendix A, as the information conveyed by the outputs about the inputs would be seriously degraded if we used random number generators).

In fact, transforming the input data into a representation which has a flat probability distribution is an extremely complicated task. This reflects the difficulty of forming high-level representations.

To see what would happen for the case of faces having higher than normal probability of occurring, we could represent the input probability distribution by pretending it has only 2 dimensions, and using a third dimension to represent the probability distribution. This gives us a surface which represents the input probability distribution:

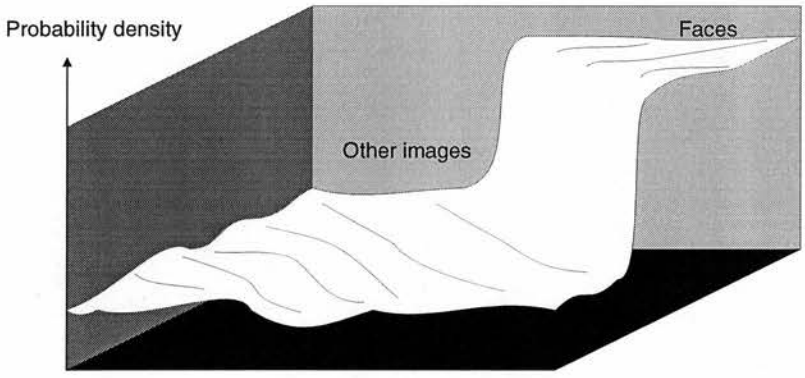



***Figure 1***
***Input probability density***

In order to make a uniform output probability distribution, the system would have to expand the region in input space, 'Faces', so that it occupies a large part of output space, while all other regions in input space, 'Other images', would have to be condensed into the remainder of the output space, making the probability distribution look like:

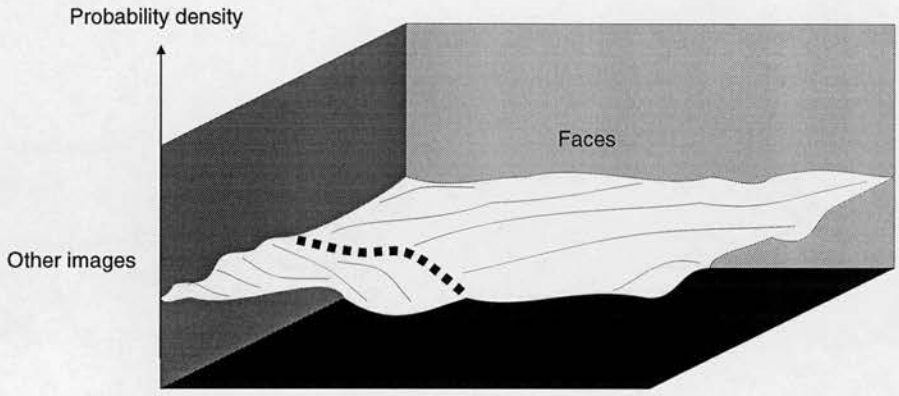



*Figure 2 Output probability density*

This is a very non-trivial operation, as the system would have to transform each input vector in one of two ways, according to whether the input vector was in input region 'Faces' or not. The simplest way of doing this would probably be to have a module which detects whether an input pattern is from within input region 'Faces' and causes the input vector to be transformed in one way if this is so, and a different way if not.

This illustrates how the seemingly abstract M.E.F. principle would lead to the development of conventional concepts such as feature detectors.

Note that the output representation automatically provides more output space for images of faces, and so can represent faces much more accurately than other images.

However, the M.E.F. principle goes further - any additional irregularities in the input probability distribution would need to be dealt with in a similar way. For example, it might be that within the 'Faces' region of input space, there is a certain area which has lower than normal probability (this could be the part which represents oriental faces, if the input data corresponds to the visual experience of a European person). In order to transform this more complicated input probability distribution into a flat probability distribution, the system would have to not only detect and selectively transform faces/non-faces but, within the space of faces, it would have to selectively detect and transform *oriental* faces. As before, this is a highly non-trivial task, and would have to involve the generation of feature detectors for oriental faces.

It is important to bear in mind that so far we have been considering a very over-simplified input probability distribution. In practise, the probability surface will be full of extremely complicated peaks and troughs, and of course this will all be in a space of many thousands

of dimensions.

Hopefully it will be clear now that the M.E.F. principle requires that this sort of process be done for every level of detail in the input probability distribution and so the M.E.F. principle corresponds very closely to conventional ideas on how data should be represented in terms of many different high-level concepts. Very importantly, the M.E.F. principle allows this statistical and hierarchical decomposition to be viewed in a particularly simple information theoretic framework.

Incidentally, viewing the maximum-entropy-filter principle as meaning that the output probability should be flat, removes any considerations of how large the output noise is (as long as the noise is constant additive noise). This is because a flat probability distribution before noise will always give a flat probability distribution after constant additive noise[7].

In some studies of systems which transform input data into a different representation, (for example Atick and Redlich [1990]), varying the output noise-width governs how the system behaves - whether data is represented very accurately, or less accurately but with more resistance to noise. Because the M.E.F. principle can be viewed as making a uniform output probability distribution, it is unaffected by these considerations (except for the case of a finite number of input patterns, when the output probability distribution cannot be flat). This is very convenient as it removes the need to tune the noise width to the task being performed.

### Uniqueness of a constant probability distribution

A slightly different way of looking at the M.E.F. process is to forget about information theory and high-level representations, and simply consider that, as has been mentioned before, all a black box such as an unsupervised neural network can do is to transform input data into a different representation.

If we look at this in terms of the input/output probability distribution, then transforming into a constant probability output distribution is, in some sense, a special transformation, because a uniform probability distribution is the simplest possible probability distribution.

[7] I don't know of a proof that a flat distribution before noise is the *only* solution, however.

### Continuous mappings

Requiring a maximum entropy representation (or, equivalently for most realistic cases, constant probability representations) doesn't constrain the representation completely. For example, we could swap chunks of output space around, which would give a very different output representation while keeping the output entropy at the maximum. In particular, this means that a maximum-entropy output representation is possible whose topology is very different from the topology of the input representation - i.e. similar input patterns are not mapped to similar output patterns or, equivalently, the mapping from input to output is not a *smooth* mapping.

The reasons for requiring a maximum-entropy representation are not affected by this problem, but to conform to the overriding intuitive idea that internal representations should be usable high-level representations, we need to require that the mapping from input representation to output representation be as smooth as possible.

Neural networks will automatically generate reasonably smooth mappings so, practically, this isn't too serious a problem.

Also, it is possible that a purely information theoretic treatment would require a smooth mapping, when there is *input* noise: If we consider the case of there being discrete patterns with input and output noise, there is always a chance that any error caused by input noise will be cancelled by the random output noise. However, this beneficial effect will be strongest when nearby input patterns correspond to nearby output patterns, as this gives the greatest chance that output noise will be large enough to move the output back into the correct pattern. Hence I conjecture that smooth mappings would in fact follow from a more complete information-theoretic treatment, as well as being generated automatically by practical systems such as neural networks.

### Input noise

Some of the work in Plumbley [1991] would seem to contradict the work presented above, as it deals with the same problem of deriving an ideal filter for particular input data, but ends up with a filter whose characteristics are affected by the signal-to-noise ratio of the input data at various frequencies. However, input noise is not considered in the M.E.F. principle, which is only about forming high-level categories from input data, rather than

processing input data which has significant noise present in such a way as to reduce the effect of the input noise.

To optimally transform input data which has input noise into a high-level representation, one would have to first increase the signal-to-noise ratio in each input line using filters such as in Plumbley [1991], and then transform the multi-input data in accordance with the M.E.F. principle.

### Psychological limitations of the M.E.F. principle

The M.E.F. principle assumes that the brain is interested in those high-level categories which are enshrined in the statistics of the input data. This works well in the case of pictures of people's faces, for example, because faces clearly form a large fraction of the visual experience of people (although only because people's eyes are directed towards other faces). Also, work such as Finch & Chater [1992], in which words are classified into very natural-looking categories purely by grouping together those words which usually occur in similar contexts, suggests that there is much useful information in the statistics of ordinary text.

However, it is probably the case that evolution has designed the brain to pay special attention to (and develop high-level representations for) some aspects of sense-data which are actually quite rare. For example, the instinctive reflex to duck when a fast-moving object approaches one's head is unlikely to be helped by the visual system learning about such objects just from experience, as this type of situation doesn't occur very often. Hence maybe there would have to be hard-wired detection of the trajectory and speed of such objects in the brain.

This idea also has relevance to the problem of dealing with input noise discussed above. How could a purely statistical system know what noise is present in input data? - such information is needed in order to determine the optimal filters in Plumbley [1991]. One might think that noise gives itself away by being a completely random element of the input, but this can apply equally to useful parts of the input data. For example, the colour of Smarties covers the whole spectrum fairly evenly, but it would be a poor visual system indeed which decides from this that the colour content of the retinal images of a pile Smarties is just random noise and so should be discarded.

Hence we cannot expect the purely statistical technique such as the M.E.F. principle to work

on its own if there is significant input noise to a system. This noise would have to be dealt with using a hard-wired method such as evolving low-pass filters in early visual pathways, as described in Plumbley [1991] or Atick & Redlich [1993].

### How to construct an ideal filter - the Filter as Inverse Generator (F.I.G.) theorem.

We have determined that the output of a filter should have maximum entropy, as this maximises the mutual information between the filter's output and the input and, more importantly, seems to be equivalent to representing the data in a high-level way.

There is a simple trick which can sometimes aid the theoretical calculation of a filter which converts some particular data into a maximum-entropy representation. This trick will be used in the next chapter and is explained below. Note that it couldn't be easily converted into a network learning rule.

The method relies on the inverse of the filter function existing. To this end, we shall assume that the filter has the same number of inputs as outputs. Note that once we have applied the method, we can discard any number of outputs from our filter, and we will still have a maximum-entropy output representation. Thus the method is still applicable to cases where the output bandwidth is smaller than the input bandwidth.

Let the set of input patterns be $\boldsymbol{A}_p$, with $p$ running from 1 to $p_{max}$, the number of input patterns. We define the filter function to be $f$, so the output of the filter is $\boldsymbol{B} = f\boldsymbol{A}$.

We can invert this equation to get $A = gB$, where $g = f^{-1}$. We shall refer to $g$ as the **generator** of the input distribution, because when fed with a completely random (maximum entropy) set of $B$'s, the set of $A$'s from $A = gB$ must necessarily have the original input distribution, if the filter $f$ from which $g$ is derived is an ideal filter.

So to find an ideal filter for an input distribution with a certain statistical property $X$, all we have to do is to find a way of generating a set with property $X$ from random numbers[8], in other words a function which transforms random numbers into a distribution with property X. We then invert this function to find the ideal filter.

This will be referred to as the 'Filter as Inverse Generator' theorem, or the F.I.G. theorem.

The reason for considering the F.I.G. theorem is that it can be easier to think in terms of

[8] The nature of the random numbers will depend on the problem being addressed - they could be random binary digits or continuous values within a certain range. See chapter 5 for an example of this.

generating a distribution from random numbers rather than converting a special distribution into a random distribution - this will be the case in chapter 5.

## Appendix A - Proof that a maximum entropy filter maximises the average mutual information between the input and output of the filter.

This result is certainly not new, but is included here because it is an important part of the M.E.F. idea.

We will show that, with constant additive noise, the mutual information between the input and output of a filter equals the output entropy of the filter, minus the entropy of the additive noise. Hence maximising the mutual information between input and output with constant noise is equivalent to maximising the output entropy.

First, we will introduce a notation for the various quantities involved.

We will be considering a system with input vector $A = \begin{pmatrix} a_1 \\ \dots \\ a_n \end{pmatrix}$ and output vector $B = \begin{pmatrix} b_1 \\ \dots \\ b_n \end{pmatrix}$, with filter function $f(A)$ and constant additive noise, so that $B = f(A) + \text{noise}$. $A$ and $B$ are both assumed to be in continuous spaces.

The mutual information $R$ between the inputs and outputs of this filter is $R = \left\langle \text{Log}\left[\frac{p(A,B)}{p(A)p(B)}\right]\right\rangle$

where:

- $p(A)$ probability of the input vector to the network being $A$.
- $p(B)$ probability of an output being $B$ (N.B the output can be $B$ for more than one input pattern $A$, and that $p(B)$ is the probability distribution *after* noise).
- $p(A, B)$ probability of the input being $A$ and the output $B$. With no noise, this would be 0 except for when $B = f(A)$, i.e. a delta function.
- $\langle \dots \rangle$ the average over all pairs of points $(A, B)$ in input and output space.

Note that the $p(A)$ and $p(B)$ will be probability densities

With discrete input patterns, the only points in input space which have non-zero probability of occurring are those points in the input data set: $A_p$, with $p$ running from 1 to the number of patterns in the dataset.

Replacing the $\langle ... \rangle$ by integrals, the expression for $R$ becomes:

$$R = \int_{\text{all input space}} dA \int_{\text{all output space}} dB\, p(A, B) \operatorname{Log}\left[\frac{p(A, B)}{p(A)p(B)}\right]$$

This can be re-written to get $R$ in terms of the entropies of the input data, output data, and the joint entropy of the input and output data. This will aid the calculation of $R$ because, for example, the entropy of the input data is fixed.

We will use $\operatorname{Log}\left[\frac{p(A,B)}{p(A)p(B)}\right] \equiv \operatorname{Log}[p(A, B)] - \operatorname{Log}[p(A)] - \operatorname{Log}[p(B)]$, and $p(A, B) = p(A)p(B \mid A)$:

$$\begin{aligned} R &= \int\int p(A, B) \operatorname{Log}\left[\frac{p(A, B)}{p(A)p(B)}\right] dA\,dB \\ &= \int\int p(A, B) \operatorname{Log}[p(A, B)]\, dA\,dB \\ &\quad - \int dA\, p(A) \operatorname{Log}[p(A)] \int dB\, p(B \mid A) \\ &\quad - \int dB\, p(B) \operatorname{Log}[p(B)] \int dA\, p(A \mid B) \end{aligned}$$

But $\int dB\, p(B \mid A) \equiv 1$, so:

$$\begin{aligned} R &= -H(A, B) - \int dA\, p(A) \operatorname{Log}[p(A)] - \int dB\, p(B) \operatorname{Log}[p(B)] \\ &= -H(A, B) + H(A) + H(B) \end{aligned}$$

Where $H(X)$ is the entropy of the probability distribution $X$.

We now have to calculate these entropies in the case where the output $B$ is a function of the input $A$ with additive noise, i.e. $B_p = f(A_p) + \text{noise}$. First we will find $H(A, B)$, making use of $p(A, B) \equiv p(A)p(B \mid A)$:

$$\begin{aligned} H(A, B) &= -\int dA \int dB\, p(A, B) \operatorname{Log}[p(A, B)] \\ &= -\int dA\, p(A) \int dB\, p(B \mid A) \operatorname{Log}[p(A)p(B \mid A)] \\ &= -\int dA\, p(A) \int dB\, p(B \mid A)\{\operatorname{Log}[p(A)] + \operatorname{Log}[p(B \mid A)]\} \\ &= -\int dA\, p(A) \operatorname{Log}[p(A)] \int dB\, p(B \mid A) - \int dA\, p(A) \int dB\, p(B \mid A) \operatorname{Log}[p(B \mid A)] \\ &= -\int dA\, p(A) \operatorname{Log}[p(A)] - \int dA\, p(A) \int dB\, p(B \mid A) \operatorname{Log}[p(B \mid A)] \\ &= H(A) - \int dA\, p(A) \int dB\, p(B \mid A) \operatorname{Log}[p(B \mid A)] \end{aligned}$$

The second integral in the second term can be seen to be constant when there is *constant additive noise* on the output $B$, and equal to the entropy of the noise. For example, with Gaussian noise of width $\sigma$ added to the outputs, $p(B \mid A)$ is a Gaussian of width $\sigma$ centred on $B = B(A)$. The integral is over all of $B$-space, so the second integral is then the entropy of this Gaussian, which is $\mathrm{Log}(\sigma)$ + a constant. For different types of noise, this will change, but provided the noise is the same for all $B$'s, this second integral will always be the constant entropy of the noise[9], which we will call $H_{noise}$. So we have for additive noise:

$$H(A, B) = H(A) + \int dA\, p(A) H_{noise}$$

$$= H(A) + H_{noise}$$

The expression for the Shannon information was $R = H(A) + H(B) - H(A, B)$. Substituting the above expression for $H(A, B)$ gives:

$$R = H(B) - H_{noise}$$

The $H(A)$ term has been cancelled by part of the $H(A, B)$ term, so the mutual information doesn't depend directly on the input entropy $H(A)$. However, the output of the filter certainly depends on the input statistics, so there is a link between the mutual information and the input entropy.

Hence we have the result that, for constant additive noise, the mutual information between the input and output of a filter is just the entropy of the output of the filter (after noise) minus the entropy of the noise, which is fixed.

## Appendix B - Proof that the probability distribution which maximises entropy is a flat distribution

This derivation is not novel. For example, see Shannon [1948].

This proof uses a technique similar to that in the calculus of variations. We assume that we know a correct maximum-entropy distribution, and that this distribution is a local-maximum - i.e. require that any very similar probability distribution should have (to first order) the same entropy. This leads to an equation which the original distribution must satisfy, whose solution is a constant probability distribution:

[9] This only works because we are assuming unbounded values for each $B_{..}$ e.g., if an output was 0.99 and the maximum $B$ is 1.00, then we couldn't realistically assume constant additive noise of width 0.4. However, in most cases with reasonably small noise, this is unlikely to be a problem.

We will work in a space $x$, with each value of $x$ being a particular pattern. For example, $x$ could be a 144-dimensional vector if we were considering patterns on a $12 \times 12$ retinal array. All integrals are over all possible input patterns $x$, so would be 144-dimensional integrals in this case.

We need to choose a probability distribution $p(x)$ to maximise the entropy of a distribution, but also to satisfy:

$$\int p(x)\,dx = 1$$

The entropy to be maximised is:

$$H = -\int p(x)\,\mathrm{Log}(p(x))\,dx$$

If $p(x)$ is optimal, then when we change $p(x)$ into a slightly different probability distribution $p(x) + \delta p(x)$, then the second probability distribution must satisfy $\int p(x) + \delta p(x) = 1$. Hence we must have:

$$0 = \int \delta p(x)\,dx \tag{1}$$

Also, the entropy of the new distribution must be (to first order) the same as previously, i.e. $\delta[H] = 0$. This is:

$$\begin{aligned} 0 &= \delta\left[-\int p(x)\,\mathrm{Log}(p(x))\,dx\right] \\ &= -\int \delta p(x)\,\mathrm{Log}(p(x))\,dx - \int p(x)\,\delta[\mathrm{Log}(p(x))]\,dx \\ &= -\int \delta p(x)\,\mathrm{Log}(p(x))\,dx - \int p(x)\,\frac{\delta p(x)}{p(x)}dx \\ &= -\int \delta p(x)\,\mathrm{Log}(p(x))\,dx - \int \delta p(x)\,dx \end{aligned} \tag{2}$$

These two equations must each hold for all $\delta p$, so we can use a Lagrange multiplier $\lambda$, and require that $\lambda(1) + (2) = 0$. This is:

$$\begin{aligned} 0 &= \lambda \int \delta p(x)\,dx - \int \delta p(x)\,\mathrm{Log}(p(x))\,dx - \int \delta p(x)\,dx \\ &= \int \delta p(x)\{\lambda - \mathrm{Log}(p(x)) - 1\} \end{aligned}$$

If our $p(x)$ is optimal, this must hold for *all* possible (small) $\delta p(x)$. The only way of ensuring this is to have $\{\lambda - \mathrm{Log}(p(x)) - 1\} = 0$. Hence we get $\mathrm{Log}(p(x)) = 1 - \lambda$.

$\lambda$ is a constant, so we have the result that $p(x) = \text{const.}$ for a maximum-entropy distribution.

To find the actual value of $\lambda$, we use $\int p(x)\,dx = 1$, so $p(x) = \frac{1}{V}$, where $V = \int dx$ is the volume of $x$-space.

N.B., all we have actually proved is that the constant-probability distribution is either a maximum or a minimum-entropy distribution. It is easy to see that it is, in fact, a maximum.

# Chapter 5 Applying the M.E.F. principle to vision

## Introduction

The M.E.F. theorem described in the previous chapter provides a general approach to the problem of how to process sense data. In this chapter, I will give a brief description of other work which has looked at how visual data should be processed, and then describe how the F.I.G. theorem can be applied to modelling early vision in a more principled way.

Vision is particularly interesting in the context of the M.E.F. principle because the natural environment has particularly striking redundancies (Field [1989]).

## Other work

Any data can be represented in many ways, two of which are particularly relevant with visual data: one can represent an image in purely spatial terms, using an array of pixels, or one can find the Fourier transform of the image and represent it in terms of amplitudes of cos/sin waves. Gabor [1946] showed that if a linear filter element is tuned to respond to pixels in a small area of the image, this filter element will necessarily give little information about the frequency spectrum of the image as a whole. Conversely, a filter element which gives accurate information about a coefficient of the Fourier representation will give very little spatial information. Gabor went on to show that the product of the errors in frequency and spatial measurements for an individual filter element has a minimum value, and that this minimum value is attained by filter elements which are Gabor functions - essentially products of Gaussians and sine waves, with the width of the gaussian function and the frequency of the sine function determining the filter element's selectivity in the normal and frequency spaces respectively.

Daugman [1985] looks more closely at the properties of Gabor filters. Daugman shows that if one is interested only in locating a two-dimensional image in space and frequency-space, a set of Gabor filters, generalised to two dimensions, with varying spatial and frequency orientations/sizes is optimal for measuring a combination of the spatial and Fourier aspects of input data.

Daugman appears to consider that the products of the uncertainties in space and frequency is

the joint entropy of the spatial and frequency measurements, but doesn't explain how or why this product is related to the information theoretic concept of entropy. This leads to a criticism of Daugman's work - there is no justification for the central premise that the brain should analyse both the spatial and frequency representations of input images. There are an infinite number of representations that could be considered, so why choose these two? Later in this chapter, the M.E.F. principle will be applied to simple visual data, and results in some filter elements performing Fourier transforms and others performing spatial localisation, similar to Daugman's filters, but crucially there are other filter elements which don't fit in to the simple dichotomy of space verses frequency representations.

Atick and Redlich have published a series of papers on the problem of determining how filters should be tuned to the environment. Their overall aim in Atick and Redlich [1990][1992] is to find a representation which carries however much mutual information about the input data that the organism needs, and squeezes this mutual information into as small a bandwidth (output power) as possible. This is similar (but not identical) to the criterion adopted here; unfortunately, they don't specify why this criterion is used and their analysis of this principle is confined to looking at only bi-variate statistics so that entropy is approximated by correlation matrices from the start. They use their principle to find a receptive field shape for cells involved in early visual processing, assuming the input data is maximum entropy, but with the proviso of having a specific correlation matrix. After a series of rather complicated simplifications, they end up with a centre-surround receptive field.

Barlow et al [1989] use the term *minimum entropy* to describe codes which are as sparse as possible while still preserving all information. However, data represented in this way will not look like completely unstructured zero-entropy data - it will contain the same structure as the original data.

Mallat [1989] derives a wavelet filter for use with visual data, formed by making different-sized copies of a set of basic filters. Thus the wavelet filter has separate elements which analyse the input data at different scales. The basic filter element shape is similar to a Gabor function. Unfortunately there is no justification for wanting a filter that analyses inputs at different scales - although this seems intuitively reasonable given the fractal nature of many real images.

A general criticism of much work in this field is that although people start off with an aim of

working within an information-theoretic framework, they often approximate with correlation statistics before making any significant progress. As a result, there is no overriding information-theoretic objective. In chapter 4, I have described a concrete objective for any filter, that it should transform input data into a maximum-entropy representation. The rest of this chapter is an example of how this general theorem can be applied to simple bi-variate statistics, and produces interesting filter elements.

### Using the M.E.F. principle for visual data.

As an example, we will consider the input set to be a set of real-life visual images. These have the empirical property that nearby pixels are highly correlated, with the correlation decreasing as the distance between the pixels increases. We can thus construct a correlation matrix $C$ which embodies this property.

In order to make the calculation of correlations simple, we will assume that each input pixel's activation can be both positive and negative, with the mean activation being zero, with a similar assumption made for the transformed representation of the input pixel patterns.

We will represent the 2D array of input pixels as a column vector $A = \begin{pmatrix} a_1 \\ \dots \\ a_n \end{pmatrix}$, whose height will be $n$, the number of pixels in the input 2D array.

Although we will be representing the input pixels in this one-dimensional array, the pixels are assumed to be physically arranged in a square array with (say) $l$ pixels along each side, for the purposes of finding the correlation matrix of the input patterns (whose elements are determined by the physical proximity of pairs of pixels). Then the number of pixels will be $n = l^2$. Hence the correlation matrix will have $n^2 = l^4$ elements. This might be a little confusing at first, but the more obvious way of representing the input pattern as a $l \times l$ matrix would make it impossible to find the correlation matrix of the input pixels.

It turns out that we can transform this input data into a maximum-entropy form using a linear filter. Using the F.I.G. theorem, described in chapter 4, we will try to generate pixels which have the same statistics as the input data (i.e. their correlation matrix is $C$) using the most general linear function $g$ (an $n \times n$ matrix) acting on a set of $n$ random numbers

$B = \begin{pmatrix} b_1 \\ ... \\ b_n \end{pmatrix}$. We generate the input data using $A = gB$, with $g$ being the inverse of the filter function whose coefficients we want to find. We require that the correlation matrix of the generated $A$'s to be equal to $C$.

Because our pixels have zero mean activations, their correlation matrix will be $\langle AA^T \rangle$, so we want $C = \langle AA^T \rangle$, where $\langle ... \rangle$ denotes averaging over all generated patterns i.e. over all random $B$'s. This is $C = \langle (gB)(gB)^T \rangle = \langle gBB^T g^T \rangle$. The matrix $g$ is constant, so we have $C = g\langle BB^T \rangle g^T$. To make the maths simple, we will assume that the random $B$'s have unit variance, so $\langle BB^T \rangle = I$, the unit matrix[10].

Hence we have $C = gg^T$. If we can solve this for $g$, we can then set $f = g^{-1}$ and use this as our ideal filter[11].

There are two techniques for solving this equation. One is to use Cholesky Decomposition which gives a $g$ which is lower diagonal. We will not pursue this method here as this means that the inverse is also lower diagonal, resulting in filters which are highly asymmetric - one filter will only depend on one pixel, the next on the first 2 pixels, and so on. The alternative, explained below, generates an infinite set of solutions. We shall consider only the simplest-to-derive solution first; later we will pick the solution which conforms to a realistic constraint in brains, making the filter elements as localised as possible.

The alternative method of solving $C = gg^T$ is to consider the diagonalisation of the correlation matrix $C$. Diagonalisation of $C$ will generate two matrices $E$ and $D$ such that $C = EDE^{-1}$. $D$ will be a diagonal matrix $\begin{pmatrix} d_1 & 0 & ... \\ 0 & d_2 & ... \\ ... & ... & ... \end{pmatrix}$, where the $d$'s are the eigenvalues of $C$, and $E$ will be a matrix whose rows are the eigenvectors of $C$.

$C$ is a correlation matrix, so is symmetrical and real. This means that $E$ will be orthogonal, i.e. $E^{-1} = E^T$, and all the $d$'s are real. (Boas [1983], page 418, or Press et al [1992], page 357). Hence we have $C = EDE^T$.

[10] This isn't quite the same as requiring that the $B$'s have maximum entropy, but any maximum-entropy distribution with constant additive noise present will have $\langle BB^T \rangle = I$.

[11] Note that it has turned out that the generated correlation matrix is affected only by the correlation matrix of our random $B$'s. The nature of higher-order statistics of the output of our ideal filter when fed with input data $A$ with correlation matrix $C$ will depend on the higher-order statistics of the input data $A$ - in effect, fixing the correlation matrix of the input data doesn't fully determine all statistics of the data set.

Because $D$ is diagonal, we can take its square route to get $\sqrt{D} = \begin{pmatrix} \sqrt{d_1} & 0 & \dots \\ 0 & \sqrt{d_2} & \dots \\ \dots & \dots & \dots \end{pmatrix}$. This is obviously diagonal also, so we can write $C = E\sqrt{D}\sqrt{D}^T E^T$.

Practical correlation matrices turn out to be positive definite, which means that the $d$'s are positive, so $\sqrt{D}$ is real.

Hence we can take $g = E\sqrt{D}$ as the solution of the equation $C = gg^T$. We then invert this to find the ideal filter matrix $f = (E\sqrt{D})^{-1}$.

This operation was carried out numerically using a correlation matrix for a 2 dimensional array of pixels where the correlation between any pair of pixels is a gaussian function $e^{-\frac{1}{2}(\frac{s}{\sigma})^2}$, where $s$ is the distance between the two pixels in the input retinal array. Note that the input pixel array was 12×12, so we represent an image as a 144-element column vector, and the correlation matrix $C$ is a 144×144 matrix.

The resulting filters are shown in Figure 1. Each small square is an image of the 144 weights from the input to one filter element. There are 144 filter elements (the same number as the number of input pixels); they are displayed in a 12×12 array but this has nothing to do with the shape of the input array . It is interesting to note that many of these filter elements can be seen to perform various Fourier transforms on the input. Other elements seem to be contrast detectors.

Note that the filter elements are both positive and negative. A mid-grey level in the diagrams represents a strength of zero, while white and black represent strongly positive and strongly negative respectively.

Many of the filters have very small amplitude. When the width of the gaussian is increased (i.e. widely separated input pixels are highly correlated), this characteristic is magnified, resulting in many filters which have very low sensitivities (figures 2a, 3a). The figures also show 'contrast-enhanced' versions of the filters so that the structure of the filter elements can be seen. Note that each filter element will have the same output variance even though they have very different element strengths, because the input data is smoothly-varying.

Although the filter elements are displayed in a rectangular array, this has no bearing on the rectangular array arrangement of the input pixels. To emphasise this, some of the filter figures show the filter elements in different shaped arrays.

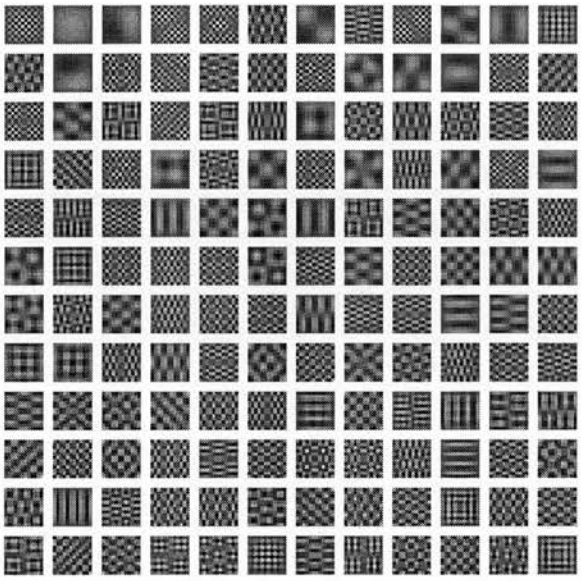


*Figure 1. Maximum-entropy filters for typical visual correlation statistics. Width of the correlation gaussian was 0.5 pixels.*

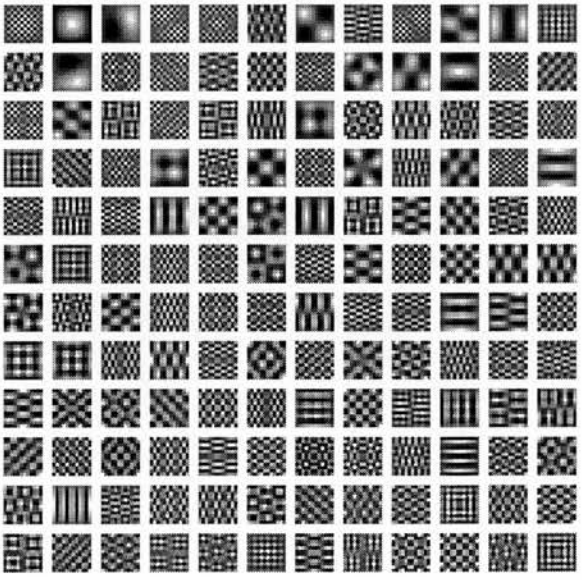


*Figure 1b. As figure 1a, but each filter has been individually contrast-enhanced to show the structure of the filter elements.*

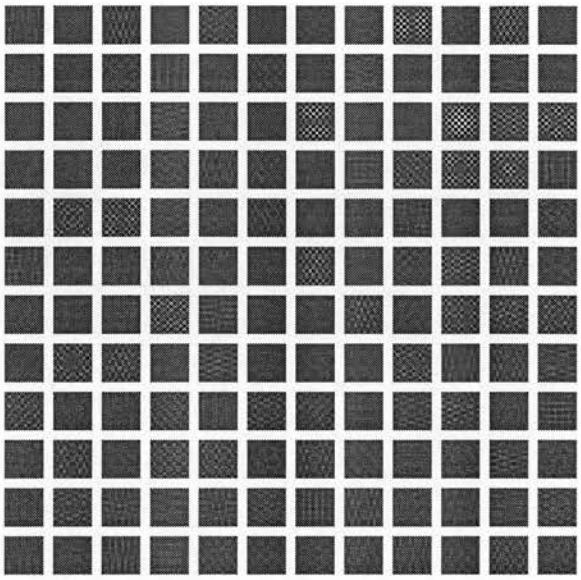


*Figure 2a. Maximum-entropy filters for typical visual correlation statistics. Width of the correlation gaussian was 1 pixel.*

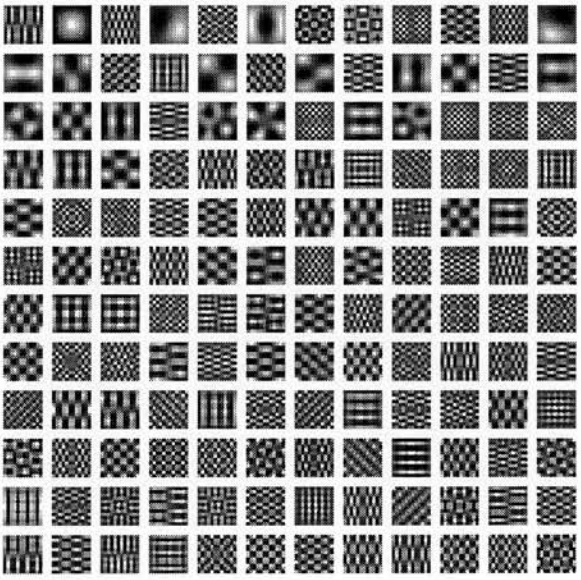


*Figure 2b. As figure 2a, but each filter has been individually contrast-enhanced to show the structure of the filter elements.*

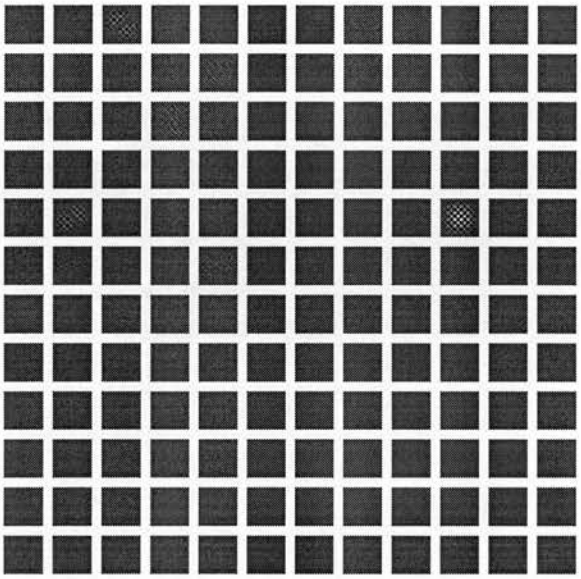


*Figure 3a.*
*As in figure 1, except the width of the gaussian was 2 pixels.*

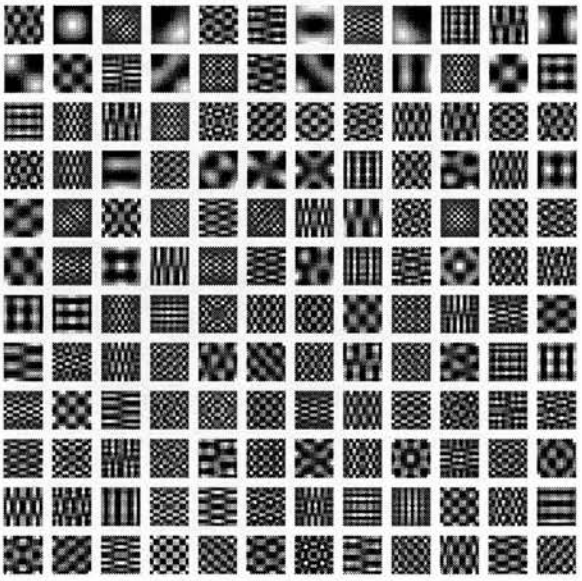


*Figure 3b.*
*As figure 2a, but each filter has been individually contrast-enhanced to show the structure of the filter elements.*

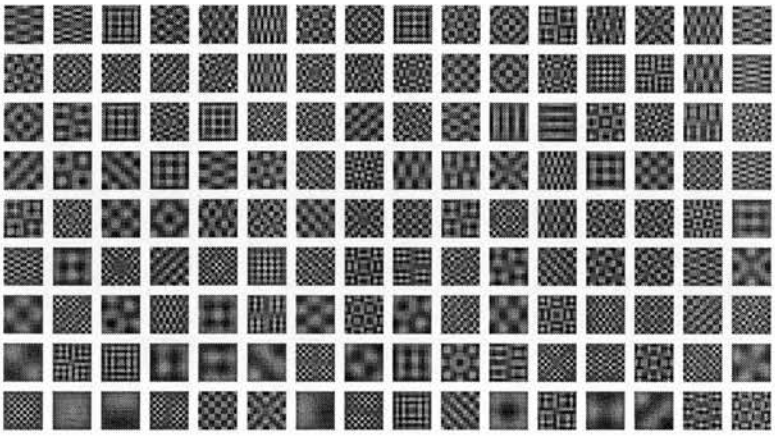


*Figure 4a. As figure 1a, but the original correlation matrix's elements are $e^{-\frac{1s}{2\sigma}}$ i.e. the exponent is linear in s, the distance between pixels. The arrangement of the filter elements is now a 9 ×16 array instead of 12 ×12. This is just a different way of displaying the filter, the calculations involved are identical to the previous cases.*

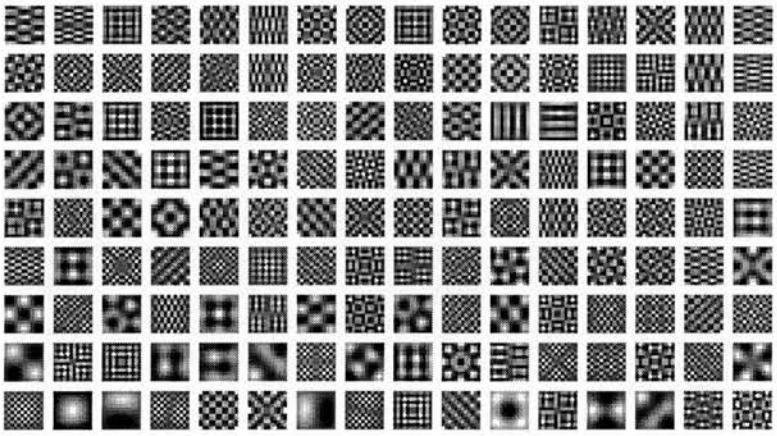


*Figure 4b. As 4a, but with contrast-enhanced filter elements.*

It should be pointed out here that the filters found using the above technique are not unique. If we transform $g \rightarrow \hat{g} = gx$, where $x$ is any orthogonal matrix, then $\hat{g}$ is also a solution of the original equation $C = gg^T$, because $xx^T \equiv I$ is the definition of an orthogonal matrix. This redundancy is used in the next section to make the filter elements localised.

Related to this non-uniqueness is the fact that there are many possible different input datasets which have the same correlation matrix. The procedure described above for determining an ideal filter imposes extra structure on the input dataset. This is easily seen by considering that the redundancy involved in specifying just the correlation matrix means that the exact set of input data is undetermined. However, by using the derived generator $g$,

one can generate an exact input dataset, so somewhere along the line parameters which were unspecified in the original problem have been assigned concrete values. This will be because of the freedom involved with the transformation $g \rightarrow \hat{g} = gx$ mentioned above.

### Localising the filters

The filters described so far have highly non-localised filter elements. This is contrary to what is found in the brain. It is important to note that any localised linear filter can be made into a non-local filter by taking linear combinations of elements, so a general method of finding filter structure such as that described in this paper will always require extra constraints in order to make it local.

One way of making a filter localised would be to divide up the input field into small squares, and make a (non-localised) ideal filter for each square. Hence each square would have a filter made up of elements like those in figures 1-4. However, this approach is not particularly elegant, and would not generalise very well to cases where there are significant redundancies between pixels in different squares. I will pursue a different approach here.

As mentioned earlier, we can transform the filters found previously by using $g \rightarrow \hat{g} = gx$, where $x$ is any orthogonal matrix. Because we are using $f = g^{-1}$, this is equivalent to $f \rightarrow xf$ where, again, $x$ is any orthogonal matrix. In this section, the results of choosing $x$ so that the resulting filters are localised will be described.

The filter elements described so far have connections from all parts of the visual field. In real visual systems, such 100% connectivity is impossible, due to wiring limitations. One can define a cost function for the wiring used by a complete filter $f = xf_0$, by multiplying the strength of each connection by some measure of the physical length of the connection. We can then attempt to minimise this cost function by choosing an appropriate $x$.

This has been done for the following cost function:

$$L = \frac{1}{n^2} \sum_{ba} h(f_{ab}) w(a, b)$$

Here, $h(f_{ab})$ is a function which gives the cost of a connection strength $f_{ab}$, and $w(a, b)$ represents the cost of supporting a connection from input $a$ to output $b$. We will assume the outputs are in a rectangular array similar to the inputs, and that each output is 0 distance from the corresponding input and that the distance between adjacent pixels is 1 unit. This

means that $w$ will be the Pythagorean distance between $a$ and $b$. For $h$, we will use $h(x) = x^2$, as this penalises large negative weights in the same way as it penalises large positive weights.

It should be stressed that these choices do not reflect any detailed knowledge of the topography of the early visual system, but are a reasonable starting point to investigating the likely consequence of forcing biologically-inspired practical constraints on the filter system.

To minimise the cost function $L$, a gradient descent was carried out: the coefficients of the orthogonal matrix $x$ were all changed by small amounts in the direction which lessened $L$. For example, this is a fairly standard procedure used in training supervised neural networks. For the details, see this chapter's appendix.

The minimisation of $L$ is very computationly intensive, so has only been carried out fully for a small filter with 16 inputs arranged as a 4×4 array, and 16 outputs also arranged in a 4×4 array. The filter which gave a minimum $L$ is shown in the next figure:

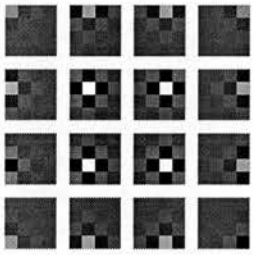


***Figure 5a Localised filters for 4×4 inputs, with gaussian correlation matrix of width 1 pixel.***

As can be seen, these filters are highly localised. As before, a mid-grey level of the filter elements actually represents zero connection strength - the strengths are both positive and negative, so zero is represented as mid-grey.

The cost function $L$ was reduced from 1.61 to 0.241 by the minimisation procedure.

N.B., this filter has been verified to obey $gg^T = C$, where $g = f^{-1}$, so is indeed an ideal filter for the specified correlation matrix.

Each filter element seems to consist of a centre-surround cell, although these differ from those found in real visual systems in that the central region of the centre-surround function is 1 pixel wide.

It is expected that if more detailed statistics were used to describe the input patterns, and the ideal filters calculated, more complicated filter elements would be required. For example, if

we were to take note of the fact that real visual scenes contain many straight lines (of which more are horizontal or vertical than by chance), we would expect the filter elements to show selectivity to straight lines at various angles. To perform the required calculations would be a lot more complicated than just solving $gg^T = C$ however, as the presence of lines would require at least 3rd-order statistics. In fact, it may well be that ideal filters cannot be constructed out of a linear filter model, but would require non-linearity. This would make the calculations even more difficult.

Nevertheless, the results presented here are very interesting, as they suggest that the principle that the brain should transform sense data into a maximum-entropy format does give rise to very natural-looking filter structures, which bear some resemblance to those found in real brains. The principle of maximum-entropy output is also very interesting theoretically, as it seems to represent the natural extension of the de-correlation ideas of Barlow to non-linear considerations.

### Comparison with Atick & Redlich [1990]

The receptive fields resulting from the localised M.E.F. principle are essentially centre-surround, but with the central region consisting of just one pixel. Real centre-surround cells have an extended centre section, as have some of the receptive fields derived in Atick & Redlich [1990]. The size of the centre region in Atick & Redlich [1990] is dependent on the noise width though, and is one pixel wide for signal-to-noise ratios greater than 2 (figure 3, Atick & Redlich [1990]).

Hence one could look on the M.E.F. principle as providing a simple low-noise approximation to the methods of Atick & Redlich. However, this ignores the generality of the M.E.F. principle, which concentrates on forming arbitrarily high-level representations, not simply removing the simplest correlations from input data and filtering out input noise.

### Appendix - Details of minimising a cost function to make a filter localised

Note that much of the following involves some rather clumsy algebra. These details of localising a filter are not relevant to the more important ideas presented earlier about maximum-entropy representations etc., as they are only of use for the special case when the input data has a particular correlation matrix, and the filter is linear. On the other hand, it is

likely that wiring constraints are significant in the brain, so any filter which is tuned to particular input statistics may need 'localising'.

The cost function chosen was $L = \sum_{ba} h(f_{ab}) w(a, b)$, where $h$ is a function which is large for large connection strengths, and $w(a, b)$ is large when input $a$ and output $b$ are physically far apart. Also, $f = xf_0$, where $f_0$ is the non-localised ideal filter, and $x$ is any orthogonal matrix.

We wish to minimise $L$ by changing $x$, while keeping $x$ orthogonal.

To do this, we need some way of specifying $x$ using $\frac{1}{2}(n-1)n$ independent parameters, as a $n \times n$ orthogonal matrix has this many degrees of freedom.

One way of doing this is to represent $x$ as the product of $\frac{1}{2}(n-1)n$ rotation matrices, each of which is a rotation matrix in a different 2D plane in $n$-space. i.e.:

$$\begin{aligned} x &= R_{21}(\theta_{21}).R_{31}(\theta_{31}).R_{32}(\theta_{32})...R_{n(n-2)}(\theta_{n(n-2)}).R_{n(n-1)}(\theta_{n(n-1)}) \\ &= \prod_{i=2}^{n} \prod_{j=1}^{i-1} R_{ij}(\theta_{ij}) \end{aligned}$$

This makes the $\theta_{ij}$ the independent parameters which determine $x$[12]. Each matrix $R_{ij}(\theta)$ is just the identity matrix, except for 4 elements: the $ii$ and $jj$ elements are $\cos\theta$, element $ji$ is -sin $\theta$, and element $ij$ is $\sin\theta$. Note that $R_{ij}$ is only defined for $i > j$, which means there are $\frac{1}{2}(n-1)n$ of them, as required.

To perform the gradient descent, we need to make each $\theta_{ij}$ to change in proportion to $\frac{\partial L}{\partial \theta_{ij}}$, i.e. repeatedly perform, for all $e, f$, the update $\theta_{ef} \rightarrow \theta_{ef} - \eta \frac{\partial L}{\partial \theta_{ef}}$, where $\eta$ is a small 'learnrate' parameter. To calculate the derivative, we use the chain rule:

$$\begin{aligned} \frac{\partial L}{\partial \theta_{ef}} &= \sum_{abcd} \frac{\partial L}{\partial f_{ab}} \frac{\partial f_{ab}}{\partial x_{cd}} \frac{\partial x_{cd}}{\partial \theta_{ef}} \\ &= \sum_{abcd} \frac{\partial h(f_{ab})}{\partial f_{ab}} w(a, b) \frac{\partial (xf^0)_{ab}}{\partial x_{cd}} \frac{\partial x_{cd}}{\partial \theta_{ef}} \\ &= \sum_{abcd} h'(f_{ab}) w(a, b) \frac{\partial}{\partial x_{cd}} \sum_{p} x_{ap} f^0_{pb} \frac{\partial x_{cd}}{\partial \theta_{ef}} \\ &= \sum_{abcd} h'(f_{ab}) w(a, b) \delta_{cd} f^0_{db} \frac{\partial x_{cd}}{\partial \theta_{ef}} \end{aligned}$$

[12] In fact this representation of $x$ always has $|x| = 1$. The most general $x$ would have $|x| = \pm 1$. The negative-determinant $x$ can be found by swapping any pair of rows of $x$. This was not done here because the localisation process worked very well with positive-determinant $x$.

$$= \sum_{bcd} h'(f_{cb})\, w(c, b)\, f^0_{db} \frac{\partial x_{cd}}{\partial \theta_{ef}}$$

All that remains is to find $\frac{\partial x_{cd}}{\partial \theta_{ef}}$. Because of the rather complicated form of $x$ when expressed in terms of the $\theta$'s, this is rather ungainly. One has to differentiate just the $R_{ef}$ term in the multiplication series $\prod_{i=2}^{n} \prod_{j=1}^{i-1} R_{ij}(\theta_{ij})$:

$$x = \prod_{i=2}^{n} \prod_{j=1}^{i-1} R_{ij}(\theta_{ij})$$

$$= \prod_{i>j} R_{ij}(\theta_{ij})$$

so: $$\frac{\partial x_{cd}}{\partial \theta_{ef}} = \left\{ \prod_{i>j,\,(i,j)<(e,f)} R_{ij}(\theta_{ij}) \right\} \left\{ \frac{dR_{ij}(\theta_{ij})}{d\theta_{ef}} \right\} \left\{ \prod_{i>j,\,(i,j)>(e,f)} R_{ij}(\theta_{ij}) \right\}$$

This completes the derivation.

# Chapter 6 A Neural network implementation of the M.E.F. principle

## Introduction

The M.E.F. principle that sense data should be transformed into a maximum entropy form seems to have interesting consequences, so it is reasonable to try to accomplish this using a neural network.

Unfortunately, although the F.I.G. theorem provides a useful approach to finding an ideal filter given the statistics of its input data, it is not much use when we want a general learning algorithm to make a network learn the best set of weights for *any* input statistics. Even if we used a supervised learning algorithm, with the aim of maximising the output entropy, the simple solution of calculating the entropy and using it as an error signal is intractable because finding the output entropy involves integrating over all of output space, which is impossible (for example, if there are 100 outputs, there will be $2^{100} \approx 10^{30}$ different points in output space to consider if we just consider each unit as being either on or off; taking into account continuous activations will make matters even worse).

The actual entropy we are trying to maximise is the entropy of a probability distribution which is formed from a discrete set of $n$ input patterns (assuming a normal neural network paradigm with discrete input patterns), where the output of the network is distorted by gaussian noise. Hence the output probability distribution is a set of $n$ gaussians superimposed. All that changing the network weights will do is move these gaussians around within output space. The entropy of these outputs is still impossible to calculate exactly though.

One can view the transformation performed by a neural network (or any other 'black box') as a mapping from one high-dimensional space to another. With 100 input lines, the input space has 100 dimensions. A feed-forward neural network will always give a mapping which is smooth, in the sense that sufficiently similar input patterns will map to similar output patterns, i.e. nearby points in the input space will map to nearby points in output space. With noise present, it is important that separate points in output space are sufficiently far apart that the noise will not confuse them. This is more important for points that occur very frequently, so one is led to the conclusion that high-frequency points should be given

more 'room' in output space.

For a general review of information-theoretic neural network learning algorithms, see Plumbley [1994].

### Making a network output high entropy

We will be considering a single layer network, trained using a number of training patterns. We will try to form a learning rule which will change the network's weights in such a way that the network becomes 'tuned' to the statistics of the training dataset, so as to maximise the information that the output conveys about the input, *for patterns in the training set.* In this way, the network will learn a coding scheme which best fits the environment it is in.

As before, we will let $\boldsymbol{A}$ and $\boldsymbol{B}$ be the input and output vectors respectively, and $p(\boldsymbol{B})$ be the probability density function for the output vector, etc. Also, input pattern number $p$ will be $\boldsymbol{A}_p$, and the output vector before noise will be $\boldsymbol{B}_p = \boldsymbol{B}(\boldsymbol{A}_p)$.

The output entropy will be $H(\boldsymbol{B}) = -\int d\boldsymbol{B}\, p(\boldsymbol{B}) \operatorname{Log}(p(\boldsymbol{B}))$

If there is no noise this is straightforward; with noise, the calculation gets tricky: we need to integrate over *all* of output space, using $p(\boldsymbol{B})$, the probability with which any particular output pattern $\boldsymbol{B}$ occurs. The problem is that with noise, any output $\boldsymbol{B}$ can arise from any input, though with very small probability if $\boldsymbol{B}$ is very different from $\boldsymbol{B}_p$, the output before noise. This is because $p(\boldsymbol{B} \mid \boldsymbol{A}_p)$ is a Gaussian centred on $\boldsymbol{B}_p = \boldsymbol{B}(\boldsymbol{A}_p)$. With $n$-dimensional Gaussian noise for example, $p(\boldsymbol{B} \mid \boldsymbol{A}_p) = \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{1}{2}\left(\frac{\boldsymbol{B}-\boldsymbol{B}_p}{\sigma}\right)^2}$, where $\boldsymbol{B}_p$ is what the network would output for pattern number $p$ if there was no noise. This probability is non-zero everywhere (though small when $\boldsymbol{B}$ is very far from $\boldsymbol{B}_p$).

We need $p(\boldsymbol{B})$ to find the output entropy. We know $p(\boldsymbol{B} \mid \boldsymbol{A}_p)$, so:

$$p(\boldsymbol{B}) = \Sigma_p p(\boldsymbol{A}_p) p(\boldsymbol{B} \mid \boldsymbol{A}_p) = \Sigma_p p(\boldsymbol{A}_p) \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{1}{2}\left(\frac{\boldsymbol{B}-\boldsymbol{B}_p}{\sigma}\right)^2}$$

More generally, $p(\boldsymbol{B}) = \Sigma_p p\left(\text{input pattern } p\right) \times p(\textit{noise} = \boldsymbol{B} - B(\boldsymbol{A}_p))$, which is the convolution of the noise with the output probability distribution before noise.

For Gaussian noise, the output probability distribution is the sum of Gaussians, each centred on the output for a particular input pattern, $B(\boldsymbol{A}_p)$. In this case, the entropy of this distribution is:

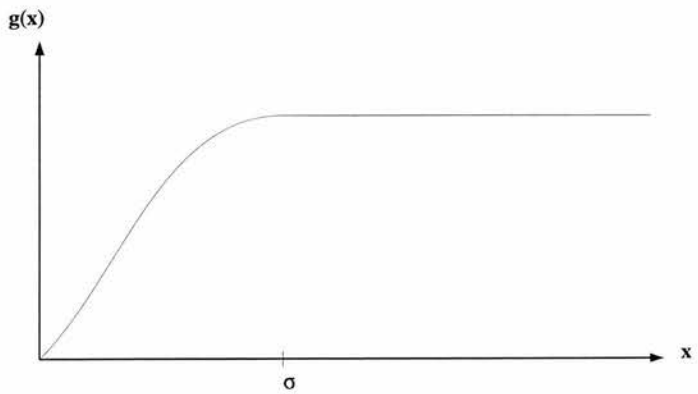
$$
\begin{aligned}
H(\boldsymbol{B}) &= -\int d\boldsymbol{B}\, p(\boldsymbol{B}) \operatorname{Log}[p(\boldsymbol{B})] \\
&= -\int d\boldsymbol{B} \left[\int d\boldsymbol{A}\, p(\boldsymbol{A}) p(\boldsymbol{B} \mid \boldsymbol{A})\right] \operatorname{Log}\left[\int d\boldsymbol{A}'\, p(\boldsymbol{A}') p(\boldsymbol{B} \mid \boldsymbol{A}')\right] \\
&= -\int d\boldsymbol{B} \left[\int d\boldsymbol{A}\, p(\boldsymbol{A}) \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{1}{2}\left(\frac{\boldsymbol{B}-\boldsymbol{B}(\boldsymbol{A})}{\sigma}\right)^2}\right] \operatorname{Log}\left[\int d\hat{\boldsymbol{A}}\, p(\hat{\boldsymbol{A}}) \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{1}{2}\left(\frac{\boldsymbol{B}-\boldsymbol{B}(\hat{\boldsymbol{A}})}{\sigma}\right)^2}\right] \\
&= \int d\boldsymbol{B}\, h\left\{\int d\boldsymbol{A}\, p(\boldsymbol{A}) \frac{1}{(\sigma\sqrt{2\pi})^n} e^{-\frac{1}{2}\left(\frac{\boldsymbol{B}-\boldsymbol{B}(\boldsymbol{A})}{\sigma}\right)^2}\right\} \qquad \text{where } h(x) = -x \operatorname{Log}(x)
\end{aligned}
$$

There is no way of calculating this analytically though.

We would like a learning rule which maximised this expression for $H(\boldsymbol{B})$. It is pretty intractable though, so we cannot simply differentiate it w.r.t. each individual weight to get a learning rule. What we *can* do is to consider what sort of output distribution $p(b)$ would give the maximum entropy. The probability distribution which gives the maximum entropy is a *flat* distribution, i.e. $p(\boldsymbol{b}) =$ constant. This is intuitive because entropy is a measure of how unpredictable something is - there is nothing more unpredictable than a variable which has a flat probability distribution.

With Gaussian noise our output probability distribution from the network is a set of identical Gaussians, each centred on the network output for a particular input pattern. We have to try and maximise the entropy of this distribution by moving the centres of the Gaussians around in output space, so that the resulting distribution is as flat as possible. Clearly, nothing is going make this distribution completely flat, but what we can do is to avoid having Gaussians overlap each other too much, as this will result in pronounced peaks and troughs.

One solution is to try and ensure that the distances between all pairs of patterns is greater than the noise width: Consider an energy function $E = \sum_{p\hat{p}} g\left[(\boldsymbol{B}_p - \boldsymbol{B}_{\hat{p}})^2\right]$, where $g(x)$ is like:



If we try to maximise this energy function, the patterns will try to be at least a distance $\sigma$ apart, resulting in a high output entropy, though not necessarily the highest possible entropy. In fact, one *cannot* get the maximum entropy using just the simple pair-wise comparison implied by $E = \sum_{p\hat{p}} g\left[(\boldsymbol{B}_p - \boldsymbol{B}_{\hat{p}})^2\right]$, as entropy is only maximised when there is *no* structure in the data. The expression for $E$ would not be effected by the presence of correlations between triplets of patterns for instance, whereas these 3-way correlations would certainly effect the output entropy[13].

We can take the derivative of $E$ with respect to each weight in the network, and so find a learning rule which maximises $E$. One ends up with $\Delta_p w_{ba} = 4A_a B'_{bp} \sum_{\hat{p}} g_{p\hat{p}}' \left(B_{bp} - B_{b\hat{p}}\right)$ (see this chapter's appendix for the derivation). However, due to the non-linear nature of the $g(x)$ above, this learning rule is highly non-local. We can simplify things by setting $g(x) = x$. The energy function in this case is simply $E = \sum_{p\hat{p}} \left(\boldsymbol{B}_p - \boldsymbol{B}_{\hat{p}}\right)^2$. Before, moving patterns much further apart than $\sigma$ made no difference to the energy. This reflects the very small increase in entropy achieved by doing this since the Gaussians already overlap very little if they are separated by more than $\sigma$. Using $g(x) = x$ would seem to be a very crude approximation indeed. However, it leads to an extremely simple and almost local learning rule, and, while it is certainly not a true measure of the output entropy, maximising it will certainly make the entropy large.

A true entropy-maximising procedure will try to move the the network's output points further apart, in output space. If a particular input pattern occurs very frequently, it will be given more room in output space. This will lead to output-space being given preferentially to frequently occurring patterns. The simpler linear energy function will do this, but also try to move *all* points further apart even if they are already more than $\sigma$ apart already, which will interfere with this process to some degree. The linear case effectively assumes a very high level of noise - in this case $\sigma$ is very large, so the curve for $g(x)$ doesn't tail off for reasonable $x$. Interestingly, this linear case ends up with all outputs at the extremes of each units output; This is reasonable, since the best a unit can do with very high noise is to try and separate the inputs into just 2 classes. Later on, I will use weight decay and an auto-adjusting threshold to make the units represent more than this minimal amount of information, while still using the basic simple learning rule which results from the choice of

[13] As an example, consider the following set of input patterns: 000 011 101 110. The correlation between any pair of inputs is zero, but there are only 4 possible input patterns, so one can represent the input data with 2 bits, despite there being 3 input bits, all uncorrelated. The input patterns are derived from the XOR truth table, where any input is the XOR of the other two inputs, which explains why the patterns' entropy is only 2 bits.

$g(x) = x$.

If we set $g(x) = x$, we get $g_{p\hat{p}}' \equiv 1$, so the learning rule becomes $\Delta_p w_{ba} = 4A_a B_{bp}' \sum_{\hat{p}} (B_{bp} - B_{b\hat{p}})$. This can be simplified because $B_{bp}'$ is not effected by the summation parameter $\hat{p}$, so can be taken out of the summation. This leads to $\Delta_p w_{ba} = \left(B_{bp} - \frac{1}{n_p} \sum_{\hat{p}} B_{b\hat{p}}\right) A_a . B_{bp}' . 4n_p$ where $n_p$ is the number of patterns. This is similar to a Hebbian rule, except that the state of the unit is replaced by the deviation of the unit from its average activity level $\frac{1}{n_p} \sum_{\hat{p}} B_{b\hat{p}}$. The $B_{bp}'$ term arises because we have performed a differentiation on a quantity which depends on the states of the units, and this involves the response function, (this is similar to what happens in the derivation of the standard back-propagation learning rule). The constant $4n_p$ can be safely included in a learning rate.

This learning rule will behave in the following way: if unit $b$ is more strongly activated than normal, the weight change will make the unit even more strongly activated next time. On the other hand, if unit $b$ is activated more weakly than normal for input pattern $p$, the weights will change so that next it will be even less activated.

The big problem with this learning rule as described is that the optimum pattern of activations for a unit is to be maximally activated for $\frac{1}{2}$ of the patterns, and totally off for the other $\frac{1}{2}$. This can be seen in two ways:

- The weight increase $\Delta_p w_{ba}$ will always move outputs away from the average. For example, if the response functions has a maximum output range of $0 \rightarrow 4$, and there are 4 patterns giving activations 1, 2, 3 and 4, the learning rule will still try to make the 3rd larger and the 2nd smaller, leading to activations like: 1, $1\frac{1}{2}$, $3\frac{1}{2}$ and 4. This clearly has a lower entropy as the previous outputs were perfectly evenly spaced.

- As an example, one can calculate the energy $E = \sum_{p\hat{p}} (B_p - B_{\hat{p}})^2$ for two 1-dimensional distributions, each with ten values, such as $\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and $\{0, 0, 0, 0, 0, 1, 1, 1, 1, 1\}$. The first turns out to be 16.5, while the second is 50.0.

This occurs as a direct result of setting $g(x) = x$, instead of a function which asymptotes for $x \gg \sigma$. The linear version is much more sensitive to movement of pairs of patterns which are already widely separated.

A possible way around this would be to compare the activation for the present pattern with only those other patterns which gave similar activations. However, this would involve very complicated calculating inside the unit. Also the unit would have to store its complete activation density - how many times it has had an activations 0.1, how many times 0.2 etc.

If we use a network of units which use the above learning rule, there is a significant problem: all the units will develop the same weights. Again, this can be understood in two ways:

- Each unit has exactly the same inputs, and the same learning rule, so unless small variations in initial weight sizes affect the way the weight vector changes, each unit will develop exactly the same weights.
- The energy function $E = \sum_{p\hat{p}} g\,(B_p - B_{\hat{p}})^2$ doesn't have any terms which compare different units' activations for the same pattern. This is because it is only an approximation to the entropy of the output probability distribution. The true output entropy is very complicated, with integrals of logs etc.

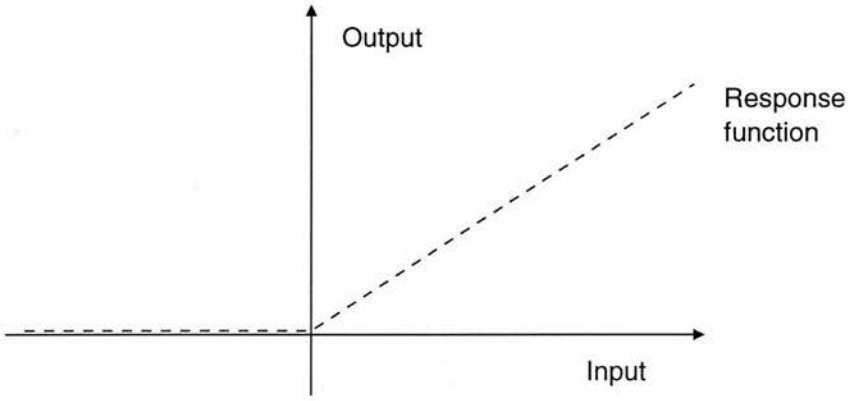
This is a general problem for neural network learning rules: back-propagation avoids it by using error terms to amplify any small initial variations in units weight vectors, while lateral connections can be looked on as another way of forcing units to behave differently. If the true expression for the output entropy could being used, it would certainly contain terms which involved comparisons between the activations of different units for the same pattern.

A particularly simple way of making units have different activation, which is suggested by the previous learning rule, is to maximise the function $S = \frac{1}{4n} \sum_{pb\hat{b}} (B_{bp} - B_{\hat{b}p})^2$, where $n$ is the number of patterns. This is maximised when the activations of all pairs of units are very different for all patterns. This function is very similar to the above one, except that the summation is $\sum_{pb\hat{b}}$, rather than $\sum_{p\hat{p}b}$. The means that it will suffer from the same problem where the optimum network performance is to have half of the units maximally on, and the other half maximally off for all patterns. To cure this, the simulations described later use weight decay to limit units' weights in order to counter this, and the response function is positive for positive net input, and zero otherwise. The learning rule obtained by differentiating $S$ is derived in a similar way as before. It turns out to be: $\Delta_p w_{ba} = f'_{bp} A_a \left(B_{a\mu} - \frac{1}{n} \sum_i B_{i\mu}\right)$, where $n$ is the number of units. This time, the rule is a Hebb rule, except that the unit's activation is replaced by the departure of the unit's activation from

the average activation of all units for the current pattern. This is *non-local*, because it involves knowledge of the average activation of all the units. Previously, the extra term in the learning rule was the average activation of *one* units activation for *all* patterns; here, the extra term is the average activation of *all* the units for *one* input pattern.

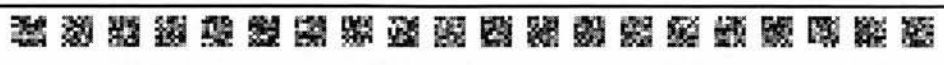
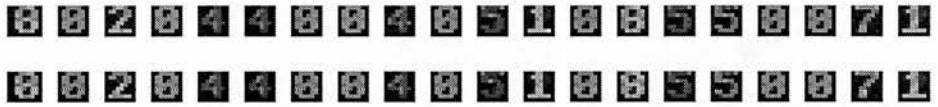
As before, the learning algorithm will always try to make the outputs 0 or 1, so I have used weight decay to prevent this happening, and also an adjustable threshold to ensure that units always have approximately equal activations. Without this last feature, if one unit was on average less activated than the others, it would always be 'pushed away' from higher activation, resulting in it being off all the time.

Even with weight decay and an adjustable threshold, there are still problems to be overcome. For instance, with outputs that can be negative as well as positive, the network will still end up with the units falling into two type, with one having opposite weights to the other. This problem is cured by ensuring that the minimum activation is 0 (which occurs in real neurons anyway) by having a response function that looks like:




To test the algorithm, I have used an 8*8 input array, with the input patterns being the numbers 0-9 in a pixel format. The use of a visual input means that we can look at the weights for each unit and get a good idea of what the unit will respond to. In this case, we would like units to respond to particular numbers. This is for two reasons: first, this would be a cognitively useful thing to do for any input data which consists of fixed patterns and, second, this will give a high output entropy (though not necessarily the highest possible output entropy if digits occur with different frequencies).

The weights evolved as follows:

| Iteration | |
|---|---|
| 0 |  |
| 20 |  |
| 40 |  |
| 60 |  |
| 80 |  |
| 100 |  |
| 120 |  |
| 140 |  |
| 160 |  |
| 180 |  |
| 200 |  |
| 220 |  |
| 240 |  |
| 260 |  |
| 280 |  |
| 300 |  |

Learnrate was 0.001, weight decay 0.001, threshold learnrate 0.1, The response function was half of a cosine wave running from (0,0) to (1,1); using a positive-only linear function as in an earlier diagram gives similar results.

In the above table, the weights for each unit are represented by the pixels in each 8*8 block. Dark pixels represent a weight of below average strength, while lighter pixels are weights of above average strength. It is readily seen that units are learning to respond to different input patterns.

After the above training, the unit states for each pattern were (blanks mean less than 0.0005):

| | Unit number | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pattern | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 0 | 0.769 | 0.790 | 0.006 | 0.804 | 0.052 | 0.052 | 0.814 | 0.800 | 0.048 | 0.819 | 0.025 | | 0.803 | 0.778 | 0.020 | 0.022 | 0.789 | 0.831 | 0.009 | |
| 1 | | | | | 0.013 | 0.015 | | | | | | 0.917 | | | | | | | | 0.924 |
| 2 | | | 0.958 | | | | | | | | | 0.057 | | | | | | | 0.146 | 0.073 |
| 3 | | | 0.002 | | | | | | | | 0.007 | | | | 0.027 | | | | | |
| 4 | | | | | 0.986 | 0.986 | | | 0.984 | | | | | | | | | | | |
| 5 | | | | | | | | | | | 0.966 | | | | 0.967 | 0.961 | | | | |
| 6 | | | | | 0.007 | 0.012 | | | 0.006 | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | 0.873 | |
| 8 | 0.274 | 0.251 | | 0.241 | | | 0.236 | 0.247 | | 0.231 | 0.002 | | 0.241 | 0.277 | 0.006 | | 0.251 | 0.224 | | |
| 9 | | | | | | | | | | | | | | | 0.001 | | | | | |

It should be pointed out that while, ideally, each input pattern should be responded to by 2 units, this is far from true - there are no units which responded strongly to the input patterns '9', '3', or '6' for example. On the other hand, the learning rule has certainly led to units responding to different input patterns.

#### Comparison with Földiák

In section 4.4 of (Földiák [1992]), a neuron network model is described (model 6) which performs a similar task to the one above, except that the input patterns were from the full standard alphabet of alpha-numeric characters. Földiák's model uses non-linear units and anti-Hebbian lateral connections between each pair of units together with an adjustable threshold for each unit which keeps the average activation of each unit constant.

It gives a much better performance than the one described above, since there is significantly less overlap in the unit responses. Also, the input patterns were presented according to the frequency with which they occur in English text, and the network generates sparser representations for high-frequency patterns.

However, it should be noted that Földiák's model uses anti-Hebbian inhibitory connections between every pair of units in order to decorrelate them, whereas the above model is allowed just one signal (the average activation of all units) to perform the same task. Also, the Földiák model would require a significant amount of time to settle to a stable output, whereas the model described above would give a stable output immediately.

## Appendix

Derivation of the learning rule for ascent of the error surface $E = \sum_{\hat{p}\hat{\hat{p}}} g\left[\sum_{\hat{b}} (B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})^2\right]$

We need $\frac{\partial E}{\partial w_{ba}}$, i.e. the derivative with respect to any individual weight. This is $\frac{\partial E}{\partial w_{ba}} = \sum_p \frac{\partial E}{\partial B_{bp}} \frac{\partial B_{bp}}{\partial w_{ba}}$, where $p$ enumerates each input pattern, since the only links between the weight $w_{ba}$ and $E$ are the states of unit $b$ for each pattern $p$, $B_{bp}$.

$\frac{\partial B_{bp}}{\partial w_{ba}}$ is easy to find: $B_{bp} = f\left(\sum_{\hat{a}} w_{b\hat{a}} A_{\hat{a}}\right)$, so $\frac{\partial B_{bp}}{\partial w_{ba}} = A_a f'\left(\sum_{\hat{a}} w_{b\hat{a}} A_{\hat{a}}\right)$.

Meanwhile:

$$\begin{aligned}
\frac{\partial E}{\partial B_{bp}} &= \frac{\partial}{\partial B_{bp}} \sum_{\hat{p}\hat{\hat{p}}} g\left[\sum_{\hat{b}} (B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})^2\right] \\
&= \sum_{\hat{p}\hat{\hat{p}}} \frac{\partial g\left[\sum_{\hat{b}} (B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})^2\right]}{\partial \sum_{\hat{b}} (B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})^2} \cdot \frac{\partial \sum_{\hat{b}} (B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})^2}{\partial B_{bp}} \\
&= \sum_{\hat{p}\hat{\hat{p}}} g'\left(\sum_{\hat{b}} (B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})^2\right) \cdot \left[\sum_{\hat{b}} 2\delta_{b\hat{b}} (\delta_{p\hat{p}} - \delta_{p\hat{\hat{p}}})(B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})\right] \\
&= \sum_{\hat{p}\hat{\hat{p}}} g_{\hat{p}\hat{\hat{p}}}' \cdot \sum_{\hat{b}} \left[2\delta_{b\hat{b}} (\delta_{p\hat{p}} - \delta_{p\hat{\hat{p}}})(B_{\hat{b}\hat{p}} - B_{\hat{b}\hat{\hat{p}}})\right] \\
&= \sum_{\hat{p}\hat{\hat{p}}} g_{\hat{p}\hat{\hat{p}}}' \cdot \left[2 (\delta_{p\hat{p}} - \delta_{p\hat{\hat{p}}})(B_{b\hat{p}} - B_{b\hat{\hat{p}}})\right]
\end{aligned}$$

The expression inside the summation is symmetrical under $\hat{p} \leftrightarrow \hat{\hat{p}}$ so we can take twice the $\delta_{p\hat{p}}$ term:

$$\frac{\partial E}{\partial B_{bp}} = 4 \sum_{\hat{p}\hat{\hat{p}}} g_{\hat{p}\hat{\hat{p}}}' \delta_{p\hat{\hat{p}}} (B_{b\hat{\hat{p}}} - B_{b\hat{p}})$$

$$\frac{\partial E}{\partial B_{bp}} = 4 \sum_{\hat{p}} g_{p\hat{p}}' (B_{bp} - B_{b\hat{p}})$$

So the required derivative is:

$$\begin{aligned}
\frac{\partial E}{\partial w_{ba}} &= \sum_p \frac{\partial B_{bp}}{\partial w_{ba}} \frac{\partial E}{\partial B_{bp}} \\
&= \sum_p 4 A_a f'_{bp} \sum_{\hat{p}} g_{p\hat{p}}' (B_{bp} - B_{b\hat{p}})
\end{aligned}$$

Note that if a learning rule is written so that $\Delta_p w_{ba}$ is the change of the weight from input $a$ to unit $b$, after presentation of input pattern $p$, then for optimum gradient ascent we need

$\sum_p \Delta_p w_{ba} \propto \frac{\partial E}{\partial w_{ba}}$. Here, this is: $\sum_p \Delta_p w_{ba} \propto \sum_p 4A_a f'_{bp} \sum_{\hat{p}} g_{p\hat{p}}{}' \left(B_{bp} - B_{b\hat{p}}\right)$. Conveniently, $\frac{\partial E}{\partial w_{ba}}$ is already in the form $\sum_p(...)$, so we can simply use, for our learning rule, $\Delta_p w_{ba} = 4A_a f'_{bp} \sum_{\hat{p}} g_{p\hat{p}}{}' \left(B_{bp} - B_{b\hat{p}}\right)$.

# Chapter 7 Conclusion

## Summary of this thesis

Somehow, sequential thought processes etc. arise out of the behaviour of the brain's networks of neurons. I think that having information available in high-level forms, possibly organised in a hierarchical way, is likely to be very important if these processes are to take place. For example, any new subject is difficult to learn but, once learnt, seems almost trivial.

Hence I suggest that learning an appropriate representation for information is a significant part of learning how to understand and use that information, and that any mechanism which could learn to represent information efficiently would therefore be invaluable to the brain. It seems that a lot of high-level information can be obtained 'for free' purely from looking at the statistics of the data. These two points taken together mean that it is likely that the brain will make use of unsupervised methods to transform sense data into efficient representations (Földiák [1992]; Hinton [1989]).

The work presented in this thesis, particularly in chapters 4-6, describes a very general, but precisely defined, mathematical objective (maximum-entropy, or uniform probability, representations) for such unsupervised systems, which corresponds very closely to intuitive ideas about what high-level representations are.

The F.I.G idea described in chapter 4 shows how the M.E.F. principle can be reversed, so that the inverse of a M.E.F. filter (generator) will transform random data into data with the same statistics as the environment. If one possesses such a generator, one effectively possesses a model of the environment. Since inverting a filter doesn't require any additional knowledge, learning to transform sense data into a maximum-entropy representation is equivalent to learning a model of the environment.

It is unlikely that a general neural network learning algorithm could be found which always learns to transform input data into a maximum-entropy representation. However, simple decorrelation of pairs of non-linear units could go some way to achieving this for certain classes of input patterns, while the use of a global average-activation signal as described in chapter 6 may also be important.

The general applicability of information theory to arbitrarily high-level representations means that these ideas may have relevance far beyond pre-processing of sense data (like the early stages of vision), and be useful for investigating how the brain performs all those high-level functions that we tend to take for granted until we try to model them on a computer.

## Future work

On a more practical level, the work described in this thesis suggests two further areas to investigate: applying the M.E.F. principle to more complicated statistics, and developing the rather poor performance of the neural network model which tries to transform its input into an maximum-entropy representation, in accordance with M.E.F.

### Applying the M.E.F. principle to more complicated statistics

The M.E.F. principle is very general, and so could be applied to a more complete statistical model of everyday images or other types of sense data.

For example, everyday images contain more straight lines than would occur by chance (of which many are either horizontal or vertical, e.g. the horizon, tree-trunks). In a pixel representation, straight lines being present result in triplets of pixels arranged in a line on the retina being correlated.

One way of embodying this in a statistical way is that, instead of the essential characteristics of the data being a particular correlation matrix, there would be a three-dimensional matrix $L$ whose elements are $L_{abc} = \langle A_a A_b A_c \rangle$, where $\langle ... \rangle$ is, again, averaging over all patterns. Certain elements of this matrix (the ones corresponding to three pixels which are in a straight line - remember that there is an element in $L$ for every set of three pixels on the retina), would be large, while the other elements would be near-zero.

This is analogous to the correlation-matrix case described in chapter 5 where only the elements of the correlation matrix which corresponded to *nearby* pairs of pixels were large.

Hence the procedure would be to try to generate pixel-images $\boldsymbol{A}$, from random maximum-entropy numbers, in such a way that the three-dimensional matrix $L_{abc} = \langle A_a A_b A_c \rangle$ was the same as some pre-determined matrix which embodies the above-average presence of straight lines. Then one could invert this generator to find the ideal filter for images which contain many straight lines.

Note that one couldn't use a simple linear generator (and filter) as in the correlation-matrix case because, for $n$ inputs, a generator would have $n^2$ degrees of freedom, while the matrix equation $L_{abc} = \langle A_a A_b A_c \rangle$ would be $n^3$ normal equations. Hence one would be trying to solve $n^3$ equations with only $n^2$ unknowns[14].

This means that the ideal filter for image data which has many straight lines would have to be a non-linear filter. This would make the analysis very difficult. There might be a way of doing some sort of numerical optimisation on a non-linear function which has enough degrees of freedom; for example define a non-linear function with enough degrees of freedom, and then do a gradient descent with an error function which is calculated as the difference between the generated matrix $L$, and the desired matrix which embodies straight lines.

An alternative is to forget about the F.I.G. theorem, and optimise a non-linear filter directly, with an error function which approximates the output entropy of the filter when presented with appropriate (i.e. line-containing) input data. This would be similar to the neural network learning maximum-entropy learning rule described in chapter 7.

#### Developing the maximum-entropy neural network learning rule

The learning rule described in chapter 7 falls some way short of reliably maximising its output entropy. In some respects, the Holy Grail of unsupervised networks learning rules in this thesis is a learning rule that always transforms its input into a higher-entropy output. The reason for this is that one could then make a multi-layer network which would always learn to output a high-entropy representation of whatever its input data and, if enough layers were present, the network would end up detecting very high-level features indeed, give or take local-minima type problems.

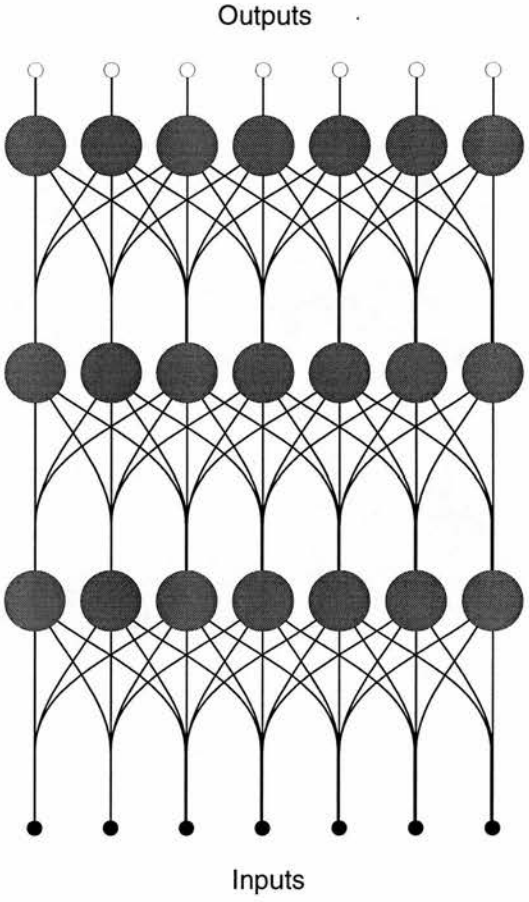
One particularly interesting way of developing the learning rule is to use it in a neural network which has limited connectivity. The reason for this is that the use of a single number representing the average activities of all the units doesn't convey very much

[14] In fact, some of these equations would be identical, because $L = \langle A_i A_j A_k \rangle$ is symmetrical under any permutation of $i, j$ and $k$, so instead of there being $n^3$ equations, there would be something like $n(n-1)(n-1)$ equations. This would still be more than $n^2$ for any reasonable $n$.

Incidentally, in chapter 5, the degeneracy of the solution for the correlation-matrix case, which was used to make the filters localised, can be viewed in a similar way: there were $n^2$ unknowns in the linear generator, but the correlation matrix necessarily had only $\frac{1}{2}n(n+1)$ degrees of freedom because it is symmetrical. The orthogonal matrix used to remove this degeneracy was $x = \prod_{i=2}^{n} \prod_{j=1}^{i-1} R_{ij}(\theta_{ij})$, so had $\frac{1}{2}n(n-1)$ coefficients $\theta_{ij}$, which is exactly the number required to determine the generator exactly.

information to each individual unit, so the units end up duplicating each other a fair amount, as can be seen in the network described in chapter 6. This problem would clearly get much worse as the number of units is increased.

Having limited connectivity would help this situation, although I have no intuitions about how large the effect would be, as very little work has been done on the effects of limited connectivity. Also, there would be a natural hierarchical structure in such a network if it had multiple layers:




In this sort of network, only small groups of units would be in danger of duplicating each other, so the learning rule from chapter 6 would probably work well. The hierarchical structure might help to make successive layers represent the input pattern in successively higher-level ways. For example, the units in each layer are progressively influenced by more of the input pattern the higher up the layer is.

It is important to note that for this sort of scheme to work well, the input data would have to

have some sort of localised structure.

A generalisation of this would be to consider an amorphous limited-connectivity network, i.e. a network which isn't organised into strict layers. The reason why this would be interesting is that sometimes a high-level representation would contain elements which are explicitly present in the input representation. For example, if the input data is a picture of a car, the high-level representation would give the make and model of the car etc., and also its colour. The colour of the car is, to a large extent at least, given explicitly by the average colour of the input pixels, so this information would have to be somehow preserved (kept explicit) through each layer, which might be difficult.

With an amorphous network, such information could be readily available at each level. This would also sometimes help with detecting correlations between high and low-level aspects of a representation (e.g. red cars are often Ferraris).

## References


Atick J. J. & Redlich A. N. [1990]
*"Towards a theory of early visual processing"*
**Neural Computation, 2, 308-320.**

Atick J. J. & Redlich A. N. [1992]
*"What does the retina know about natural scenes?"*
**Neural Computation, 4, 196-210.**

Atick J. J. & Redlich A. N. [1993]
*"Convergent algorithm for sensory receptive field development"*
**Neural Computation, 5, 45-60.**

Barlow H. B. (1989)
*"Unsupervised learning"*
**Neural Computation, 1, 295-311.**

Barlow H. B., Kaushal T. P. & Mitchison G. J. (1989)
*"Finding minimum entropy codes"*
**Neural Computation, 1, 412-423.**

Barto A. G. & Anandan P. (1985)
*"Pattern recognising stochastic learning automata"*
**IEEE Transactions on Systems, Man, and Cybernetics 15, 360-375.**

Barto A. G. & Jordan, M. I. (1987)
*"Gradient following without back-propagation in layered networks"*
**IEEE First International Conference on Neural Networks (San Diego 1987), eds. Caudill, M. & Butler, C., vol. II, 629-636. New York: IEEE.**

Boas M. L. (1983)
*"Mathematical methods in the physical sciences"*
**John Wiley & Sons, Inc.**

Bullinaria J. A. (1994)
*"Internal representations of a connectionist model of reading aloud"*
**Proceedings of sixteenth annual conference of the cognitive science society" 78-83.**

Bullinaria J. A. (1995)
*"Neural network models of reading: solving the alignment problem without Wickelfeatures"*
**In: "Connectionist models of memory and language", Levy J.P., Bairaktaris D., Bullinaria J.A., Cairns P. (eds).**

Bullinaria J. A. (1996)
*Modelling reading, spelling and past tense learning with artificial neural networks"*
**Brain and Language (in press)**

Coltheart M., Curtis B. & Atkins P. (1992)
*"Models of reading aloud: dual-route and parallel-distributed-processing approaches"*
**Macquarie University pre-print.**

Coltheart M., Patterson K. E. & Marshall J. C. (1980)
*"Deep dyslexia"*

**Routledge and Keegan.**

Damper R. I. (1995)
*"Self-learning and connectionist approaches to text-phoneme conversion"*
**In: "Connectionist models of memory and language", Levy J.P., Bairaktaris D., Bullinaria J.A., Cairns P. (eds).**

Daugman J. G. (1985)
*"Uncertainty relation for resolution in space, spatial frequency, and orientation optimised by two-dimensional visual cortical filters"*
**Journal of the Optical Society of America, Vol. 2, No. 7.**

Dedina M. J. & Nusbaum H. C. (1986)
*"PRONOUNCE: a program for pronunciation by analogy"*
**Speech research laboratory progress report 12, Indiana University, Bloomington, IN.**

Dedina M. J. & Nusbaum H. C. (1991)
*"PRONOUNCE: a program for pronunciation by analogy"*
**Computer Speech and Language, 5, 55-64.**

Douglas R. J. & Martin, K. A. C. (1990)
*"Neocortex"*
**In G.M Sheperd (Ed) "The synaptic organisation of the brain", 389-438. O.U.P..**

Field D. J. (1989)
*"What the statistics of natural images tell us about visual coding"*
**Proceedings of S.P.I.E. 1077 (269-276), Los Angeles, California**

Finch S. & Chater N. (1992)
*"Bootstrapping syntactic categories using statistical methods"*
**"Background and Experiments in Machine Learning of Natural Language, proceedings of the first SHOE workshop", Daelemans W. & Powers D. (eds.), 230-235.**

Finch S., Chater N. & Redington M. (1995)
*"Acquiring syntactic information from distributional statistics"*
**In: "Connectionist models of memory and language", Levy J.P., Bairaktaris D., Bullinaria J.A., Cairns P. (eds).**

Földiák P. (1990)
*"Forming sparse representations by local anti-Hebbian learning"*
**Biological Cybernetics, 64, 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 No. CUED/F-INFENG/TR 91. Engineering Department, University of Cambridge.**

Fukushima K. (1975)
*"Neocognitron: a self-organising neural network model for a mechanism of pattern recognition unaffected by shift in position"*
**Biological Cybernetics 36, 193-202.**

Fukushima K. (1980)
*"Cognition: a self-organising multi-layered neural network"*
**Biological Cybernetics 20, 121-136.**

Gabor D. (1946)
*"Theory of communication"*
**Journal of the Institute of Electrical Engineers, 93, 429-457**

Glushko R. J. (1979)
*"The organisation and activation of orthographic knowledge in reading aloud"*
**Journal of Experimental Psychology: Human Perception and Performance, 5, 674-691.**

Glushko R. J. (1981)
*"Principles for pronouncing print: the psychology of phonography"*
**In: Interactive processes in reading, Lesgold A.M. & Perfetti C.A. (eds), Lawrence Erlbaum Assoc., Hillsdale, NJ. 61-84.**

Grossman T. (1990)
*"The CHIR Algorithm for feed-forward networks with binary weights"*
**Advances in Neural Information Processing Systems II (Denver 1989), ed. Touretzky, D. S., 516-523. San Mateo: Morgan Kaufmann.**

Hodgkin A. K. & Huxley, A. F. (1952)
*"Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo"*
**Journal of Physiology 116 449-472.**

Hertz J., Krogh A. & Palmer R. G. (1991)
*"Introduction to the theory of neural computation"*
**Lecture notes volume 1, Santa Fe Institute - Studies in the sciences of complexity.**

Hinton G. E. (1989)
*"Connectionist learning procedures"*
**Artificial Intelligence, 40, 185-234**

Humphreys G. W. & Evett L. J. (1985)
*"Are there independent lexical and nonlexical routes in word processing? An evaluation of the dual-route theory of reading"*
**The Behavioural and Brain Sciences, 8, 689-740.**

Kohonen T. (1982)
*"Self-organised formation of topologically correct feature maps"*
**Biological Cybernetics 43, 59-69.**

Kohonen T. (1989)
*"Self-organisation an associative memory"*
**3rd edition. Berlin: Springer-Verlag.**

Kuffler S. W., Nicholls, J. G. & Martin, A. R. (1984)
*"From neuron to brain"*
**Second edition, Sinauer Associates Inc. Publishers, Sunderland, Massachusetts**

Linsker R. (1986)
*"From basic network principles to neural architecture"*
**Proceedings of the national academy of sciences, USA 83, 7508-7512,**

**8390-8394, 8779-8783.**

Linsker R. (1988)
*"Self-organisation in a perceptual network"*
**Computer, March 1988, 105-117**

Mallat S. G. (1989)
*"A theory for multiresolution signal decomposition: the wavelet representation"*
**I.E.E.E. Transactions on Pattern Analysis and Machine Intelligence. Vol. 11, No. 7.**

von der Malsburg Ch. (1973)
*"Self-organisation of orientation sensitive cells in the striate cortex"*
**Kybernetik 14, 85-100. Reprinted in Anderson J. A. and Rosenfeld E. (eds) (1988) "Neurocomputing: Foundations of research". Cambridge: MIT Press.**

Marr D. (1982)
*"Vision"*
**San Francisco: Freeman.**

Morell F. (1972)
*"Integrative properties of parastriate neurons"*
**In: Brain and Human Behaviour, A. G. Karczmar & J. Eccles (Eds).**

Morrison R. E. (1983)
*"Retinal image size and the perceptual span in reading"*
**in: "Eye Movements and Reading", Rayner, K. (ed.) , 31-40.**

Oja E. (1982)
*"A simplified neuron model as a principal component analyser"*
**Journal of Mathematical Biology 15, 267-273.**

Oja E. (1989)
*"Neural networks, principal components and sub-spaces"*
**International journal of neural systems 1, 61-68.**

Parker D. B. (1985)
*"Learning logic"*
**Technical report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA.**

Plumbley M. D. (1991)
*"On information theory and neural networks"*
**CUED/F-INFENG/TR.78**

Plumbley M. D. (1994)
*"Information theory and neural network learning algorithms"*
**Neural computing research and applications, proc. of the 2nd Irish Neural Networks Conference.**

Press W. H., Teukolsky S. A., Vetterling W. T. & Flannery B. P. (1992)
*"Numerical recipes in C: the art of scientific computing"*
**William H. Press.**

Rissanen J. (1989)
*"Stochastic complexity in statistical enquiry"*

**World Scientific Publishing Co., Singapore**

Rosenblatt F. (1962)
*"Principles of Neurodynamics"*
**New York: Spartan.**

Rumelhart D. E., Hinton G. E., & Williams R. J. (1986)
*"Learning internal representations by error back propagation"*
**In: D. E. Rumelhart & J. McClelland (eds.), Parallel Distributed Processing. Cambridge, Mass: MIT Press**

Rumelhart D. E. & McClelland J. (1986)
*"On learning the past tenses of English verbs"*
**In: D. E. Rumelhart & J. McClelland (eds.), Parallel Distributed Processing. Cambridge, Mass: MIT Press**

Rumelhart D. E. & Zipser D. (1985)
*"Feature discovery by competitive learning"*
**Cognitive Science 9, 75-112.**

Sanger T. D. (1989)
*"Optimal unsupervised learning in a single-layer linear feed-forward neural network"*
**Neural Networks 2, 459-473.**

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

Sejnowski T. J. (1986)
*"Open questions about computation in the cerebral cortex"*
**In: D. E. Rumelhart & J. McClelland (eds.), "Parallel Distributed Processing" volume 2. Cambridge, Mass: MIT Press**

Sejnowski T. J. & Rosenberg C. R. (1987)
*"Parallel Networks that Learn to Pronounce English Text"*
**Complex Systems, 1, 145-168.**

Shannon C. E. (1948)
*"A mathematical theory of communication"*
**Bell systems technical journal, 27:370-423, 623-656, 1948.**

Shannon C. E. (1963)
*"The Mathematical Theory of Communication"*
**"The Mathematical Theory of Communication", Shannon C. E. & Weaver W. University of Illinois Press.**

Smith J. P. (1993)
*"Using unsupervised feature detectors in a model of reading aloud"*
**The Irish Journal of Psychology, 14, 3, 397-409.**

Sullivan K. P. H. & Damper R. I. (1990)
*"A Pychologically-governed approach to novel-word pronunciation within a text-to-speech system"*
**Proc. IEEE International conference on Acoustic Speech Signal Processing. (ICASSP 1990), Albuquerque, NM, Vol. 1, 341-344.**

Sullivan K. P. H. & Damper R. I. (1991)

*"Speech synthesis by analogy: recent advances and results"*
**Proc. IEEE International conference on Acoustic Speech Signal Processing. (ICASSP 1991), Toronto, Canada, Vol. 2, 761-764.**

Sullivan K. P. H. & Damper R. I. (1992)
*"Novel-word pronunciation within a text-to-speech system"*
**in: "Talking Machines, Models, and Designs",**
**G. Bailly, C. Benoît, and T. R. Sawallis (ed.), 183-195.**

Wolf J. G. (1982)
*"Language acquisition, data compression and generalisation"*
**Language and communication 2:57-89**

Werbos P. (1974)
*"Beyond regression: new tools for prediction and analysis in the behavioural sciences.*
**Ph.D. thesis, Harvard University.**