Memory, Attention, and Decision-Making

A Unifying Computational Neuroscience Approach

Edmund T. Rolls
University of Oxford
Department of Experimental Psychology
Oxford
England

OXFORD UNIVERSITY PRESS • OXFORD 2008

Preface

The overall aim of this book is to provide insight into how memory systems in the brain work; how the operation of these systems is fundamental to understanding many aspects of brain function including perception, decision-making, and action selection; and how interactions between these systems provide an account of many cognitive phenomena including attention and emotion. It is shown that to do this, a neurocomputational approach is needed. The book provides Appendices that describe many of the building blocks of the neurocomputational approach, and that are designed to be useful for teaching.

To understand how the brain works, including how it functions in memory, attention, and decision-making, it is necessary to combine different approaches, including neural computation. Neurophysiology at the single neuron level is needed because this is the level at which information is exchanged between the computing elements of the brain. Evidence from the effects of brain damage, including that available from neuropsychology, is needed to help understand what different parts of the system do, and indeed what each part is necessary for. Neuroimaging is useful to indicate where in the human brain different processes take place, and to show which functions can be dissociated from each other. Knowledge of the biophysical and synaptic properties of neurons is essential to understand how the computing elements of the brain work, and therefore what the building blocks of biologically realistic computational models should be. Knowledge of the anatomical and functional architecture of the cortex is needed to show what types of neuronal network actually perform the computation. And finally the approach of neural computation is needed, as this is required to link together all the empirical evidence to produce an understanding of how the system actually works. This book utilizes evidence from all these disciplines to develop an understanding of how different types of memory, perception, attention, and decision-making are implemented by processing in the brain.

I emphasize that to understand memory, perception, attention, and decision-making in the brain, we are dealing with large-scale computational systems with interactions between the parts, and that this understanding requires analysis at the computational and global level of the operation of many neurons to perform together a useful function. Understanding at the molecular level is important for helping to understand how these large-scale computational processes are implemented in the brain, but will not by itself give any account of what computations are performed to implement these cognitive functions. Instead, understanding cognitive functions such as object recognition, memory recall, attention, and decision-making requires single neuron data to be closely linked to computational models of how the interactions between large numbers of neurons and many networks of neurons allow these cognitive problems to be solved. The single neuron level is important in this approach, for the single neurons can be thought of as the computational units of the system, and is the level at which the information is exchanged by the spiking activity between the computational elements of the brain. The single neuron level is therefore, because it is the level at which information is communicated between the computing elements of the brain, the fundamental level of information processing, and the level at which the information can be read out (by recording the spiking activity) in order to understand what information is being represented and processed in each brain area.

With its focus on how the brain works at the computational neuroscience level, this book

is distinct from the many excellent books on neuroscience that describe much evidence about brain structure and function, but do not aim to provide an understanding of how the brain works at the computational level. This book aims to forge an understanding of how some key brain systems may operate at the computational level, so that we can understand how the brain actually performs some of its complex and necessarily computational functions in memory, perception, attention, and decision-making.

A test of whether one's understanding is correct is to simulate the processing on a computer, and to show whether the simulation can perform the tasks of memory systems in the brain, and whether the simulation has similar properties to the real brain. The approach of neural computation leads to a precise definition of how the computation is performed, and to precise and quantitative tests of the theories produced. How memory systems in the brain work is a paradigm example of this approach, because memory-like operations which involve altered functionality as a result of synaptic modification are at the heart of how all computations in the brain are performed. It happens that attention and decision-making can be understood in terms of interactions between and fundamental operations in memory systems in the brain, and therefore it is natural to treat these areas of cognitive neuroscience as well as memory in this book. The same fundamental concepts based on the operation of neuronal circuitry can be applied to all these functions, as is shown in this book.

One of the distinctive properties of this book is that it links the neural computation approach not only firmly to neuronal neurophysiology, which provides much of the primary data about how the brain operates, but also to psychophysical studies (for example of attention); to neuropsychological studies of patients with brain damage; and to functional magnetic resonance imaging (fMRI) (and other neuroimaging) approaches. The empirical evidence that is brought to bear is largely from non-human primates and from humans, because of the considerable similarity of their memory and related systems, and the overall aims to understand how memory and related functions are implemented in the human brain, and the disorders that arise after brain damage.

The overall aims of the book are developed further, and the plan of the book is described, in Chapter 1, Section 1.1.

Part of the material described in the book reflects work performed in collaboration with many colleagues, whose tremendous contributions are warmly appreciated. The contributions of many will be evident from the references cited in the text. Especial appreciation is due to Gustavo Deco, Simon M. Stringer, and Alessandro Treves who have contributed greatly in an always interesting and fruitful research collaboration on computational aspects of brain function, and to many neurophysiology and functional neuroimaging colleagues who have contributed to the empirical discoveries that provide the foundation to which the computational neuroscience must always be closely linked, and whose names are cited throughout the text. Much of the work described would not have been possible without financial support from a number of sources, particularly the Medical Research Council of the UK, the Human Frontier Science Program, the Wellcome Trust, and the James S. McDonnell Foundation. The book was typeset by the author using LaTex and WinEdt.

The cover shows part of the picture Psyche Opening the Golden Box painted in 1903 by J. W. Waterhouse. The metaphor is to look inside the system of the mind and the brain, in order to understand how the brain functions, and thereby better to understand and treat its disorders. Updates to the publications cited in this book are available at http://www.oxcns.org.

I dedicate this work to the overlapping group: my family, friends, and colleagues – in salutem praesentium, in memoriam absentium.

Contents

Appendix 1 Introduction to linear algebra for neural networks

In this Appendix we review some simple elements of linear algebra relevant to understanding neural networks. This will provide a useful basis for a quantitative understanding of how neural networks operate (see Appendix B).

A.1 Vectors

A vector is an ordered set of numbers. An example of a vector is the set of numbers

$$\begin{bmatrix} 7 \\ 4 \end{bmatrix}$$

If we denote the jth element of this vector as w_j, then $w_1 = 7$, and $w_2 = 4$. We can denote the whole vector by $\mathbf{w}$. This notation is very economical. If the vector has 10,000 elements, then we can still refer to it in mathematical operations as $\mathbf{w}$. $\mathbf{w}$ might refer to the vector of 10,000 synaptic weights on the dendrites of a neuron. Another example of a vector is the set of firing rates of the axons that make synapses onto a dendrite, as shown in Fig. 1.2. The firing rate x of each axon forming the input vector can be indexed by j, and is denoted by x_j. The vector would be denoted by $\mathbf{x}$.

Certain mathematical operations can be performed with vectors. We start with the operation which is fundamental to simple models of neural networks, the inner product or dot product of two vectors.

A.1.1 The inner or dot product of two vectors

The operation of computing the activation h of a neuron from the firing rate on its input axons multiplied by the corresponding synaptic weight can be expressed as:

$$h = \sum_j x_j w_j \tag{A.1}$$

where $\sum_j$ indicates that the sum is over the C input axons to each neuron, indexed by j. Denoting the firing rate vector as $\mathbf{x}$ and the synaptic weight vector as $\mathbf{w}$, we can write

$$h = \mathbf{x} \cdot \mathbf{w} \qquad . \tag{A.2}$$

If the weight vector is

$$\mathbf{w} = \begin{bmatrix} 9 \\ 5 \\ 2 \end{bmatrix}$$

and the firing rate input vector is

$$\mathbf{x} = \begin{bmatrix} 3 \\ 6 \\ 7 \end{bmatrix}$$

then we can write

$$\mathbf{x} \cdot \mathbf{w} = (3 \cdot 9) + (6 \cdot 5) + (7 \cdot 2) = 71 \quad . \tag{A.3}$$

Thus in the inner or dot product, we multiply the corresponding terms, and then sum the result. As this is the simple mathematical operation that is used to compute the activation h in the most simplified abstraction of a neuron (see Chapter 1), we see that it is indeed the fundamental operation underlying many types of neural network. We will shortly see that some of the properties of neuronal networks can be understood in terms of the properties of the dot product. We next review a number of basic aspects of vectors and inner products between vectors.

There is a simple geometrical interpretation of vectors, at least in low-dimensional spaces. If we define, for example, x and y axes at right angles to each other in a two-dimensional space, then any two-component vector can be thought of as having a direction and length in that space that can be defined by the values of the two elements of the vector. If the first element is taken to correspond to x and the second to y, then the x axis lies in the direction [1,0] in the space, and the y axis in the direction [0,1], as shown in Fig. A.1. The line to point [1,1] in the space then lies at 45° to both axes, as shown in Fig. A.1.

A.1.2 The length of a vector

Consider taking the inner product of a vector

$$\mathbf{w} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

with itself. Then

$$||\mathbf{w}|| = \sqrt{\mathbf{w} \cdot \mathbf{w}} = \sqrt{4^2 + 3^2} = 5. \tag{A.4}$$

This is the length of the vector. We can represent this operation in the two-dimensional graph shown in Fig. A.1. In this case, the coordinates where vector $\mathbf{w}$ ends in the space are [1,1]. The length of the vector (from [0,0]) to [1,1] is obtained by Pythagoras' theorem. Pythagoras' theorem states that the length of the vector $\mathbf{w}$ is equal to the square root of the sum of the squares of the two sides. Thus we define the length of the vector $\mathbf{w}$ as

$$||\mathbf{w}|| = \sqrt{\mathbf{w} \cdot \mathbf{w}} \tag{A.5}$$

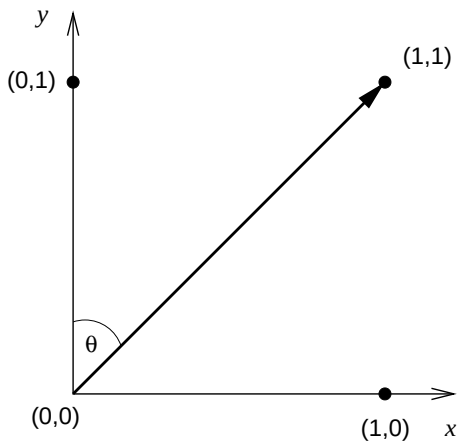


Fig. A.1 Illustration of a vector in a two-dimensional space. The basis for the space is made up of the x axis in the [1,0] direction, and the y axis in the [0,1] direction. (The first element of each vector is then the x value, and the second the y value. The values of x and y for different points, marked by a dot, in the space are shown. The origins of the axes are at point 0,0.) The [1,1] vector projects in the [1,1] (or 45^o) direction to the point 1,1, with length 1.414.

In the [1,1] case, this value is $\sqrt{2}$ = 1.414.

A.1.3 Normalizing the length of a vector

We can scale a vector in such a way that its length is equal to 1 by dividing it by its length. If we form the dot product of two normalized vectors, its maximum value will be 1, and its minimum value -1.

A.1.4 The angle between two vectors: the normalized dot product

The angle between two vectors $\mathbf{x}$ and $\mathbf{w}$ is defined in terms of the inner product as follows:

$$\cos\theta = \frac{\mathbf{x} \cdot \mathbf{w}}{||\mathbf{x}||\,||\mathbf{w}||} \tag{A.6}$$

For example, the angle between two vectors

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

where the length of vector $\mathbf{x}$ is $\sqrt{0.0 + 1.1} = 1$ and of vector $\mathbf{w}$ is $\sqrt{1.1 + 1.1} = \sqrt{2}$ is

$$\cos\theta = \frac{(0.1) + (1.1)}{1.\sqrt{2}} = 0.707. \tag{A.7}$$

Thus $\theta = \cos^{-1}(0.707) = 45^o$.

We can give a simple geometrical interpretation of this as shown in Fig. A.1. However, equation A.6 is much easier to use in a high-dimensional space!

The dot product reflects the similarity between two vectors. Once the length of the vectors is fixed, the higher their dot product, the more similar are the two vectors. By normalizing the dot product, that is by dividing by the lengths of each vector as shown in equation A.6, we

obtain a value that varies from -1 to $+1$. This normalized dot product is then just the cosine of the angle between the vectors, and is a very useful measure of the similarity between any two vectors, because it always lies in the range -1 to $+1$. It is closely related to the (Pearson product–moment) correlation coefficient between any two vectors, as we see if we write the equation in terms of its components

$$\cos\theta = \frac{\sum_j x_j w_j}{(\sum_j {x_j}^2)^{1/2} (\sum_j {w_j}^2)^{1/2}} \tag{A.8}$$

which is just the formula for the correlation coefficient between two sets of numbers with zero mean (or with the mean value removed by subtracting the mean of the components of each vector from each component of that vector).

Now consider two vectors that have a dot product of zero, that is where $\cos\theta = 0$ or the angle between the vectors is 90°. Such vectors are described as orthogonal (literally at right angles) to each other. If our two orthogonal vectors were $\mathbf{x}$ and $\mathbf{w}$, then the activation of the neuron, measured by the dot product of these two vectors, would be zero. If our two orthogonal vectors each had a mean of zero, their correlation would also be zero: the two vectors can then be described as unrelated or independent.

If, instead, the two vectors had zero angle between them, that is if $\cos\theta = 1$, then the dot product would be maximal (given the vectors' lengths), the normalized dot product would be 1, and the two vectors would be described as identical to each other apart from their length. Note that in this case their correlation would also be 1, even if the two vectors did not have zero mean components.

For intermediate similarities of the two vectors, the degree of similarity would be expressed by the relative magnitude of the dot product, or by the normalized dot product of the two vectors, which is just the cosine of the angle between them. These measures are closely related to the correlation between two vectors.

Thus we can think of the simple operation performed by neurons as measuring the similarity between their current input vector and their synaptic weight vector. Their activation, h, is this dot product. It is because of this simple operation that neurons can generalize to similar inputs; can still produce useful outputs if some of their inputs or synaptic weights are damaged or missing, that is they can show graceful degradation or fault tolerance; and can be thought of as learning to point their weight vectors towards input patterns, which is very useful in enabling neurons to categorize their inputs in competitive networks (see Section B.4).

A.1.5 The outer product of two vectors

Let us take a row vector having as components the firing rates of a set of output neurons in a pattern associator or competitive network, which we might denote as $\mathbf{y}$, with components y_i and the index i running from 1 to the number N of output neurons. $\mathbf{y}$ is then a shorthand for writing down each component, e.g. [7,2,5,2,...], to indicate that the firing rate of neuron 1 is 7, etc. To avoid confusion, we continue in the following to denote the firing rate of input neuron j as x_j. Now recall (see Chapter 1 and Section B.2) how the synaptic weights are formed in a pattern associator using a Hebb rule as follows:

$$\delta w_{ij} = \alpha y_i x_j \tag{A.9}$$

where δw_{ij} is the change of the synaptic weight w_{ij} which results from the simultaneous (or conjunctive) presence of presynaptic firing x_j and postsynaptic firing or activation y_i, and α

is a learning rate constant which specifies how much the synapses alter on any one pairing. In a more compact vector notation, this expression would be

$$\delta \mathbf{w}_i = \alpha \mathbf{y}_i \mathbf{x}' \tag{A.10}$$

where the firing rates on the axons form a column vector with the values, for example, as follows[37]:

$$\mathbf{x}' = \begin{matrix} [2] \\ [0] \\ [3] \\ .... \end{matrix}$$

The weights are then updated by a change proportional (the α factor) to the following matrix (Table A.1):

Table A.1 Multiplication of a row vector [7 2 5 ....] by a column vector to form the external or tensor product, representing for example the changes to a matrix of synaptic weights $\mathbf{W}$

	[7	2	5	.....]
[2]	14	4	10	.....
[0]	0	0	0	.....
[3]	21	6	15	.....
.....	.....	.....	.....	.....

This multiplication of the two vectors is called the outer, or tensor, product, and forms a matrix, in this case of (alterations to) synaptic weights. Thus we see that the operation of altering synaptic weights in a network can be thought of as forming a matrix of weight changes, which can then be used to alter the existing matrix of synaptic weights.

A.1.6 Linear and non-linear systems

The operations with which we have been concerned in this Appendix so far are linear operations. We should note that if two matrices operate linearly, we can form their product by matrix multiplication, and then replace the two matrices with the single matrix that is their product. We can thus effectively replace two synaptic matrices in a linear multilayer neural network with one synaptic matrix, the product of the two matrices. For this reason, multilayer neural networks if linear cannot achieve more than can be achieved in a single-layer linear network. It is only in non-linear networks that more can be achieved, in terms of mapping input vectors through the synaptic weight matrices, to produce particular mappings to output vectors. Much of the power of many networks in the brain comes from the fact that they are multilayer non-linear networks (in that the computing elements in each network, the neurons, have non-linear properties such as thresholds, and saturation at high levels of output). Because the matrix by matrix multiplication operations of linear algebra cannot be applied directly to

[37] The prime after the $\mathbf{x}$ is used here to remind us that this vector is a column vector, which can be thought of as a transformed row vector, and the prime indicates the transformed vector. We do not use the prime for most of this book in order to keep the notation uncluttered.

the operation of neural networks in the brain, we turn instead back to other aspects of linear algebra, which can help us to understand which classes of pattern can be successfully learned by different types of neural network.

A.1.7 Linear combinations of vectors, linear independence, and linear separability

We can multiply a vector by a scalar (a single value, e.g. 2) thus:

$$2 \; . \; \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 8 \\ 2 \\ 6 \end{bmatrix}$$

We can add two vectors thus:

$$\begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 2 \\ 7 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 5 \end{bmatrix}$$

The sum of the two vectors is an example of a linear combination of vectors, which is in general a weighted sum of several vectors, component by component. Thus, the linear combination of vectors $\mathbf{v}_1$, $\mathbf{v}_2$, ... to form a vector $\mathbf{v}_s$ is expressed by the sum

$$\mathbf{v}_s = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + ... \tag{A.11}$$

where c_1 and c_2 are scalars.

By adding vectors in this way, we can produce any vector in the space spanned by a set of vectors as a linear combination of vectors in the set. If in a set of n vectors at least one can be written as a linear combination of the others, then the vectors are described as **linearly dependent**. If in a set of n vectors none can be written as a linear combination of the others, then the vectors are described as **linearly independent**. A linearly independent set of vectors has the properties that any vector in the space spanned by the set can be written in only one way as a linear combination of the set, and the space has dimension $d = n$. In contrast, a vector in a space spanned by a linearly dependent set can be written in an infinite number of equivalent ways, and the dimension d of the space is less than n.

Consider a set of linearly dependent vectors and the d-dimensional space they span. Two subsets of this set are described as **linearly separable** if the vectors of one subset (that is, their endpoints) can be separated from those of the other by a hyperplane, that is a subspace of dimension $d - 1$. *Subsets formed from a set of linearly independent vectors are always linearly separable.* For example, the four vectors:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

are linearly dependent, because the fourth can be formed by a linear combination of the second and third (and also because the first, being the null vector, can be formed by multiplying any other vector by zero, a specific linear combination). In fact, $n = 4$ and $d = 2$. If we split this set into subset A including the first and fourth vector, and subset B including the second

and third, the two subsets are not linearly separable, because there is no way to draw a line (which is the subspace of dimension $d - 1 = 1$) to separate the two subsets A and B. We will encounter this set of vectors in Appendix B, and this is the geometrical interpretation of why a one-layer, one-output neuron network cannot separate these patterns. Such a network (a simple perceptron) is equivalent to its (single) weight vector, and in turn the weight vector defines a set of parallel $d - 1$ dimensional hyperplanes. (Here $d = 2$, so a hyperplane is simply a line, any line perpendicular to the weight vector.) No line can be found that separates the first and fourth vector from the second and third, whatever the weight vector the line is perpendicular to, and hence no perceptron exists that performs the required classification (see Section A.2.1). To separate such patterns, a multilayer network with non-linear neurons is needed (see Appendix B).

Any set of linearly independent vectors comprise the basis of the space they span, and they are called basis vectors. All possible vectors in the space spanned by these vectors can be formed as linear combinations of these vectors. If the vectors of the basis are in addition mutually orthogonal, the basis is an orthogonal basis, and it is, further, an orthonormal basis if the vectors are chosen to be of unit length. Given any space of vectors with a preassigned meaning to each of their components (for example the space of patterns of activation, in which each component is the activation of a particular unit), the most natural, canonical choice for a basis is the set of vectors in which each vector has one component, in turn, with value 1, and all the others with value 0. For example, in the $d = 2$ space considered earlier, the natural choice is to take as basis vectors

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

and

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

from which all vectors in the space can be created. This can be seen from Fig. A.1. (A vector in the [–1,–1] direction would have the opposite direction of the vector shown in Fig. A.1.)

If we had three vectors that were all in the same plane in a three-dimensional (x, y, z) space, then the space they spanned would be less than three-dimensional. For example, the three vectors

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} -1 \\ -1 \\ 0 \end{bmatrix}$$

all lie in the same z plane, and span only a two-dimensional space. (All points in the space could be shown in the plane of the paper in Fig. A.1.)

A.2 Application to understanding simple neural networks

The operation of simple one-layer networks can be understood in terms of these concepts.

A.2.1 Capability and limitations of single-layer networks: linear separability and capacity

Single-layer perceptrons perform pattern classification, and can be trained by an associative (Hebb) learning rule or by an error-correction (delta) rule (see Appendix B). That is, each neuron classifies the input patterns it receives into classes determined by the teacher. Single-layer perceptrons are thus supervised networks, with a separate teacher for each output neuron. The classification is most clearly understood if the output neurons are binary, or are strongly non-linear, but the network will still try to obtain an optimal mapping with linear or near-linear output neurons.

When each neuron operates as a binary classifier, we can consider how many input patterns p can be classified by each neuron, and the classes of pattern that can be correctly classified. The result is that the maximum number of patterns that can be correctly classified by a neuron with C inputs is

$$p_{\max} = 2C \tag{A.12}$$

when the inputs have random continuous-valued inputs, but the patterns must be linearly separable (see Hertz et al. (1991)). More generally, a network with a single binary unit can implement a classification between two subspaces of a space of possible input patterns provided that the p actual patterns given as examples of the correct classification are linearly separable.

The linear separability requirement can be made clear by considering a geometric interpretation of the logical AND problem, which is linearly separable, and the XOR (exclusive OR) problem, which is not linearly separable. The truth tables for the AND and XOR functions are shown in Table A.2 (there are two inputs, x_1 and x_2, and one output neuron):

Table A.2 Truth table for AND and XOR functions performed by a single output neuron with two inputs. 1 = active or firing; 0 = inactive.

Inputs		Output	
x_1	x_2	AND	XOR
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

For the AND function, we can plot the mapping required in a 2D graph as shown in Fig. A.2. A line can be drawn to separate the input coordinates for which 0 is required as the output from those for which 1 is required as the output. The problem is thus linearly separable. A neuron with two inputs can set its weights to values which draw this line through this space, and such a one-layer network can thus solve the AND function.

For the XOR function, we can plot the mapping required in a 2D graph as shown in Fig. A.3. No straight line can be drawn to separate the input coordinates for which 0 is required as the output from those for which 1 is required as the output. The problem is thus not linearly separable. For a one-layer network, no set of weights can be found that will perform the XOR, or any other non-linearly separable function.

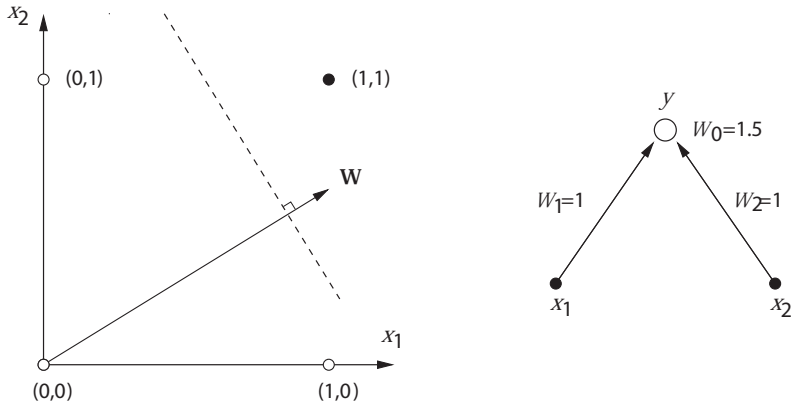


Fig. A.2 Left: the AND function shown in a 2D space. Input values for the two neurons are shown along the two axes of the space. The outputs required are plotted at the coordinates where the inputs intersect, and the values of the output required are shown as an open circle for 0, and a filled circle for 1. The AND function is linearly separable, in that a line can be drawn in the space which separates the coordinates for which 0 output is required from those from which a 1 output is required. $\mathbf{w}$ shows the direction of the weight vector. Right: a one-layer neural network can set its two weights w_1 and w_2 to values which allow the output neuron to be activated only if both inputs are present. In this diagram, w_0 is used to set a threshold for the neuron, and is connected to an input with value 1. The neuron thus fires only if the threshold of 1.5 is exceeded, which happens only if both inputs to the neuron are 1.

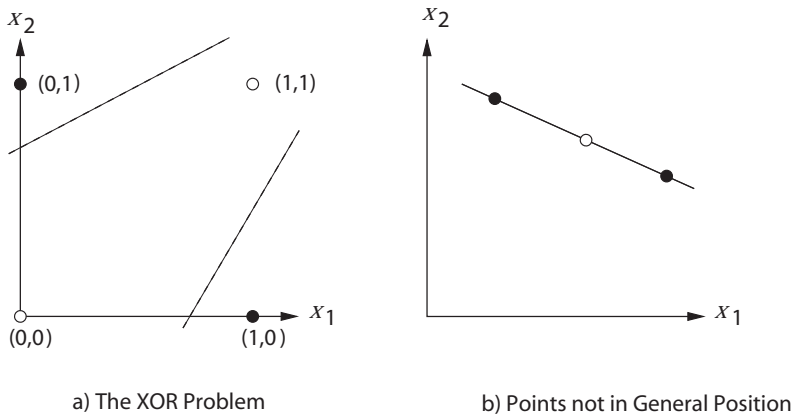


Fig. A.3 The XOR function shown in a 2D space. Input values for the two neurons are shown along the two axes of the space. The outputs required are plotted at the coordinates where the inputs intersect, and the values of the output required are shown as an open circle for 0, and a filled circle for 1. The XOR function is not linearly separable, in that a line cannot be drawn in the space to separate the coordinates for those from which a 0 output is required from those from which a 1 output is required. A one-layer neural network cannot set its two weights to values which allow the output neuron to be activated appropriately for the XOR function.

Although the inability of one-layer networks with binary neurons to solve non-linearly separable problems is a limitation, it is not in practice a major limitation on the processing that can be performed in a neural network for a number of reasons. First, if the inputs can take continuous values, then if the patterns are drawn from a random distribution, the one-layer network can map up to $2C$ of them. Second, as described for pattern associators, and for one-layer error-correcting perceptrons (see Appendix B), these networks could be preceded

by an expansion recoding network such as a competitive network with more output than input neurons. This effectively provides a two-layer network for solving the problem, and multilayer networks are in general capable of solving arbitrary mapping problems. Ways in which such multilayer networks might be trained are discussed in Chapter 4 and Appendix B.

More generally, a binary output unit provides by its operation a hyperplane (the hyperplane orthogonal to its synaptic weight vector as shown in Fig. A.2) that divides the input space in two. The input space is of dimension C, if C is the number of input axons or connections. A one-layer network with a number n of binary output units is equivalent to n hyperplanes, that could potentially divide the input space into as many as 2^n regions, each corresponding to input patterns leading to a different output. However the number p of *arbitrary* examples of the correct classification (each example consisting of an input pattern and its required correct output) that the network may be able to implement is well below 2^n, and in fact depends on C not on n. This is because for p too large it will be impossible to position the n weight vectors such that all examples of input vectors for which the first output unit is required to be 'on' fall on one side of the hyperplane associated with the first weight vector, all those for which it is required to be 'off' fall on the other side, and simultaneously the same holds with respect to the second output unit (a different dichotomy), the third, and so on. The limit on p, which can be thought of also as the number of independent associations implemented by the network, when this is viewed as a heteroassociator (i.e. pattern associator) with binary outputs, can be calculated with the Gardner method (Gardner 1987, Gardner 1988) and depends on the statistics of the patterns. For input patterns that are also binary, random and with equal probability for each of the two states on every unit, the limit is $p_c = 2C$ (see further Appendix B, and Rolls and Treves (1998) Appendix A3).

A.2.2 Non-linear networks: neurons with non-linear activation functions

These concepts also help one to understand further the limitation of linear systems, and the power of non-linear systems. Consider the dot product operation by which the neuronal activation h is computed:

$$h = \sum_j x_j w_j. \tag{A.13}$$

If the output firing is just a linear function of the activation, any input pattern will produce a non-zero output unless it happens to be exactly orthogonal to the weight vector. For positive-only firing rates and synaptic weights, being orthogonal means taking non-zero values only on non-corresponding components. Since with distributed representations the non-zero components of different input firing vectors will in general be overlapping (i.e. some corresponding components in both firing rate vectors will be on, that is the vectors will overlap), this will result effectively in interference between any two different patterns that for example have to be associated to different outputs. Thus a basic limitation of linear networks is that they can perform pattern association perfectly only if the input patterns $\mathbf{x}$ are orthogonal; and for positive-only patterns that represent actual firing rates only if the different firing rate vectors are non-overlapping. Further, linear networks cannot of course perform any classification, just because they act linearly. (Classification implies producing output states that are clearly defined as being in one class, and not in other classes.) For example, in a linear network, if a pattern is presented that is intermediate between two patterns $\mathbf{v}_1$ and $\mathbf{v}_2$, such as $c_1\mathbf{v}_1 + c_2\mathbf{v}_2$, then the output pattern will be a linear combination of the outputs produced by $\mathbf{v}_1$ and $\mathbf{v}_2$ (e.g. $c_1\mathbf{o}_1 + c_2\mathbf{o}_2$), rather than being classified into $\mathbf{o}_1$ or $\mathbf{o}_2$. In contrast, with non-linear neurons, the patterns need not be orthogonal, only linearly separable, for a one-layer network

to be able to correctly classify the patterns (provided that a sufficiently powerful learning rule is used – see Appendix B).

The networks just described, and most of those described in this book, are trained with a local learning rule, in which the pre- and post-synaptic terms needed to alter the synaptic weights are available locally in the synapses, in terms for example of the release of transmitter from the presynaptic terminal, and the depolarization of the postsynaptic neuron. This type of network is considered because this is a biologically plausible constraint (see Section B.13). It is much less biologically plausible to use an algorithm such as multilayer error backpropagation which calculates the correction to the value of a synapse that is needed taking into account the values of the errors of the neurons at later stages of the system and the strengths of all the synapses to these neurons (see Section B.12). This use of a local learning rule is a major difference of the networks described in this book, which is directed at neurally plausible computation, from connectionist networks, which typically assume non-local learning rules and which therefore operate very differently from real neural networks in the brain (see Section B.12 and McLeod, Plunkett and Rolls (1998)).

A.2.3 Non-linear networks: neurons with non-linear activations

Most of the networks described in this book calculate the activation h of each neuron as the linear product of the input firing weighted by the synaptic weight vector (see equation A.13). This corresponds in a real neuron to receiving currents from each of its synapses which sum to produce depolarization of the neuronal cell body and the spike initiation region which is located very close to the cell body. This is a reasonable reflection of what does happen in many neurons, especially those with large dendrites such as pyramidal cells (Koch 1999). This calculation of the activation h by a linear summation not only approximates to what happens in many real neurons, but is also a useful simplification which makes tractable the analysis of many classes of network that utilize such neurons. These analyses provide insight into the operation of networks of neurons, even if the linear summation assumption is not perfectly realized. Having computed the activation linearly, the neurons do of course for essentially all the networks described in this book, then utilize a non-linear activation function, which, as described above, provides the networks with much of their interesting computational power. Given that the activation functions of the neurons are non-linear, some non-linearity in the summation expressed in equation A.13 may in practice be lumped into the non-linearity in the activation function.

However, another class of neuron that is implemented in some networks in the brain utilizes non-linearity in the calculation of the activation h of the neuron, which reflects a local product of two inputs to a neuron. This could arise for example if one synapse makes a presynaptic contact with another synapse which in turn connects to the dendrite, or if two synapses are close together on a thin dendrite. In such situations, the current injected into the neuron could reflect the conjoint firing of the two classes of input (Koch 1999). The dendrite as a whole could then sum all such products into the cell body, leading to the description **Sigma-Pi**. This could be expressed by equation A.14

$$h = \sum_j \sum_k w_{jk} x_j x^c{}_k \tag{A.14}$$

where x_j is the firing rate of input cell j, $x^c{}_k$ is the firing rate of input cell k of class c, and w_{jk} is the connection strength. Such Sigma-Pi neurons were utilized in the model described in Section B.5.5 of how idiothetic inputs could update a continuous attractor network. Another possible application is to learning invariant representations in neural networks. For example, the x^c input in equation A.14 could be a signal that varies with the shift required to compute

translation invariance, effectively mapping the appropriate set of x_j inputs through to the output neurons depending on the shift required (Mel, Ruderman and Archie 1998, Mel and Fiser 2000, Olshausen, Anderson and Van Essen 1993, Olshausen, Anderson and Van Essen 1995).

To train such a Sigma-Pi network requires that combinations of the two presynaptic inputs to a neuron be learned onto a neuron, using for example associativity with the post-synaptic term y, as exemplified in equation A.15

$$\delta w_{jk} = \alpha y x_j x^c{}_k. \tag{A.15}$$

This learning principle is exemplified in the model described in Section B.5.5 of how idiothetic inputs could update a continuous attractor network to perform path integration.

Sigma-Pi networks are clearly very powerful, but require rather specialized anatomical and biophysical arrangements (see Koch (1999)), and hence we do not use them unless they become very necessary in models of neural network operations in the brain. We have shown that in at least some applications such as path integration, it is possible to replace a Sigma-Pi network with a competitive network followed by a pattern association network (Stringer and Rolls 2006).

Appendix 2 Neural network models

B.1 Introduction

Formal models of neural networks are needed in order to provide a basis for understanding the processing and memory functions performed by real neuronal networks in the brain. The formal models included in this Appendix all describe fundamental types of network found in different brain regions, and the computations they perform. Each of the types of network described can be thought of as providing one of the fundamental building blocks that the brain uses. Often these building blocks are combined within a brain area to perform a particular computation.

The aim of this Appendix is to describe a set of fundamental networks used by the brain, including the parts of the brain involved in memory, attention, decision-making, and the building of perceptual representations. As each type of network is introduced, we will point briefly to parts of the brain in which each network is found. Understanding these models provides a basis for understanding the theories of how different types of memory functions are performed. The descriptions of these networks are kept relatively concise in this Appendix. More detailed descriptions of some of the quantitative aspects of storage in pattern associators and autoassociators are provided in the Appendices of Rolls and Treves (1998) *Neural Networks and Brain Function*. Another book that provides a clear and quantitative introduction to some of these networks is Hertz, Krogh and Palmer (1991) *Introduction to the Theory of Neural Computation*, and other useful sources include Dayan and Abbott (2001), Amit (1989) (for attractor networks), Koch (1999) (for a biophysical approach), Wilson (1999) (on spiking networks), Gerstner and Kistler (2002) (on spiking networks), and Rolls and Deco (2002).

Some of the background to the operation of the types of neuronal network described here, including a brief review of the evidence on neuronal structure and function, and on synaptic plasticity and the rules by which synaptic strength is modified, much based on studies with long-term potentiation, is provided in Chapter 1.

The network models on which we focus in this Appendix utilize a local learning rule, that is a rule for synaptic modification, in which the signals needed to alter the synaptic strength are present in the pre- and post-synaptic neurons. We focus on these networks because use of a local learning rule is biologically plausible. We discuss the issue of biological plausibility of the networks described, and show how they differ from less biologically plausible networks such as multilayer backpropagation of error networks, in Section B.13.

B.2 Pattern association memory

A fundamental operation of most nervous systems is to learn to associate a first stimulus with a second that occurs at about the same time, and to retrieve the second stimulus when the first is presented. The first stimulus might be the sight of food, and the second stimulus the taste of food. After the association has been learned, the sight of food would enable its taste to be retrieved. In classical conditioning, the taste of food might elicit an unconditioned response of

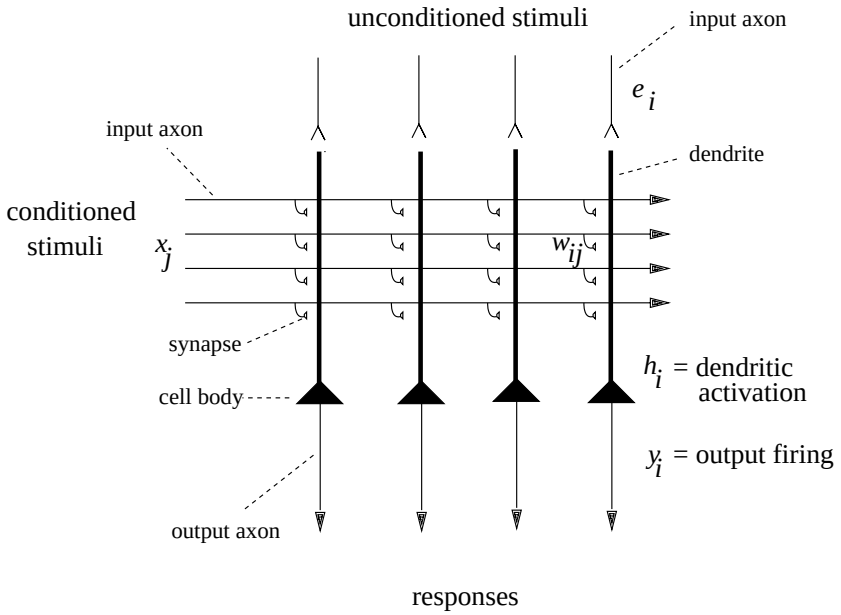


Fig. B.1 A pattern association memory. An unconditioned stimulus has activity or firing rate e_i for the ith neuron, and produces firing y_i of the ith neuron. An unconditioned stimulus may be treated as a vector, across the set of neurons indexed by i, of activity $\mathbf{e}$. The firing rate response can also be thought of as a vector of firing $\mathbf{y}$. The conditioned stimuli have activity or firing rate x_j for the jth axon, which can also be treated as a vector $\mathbf{x}$.

salivation, and if the sight of the food is paired with its taste, then the sight of that food would by learning come to produce salivation. Pattern associators are thus used where the outputs of the visual system interface to learning systems in the orbitofrontal cortex and amygdala that learn associations between the sight of objects and their taste or touch in stimulus–reinforcer association learning (see Chapter 3). Pattern association is also used throughout the cerebral (neo)cortical areas, as it is the architecture that describes the backprojection connections from one cortical area to the preceding cortical area (see Chapters 1, 2 and 6). Pattern association thus contributes to implementing top-down influences in attention, including the effects of attention from higher to lower cortical areas, and thus between the visual object and spatial processing streams (Rolls and Deco 2002) (see Chapter 6); the effects of mood on memory and visual information processing (see Section 3.11); the recall of visual memories; and the operation of short-term memory systems (see Chapter 5).

B.2.1 Architecture and operation

The essential elements necessary for pattern association, forming what could be called a prototypical pattern associator network, are shown in Fig. B.1. What we have called the second or unconditioned stimulus pattern is applied through unmodifiable synapses generating an input to each neuron, which, being external with respect to the synaptic matrix we focus on, we can call the external input e_i for the ith neuron. [We can also treat this as a vector, $\mathbf{e}$, as indicated in the legend to Fig. B.1. Vectors and simple operations performed with them are summarized in Appendix A. This unconditioned stimulus is dominant in producing or forcing the firing of the output neurons (y_i for the ith neuron, or the vector $\mathbf{y}$)]. At the same time, the first or conditioned stimulus pattern consisting of the set of firings on the horizontally running

input axons in Fig. B.1 (x_j for the jth axon) (or equivalently the vector $\mathbf{x}$) is applied through modifiable synapses w_{ij} to the dendrites of the output neurons. The synapses are modifiable in such a way that if there is presynaptic firing on an input axon x_j paired during learning with postsynaptic activity on neuron i, then the strength or weight w_{ij} between that axon and the dendrite increases. This simple learning rule is often called the Hebb rule, after Donald Hebb who in 1949 formulated the hypothesis that if the firing of one neuron was regularly associated with another, then the strength of the synapse or synapses between the neurons should increase[38]. After learning, presenting the pattern $\mathbf{x}$ on the input axons will activate the dendrite through the strengthened synapses. If the cue or conditioned stimulus pattern is the same as that learned, the postsynaptic neurons will be activated, even in the absence of the external or unconditioned input, as each of the firing axons produces through a strengthened synapse some activation of the postsynaptic element, the dendrite. The total activation h_i of each postsynaptic neuron i is then the sum of such individual activations. In this way, the 'correct' output neurons, that is those activated during learning, can end up being the ones most strongly activated, and the second or unconditioned stimulus can be effectively recalled. The recall is best when only strong activation of the postsynaptic neuron produces firing, that is if there is a threshold for firing, just like real neurons. The advantages of this are evident when many associations are stored in the memory, as will soon be shown.

Next we introduce a more precise description of the above by writing down explicit mathematical rules for the operation of the simple network model of Fig. B.1, which will help us to understand how pattern association memories in general operate. (In this description we introduce simple vector operations, and, for those who are not familiar with these, refer the reader to for example Appendix 1 of Rolls and Deco (2002).) We have denoted above a conditioned stimulus input pattern as $\mathbf{x}$. Each of the axons has a firing rate, and if we count or index through the axons using the subscript j, the firing rate of the first axon is x_1, of the second x_2, of the jth x_j, etc. The whole set of axons forms a vector, which is just an ordered (1, 2, 3, etc.) set of elements. The firing rate of each axon x_j is one element of the firing rate vector $\mathbf{x}$. Similarly, using i as the index, we can denote the firing rate of any output neuron as y_i, and the firing rate output vector as $\mathbf{y}$. With this terminology, we can then identify any synapse onto neuron i from neuron j as w_{ij} (see Fig. B.1). In this book, the first index, i, always refers to the receiving neuron (and thus signifies a dendrite), while the second index, j, refers to the sending neuron (and thus signifies a conditioned stimulus input axon in Fig. B.1). We can now specify the learning and retrieval operations as follows:

B.2.1.1 Learning

The firing rate of every output neuron is forced to a value determined by the unconditioned (or external or forcing stimulus) input e_i. In our simple model this means that for any one neuron i,

$$y_i = \mathrm{f}(e_i) \tag{B.1}$$

which indicates that the firing rate is a function of the dendritic activation, taken in this case to reduce essentially to that resulting from the external forcing input (see Fig. B.1). The function f is called the activation function (see Fig. 1.3), and its precise form is irrelevant, at least during this learning phase. For example, the function at its simplest could be taken to be linear, so that the firing rate would be just proportional to the activation.

[38] In fact, the terms in which Hebb put the hypothesis were a little different from an association memory, in that he stated that if one neuron regularly comes to elicit firing in another, then the strength of the synapses should increase. He had in mind the building of what he called cell assemblies. In a pattern associator, the conditioned stimulus need not produce before learning any significant activation of the output neurons. The connection strengths must simply increase if there is associated pre- and postsynaptic firing when, in pattern association, most of the postsynaptic firing is being produced by a different input.

The Hebb rule can then be written as follows:

$$\delta w_{ij} = \alpha y_i x_j \tag{B.2}$$

where δw_{ij} is the change of the synaptic weight w_{ij} that results from the simultaneous (or conjunctive) presence of presynaptic firing x_j and postsynaptic firing or activation y_i, and α is a learning rate constant that specifies how much the synapses alter on any one pairing.

The Hebb rule is expressed in this multiplicative form to reflect the idea that both pre-synaptic and postsynaptic activity must be present for the synapses to increase in strength. The multiplicative form also reflects the idea that strong pre- and postsynaptic firing will produce a larger change of synaptic weight than smaller firing rates. It is also assumed for now that before any learning takes place, the synaptic strengths are small in relation to the changes that can be produced during Hebbian learning. We will see that this assumption can be relaxed later when a modified Hebb rule is introduced that can lead to a reduction in synaptic strength under some conditions.

B.2.1.2 Recall

When the conditioned stimulus is present on the input axons, the total activation h_i of a neuron i is the sum of all the activations produced through each strengthened synapse w_{ij} by each active neuron x_j. We can express this as

$$h_i = \sum_{j=1}^{C} x_j w_{ij} \tag{B.3}$$

where $\sum_{j=1}^{C}$ indicates that the sum is over the C input axons (or connections) indexed by j to each neuron.

The multiplicative form here indicates that activation should be produced by an axon only if it is firing, and only if it is connected to the dendrite by a strengthened synapse. It also indicates that the strength of the activation reflects how fast the axon x_j is firing, and how strong the synapse w_{ij} is. The sum of all such activations expresses the idea that summation (of synaptic currents in real neurons) occurs along the length of the dendrite, to produce activation at the cell body, where the activation h_i is converted into firing y_i. This conversion can be expressed as

$$y_i = \mathrm{f}(h_i) \tag{B.4}$$

where the function f is again the activation function. The form of the function now becomes more important. Real neurons have thresholds, with firing occurring only if the activation is above the threshold. A threshold linear activation function is shown in Fig. 1.3b on page 6. This has been useful in formal analysis of the properties of neural networks. Neurons also have firing rates that become saturated at a maximum rate, and we could express this as the sigmoid activation function shown in Fig. 1.3c. Yet another simple activation function, used in some models of neural networks, is the binary threshold function (Fig. 1.3d), which indicates that if the activation is below threshold, there is no firing, and that if the activation is above threshold, the neuron fires maximally. Whatever the exact shape of the activation function, some non-linearity is an advantage, for it enables small activations produced by interfering memories to be minimized, and it can enable neurons to perform logical operations, such as to fire or respond only if two or more sets of inputs are present simultaneously.

B.2.2 A simple model

An example of these learning and recall operations is provided in a simple form as follows. The neurons will have simple firing rates, which can be 0 to represent no activity, and 1 to indicate high firing. They are thus binary neurons, which can assume one of two firing rates. If we have a pattern associator with six input axons and four output neurons, we could represent the network before learning, with the same layout as in Fig. B.1, as shown in Fig. B.2:

	U	C	S	
	1	1	0	0
	↓	↓	↓	↓
CS				
1 →	0	0	0	0
0 →	0	0	0	0
1 →	0	0	0	0
0 →	0	0	0	0
1 →	0	0	0	0
0 →	0	0	0	0

Fig. B.2 Pattern association: before synaptic modification. The unconditioned stimulus (UCS) firing rates are shown as 1 if high and 0 if low as a row vector being applied to force firing of the four output neurons. The six conditioned stimulus (CS) firing rates are shown as a column vector being applied to the vertical dendrites of the output neurons which have initial synaptic weights of 0.

where $\mathbf{x}$ or the conditioned stimulus (CS) is 101010, and $\mathbf{y}$ or the firing produced by the unconditioned stimulus (UCS) is 1100. (The arrows indicate the flow of signals.) The synaptic weights are initially all 0.

After pairing the CS with the UCS during one learning trial, some of the synaptic weights will be incremented according to equation B.2, so that after learning this pair the synaptic weights will become as shown in Fig. B.3:

	U	C	S	
	1	1	0	0
	↓	↓	↓	↓
CS				
1 →	1	1	0	0
0 →	0	0	0	0
1 →	1	1	0	0
0 →	0	0	0	0
1 →	1	1	0	0
0 →	0	0	0	0

Fig. B.3 Pattern association: after synaptic modification. The synapses where there is conjunctive pre- and post-synaptic activity have been strengthened to value 1.

We can represent what happens during recall, when, for example, we present the CS that has been learned, as shown in Fig. B.4:

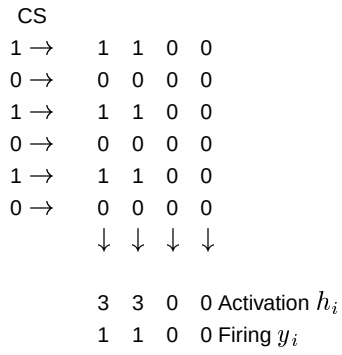


Fig. B.4 Pattern association: recall. The activation h_i of each neuron i is converted with a threshold of 2 to the binary firing rate y_i (1 for high, and 0 for low).

The activation of the four output neurons is 3300, and if we set the threshold of each output neuron to 2, then the output firing is 1100 (where the binary firing rate is 0 if below threshold, and 1 if above). The pattern associator has thus achieved recall of the pattern 1100, which is correct.

We can now illustrate how a number of different associations can be stored in such a pattern associator, and retrieved correctly. Let us associate a new CS pattern 110001 with the UCS 0101 in the same pattern associator. The weights will become as shown next in Fig. B.5 after learning:

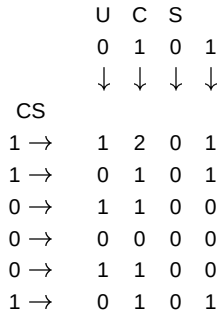


Fig. B.5 Pattern association: synaptic weights after learning a second pattern association.

If we now present the second CS, the retrieval is as shown in Fig. B.6:

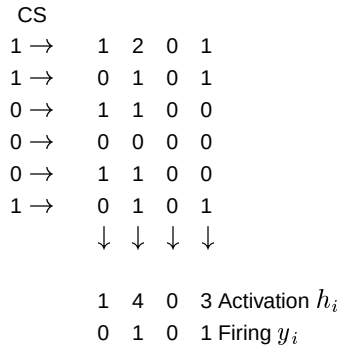


Fig. B.6 Pattern association: recall with the second CS.

The binary output firings were again produced with the threshold set to 2. Recall is perfect.

This illustration shows the value of some threshold non-linearity in the activation function of the neurons. In this case, the activations did reflect some small cross-talk or interference from the previous pattern association of CS1 with UCS1, but this was removed by the threshold operation, to clean up the recall firing. The example also shows that when further associations are learned by a pattern associator trained with the Hebb rule, equation B.2, some synapses will reflect increments above a synaptic strength of 1. It is left as an exercise to the reader to verify that recall is still perfect to CS1, the vector 101010. (The activation vector $\mathbf{h}$ is 3401, and the output firing vector $\mathbf{y}$ with the same threshold of 2 is 1100, which is perfect recall.)

B.2.3 The vector interpretation

The way in which recall is produced, equation B.3, consists for each output neuron i of multiplying each input firing rate x_j by the corresponding synaptic weight w_{ij} and summing the products to obtain the activation h_i. Now we can consider the firing rates x_j where j varies from 1 to N', the number of axons, to be a vector. (A vector is simply an ordered set of numbers – see Appendix A.) Let us call this vector $\mathbf{x}$. Similarly, on a neuron i, the synaptic weights can be treated as a vector, $\mathbf{w}_i$. (The subscript i here indicates that this is the weight vector on the ith neuron.) The operation we have just described to obtain the activation of an output neuron can now be seen to be a simple multiplication operation of two vectors to produce a single output value (called a scalar output). This is the inner product or dot product of two vectors, and can be written

$$h_i = \mathbf{x} \cdot \mathbf{w}_i. \tag{B.5}$$

The inner product of two vectors indicates how similar they are. If two vectors have corresponding elements the same, then the dot product will be maximal. If the two vectors are similar but not identical, then the dot product will be high. If the two vectors are completely different, the dot product will be 0, and the vectors are described as orthogonal. (The term orthogonal means at right angles, and arises from the geometric interpretation of vectors, which is summarized in Appendix A.) Thus the dot product provides a direct measure of how similar two vectors are.

It can now be seen that a fundamental operation many neurons perform is effectively to compute how similar an input pattern vector $\mathbf{x}$ is to their stored weight vector $\mathbf{w}_i$. The

similarity measure they compute, the dot product, is a very good measure of similarity, and indeed, the standard (Pearson product–moment) correlation coefficient used in statistics is the same as a normalized dot product with the mean subtracted from each vector, as shown in Appendix A. (The normalization used in the correlation coefficient results in the coefficient varying always between $+1$ and -1, whereas the actual scalar value of a dot product clearly depends on the length of the vectors from which it is calculated.)

With these concepts, we can now see that during learning, a pattern associator adds to its weight vector a vector $\delta \mathbf{w}_i$ that has the same pattern as the input pattern $\mathbf{x}$, if the postsynaptic neuron i is strongly activated. Indeed, we can express equation B.2 in vector form as

$$\delta \mathbf{w}_i = \alpha y_i \mathbf{x}. \tag{B.6}$$

We can now see that what is recalled by the neuron depends on the similarity of the recall cue vector $\mathbf{x}_r$ to the originally learned vector $\mathbf{x}$. The fact that during recall the output of each neuron reflects the similarity (as measured by the dot product) of the input pattern $\mathbf{x}_r$ to each of the patterns used originally as $\mathbf{x}$ inputs (conditioned stimuli in Fig. B.1) provides a simple way to appreciate many of the interesting and biologically useful properties of pattern associators, as described next.

B.2.4 Properties

B.2.4.1 Generalization

During recall, pattern associators generalize, and produce appropriate outputs if a recall cue vector $\mathbf{x}_r$ is similar to a vector that has been learned already. This occurs because the recall operation involves computing the dot (inner) product of the input pattern vector $\mathbf{x}_r$ with the synaptic weight vector $\mathbf{w}_i$, so that the firing produced, y_i, reflects the similarity of the current input to the previously learned input pattern $\mathbf{x}$. (Generalization will occur to input cue or conditioned stimulus patterns $\mathbf{x}_r$ that are incomplete versions of an original conditioned stimulus $\mathbf{x}$, although the term completion is usually applied to the autoassociation networks described in Section B.3.)

This is an extremely important property of pattern associators, for input stimuli during recall will rarely be absolutely identical to what has been learned previously, and automatic generalization to similar stimuli is extremely useful, and has great adaptive value in biological systems.

Generalization can be illustrated with the simple binary pattern associator considered above. (Those who have appreciated the vector description just given might wish to skip this illustration.) Instead of the second CS, pattern vector 110001, we will use the similar recall cue 110100, as shown in Fig. B.7:

CS					
1 →	1	2	0	1	
1 →	0	1	0	1	
0 →	1	1	0	0	
1 →	0	0	0	0	
0 →	1	1	0	0	
0 →	0	1	0	1	
	↓	↓	↓	↓	
	1	3	0	2	Activation h_i
	0	1	0	1	Firing y_i

Fig. B.7 Pattern association: generalization using an input vector similar to the second CS.

It is seen that the output firing rate vector, 0101, is exactly what should be recalled to CS2 (and not to CS1), so correct generalization has occurred. Although this is a small network trained with few examples, the same properties hold for large networks with large numbers of stored patterns, as described more quantitatively in Section B.2.7.1 on capacity below and in Appendix A3 of Rolls and Treves (1998).

B.2.4.2 Graceful degradation or fault tolerance

If the synaptic weight vector $\mathbf{w}_i$ (or the weight matrix, which we can call $\mathbf{W}$) has synapses missing (e.g. during development), or loses synapses, then the activation h_i or $\mathbf{h}$ is still reasonable, because h_i is the dot product (correlation) of $\mathbf{x}$ with $\mathbf{w}_i$. The result, especially after passing through the activation function, can frequently be perfect recall. The same property arises if for example one or some of the conditioned stimulus (CS) input axons are lost or damaged. This is a very important property of associative memories, and is not a property of conventional computer memories, which produce incorrect data if even only 1 storage location (for 1 bit or binary digit of data) of their memory is damaged or cannot be accessed. This property of graceful degradation is of great adaptive value for biological systems.

We can illustrate this with a simple example. If we damage two of the synapses in Fig. B.6 to produce the synaptic matrix shown in Fig. B.8 (where x indicates a damaged synapse which has no effect, but was previously 1), and now present the second CS, the retrieval is as follows:

CS					
1 →	1	2	0	1	
1 →	0	1	0	x	
0 →	1	1	0	0	
0 →	0	0	0	0	
0 →	1	x	0	0	
1 →	0	1	0	1	
	↓	↓	↓	↓	
	1	4	0	2	Activation h_i
	0	1	0	1	Firing y_i

Fig. B.8 Pattern association: graceful degradation when some synapses are damaged (x).

The binary output firings were again produced with the threshold set to 2. The recalled vector, 0101, is perfect. This illustration again shows the value of some threshold non-linearity in the activation function of the neurons. It is left as an exercise to the reader to verify that recall is still perfect to CS1, the vector 101010. (The output activation vector $\mathbf{h}$ is 3301, and the output firing vector $\mathbf{y}$ with the same threshold of 2 is 1100, which is perfect recall.)

B.2.4.3 The importance of distributed representations for pattern associators

A distributed representation is one in which the firing or activity of all the elements in the vector is used to encode a particular stimulus. For example, in a conditioned stimulus vector CS1 that has the value 101010, we need to know the state of all the elements to know which stimulus is being represented. Another stimulus, CS2, is represented by the vector 110001. We can represent many different events or stimuli with such overlapping sets of elements, and because in general any one element cannot be used to identify the stimulus, but instead the information about which stimulus is present is distributed over the population of elements or neurons, this is called a distributed representation (see Section 1.6). If, for binary neurons, half the neurons are in one state (e.g. 0), and the other half are in the other state (e.g. 1), then the representation is described as fully distributed. The CS representations above are thus fully distributed. If only a smaller proportion of the neurons is active to represent a stimulus, as in the vector 100001, then this is a sparse representation. For binary representations, we can quantify the sparseness by the proportion of neurons in the active (1) state.

In contrast, a local representation is one in which all the information that a particular stimulus or event has occurred is provided by the activity of one of the neurons, or elements in the vector. One stimulus might be represented by the vector 100000, another stimulus by the vector 010000, and a third stimulus by the vector 001000. The activity of neuron or element 1 would indicate that stimulus 1 was present, and of neuron 2, that stimulus 2 was present. The representation is local in that if a particular neuron is active, we know that the stimulus represented by that neuron is present. In neurophysiology, if such cells were present, they might be called 'grandmother cells' (cf. Barlow (1972), (1995)), in that one neuron might represent a stimulus in the environment as complex and specific as one's grandmother. Where the activity of a number of cells must be taken into account in order to represent a stimulus (such as an individual taste), then the representation is sometimes described as using ensemble encoding.

The properties just described for associative memories, generalization, and graceful degradation are only implemented if the representation of the CS or $\mathbf{x}$ vector is distributed. This occurs because the recall operation involves computing the dot (inner) product of the input pattern vector $\mathbf{x}_r$ with the synaptic weight vector $\mathbf{w}_i$. This allows the activation h_i to reflect the similarity of the current input pattern to a previously learned input pattern $\mathbf{x}$ only if several or many elements of the $\mathbf{x}$ and $\mathbf{x}_r$ vectors are in the active state to represent a pattern. If local encoding were used, e.g. 100000, then if the first element of the vector (which might be the firing of axon 1, i.e. x_1, or the strength of synapse $i1$, w_{i1}) is lost, the resulting vector is not similar to any other CS vector, and the activation is 0. In the case of local encoding, the important properties of associative memories, generalization and graceful degradation do not thus emerge. Graceful degradation and generalization are dependent on distributed representations, for then the dot product can reflect similarity even when some elements of the vectors involved are altered. If we think of the correlation between Y and X in a graph, then this correlation is affected only a little if a few X, Y pairs of data are lost (see Appendix A).

B.2.5 Prototype extraction, extraction of central tendency, and noise reduction

If a set of similar conditioned stimulus vectors $\mathbf{x}$ are paired with the same unconditioned stimulus e_i, the weight vector $\mathbf{w}_i$ becomes (or points towards) the sum (or with scaling, the average) of the set of similar vectors $\mathbf{x}$. This follows from the operation of the Hebb rule in equation B.2. When tested at recall, the output of the memory is then best to the average input pattern vector denoted $< \mathbf{x} >$. If the average is thought of as a prototype, then even though the prototype vector $< \mathbf{x} >$ itself may never have been seen, the best output of the neuron or network is to the prototype. This produces 'extraction of the prototype' or 'central tendency'. The same phenomenon is a feature of human memory performance (see McClelland and Rumelhart (1986) Chapter 17), and this simple process with distributed representations in a neural network accounts for the psychological phenomenon.

If the different exemplars of the vector $\mathbf{x}$ are thought of as noisy versions of the true input pattern vector $< \mathbf{x} >$ (with incorrect values for some of the elements), then the pattern associator has performed 'noise reduction', in that the output produced by any one of these vectors will represent the output produced by the true, noiseless, average vector $< \mathbf{x} >$.

B.2.6 Speed

Recall is very fast in a real neuronal network, because the conditioned stimulus input firings x_j ($j = 1, C$ axons) can be applied simultaneously to the synapses w_{ij}, and the activation h_i can be accumulated in one or two time constants of the dendrite (e.g. 10–20 ms). Whenever the threshold of the cell is exceeded, it fires. Thus, in effectively one step, which takes the brain no more than 10–20 ms, all the output neurons of the pattern associator can be firing with rates that reflect the input firing of every axon. This is very different from a conventional digital computer, in which computing h_i in equation B.3 would involve C multiplication and addition operations occurring one after another, or $2C$ time steps.

The brain performs parallel computation in at least two senses in even a pattern associator. One is that for a single neuron, the separate contributions of the firing rate x_j of each axon j multiplied by the synaptic weight w_{ij} are computed in parallel and added in the same timestep. The second is that this can be performed in parallel for all neurons $i = 1, N$ in the network, where there are N output neurons in the network. It is these types of parallel and time-continuous (see Section B.6) processing that enable these classes of neuronal network in the brain to operate so fast, in effectively so few steps.

Learning is also fast ('one-shot') in pattern associators, in that a single pairing of the conditioned stimulus $\mathbf{x}$ and the unconditioned stimulus (UCS) $\mathbf{e}$ which produces the unconditioned output firing $\mathbf{y}$ enables the association to be learned. There is no need to repeat the pairing in order to discover over many trials the appropriate mapping. This is extremely important for biological systems, in which a single co-occurrence of two events may lead to learning that could have life-saving consequences. (For example, the pairing of a visual stimulus with a potentially life-threatening aversive event may enable that event to be avoided in future.) Although repeated pairing with small variations of the vectors is used to obtain the useful properties of prototype extraction, extraction of central tendency, and noise reduction, the essential properties of generalization and graceful degradation are obtained with just one pairing. The actual time scales of the learning in the brain are indicated by studies of associative synaptic modification using long-term potentiation paradigms (LTP, see Section 1.5). Co-occurrence or near simultaneity of the CS and UCS is required for periods of as little as 100 ms, with expression of the synaptic modification being present within typically a few seconds.

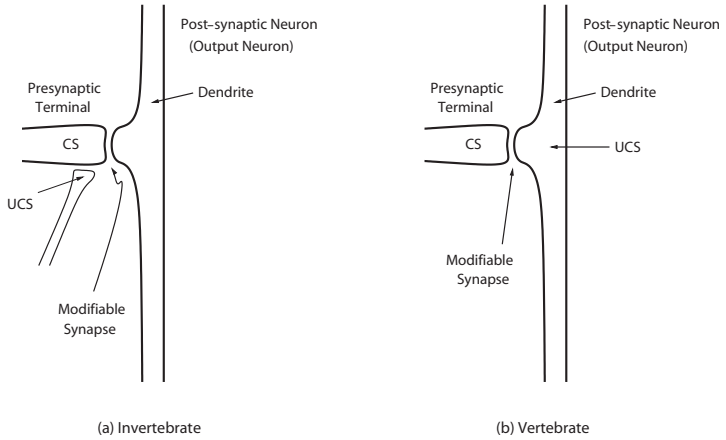


Fig. B.9 (b) In vertebrate pattern association learning, the unconditioned stimulus (UCS) may be made available at all the conditioned stimulus (CS) terminals onto the output neuron because the dendrite of the postsynaptic neuron is electrically short, so that the effect of the UCS spreads for long distances along the dendrite. (a) In contrast, in at least some invertebrate association learning systems, the unconditioned stimulus or teaching input makes a synapse onto the presynaptic terminal carrying the conditioned stimulus.

B.2.7 Local learning rule

The simplest learning rule used in pattern association neural networks, a version of the Hebb rule, is, as shown in equation B.2 above,

$$\delta w_{ij} = \alpha y_i x_j.$$

This is a local learning rule in that the information required to specify the change in synaptic weight is available locally at the synapse, as it is dependent only on the presynaptic firing rate x_j available at the synaptic terminal, and the postsynaptic activation or firing y_i available on the dendrite of the neuron receiving the synapse (see Fig. B.9b). This makes the learning rule biologically plausible, in that the information about how to change the synaptic weight does not have to be carried from a distant source, where it is computed, to every synapse. Such a non-local learning rule would not be biologically plausible, in that there are no appropriate connections known in most parts of the brain to bring in the synaptic training or teacher signal to every synapse.

Evidence that a learning rule with the general form of equation B.2 is implemented in at least some parts of the brain comes from studies of long-term potentiation, described in Section 1.5. Long-term potentiation (LTP) has the synaptic specificity defined by equation B.2, in that only synapses from active afferents, not those from inactive afferents, become strengthened. Synaptic specificity is important for a pattern associator, and most other types of neuronal network, to operate correctly. The number of independently modifiable synapses on each neuron is a primary factor in determining how many different memory patterns can be stored in associative memories (see Sections B.2.7.1 and B.3.3.7).

Another useful property of real neurons in relation to equation B.2 is that the postsynaptic term, y_i, is available on much of the dendrite of a cell, because the electrotonic length of the dendrite is short. In addition, active propagation of spiking activity from the cell body along the dendrite may help to provide a uniform postsynaptic term for the learning. Thus if a neuron is strongly activated with a high value for y_i, then any active synapse onto the cell will be capable of being modified. This enables the cell to learn an association between the pattern

of activity on all its axons and its postsynaptic activation, which is stored as an addition to its weight vector $\mathbf{w}_i$. Then later on, at recall, the output can be produced as a vector dot product operation between the input pattern vector $\mathbf{x}$ and the weight vector $\mathbf{w}_i$, so that the output of the cell can reflect the correlation between the current input vector and what has previously been learned by the cell.

It is interesting that at least many invertebrate neuronal systems may operate very differently from those described here, as described by Rolls and Treves (1998) (see Fig. B.9a). If there were 5,000 conditioned stimulus inputs to a neuron, the implication is that every one would need to have a presynaptic terminal conveying the same UCS to each presynaptic terminal, which is hardly plausible. The implication is that at least some invertebrate neural systems operate very differently to those in vertebrates and, in such systems, the useful properties that arise from using distributed CS representations such as generalization would not arise in the same simple way as a property of the network.

B.2.7.1 Capacity

The question of the storage capacity of a pattern associator is considered in detail in Appendix A3 of Rolls and Treves (1998). It is pointed out there that, for this type of associative network, the number of memories that it can hold simultaneously in storage has to be analysed together with the retrieval quality of each output representation, and then only for a given quality of the representation provided in the input. This is in contrast to autoassociative nets (Section B.3), in which a critical number of stored memories exists (as a function of various parameters of the network), beyond which attempting to store additional memories results in it becoming impossible to retrieve essentially anything. With a pattern associator, instead, one will always retrieve something, but this something will be very small (in information or correlation terms) if too many associations are simultaneously in storage and/or if too little is provided as input.

The conjoint quality–capacity input analysis can be carried out, for any specific instance of a pattern associator, by using formal mathematical models and established analytical procedures (see e.g. Treves (1995), Rolls and Treves (1998), Treves (1990) and Rolls and Treves (1990)). This, however, has to be done case by case. It is anyway useful to develop some intuition for how a pattern associator operates, by considering what its capacity would be in certain well-defined simplified cases.

Linear associative neuronal networks These networks are made up of units with a linear activation function, which appears to make them unsuitable to represent real neurons with their positive-only firing rates. However, even purely linear units have been considered as provisionally relevant models of real neurons, by assuming that the latter operate sometimes in the linear regime of their transfer function. (This implies a high level of spontaneous activity, and may be closer to conditions observed early on in sensory systems rather than in areas more specifically involved in memory.) As usual, the connections are trained by a Hebb (or similar) associative learning rule. The capacity of these networks can be defined as the total number of associations that can be learned independently of each other, given that the linear nature of these systems prevents anything more than a linear transform of the inputs. This implies that if input pattern C can be written as the weighted sum of input patterns A and B, the output to C will be just the same weighted sum of the outputs to A and B. If there are N' input axons, then there can be only at most N' mutually independent input patterns (i.e. none able to be written as a weighted sum of the others), and therefore the capacity of linear networks, defined above, is just N', or equal to the number of inputs to each neuron. In general, a random set of less than N' vectors (the CS input pattern vectors) will tend to be mutually independent but not mutually orthogonal (at $90\,\mathrm{deg}$ to each other) (see Appendix A). If they are not orthogonal (the normal situation), then the dot product of them is not 0,

and the output pattern activated by one of the input vectors will be partially activated by other input pattern vectors, in accordance with how similar they are (see equations B.5 and B.6). This amounts to interference, which is therefore the more serious the less orthogonal, on the whole, is the set of input vectors.

Since input patterns are made of elements with positive values, if a simple Hebbian learning rule like the one of equation B.2 is used (in which the input pattern enters directly with no subtraction term), the output resulting from the application of a stored input vector will be the sum of contributions from all other input vectors that have a non-zero dot product with it (see Appendix A), and interference will be disastrous. The only situation in which this would not occur is when different input patterns activate completely different input lines, but this is clearly an uninteresting circumstance for networks operating with distributed representations. A solution to this issue is to use a modified learning rule of the following form:

$$\delta w_{ij} = \alpha y_i (x_j - x) \tag{B.7}$$

where x is a constant, approximately equal to the average value of x_j. This learning rule includes (in proportion to y_i) increasing the synaptic weight if $(x_j - x) > 0$ (long-term potentiation), and decreasing the synaptic weight if $(x_j - x) < 0$ (heterosynaptic long-term depression). It is useful for x to be roughly the average activity of an input axon x_j across patterns, because then the dot product between the various patterns stored on the weights and the input vector will tend to cancel out with the subtractive term, except for the pattern equal to (or correlated with) the input vector itself. Then up to N' input vectors can still be learned by the network, with only minor interference (provided of course that they are mutually independent, as they will in general tend to be).

Table B.1 Effects of pre- and post-synaptic activity on synaptic modification

		Post-synaptic activation	
		0	high
Presynaptic firing	0	No change	Heterosynaptic LTD
	high	Homosynaptic LTD	LTP

This modified learning rule can also be described in terms of a contingency table (Table B.1) showing the synaptic strength modifications produced by different types of learning rule, where LTP indicates an increase in synaptic strength (called long-term potentiation in neurophysiology), and LTD indicates a decrease in synaptic strength (called long-term depression in neurophysiology). Heterosynaptic long-term depression is so-called because it is the decrease in synaptic strength that occurs to a synapse that is other than that through which the postsynaptic cell is being activated. This heterosynaptic long-term depression is the type of change of synaptic strength that is required (in addition to LTP) for effective subtraction of the average presynaptic firing rate, in order, as it were, to make the CS vectors appear more orthogonal to the pattern associator. The rule is sometimes called the Singer–Stent rule, after work by Singer (1987) and Stent (1973), and was discovered in the brain by Levy (Levy 1985, Levy and Desmond 1985) (see also Brown, Kairiss and Keenan (1990b)). Homosynaptic long-term depression is so-called because it is the decrease in synaptic strength that occurs to a synapse which is (the same as that which is) active. For it to occur, the postsynaptic neuron must

simultaneously be inactive, or have only low activity. (This rule is sometimes called the BCM rule after the paper of Bienenstock, Cooper and Munro (1982); see Rolls and Deco (2002), Chapter 7).

Associative neuronal networks with non-linear neurons With non-linear neurons, that is with at least a threshold in the activation function so that the output firing y_i is 0 when the activation h_i is below the threshold, the capacity can be measured in terms of the number of different clusters of output pattern vectors that the network produces. This is because the non-linearities now present (one per output neuron) result in some clustering of the outputs produced by all possible (conditioned stimulus) input patterns $\mathbf{x}$. Input patterns that are similar to a stored input vector can produce, due to the non-linearities, output patterns even closer to the stored output; and vice versa, sufficiently dissimilar inputs can be assigned to different output clusters thereby increasing their mutual dissimilarity. As with the linear counterpart, in order to remove the correlation that would otherwise occur between the patterns because the elements can take only positive values, it is useful to use a modified Hebb rule of the form shown in equation B.7.

With fully distributed output patterns, the number p of associations that leads to different clusters is of order C, the number of input lines (axons) per output neuron (that is, of order N' for a fully connected network), as shown in Appendix A3 of Rolls and Treves (1998). If sparse patterns are used in the output, or alternatively if the learning rule includes a non-linear postsynaptic factor that is effectively equivalent to using sparse output patterns, the coefficient of proportionality between p and C can be much higher than one, that is, many more patterns can be stored than inputs onto each output neuron (see Appendix A3 of Rolls and Treves (1998)). Indeed, the number of different patterns or prototypes p that can be stored can be derived for example in the case of binary units (Gardner 1988) to be

$$p \approx C/[a_o log(1/a_o)] \tag{B.8}$$

where a_o is the sparseness of the output firing pattern $\mathbf{y}$ produced by the unconditioned stimulus. p can in this situation be much larger than C (see Appendix A3 of Rolls and Treves (1998), Rolls and Treves (1990) and Treves (1990)). This is an important result for encoding in pattern associators, for it means that provided that the activation functions are non-linear (which is the case with real neurons), there is a very great advantage to using sparse encoding, for then many more than C pattern associations can be stored. Sparse representations may well be present in brain regions involved in associative memory for this reason (see Appendix C).

The non-linearity inherent in the NMDA receptor-based Hebbian plasticity present in the brain may help to make the stored patterns more sparse than the input patterns, and this may be especially beneficial in increasing the storage capacity of associative networks in the brain by allowing participation in the storage of especially those relatively few neurons with high firing rates in the exponential firing rate distributions typical of neurons in sensory systems (see Appendix C).

B.2.7.2 Interference

Interference occurs in linear pattern associators if two vectors are not orthogonal, and is simply dependent on the angle between the originally learned vector and the recall cue or CS vector (see Appendix A), for the activation of the output neuron depends simply on the dot product of the recall vector and the synaptic weight vector (equation B.5). Also in non-linear pattern associators (the interesting case for all practical purposes), interference may occur if two CS patterns are not orthogonal, though the effect can be controlled with sparse encoding of the UCS patterns, effectively by setting high thresholds for the firing of output units. In

Input A	1	0	1
Input B	0	1	1
Required Output	1	1	0

Fig. B.10 A non-linearly separable mapping.

other words, the CS vectors need not be strictly orthogonal, but if they are too similar, some interference will still be likely to occur.

The fact that interference is a property of neural network pattern associator memories is of interest, for interference is a major property of human memory. Indeed, the fact that interference is a property of human memory and of neural network association memories is entirely consistent with the hypothesis that human memory is stored in associative memories of the type described here, or at least that network associative memories of the type described represent a useful exemplar of the class of parallel distributed storage network used in human memory.

It may also be suggested that one reason that interference is tolerated in biological memory is that it is associated with the ability to generalize between stimuli, which is an invaluable feature of biological network associative memories, in that it allows the memory to cope with stimuli that will almost never be identical on different occasions, and in that it allows useful analogies that have survival value to be made.

B.2.7.3 Expansion recoding

If patterns are too similar to be stored in associative memories, then one solution that the brain seems to use repeatedly is to expand the encoding to a form in which the different stimulus patterns are less correlated, that is, more orthogonal, before they are presented as CS stimuli to a pattern associator. The problem can be highlighted by a non-linearly separable mapping (which captures part of the eXclusive OR (XOR) problem), in which the mapping that is desired is as shown in Fig. B.10. The neuron has two inputs, A and B.

This is a mapping of patterns that is impossible for a one-layer network, because the patterns are not linearly separable[39]. A solution is to remap the two input lines A and B to three input lines 1–3, that is to use expansion recoding, as shown in Fig. B.11. This can be performed by a competitive network (see Section B.4). The synaptic weights on the dendrite of the output neuron could then learn the following values using a simple Hebb rule, equation B.2, and the problem could be solved as in Fig. B.12. The whole network would look like that shown in Fig. B.11.

Rolls and Treves (1998) show that competitive networks could help with this type of recoding, and could provide very useful preprocessing for a pattern associator in the brain. It is possible that the lateral nucleus of the amygdala performs this function, for it receives inputs from the temporal cortical visual areas, and may preprocess them before they become the inputs to associative networks at the next stage of amygdala processing (see Fig. 3.41). The granule cells of the cerebellum may operate similarly (see Rolls and Treves (1998)).

[39] See Appendix A. There is no set of synaptic weights in a one-layer net that could solve the problem shown in Fig. B.10. Two classes of patterns are not linearly separable if no hyperplane can be positioned in their N-dimensional space so as to separate them (see Appendix A). The XOR problem has the additional constraint that $A = 0, B = 0$ must be mapped to Output = 0.

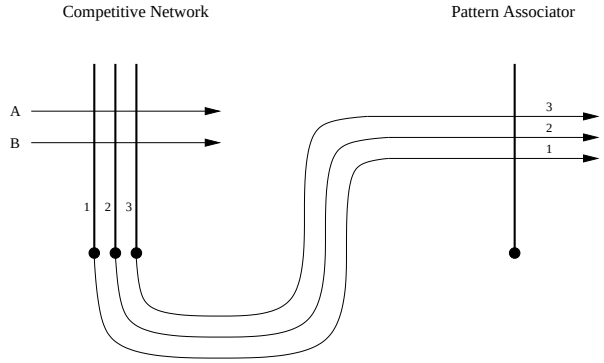


Fig. B.11 Expansion recoding. A competitive network followed by a pattern associator that can enable patterns that are not linearly separable to be learned correctly.

	Synaptic weight
Input 1 (A=1, B=0)	1
Input 2 (A=0, B=1)	1
Input 3 (A=1, B=1)	0

Fig. B.12 Synaptic weights on the dendrite of the output neuron in Fig. B.11.

B.2.8 Implications of different types of coding for storage in pattern associators

Throughout this section, we have made statements about how the properties of pattern associators – such as the number of patterns that can be stored, and whether generalization and graceful degradation occur – depend on the type of encoding of the patterns to be associated. (The types of encoding considered, local, sparse distributed, and fully distributed, are described above.) We draw together these points in Table B.2.

Table B.2 Coding in associative memories*

	Local	**Sparse distributed**	**Fully distributed**
Generalization, completion, graceful degradation	No	Yes	Yes
Number of patterns that can be stored	N (large)	of order $C/[a_o \log(1/a_o)]$ (can be larger)	of order C (usually smaller than N)
Amount of information in each pattern (values if binary)	Minimal (log(N) bits)	Intermediate ($N a_o \log(1/a_o)$ bits)	Large (N bits)

* N refers here to the number of output units, and C to the average number of inputs to each output unit. a_o is the sparseness of output patterns, or roughly the proportion of output units activated by a UCS pattern. Note: logs are to the base 2.

The amount of information that can be stored in each pattern in a pattern associator is considered in Appendix A3 of Rolls and Treves (1998).

In conclusion, the architecture and properties of pattern association networks make them very appropriate for stimulus–reinforcer association learning. Their high capacity enables them to learn the reinforcement associations for very large numbers of different stimuli.

B.3 Autoassociation or attractor memory

Autoassociative memories, or attractor neural networks, store memories, each one of which is represented by a pattern of neural activity. The memories are stored in the recurrent synaptic connections between the neurons of the network, for example in the recurrent collateral connections between cortical pyramidal cells. Autoassociative networks can then recall the appropriate memory from the network when provided with a fragment of one of the memories. This is called completion. Many different memories can be stored in the network and retrieved correctly. A feature of this type of memory is that it is content addressable; that is, the information in the memory can be accessed if just the contents of the memory (or a part of the contents of the memory) are used. This is in contrast to a conventional computer, in which the address of what is to be accessed must be supplied, and used to access the contents of the memory. Content addressability is an important simplifying feature of this type of memory, which makes it suitable for use in biological systems. The issue of content addressability will be amplified below.

An autoassociation memory can be used as a short-term memory, in which iterative processing round the recurrent collateral connection loop keeps a representation active by continuing neuronal firing. The short-term memory reflected in continuing neuronal firing for several hundred milliseconds after a visual stimulus is removed which is present in visual cortical areas such as the inferior temporal visual cortex (see Chapter 4) is probably implemented in this way. This short-term memory is one possible mechanism that contributes to the implementation of the trace memory learning rule which can help to implement invariant object recognition as described in Chapter 4. Autoassociation memories also appear to be used in a short-term memory role in the prefrontal cortex. In particular, the temporal visual cortical areas have connections to the ventrolateral prefrontal cortex which help to implement the short-term memory for visual stimuli (in for example delayed match to sample tasks, and visual search tasks, as described in Section 5.1). In an analogous way the parietal cortex has connections to the dorsolateral prefrontal cortex for the short-term memory of spatial responses (see Section 5.1). These short-term memories provide a mechanism that enables attention to be maintained through backprojections from prefrontal cortex areas to the temporal and parietal areas that send connections to the prefrontal cortex, as described in Chapter 6. Autoassociation networks implemented by the recurrent collateral synapses between cortical pyramidal cells also provide a mechanism for constraint satisfaction and also noise reduction whereby the firing of neighbouring neurons can be taken into account in enabling the network to settle into a state that reflects all the details of the inputs activating the population of connected neurons, as well as the effects of what has been set up during developmental plasticity as well as later experience. Attractor networks are also effectively implemented by virtue of the forward and backward connections between cortical areas (see Sections 1.11 and 5.1). An autoassociation network with rapid synaptic plasticity can learn each memory in one trial. Because of its 'one-shot' rapid learning, and ability to complete, this type of network is well suited for episodic memory storage, in which each past episode must be stored and recalled later from a fragment, and kept separate from other episodic memories (see Chapter 2).

B.3.1 Architecture and operation

The prototypical architecture of an autoassociation memory is shown in Fig. B.13. The external input e_i is applied to each neuron i by unmodifiable synapses. This produces firing y_i of each neuron, or a vector of firing on the output neurons $\mathbf{y}$. Each output neuron i is connected by a recurrent collateral connection to the other neurons in the network, via modifiable connection weights w_{ij}. This architecture effectively enables the output firing vector $\mathbf{y}$ to be associated

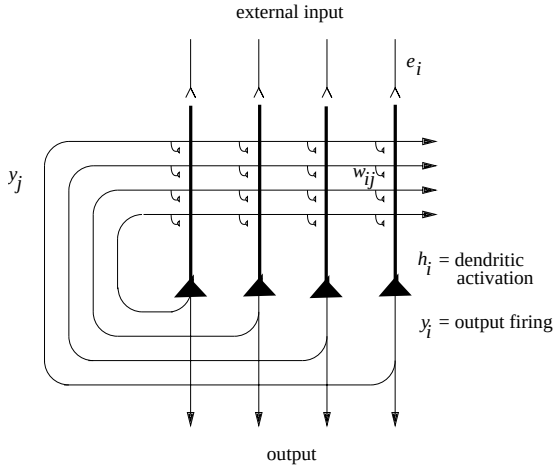


Fig. B.13 The architecture of an autoassociative neural network.

during learning with itself. Later on, during recall, presentation of part of the external input will force some of the output neurons to fire, but through the recurrent collateral axons and the modified synapses, other neurons in $\mathbf{y}$ can be brought into activity. This process can be repeated a number of times, and recall of a complete pattern may be perfect. Effectively, a pattern can be recalled or recognized because of associations formed between its parts. This of course requires distributed representations.

Next we introduce a more precise and detailed description of the above, and describe the properties of these networks. Ways to analyze formally the operation of these networks are introduced in Appendix A4 of Rolls and Treves (1998) and by Amit (1989).

B.3.1.1 Learning

The firing of every output neuron i is forced to a value y_i determined by the external input e_i. Then a Hebb-like associative local learning rule is applied to the recurrent synapses in the network:

$$\delta w_{ij} = \alpha y_i y_j. \tag{B.9}$$

It is notable that in a fully connected network, this will result in a symmetric matrix of synaptic weights, that is the strength of the connection from neuron 1 to neuron 2 will be the same as the strength of the connection from neuron 2 to neuron 1 (both implemented via recurrent collateral synapses).

It is a factor that is sometimes overlooked that there must be a mechanism for ensuring that during learning y_i does approximate e_i, and must not be influenced much by activity in the recurrent collateral connections, otherwise the new external pattern $\mathbf{e}$ will not be stored in the network, but instead something will be stored that is influenced by the previously stored memories. It is thought that in some parts of the brain, such as the hippocampus, there are processes that help the external connections to dominate the firing during learning (see Chapter 2, Treves and Rolls (1992) and Rolls and Treves (1998)).

B.3.1.2 Recall

During recall, the external input e_i is applied, and produces output firing, operating through the non-linear activation function described below. The firing is fed back by the recurrent collateral axons shown in Fig. B.13 to produce activation of each output neuron through the modified synapses on each output neuron. The activation h_i produced by the recurrent

collateral effect on the ith neuron is, in the standard way, the sum of the activations produced in proportion to the firing rate of each axon y_j operating through each modified synapse w_{ij}, that is,

$$h_i = \sum_j y_j w_{ij} \tag{B.10}$$

where $\sum_j$ indicates that the sum is over the C input axons to each neuron, indexed by j.

The output firing y_i is a function of the activation produced by the recurrent collateral effect (internal recall) and by the external input (e_i):

$$y_i = \mathrm{f}(h_i + e_i) \tag{B.11}$$

The activation function should be non-linear, and may be for example binary threshold, linear threshold, sigmoid, etc. (see Fig. 1.3). The threshold at which the activation function operates is set in part by the effect of the inhibitory neurons in the network (not shown in Fig. B.13). The connectivity is that the pyramidal cells have collateral axons that excite the inhibitory interneurons, which in turn connect back to the population of pyramidal cells to inhibit them by a mixture of shunting (divisive) and subtractive inhibition using GABA (gamma-amino-butyric acid) terminals, as described in Section B.6. There are many fewer inhibitory neurons than excitatory neurons (in the order of 5–10%, see Table 1.1) and of connections to and from inhibitory neurons (see Table 1.1), and partly for this reason the inhibitory neurons are considered to perform generic functions such as threshold setting, rather than to store patterns by modifying their synapses. Similar inhibitory processes are assumed for the other networks described in this Appendix. The non-linear activation function can minimize interference between the pattern being recalled and other patterns stored in the network, and can also be used to ensure that what is a positive feedback system remains stable. The network can be allowed to repeat this recurrent collateral loop a number of times. Each time the loop operates, the output firing becomes more like the originally stored pattern, and this progressive recall is usually complete within 5–15 iterations.

B.3.2 Introduction to the analysis of the operation of autoassociation networks

With complete connectivity in the synaptic matrix, and the use of a Hebb rule, the matrix of synaptic weights formed during learning is symmetric. The learning algorithm is fast, 'one-shot', in that a single presentation of an input pattern is all that is needed to store that pattern.

During recall, a part of one of the originally learned stimuli can be presented as an external input. The resulting firing is allowed to iterate repeatedly round the recurrent collateral system, gradually on each iteration recalling more and more of the originally learned pattern. Completion thus occurs. If a pattern is presented during recall that is similar but not identical to any of the previously learned patterns, then the network settles into a stable recall state in which the firing corresponds to that of the previously learned pattern. The network can thus generalize in its recall to the most similar previously learned pattern. The activation function of the neurons should be non-linear, since a purely linear system would not produce any categorization of the input patterns it receives, and therefore would not be able to effect anything more than a trivial (i.e. linear) form of completion and generalization.

Recall can be thought of in the following way, relating it to what occurs in pattern associators. The external input $\mathbf{e}$ is applied, produces firing $\mathbf{y}$, which is applied as a recall cue on the recurrent collaterals as $\mathbf{y}^{\mathrm{T}}$. (The notation $\mathbf{y}^{\mathrm{T}}$ signifies the transpose of $\mathbf{y}$, which is implemented by the application of the firing of the neurons $\mathbf{y}$ back via the recurrent collateral

axons as the next set of inputs to the neurons.) The activity on the recurrent collaterals is then multiplied by the synaptic weight vector stored during learning on each neuron to produce the new activation h_i which reflects the similarity between $\mathbf{y}^{\mathrm{T}}$ and one of the stored patterns. Partial recall has thus occurred as a result of the recurrent collateral effect. The activations h_i after thresholding (which helps to remove interference from other memories stored in the network, or noise in the recall cue) result in firing y_i, or a vector of all neurons $\mathbf{y}$, which is already more like one of the stored patterns than, at the first iteration, the firing resulting from the recall cue alone, $\mathbf{y} = \mathrm{f}(\mathbf{e})$. This process is repeated a number of times to produce progressive recall of one of the stored patterns.

Autoassociation networks operate by effectively storing associations between the elements of a pattern. Each element of the pattern vector to be stored is simply the firing of a neuron. What is stored in an autoassociation memory is a set of pattern vectors. The network operates to recall one of the patterns from a fragment of it. Thus, although this network implements recall or recognition of a pattern, it does so by an association learning mechanism, in which associations between the different parts of each pattern are learned. These memories have sometimes been called autocorrelation memories (Kohonen 1977), because they learn correlations between the activity of neurons in the network, in the sense that each pattern learned is defined by a set of simultaneously active neurons. Effectively each pattern is associated by learning with itself. This learning is implemented by an associative (Hebb-like) learning rule.

The system formally resembles spin glass systems of magnets analyzed quantitatively in statistical mechanics. This has led to the analysis of (recurrent) autoassociative networks as dynamical systems made up of many interacting elements, in which the interactions are such as to produce a large variety of basins of attraction of the dynamics. Each basin of attraction corresponds to one of the originally learned patterns, and once the network is within a basin it keeps iterating until a recall state is reached that is the learned pattern itself or a pattern closely similar to it. (Interference effects may prevent an exact identity between the recall state and a learned pattern.) This type of system is contrasted with other, simpler, systems of magnets (e.g. ferromagnets), in which the interactions are such as to produce only a limited number of related basins, since the magnets tend to be, for example, all aligned with each other. The states reached within each basin of attraction are called attractor states, and the analogy between autoassociator neural networks and physical systems with multiple attractors was drawn by Hopfield (1982) in a very influential paper. He was able to show that the recall state can be thought of as the local minimum in an energy landscape, where the energy would be defined as

$$E = -\frac{1}{2}\sum_{i,j} w_{ij}(y_i - < y >)(y_j - < y >). \tag{B.12}$$

This equation can be understood in the following way. If two neurons are both firing above their mean rate (denoted by $< y >$), and are connected by a weight with a positive value, then the firing of these two neurons is consistent with each other, and they mutually support each other, so that they contribute to the system's tendency to remain stable. If across the whole network such mutual support is generally provided, then no further change will take place, and the system will indeed remain stable. If, on the other hand, either of our pair of neurons was not firing, or if the connecting weight had a negative value, the neurons would not support each other, and indeed the tendency would be for the neurons to try to alter ('flip' in the case of binary units) the state of the other. This would be repeated across the whole network until a situation in which most mutual support, and least 'frustration', was reached. What makes it possible to define an energy function and for these points to hold is that the matrix is symmetric (see Hopfield (1982), Hertz, Krogh and Palmer (1991), Amit (1989)).

Physicists have generally analyzed a system in which the input pattern is presented and then immediately removed, so that the network then 'falls' without further assistance (in what is referred to as the unclamped condition) towards the minimum of its basin of attraction. A more biologically realistic system is one in which the external input is left on contributing to the recall during the fall into the recall state. In this clamped condition, recall is usually faster, and more reliable, so that more memories may be usefully recalled from the network. The approach using methods developed in theoretical physics has led to rapid advances in the understanding of autoassociative networks, and its basic elements are described in Appendix A4 of Rolls and Treves (1998), and by Hertz, Krogh and Palmer (1991) and Amit (1989).

B.3.3 Properties

The internal recall in autoassociation networks involves multiplication of the firing vector of neuronal activity by the vector of synaptic weights on each neuron. This inner product vector multiplication allows the similarity of the firing vector to previously stored firing vectors to be provided by the output (as effectively a correlation), if the patterns learned are distributed. As a result of this type of 'correlation computation' performed if the patterns are distributed, many important properties of these networks arise, including pattern completion (because part of a pattern is correlated with the whole pattern), and graceful degradation (because a damaged synaptic weight vector is still correlated with the original synaptic weight vector). Some of these properties are described next.

B.3.3.1 Completion

Perhaps the most important and useful property of these memories is that they complete an incomplete input vector, allowing recall of a whole memory from a small fraction of it. The memory recalled in response to a fragment is that stored in the memory that is closest in pattern similarity (as measured by the dot product, or correlation). Because the recall is iterative and progressive, the recall can be perfect.

This property and the associative property of pattern associator neural networks are very similar to the properties of human memory. This property may be used when we recall a part of a recent memory of a past episode from a part of that episode. The way in which this could be implemented in the hippocampus is described in Chapter 2.

B.3.3.2 Generalization

The network generalizes in that an input vector similar to one of the stored vectors will lead to recall of the originally stored vector, provided that distributed encoding is used. The principle by which this occurs is similar to that described for a pattern associator.

B.3.3.3 Graceful degradation or fault tolerance

If the synaptic weight vector $\mathbf{w}_i$ on each neuron (or the weight matrix) has synapses missing (e.g. during development), or loses synapses (e.g. with brain damage or ageing), then the activation h_i (or vector of activations $\mathbf{h}$) is still reasonable, because h_i is the dot product (correlation) of $\mathbf{y}^{\mathrm{T}}$ with $\mathbf{w}_i$. The same argument applies if whole input axons are lost. If an output neuron is lost, then the network cannot itself compensate for this, but the next network in the brain is likely to be able to generalize or complete if its input vector has some elements missing, as would be the case if some output neurons of the autoassociation network were damaged.

B.3.3.4 Prototype extraction, extraction of central tendency, and noise reduction

These arise when a set of similar input pattern vectors $\{\mathbf{e}\}$ (which induce firing of the output neurons $\{\mathbf{y}\}$) are learned by the network. The weight vectors $\mathbf{w}_i$ (or strictly $\mathbf{w}_i^{\mathrm{T}}$) become (or point towards) the average $\{< \mathbf{y} >\}$ of that set of similar vectors. This produces 'extraction of the prototype' or 'extraction of the central tendency', and 'noise reduction'. This process can result in better recognition or recall of the prototype than of any of the exemplars, even though the prototype may never itself have been presented. The general principle by which the effect occurs is similar to that by which it occurs in pattern associators. It of course only occurs if each pattern uses a distributed representation.

Related to outputs of the visual system to long-term memory systems (see Chapter 2), there has been intense debate about whether when human memories are stored, a prototype of what is to be remembered is stored, or whether all the instances or the exemplars are each stored separately so that they can be individually recalled (McClelland and Rumelhart (1986), Chapter 17, p. 172). Evidence favouring the prototype view is that if a number of different examples of an object are shown, then humans may report more confidently that they have seen the prototype before than any of the different exemplars, even though the prototype has never been shown (Posner and Keele 1968, Rosch 1975). Evidence favouring the view that exemplars are stored is that in categorization and perceptual identification tasks the responses made are often sensitive to the congruity between particular training stimuli and particular test stimuli (Brooks 1978, Medin and Schaffer 1978, Jacoby 1983a, Jacoby 1983b, Whittlesea 1983). It is of great interest that both types of phenomena can arise naturally out of distributed information storage in a neuronal network such as an autoassociator. This can be illustrated by the storage in an autoassociation memory of sets of stimuli that are all somewhat different examples of the same pattern. These can be generated, for example, by randomly altering each of the input vectors from the input stimulus. After many such randomly altered exemplars have been learned by the network, recall can be tested, and it is found that the network responds best to the original (prototype) input vector, with which it has never been presented. The reason for this is that the autocorrelation components that build up in the synaptic matrix with repeated presentations of the exemplars represent the average correlation between the different elements of the vector, and this is highest for the prototype. This effect also gives the storage some noise immunity, in that variations in the input that are random noise average out, while the signal that is constant builds up with repeated learning.

B.3.3.5 Speed

The recall operation is fast on each neuron on a single iteration, because the pattern $\mathbf{y}^{\mathrm{T}}$ on the axons can be applied simultaneously to the synapses $\mathbf{w}_i$, and the activation h_i can be accumulated in one or two time constants of the dendrite (e.g. 10–20 ms). If a simple implementation of an autoassociation net such as that described by Hopfield (1982) is simulated on a computer, then 5–15 iterations are typically necessary for completion of an incomplete input cue $\mathbf{e}$. This might be taken to correspond to 50–200 ms in the brain, rather too slow for any one local network in the brain to function. However, it has been shown that if the neurons are treated not as McCulloch–Pitts neurons which are simply 'updated' at each iteration, or cycle of timesteps (and assume the active state if the threshold is exceeded), but instead are analyzed and modelled as 'integrate-and-fire' neurons in real continuous time, then the network can effectively 'relax' into its recall state very rapidly, in one or two time constants of the synapses (see Section B.6 and Treves (1993), Battaglia and Treves (1998a) and Appendix A5 of Rolls and Treves (1998)). This corresponds to perhaps 20 ms in the brain.

One factor in this rapid dynamics of autoassociative networks with brain-like 'integrate-and-fire' membrane and synaptic properties is that with some spontaneous activity, some of the neurons in the network are close to threshold already before the recall cue is applied, and hence some of the neurons are very quickly pushed by the recall cue into firing, so that information starts to be exchanged very rapidly (within 1–2 ms of brain time) through the modified synapses by the neurons in the network. The progressive exchange of information starting early on within what would otherwise be thought of as an iteration period (of perhaps 20 ms, corresponding to a neuronal firing rate of 50 spikes/s) is the mechanism accounting for rapid recall in an autoassociative neuronal network made biologically realistic in this way. Further analysis of the fast dynamics of these networks if they are implemented in a biologically plausible way with 'integrate-and-fire' neurons is provided in Section B.6, in Appendix A5 of Rolls and Treves (1998), and by Treves (1993). *The general approach applies to other networks with recurrent connections, not just autoassociators, and the fact that such networks can operate much faster than it would seem from simple models that follow discrete time dynamics is probably a major factor in enabling these networks to provide some of the building blocks of brain function.*

Learning is fast, 'one-shot', in that a single presentation of an input pattern $\mathbf{e}$ (producing $\mathbf{y}$) enables the association between the activation of the dendrites (the post-synaptic term h_i) and the firing of the recurrent collateral axons $\mathbf{y}^{\mathrm{T}}$, to be learned. Repeated presentation with small variations of a pattern vector is used to obtain the properties of prototype extraction, extraction of central tendency, and noise reduction, because these arise from the averaging process produced by storing very similar patterns in the network.

B.3.3.6 Local learning rule

The simplest learning used in autoassociation neural networks, a version of the Hebb rule, is (as in equation B.9)

$$\delta w_{ij} = \alpha y_i y_j.$$

The rule is a local learning rule in that the information required to specify the change in synaptic weight is available locally at the synapse, as it is dependent only on the presynaptic firing rate y_j available at the synaptic terminal, and the postsynaptic activation or firing y_i available on the dendrite of the neuron receiving the synapse. This makes the learning rule biologically plausible, in that the information about how to change the synaptic weight does not have to be carried to every synapse from a distant source where it is computed. As with pattern associators, since firing rates are positive quantities, a potentially interfering correlation is induced between different pattern vectors. This can be removed by subtracting the mean of the presynaptic activity from each presynaptic term, using a type of long-term depression. This can be specified as

$$\delta w_{ij} = \alpha y_i (y_j - z) \tag{B.13}$$

where α is a learning rate constant. This learning rule includes (in proportion to y_i) increasing the synaptic weight if $(y_j - z) > 0$ (long-term potentiation), and decreasing the synaptic weight if $(y_j - z) < 0$ (heterosynaptic long-term depression). This procedure works optimally if z is the average activity $< y_j >$ of an axon across patterns.

Evidence that a learning rule with the general form of equation B.9 is implemented in at least some parts of the brain comes from studies of long-term potentiation, described in Section 1.5. One of the important potential functions of heterosynaptic long-term depression is its ability to allow in effect the average of the presynaptic activity to be subtracted from the presynaptic firing rate (see Appendix A3 of Rolls and Treves (1998), and Rolls and Treves (1990)).

Autoassociation networks can be trained with the error-correction or delta learning rule described in Section B.11. Although a delta rule is less biologically plausible than a Hebb-like rule, a delta rule can help to store separately patterns that are very similar (see McClelland and Rumelhart (1988), Hertz, Krogh and Palmer (1991)).

B.3.3.7 Capacity

One measure of storage capacity is to consider how many orthogonal patterns could be stored, as with pattern associators. If the patterns are orthogonal, there will be no interference between them, and the maximum number p of patterns that can be stored will be the same as the number N of output neurons in a fully connected network. Although in practice the patterns that have to be stored will hardly be orthogonal, this is not a purely academic speculation, since it was shown how one can construct a synaptic matrix that effectively orthogonalizes any set of (linearly independent) patterns (Kohonen 1977, Kohonen 1989, Personnaz, Guyon and Dreyfus 1985, Kanter and Sompolinsky 1987). However, this matrix cannot be learned with a local, one-shot learning rule, and therefore its interest for autoassociators in the brain is limited. The more general case of random non-orthogonal patterns, and of Hebbian learning rules, is considered next.

With non-linear neurons used in the network, the capacity can be measured in terms of the number of input patterns $\mathbf{y}$ (produced by the external input $\mathbf{e}$, see Fig. B.13) that can be stored in the network and recalled later whenever the network settles within each stored pattern's basin of attraction. The first quantitative analysis of storage capacity (Amit, Gutfreund and Sompolinsky 1987) considered a fully connected Hopfield (1982) autoassociator model, in which units are binary elements with an equal probability of being 'on' or 'off' in each pattern, and the number C of inputs per unit is the same as the number N of output units. (Actually it is equal to $N - 1$, since a unit is taken not to connect to itself.) Learning is taken to occur by clamping the desired patterns on the network and using a modified Hebb rule, in which the mean of the presynaptic and postsynaptic firings is subtracted from the firing on any one learning trial (this amounts to a covariance learning rule, and is described more fully in Appendix A4 of Rolls and Treves (1998)). With such fully distributed random patterns, the number of patterns that can be learned is (for C large) $p \approx 0.14C = 0.14N$, hence well below what could be achieved with orthogonal patterns or with an 'orthogonalizing' synaptic matrix. Many variations of this 'standard' autoassociator model have been analyzed subsequently.

Treves and Rolls (1991) have extended this analysis to autoassociation networks that are much more biologically relevant in the following ways. First, some or many connections between the recurrent collaterals and the dendrites are missing (this is referred to as diluted connectivity, and results in a non-symmetric synaptic connection matrix in which w_{ij} does not equal w_{ji}, one of the original assumptions made in order to introduce the energy formalism in the Hopfield model). Second, the neurons need not be restricted to binary threshold neurons, but can have a threshold linear activation function (see Fig. 1.3). This enables the neurons to assume real continuously variable firing rates, which are what is found in the brain (Rolls and Tovee 1995b, Treves, Panzeri, Rolls, Booth and Wakeman 1999). Third, the representation need not be fully distributed (with half the neurons 'on', and half 'off'), but instead can have a small proportion of the neurons firing above the spontaneous rate, which is what is found in parts of the brain such as the hippocampus that are involved in memory (see Treves and Rolls (1994), and Chapter 6 of Rolls and Treves (1998)). Such a representation is defined as being sparse, and the sparseness a of the representation can be measured, by extending the binary notion of the proportion of neurons that are firing, as

$$a = \frac{(\sum_{i=1}^{N} y_i/N)^2}{\sum_{i=1}^{N} y_i^2/N} \tag{B.14}$$

where y_i is the firing rate of the ith neuron in the set of N neurons. Treves and Rolls (1991) have shown that such a network does operate efficiently as an autoassociative network, and can store (and recall correctly) a number of different patterns p as follows

$$p \approx \frac{C^{\mathrm{RC}}}{a \ln(\frac{1}{a})} k \tag{B.15}$$

where C^{RC} is the number of synapses on the dendrites of each neuron devoted to the recurrent collaterals from other neurons in the network, and k is a factor that depends weakly on the detailed structure of the rate distribution, on the connectivity pattern, etc., but is roughly in the order of 0.2–0.3.

The main factors that determine the maximum number of memories that can be stored in an autoassociative network are thus the number of connections on each neuron devoted to the recurrent collaterals, and the sparseness of the representation. For example, for $C^{\mathrm{RC}} = 12,000$ and $a = 0.02$, p is calculated to be approximately $36,000$. This storage capacity can be realized, with little interference between patterns, if the learning rule includes some form of heterosynaptic long-term depression that counterbalances the effects of associative long-term potentiation (Treves and Rolls (1991); see Appendix A4 of Rolls and Treves (1998)). It should be noted that the number of neurons N (which is greater than C^{RC}, the number of recurrent collateral inputs received by any neuron in the network from the other neurons in the network) is not a parameter that influences the number of different memories that can be stored in the network. The implication of this is that increasing the number of neurons (without increasing the number of connections per neuron) does not increase the number of different patterns that can be stored (see Rolls and Treves (1998) Appendix A4), although it may enable simpler encoding of the firing patterns, for example more orthogonal encoding, to be used. This latter point may account in part for why there are generally in the brain more neurons in a recurrent network than there are connections per neuron (see e.g. Chapter 2).

The non-linearity inherent in the NMDA receptor-based Hebbian plasticity present in the brain may help to make the stored patterns more sparse than the input patterns, and this may be especially beneficial in increasing the storage capacity of associative networks in the brain by allowing participation in the storage of especially those relatively few neurons with high firing rates in the exponential firing rate distributions typical of neurons in sensory systems (see Sections B.4.9.3 and C.3.1).

B.3.3.8 Context

The environmental context in which learning occurs can be a very important factor that affects retrieval in humans and other animals. Placing the subject back into the same context in which the original learning occurred can greatly facilitate retrieval.

Context effects arise naturally in association networks if some of the activity in the network reflects the context in which the learning occurs. Retrieval is then better when that context is present, for the activity contributed by the context becomes part of the retrieval cue for the memory, increasing the correlation of the current state with what was stored. (A strategy for retrieval arises simply from this property. The strategy is to keep trying to recall as many fragments of the original memory situation, including the context, as possible, as this will provide a better cue for complete retrieval of the memory than just a single fragment.)

The effects that mood has on memory including visual memory retrieval may be accounted for by backprojections from brain regions such as the amygdala and orbitofrontal cortex in

which the current mood, providing a context, is represented, to brain regions involved in memory such as the perirhinal cortex, and in visual representations such as the inferior temporal visual cortex (see Rolls and Stringer (2001b) and Section 3.11). The very well-known effects of context in the human memory literature could arise in the simple way just described. An implication of the explanation is that context effects will be especially important at late stages of memory or information processing systems in the brain, for there information from a wide range of modalities will be mixed, and some of that information could reflect the context in which the learning takes place. One part of the brain where such effects may be strong is the hippocampus, which is implicated in the memory of recent episodes, and which receives inputs derived from most of the cortical information processing streams, including those involved in space (see Chapter 2).

B.3.3.9 Mixture states

If an autoassociation memory is trained on pattern vectors $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{A} + \mathbf{B}$ (i.e. $\mathbf{A}$ and $\mathbf{B}$ are both included in the joint vector $\mathbf{A} + \mathbf{B}$; that is if the vectors are not linearly independent), then the autoassociation memory will have difficulty in learning and recalling these three memories as separate, because completion from either $\mathbf{A}$ or $\mathbf{B}$ to $\mathbf{A} + \mathbf{B}$ tends to occur during recall. (The ability to separate such patterns is referred to as configurational learning in the animal learning literature, see e.g. Sutherland and Rudy (1991).) This problem can be minimized by re-representing $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{A} + \mathbf{B}$ in such a way that they are different vectors before they are presented to the autoassociation memory. This can be performed by recoding the input vectors to minimize overlap using, for example, a competitive network, and possibly involving expansion recoding, as described for pattern associators (see Section B.2, Fig. B.11). It is suggested that this is a function of the dentate granule cells in the hippocampus, which precede the CA3 recurrent collateral network (Treves and Rolls 1992, Treves and Rolls 1994) (see Chapter 2).

B.3.3.10 Memory for sequences

One of the first extensions of the standard autoassociator paradigm that has been explored in the literature is the capability to store and retrieve not just individual patterns, but whole sequences of patterns. Hopfield, in the same 1982 paper, suggested that this could be achieved by adding to the standard connection weights, which associate a pattern with itself, a new, asymmetric component, which associates a pattern with the next one in the sequence. In practice this scheme does not work very well, unless the new component is made to operate on a slower time scale than the purely autoassociative component (Kleinfeld 1986, Sompolinsky and Kanter 1986). With two different time scales, the autoassociative component can stabilize a pattern for a while, before the heteroassociative component moves the network, as it were, into the next pattern. The heteroassociative retrieval cue for the next pattern in the sequence is just the previous pattern in the sequence. A particular type of 'slower' operation occurs if the asymmetric component acts after a delay τ. In this case, the network sweeps through the sequence, staying for a time of order τ in each pattern.

One can see how the necessary ingredient for the storage of sequences is only a minor departure from purely Hebbian learning: in fact, the (symmetric) autoassociative component of the weights can be taken to reflect the Hebbian learning of strictly simultaneous conjunctions of pre- and post-synaptic activity, whereas the (asymmetric) heteroassociative component can be implemented by Hebbian learning of each conjunction of postsynaptic activity with presynaptic activity shifted a time τ in the past. Both components can then be seen as resulting from a generalized Hebbian rule, which increases the weight whenever postsynaptic activity is paired with presynaptic activity occurring within a given time range, which may extend from a few hundred milliseconds in the past up to include strictly simultaneous activity. This is

similar to a trace rule (see Chapter 4), which itself matches very well the observed conditions for induction of long-term potentiation, and appears entirely plausible. The learning rule necessary for learning sequences, though, is more complex than a simple trace rule in that the time-shifted conjunctions of activity that are encoded in the weights must in retrieval produce activations that are time-shifted as well (otherwise one falls back into the Hopfield (1982) proposal, which does not quite work). The synaptic weights should therefore keep separate 'traces' of what was simultaneous and what was time-shifted during the original experience, and this is not very plausible.

Levy and colleagues (Levy, Wu and Baxter 1995, Wu, Baxter and Levy 1996) have investigated these issues further, and the temporal asymmetry that may be present in LTP (see Section 1.5) has been suggested as a mechanism that might provide some of the temporal properties that are necessary for the brain to store and recall sequences (Minai and Levy 1993, Abbott and Blum 1996, Markram, Pikus, Gupta and Tsodyks 1998, Abbott and Nelson 2000). A problem with this suggestion is that, given that the temporal dynamics of attractor networks are inherently very fast when the networks have continuous dynamics (see Section B.6), and that the temporal asymmetry in LTP may be in the order of only milliseconds to a few tens of milliseconds (see Section 1.5), the recall of the sequences would be very fast, perhaps 10–20 ms per step of the sequence, with every step of a 10-step sequence effectively retrieved and gone in a quick-fire session of 100–200 ms.

Another way in which a delay could be inserted in a recurrent collateral path in the brain is by inserting another cortical area in the recurrent path. This could fit in with the cortico-cortical backprojection connections described in Section 1.11, which would introduce some conduction delay (see Panzeri, Rolls, Battaglia and Lavis (2001)).

B.3.4 Use of autoassociation networks in the brain

Because of its 'one-shot' rapid learning, and ability to complete, this type of network is well suited for episodic memory storage, in which each episode must be stored and recalled later from a fragment, and kept separate from other episodic memories. It does not take a long time (the 'many epochs' of backpropagation networks) to train this network, because it does not have to 'discover the structure' of a problem. Instead, it stores information in the form in which it is presented to the memory, without altering the representation. An autoassociation network may be used for this function in the CA3 region of the hippocampus (see Chapter 2, and Rolls and Treves (1998) Chapter 6).

An autoassociation memory can also be used as a short-term memory, in which iterative processing round the recurrent collateral loop keeps a representation active until another input cue is received. This may be used to implement many types of short-term memory in the brain (see Section 5.1). For example, it may be used in the perirhinal cortex and adjacent temporal lobe cortex to implement short-term visual object memory (Miyashita and Chang 1988, Amit 1995); in the dorsolateral prefrontal cortex to implement a short-term memory for spatial responses (Goldman-Rakic 1996); and in the prefrontal cortex to implement a short-term memory for where eye movements should be made in space (see Chapter 5). Such an autoassociation memory in the temporal lobe visual cortical areas may be used to implement the firing that continues for often 300 ms after a very brief (16 ms) presentation of a visual stimulus (Rolls and Tovee 1994) (see e.g. Fig. C.17), and may be one way in which a short memory trace is implemented to facilitate invariant learning about visual stimuli (see Chapter 4). In all these cases, the short-term memory may be implemented by the recurrent excitatory collaterals that connect nearby pyramidal cells in the cerebral cortex. The connectivity in this system, that is the probability that a neuron synapses on a nearby neuron, may be in the region of 10% (Braitenberg and Schuz 1991, Abeles 1991).

The recurrent connections between nearby neocortical pyramidal cells may also be important in defining the response properties of cortical cells, which may be triggered by external inputs (from for example the thalamus or a preceding cortical area), but may be considerably dependent on the synaptic connections received from nearby cortical pyramidal cells.

The cortico-cortical backprojection connectivity described in Chapters 1 and 2 can be interpreted as a system that allows the forward-projecting neurons in one cortical area to be linked autoassociatively with the backprojecting neurons in the next cortical area (see Section 1.11 and Chapter 2). This would be implemented by associative synaptic modification in for example the backprojections. This particular architecture may be especially important in constraint satisfaction (as well as recall), that is it may allow the networks in the two cortical areas to settle into a mutually consistent state. This would effectively enable information in higher cortical areas, which would include information from more divergent sources, to influence the response properties of neurons in earlier cortical processing stages. This is an important function in cortical information processing of interacting associative networks.

B.4 Competitive networks, including self-organizing maps

B.4.1 Function

Competitive neural networks learn to categorize input pattern vectors. Each category of inputs activates a different output neuron (or set of output neurons – see below). The categories formed are based on similarities between the input vectors. Similar, that is correlated, input vectors activate the same output neuron. In that the learning is based on similarities in the input space, and there is no external teacher that forces classification, this is an unsupervised network. The term categorization is used to refer to the process of placing vectors into categories based on their similarity. The term classification is used to refer to the process of placing outputs in particular classes as instructed or taught by a teacher. Examples of classifiers are pattern associators, one-layer delta-rule perceptrons, and multilayer perceptrons taught by error backpropagation (see Sections B.2, B.3, B.11 and B.12). In supervised networks there is usually a teacher for each output neuron.

The categorization produced by competitive nets is of great potential importance in perceptual systems including the whole of the visual cortical processing hierarchies, as described in Chapter 4. Each category formed reflects a set or cluster of active inputs x_j which occur together. This cluster of coactive inputs can be thought of as a feature, and the competitive network can be described as building feature analyzers, where a feature can now be defined as a correlated set of inputs. During learning, a competitive network gradually discovers these features in the input space, and the process of finding these features without a teacher is referred to as self-organization.

Another important use of competitive networks is to remove redundancy from the input space, by allocating output neurons to reflect a set of inputs that co-occur.

Another important aspect of competitive networks is that they separate patterns that are somewhat correlated in the input space, to produce outputs for the different patterns that are less correlated with each other, and may indeed easily be made orthogonal to each other. This has been referred to as orthogonalization.

Another important function of competitive networks is that partly by removing redundancy from the input information space, they can produce sparse output vectors, without losing information. We may refer to this as sparsification.

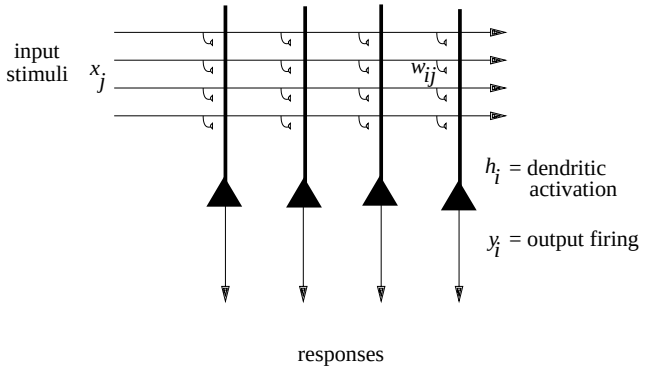


Fig. B.14 The architecture of a competitive network.

B.4.2 Architecture and algorithm

B.4.2.1 Architecture

The basic architecture of a competitive network is shown in Fig. B.14. It is a one-layer network with a set of inputs that make modifiable excitatory synapses w_{ij} with the output neurons. The output cells compete with each other (for example by mutual inhibition) in such a way that the most strongly activated neuron or neurons win the competition, and are left firing strongly. The synaptic weights, w_{ij}, are initialized to random values before learning starts. If some of the synapses are missing, that is if there is randomly diluted connectivity, that is not a problem for such networks, and can even help them (see below).

In the brain, the inputs arrive through axons, which make synapses with the dendrites of the output or principal cells of the network. The principal cells are typically pyramidal cells in the cerebral cortex. In the brain, the principal cells are typically excitatory, and mutual inhibition between them is implemented by inhibitory interneurons, which receive excitatory inputs from the principal cells. The inhibitory interneurons then send their axons to make synapses with the pyramidal cells, typically using GABA (gamma-aminobutyric acid) as the inhibitory transmitter.

B.4.2.2 Algorithm

1. Apply an input vector $\mathbf{x}$ and calculate the activation h_i of each neuron

$$h_i = \sum_j x_j w_{ij} \tag{B.16}$$

where the sum is over the C input axons, indexed by j. (It is useful to normalize the length of each input vector $\mathbf{x}$. In the brain, a scaling effect is likely to be achieved both by feedforward inhibition, and by feedback inhibition among the set of input cells (in a preceding network) that give rise to the axons conveying $\mathbf{x}$.)

The output firing y_i^1 is a function of the activation of the neuron

$$y_i^1 = \mathrm{f}(h_i). \tag{B.17}$$

The function f can be linear, sigmoid, monotonically increasing, etc. (see Fig. 1.3).

2. Allow competitive interaction between the output neurons by a mechanism such as lateral or mutual inhibition (possibly with self-excitation), to produce a contrast-enhanced version of the firing rate vector

$$y_i = \mathrm{g}(y_i^1). \tag{B.18}$$

Function g is typically a non-linear operation, and in its most extreme form may be a winner-take-all function, in which after the competition one neuron may be 'on', and the others 'off'. Algorithms that produce softer competition without a single winner to produce a distributed representation are described in Section B.4.9.4 below.

3. Apply an associative Hebb-like learning rule

$$\delta w_{ij} = \alpha y_i x_j. \tag{B.19}$$

4. Normalize the length of the synaptic weight vector on each dendrite to prevent the same few neurons always winning the competition:

$$\sum_j (w_{ij})^2 = 1. \tag{B.20}$$

(A less efficient alternative is to scale the sum of the weights to a constant, e.g. 1.0.)

5. Repeat steps 1–4 for each different input stimulus $\mathbf{x}$, in random sequence, a number of times.

B.4.3 Properties

B.4.3.1 Feature discovery by self-organization

Each neuron in a competitive network becomes activated by a set of consistently coactive, that is correlated, input axons, and gradually learns to respond to that cluster of coactive inputs. We can think of competitive networks as discovering features in the input space, where features can now be defined by a set of consistently coactive inputs. Competitive networks thus show how feature analyzers can be built, with no external teacher. The feature analyzers respond to correlations in the input space, and the learning occurs by self-organization in the competitive network. Competitive networks are therefore well suited to the analysis of sensory inputs. Ways in which they may form fundamental building blocks of sensory systems are described in Chapter 4.

The operation of competitive networks can be visualized with the help of Fig. B.15. The input patterns are represented as dots on the surface of a sphere. (The patterns are on the surface of a sphere because they are normalized to the same length.) The directions of the weight vectors of the three neurons are represented by '$\times$'s. The effect of learning is to move the weight vector of each of the neurons to point towards the centre of one of the clusters of inputs. If the neurons are winner-take-all, the result of the learning is that although there are correlations between the input stimuli, the outputs of the three neurons are orthogonal. In this sense, orthogonalization is performed. At the same time, given that each of the patterns within a cluster produces the same output, the correlations between the patterns within a cluster become higher. In a winner-take-all network, the within-pattern correlation becomes 1, and the patterns within a cluster have been placed within the same category.

B.4.3.2 Removal of redundancy

In that competitive networks recode sets of correlated inputs to one or a few output neurons, then redundancy in the input representation is removed. Identifying and removing redundancy in sensory inputs is an important part of processing in sensory systems (cf. Barlow (1989)), in part because a compressed representation is more manageable as an output of sensory systems. The reason for this is that neurons in the receiving systems, for example pattern associators in the orbitofrontal cortex or autoassociation networks in the hippocampus, can then operate with the limited numbers of inputs that each neuron can receive. For example,

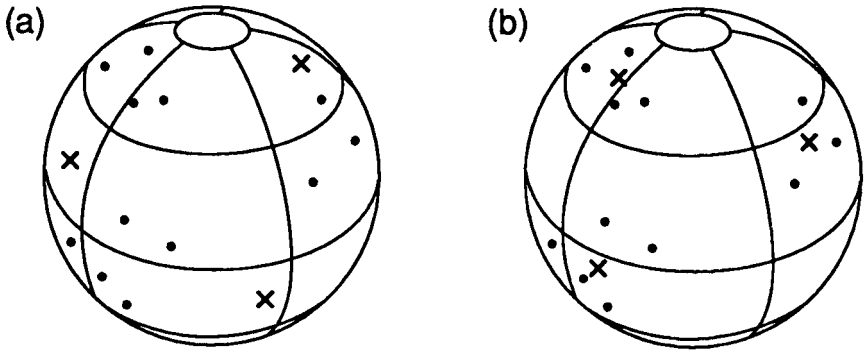


Fig. B.15 Competitive learning. The dots represent the directions of the input vectors, and the '×'s the weights for each of three output neurons. (a) Before learning. (b) After learning. (After Rumelhart and Zipser 1986.)

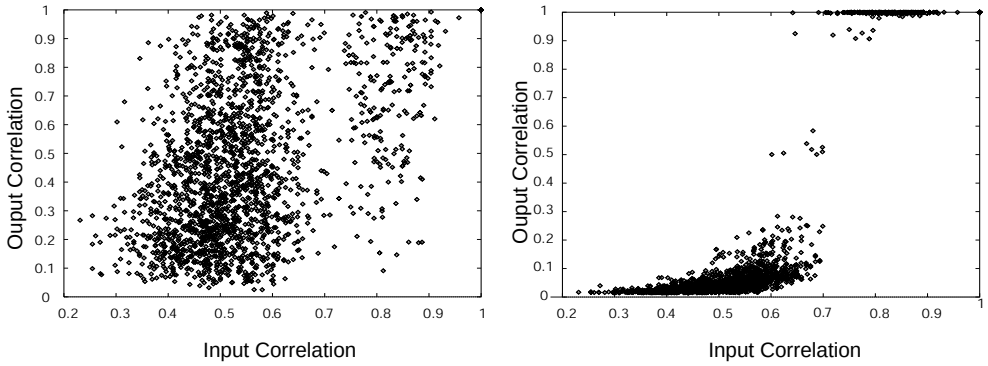
although the information that a particular face is being viewed is present in the 10^6 fibres in the optic nerve, the information is unusable by associative networks in this form, and is compressed through the visual system until the information about which of many hundreds of faces is present can be represented by less than 100 neurons in the temporal cortical visual areas (Rolls, Treves and Tovee 1997b, Abbott, Rolls and Tovee 1996). (Redundancy can be defined as the difference between the maximum information content of the input data stream (or channel capacity) and its actual content; see Appendix C.)

The recoding of input pattern vectors into a more compressed representation that can be conveyed by a much reduced number of output neurons of a competitive network is referred to in engineering as vector quantization. With a winner-take-all competitive network, each output neuron points to or stands for one of or a cluster of the input vectors, and it is more efficient to transmit the states of the few output neurons than the states of all the input elements. (It is more efficient in the sense that the information transmission rate required, that is the capacity of the channel, can be much smaller.) Vector quantization is of course possible when the input representation contains redundancy.

B.4.3.3 Orthogonalization and categorization

Figure B.15 shows visually how competitive networks reduce the correlation between different clusters of patterns, by allocating them to different output neurons. This is described as orthogonalization. It is a process that is very usefully applied to signals before they are used as inputs to associative networks (pattern associators and autoassociators) trained with Hebbian rules (see Sections B.2 and B.3), because it reduces the interference between patterns stored in these memories. The opposite effect in competitive networks, of bringing closer together very similar input patterns, is referred to as categorization.

These two processes are also illustrated in Fig. B.16, which shows that in a competitive network, very similar input patterns (with correlations higher in this case than approximately 0.8) produce more similar outputs (close to 1.0), whereas the correlations between pairs of input patterns that are smaller than approximately 0.7 become much smaller in the output representation. (This simulation used soft competition between neurons with graded firing rates.)



a) Before Learning b) After Learning

Fig. B.16 Orthogonalization and categorization in a competitive network: (a) before learning; (b) after learning. The correlations between pairs of output vectors (abscissa) are plotted against the correlations of the corresponding pairs of input vectors that generated the output pair, for all possible pairs in the input set. The competitive net learned for 16 cycles. One cycle consisted of presenting the complete input set of stimuli in a renewing random sequence. The correlation measure shown is the cosine of the angle between two vectors (i.e. the normalized dot product). The network used had 64 input axons to each of 8 output neurons. The net was trained with 64 stimuli, made from 8 initial random binary vectors with each bit having a probability of 0.5 of being 1, from each of which 8 noisy exemplars were made by randomly altering 10 % of the 64 elements. Soft competition was used between the output neurons. (A normalized exponential activation function described in Section B.4.9.4 was used to implement the soft competition.) The sparseness a of the input patterns thus averaged 0.5; and the sparseness a of the output firing vector after learning was close to 0.17 (i.e. after learning, primarily one neuron was active for each input pattern; before learning, the average sparseness of the output patterns produced by each of the inputs was 0.39).

B.4.3.4 Sparsification

Competitive networks can produce more sparse representations than those that they receive, depending on the degree of competition. With the greatest competition, winner-take-all, only one output neuron remains active, and the representation is at its most sparse. This effect can be understood further using Figs. B.15 and B.16. This sparsification is useful to apply to representations before input patterns are applied to associative networks, because sparse representations allow many different pattern associations or memories to be stored in these networks (see Sections B.2 and B.3).

B.4.3.5 Capacity

In a competitive net with N output neurons and a simple winner-take-all rule for the competition, it is possible to learn up to N output categories, in that each output neuron may be allocated a category. When the competition acts in a less rudimentary way, the number of categories that can be learned becomes a complex function of various factors, including the number of modifiable connections per cell and the degree of dilution, or incompleteness, of the connections. Such a function has not yet been described analytically in general, but an upper bound on it can be deduced for the particular case in which the learning is fast, and can be achieved effectively in one shot, or one presentation of each pattern. In that case, the number of categories that can be learned (by the self-organizing process) will at most be equal to the number of associations that can be formed by the corresponding pattern associators, a process that occurs with the additional help of the driving inputs, which effectively determine the categorization in the pattern associator.

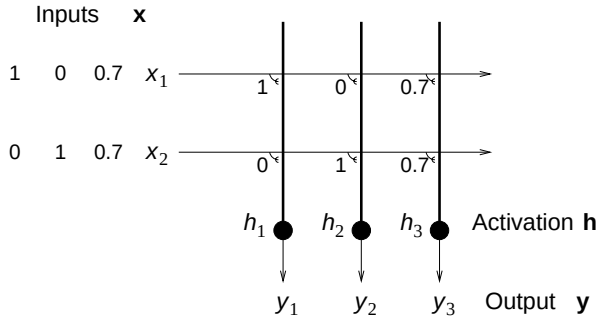


Fig. B.17 Separation of linearly dependent patterns by a competitive network. The network was trained on patterns 10, 01, and 11, applied on the inputs x_1 and x_2. After learning, the network allocated output neuron 1 to pattern 10, neuron 2 to pattern 01, and neuron 3 to pattern 11. The weights in the network produced during the learning are shown. Each input pattern was normalized to unit length, and thus for pattern 11, x_1=0.7 and x_2=0.7, as shown. Because the weight vectors were also normalized to unit length, w_{31}=0.7 and w_{32}=0.7.

Separate constraints on the capacity result if the output vectors are required to be strictly orthogonal. Then, if the output firing rates can assume only positive values, the maximum number p of categories arises, obviously, in the case when only one output neuron is firing for any stimulus, so that up to N categories are formed. If ensemble encoding of output neurons is used (soft competition), again under the orthogonality requirement, then the number of output categories that can be learned will be reduced according to the degree of ensemble encoding. The p categories in the ensemble-encoded case reflect the fact that the between-cluster correlations in the output space are lower than those in the input space. The advantages of ensemble encoding are that dendrites are more evenly allocated to patterns (see Section B.4.9.5), and that correlations between different input stimuli can be reflected in correlations between the corresponding output vectors, so that later networks in the system can generalize usefully. This latter property is of crucial importance, and is utilized for example when an input pattern is presented that has not been learned by the network. The relative similarity of the input pattern to previously learned patterns is indicated by the relative activation of the members of an ensemble of output neurons. This makes the number of different representations that can be reflected in the output of competitive networks with ensemble encoding much higher than with winner-take-all representations, even though with soft competition all these representations cannot strictly be learned.

B.4.3.6 Separation of non-linearly separable patterns

A competitive network can not only separate (e.g. by activating different output neurons) pattern vectors that overlap in almost all elements, but can also help with the separation of vectors that are not linearly separable. An example is that three patterns **A**, **B**, and **A+B** will lead to three different output neurons being activated (see Fig. B.17). For this to occur, the length of the synaptic weight vectors must be normalized (to for example unit length), so that they lie on the surface of a sphere or hypersphere (see Fig. B.15). (If the weight vectors of each neuron are scaled to the same sum, then the weight vectors do not lie on the surface of a hypersphere, and the ability of the network to separate patterns is reduced.)

The property of pattern separation makes a competitive network placed before an autoassociator network very valuable, for it enables the autoassociator to store the three patterns separately, and to recall **A+B** separately from **A** and **B**. This is referred to as the configuration learning problem in animal learning theory (Sutherland and Rudy 1991). Placing a compet-

itive network before a pattern associator will enable a linearly inseparable problem to be solved. For example, three different output neurons of a two-input competitive network could respond to the patterns 01, 10, and 11, and a pattern associator can learn different outputs for neurons 1–3, which are orthogonal to each other (see Fig. B.11). This is an example of expansion recoding (cf. Marr (1969), who used a different algorithm to obtain the expansion). The sparsification that can be produced by the competitive network can also be advantageous in preparing patterns for presentation to a pattern associator or autoassociator, because the sparsification can increase the number of memories that can be associated or stored.

B.4.3.7 Stability

These networks are generally stable if the input statistics are stable. If the input statistics keep varying, then the competitive network will keep following the input statistics. If this is a problem, then a critical period in which the input statistics are learned, followed by stabilization, may be useful. This appears to be a solution used in developing sensory systems, which have critical periods beyond which further changes become more difficult. An alternative approach taken by Carpenter and Grossberg in their 'Adaptive Resonance Theory' (Carpenter 1997) is to allow the network to learn only if it does not already have categorizers for a pattern (see Hertz, Krogh and Palmer (1991), p. 228).

Diluted connectivity can help stability, by making neurons tend to find inputs to categorize in only certain parts of the input space, and then making it difficult for the neuron to wander randomly throughout the space later.

B.4.3.8 Frequency of presentation

If some stimuli are presented more frequently than others, then there will be a tendency for the weight vectors to move more rapidly towards frequently presented stimuli, and more neurons may become allocated to the frequently presented stimuli. If winner-take-all competition is used, the result is that the neurons will tend to become allocated during the learning process to the more frequently presented patterns. If soft competition is used, the tendency of neurons to move from patterns that are infrequently or never presented can be reduced by making the competition fairly strong, so that only a few neurons show any learning when each pattern is presented. Provided that the competition is moderately strong (see Section B.4.9.4), the result is that more neurons are allocated to frequently presented patterns, but one or some neurons are allocated to infrequently presented patterns. These points can all be easily demonstrated in simulations.

In an interesting development, it has been shown that if objects consisting of groups of features are presented during training always with another object present, then separate representations of each object can be formed provided that each object is presented many times, but on each occasion is paired with a different object (Stringer and Rolls 2007b, Stringer, Rolls and Tromans 2007b). This is related to the fact that in this scenario the frequency of co-occurrence of features within the same object is greater than that of features between different objects (see Section 4.5.6.2).

B.4.3.9 Comparison to principal component analysis (PCA) and cluster analysis

Although competitive networks find clusters of features in the input space, they do not perform hierarchical cluster analysis as typically performed in statistics. In hierarchical cluster analysis, input vectors are joined starting with the most correlated pair, and the level of the joining of vectors is indicated. Competitive nets produce different outputs (i.e. activate different output neurons) for each cluster of vectors (i.e. perform vector quantization), but do not compute the level in the hierarchy, unless the network is redesigned (see Hertz, Krogh and Palmer (1991)).

The feature discovery can also be compared to principal component analysis (PCA). (In PCA, the first principal component of a multidimensional space points in the direction of the vector that accounts for most of the variance, and subsequent principal components account for successively less of the variance, and are mutually orthogonal.) In competitive learning with a winner-take-all algorithm, the outputs are mutually orthogonal, but are not in an ordered series according to the amount of variance accounted for, unless the training algorithm is modified. The modification amounts to allowing each of the neurons in a winner-take-all network to learn one at a time, in sequence. The first neuron learns the first principal component. (Neurons trained with a modified Hebb rule learn to maximize the variance of their outputs – see Hertz, Krogh and Palmer (1991).) The second neuron is then allowed to learn, and because its output is orthogonal to the first, it learns the second principal component. This process is repeated. Details are given by Hertz, Krogh and Palmer (1991), but as this is not a biologically plausible process, it is not considered in detail here. We note that simple competitive learning is very helpful biologically, because it can separate patterns, but that a full ordered set of principal components as computed by PCA would probably not be very useful in biologically plausible networks. Our point here is that biological neuronal networks may operate well if the variance in the input representation is distributed across many input neurons, whereas principal component analysis would tend to result in most of the variance being allocated to a few neurons, and the variance being unevenly distributed across the neurons.

B.4.4 Utility of competitive networks in information processing by the brain

B.4.4.1 Feature analysis and preprocessing

Neurons that respond to correlated combinations of their inputs can be described as feature analyzers. Neurons that act as feature analyzers perform useful preprocessing in many sensory systems (see e.g. Chapter 8 of Rolls and Treves (1998)). The power of competitive networks in multistage hierarchical processing to build combinations of what is found at earlier stages, and thus effectively to build higher-order representations, is also described in Chapter 4 of this book. An interesting recent development is that competitive networks can learn about individual objects even when multiple objects are presented simultaneously, provided that each object is presented several times more frequently than it is paired with any other individual object (Stringer and Rolls 2007b) (see Section 4.5.6.2). This property arises because learning in competitive networks is primarily about forming representations of objects defined by a high correlation of coactive features in the input space (Stringer and Rolls 2007b).

B.4.4.2 Removal of redundancy

The removal of redundancy by competition is thought to be a key aspect of how sensory systems, including the ventral cortical visual system, operate. Competitive networks can also be thought of as performing dimension reduction, in that a set of correlated inputs may be responded to as one category or dimension by a competitive network. The concept of redundancy removal can be linked to the point that individual neurons trained with a modified Hebb rule point their weight vector in the direction of the vector that accounts for most of the variance in the input, that is (acting individually) they find the first principal component of the input space (see Section B.4.3.9 and Hertz, Krogh and Palmer (1991)). Although networks with anti-Hebbian synapses between the principal cells (in which the anti-Hebbian learning forces neurons with initially correlated activity to effectively inhibit each other) (F öldiák 1991), and networks that perform Independent Component Analysis (Bell and Sejnowski 1995), could

in principle remove redundancy more effectively, it is not clear that they are implemented biologically. In contrast, competitive networks are more biologically plausible, and illustrate redundancy reduction. The more general use of an unsupervised competitive preprocessor is discussed below (see Fig. B.24).

B.4.4.3 Orthogonalization

The orthogonalization performed by competitive networks is very useful for preparing signals for presentation to pattern associators and autoassociators, for this re-representation decreases interference between the patterns stored in such networks. Indeed, this can be essential if patterns are overlapping and not linearly independent, e.g. 01, 10, and 11. If three such binary patterns were presented to an autoassociative network, it would not form separate representations of them, because either of the patterns 01 or 10 would result by completion in recall of the 11 pattern. A competitive network allows a separate neuron to be allocated to each of the three patterns, and this set of orthogonal representations can be learned by associative networks (see Fig. B.17).

B.4.4.4 Sparsification

The sparsification performed by competitive networks is very useful for preparing signals for presentation to pattern associators and autoassociators, for this re-representation increases the number of patterns that can be associated or stored in such networks (see Sections B.2 and B.3).

B.4.4.5 Brain systems in which competitive networks may be used for orthogonalization and sparsification

One system is the hippocampus, in which the dentate granule cells are believed to operate as a competitive network in order to prepare signals for presentation to the CA3 autoassociative network (see Chapter 2). In this case, the operation is enhanced by expansion recoding, in that (in the rat) there are approximately three times as many dentate granule cells as there are cells in the preceding stage, the entorhinal cortex. This expansion recoding will itself tend to reduce correlations between patterns (cf. Marr (1970), and Marr (1969)).

Also in the hippocampus, the CA1 neurons are thought to act as a competitive network that recodes the separate representations of each of the parts of an episode that must be separately represented in CA3, into a form more suitable for the recall using pattern association performed by the backprojections from the hippocampus to the cerebral cortex (see Chapter 2 and Rolls and Treves (1998) Chapter 6).

The granule cells of the cerebellum may perform a similar function, but in this case the principle may be that each of the very large number of granule cells receives a very small random subset of inputs, so that the outputs of the granule cells are decorrelated with respect to the inputs (Marr (1969); see Rolls and Treves (1998) Chapter 9).

B.4.5 Guidance of competitive learning

Although competitive networks are primarily unsupervised networks, it is possible to influence the categories found by supplying a second input, as follows (Rolls 1989a). Consider a competitive network as shown in Fig. B.18 with the normal set of inputs **A** to be categorized, and with an additional set of inputs **B** from a different source. Both sets of inputs work in the normal way for a competitive network, with random initial weights, competition between the output neurons, and a Hebb-like synaptic modification rule that normalizes the lengths of the synaptic weight vectors onto each neuron. The idea then is to use the **B** inputs to influence the categories formed by the **A** input vectors. The influence of the **B** vectors works best if

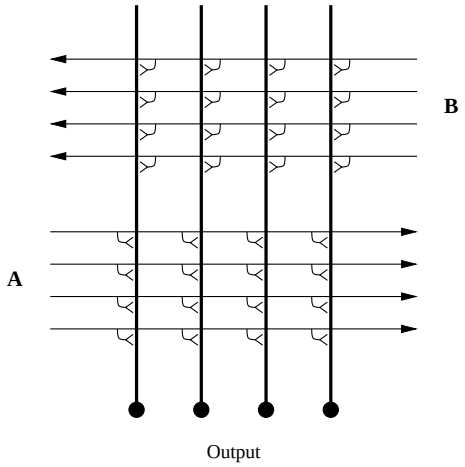


Fig. B.18 Competitive net receiving a normal set of inputs **A**, but also another set of inputs **B** that can be used to influence the categories formed in response to **A** inputs.

they are orthogonal to each other. Consider any two **A** vectors. If they occur together with the same **B** vector, then the categories produced by the **A** vectors will be more similar than they would be without the influence of the **B** vectors. The categories will be pulled closer together if soft competition is used, or will be more likely to activate the same neuron if winner-take-all competition is used. Conversely, if any two **A** vectors are paired with two different, preferably orthogonal, **B** vectors, then the categories formed by the **A** vectors will be drawn further apart than they would be without the **B** vectors. The differences in categorization remain present after the learning when just the **A** inputs are used.

This guiding function of one of the inputs is one way in which the consequences of sensory stimuli could be fed back to a sensory system to influence the categories formed when the **A** inputs are presented. This could be one function of backprojections in the cerebral cortex (Rolls 1989c, Rolls 1989a). In this case, the **A** inputs of Fig. B.18 would be the forward inputs from a preceding cortical area, and the **B** inputs backprojecting axons from the next cortical area, or from a structure such as the amygdala or hippocampus. If two **A** vectors were both associated with positive reinforcement that was fed back as the same **B** vector from another part of the brain, then the two **A** vectors would be brought closer together in the representational space provided by the output of the neurons. If one of the **A** vectors was associated with positive reinforcement, and the other with negative reinforcement, then the output representations of the two **A** vectors would be further apart. This is one way in which external signals could influence in a mild way the categories formed in sensory systems. Another is that if any **B** vector only occurred for important sensory **A** inputs (as shown by the immediate consequences of receiving those sensory inputs), then the **A** inputs would simply be more likely to have any representation formed than otherwise, due to strong activation of neurons only when combined **A** and **B** inputs are present.

A similar architecture could be used to provide mild guidance for one sensory system (e.g. olfaction) by another (e.g. taste), as shown in Fig. B.19. (Another example of where this architecture could be used is convergence in the visual system at the next cortical stage of processing, with guiding feedback to influence the categories formed in the different regions of the preceding cortical area, as illustrated in Section 1.11.) The idea is that the taste inputs would be more orthogonal to each other than the olfactory inputs, and that the taste inputs would influence the categories formed in the olfactory input categorizer in layer 1, by feedback

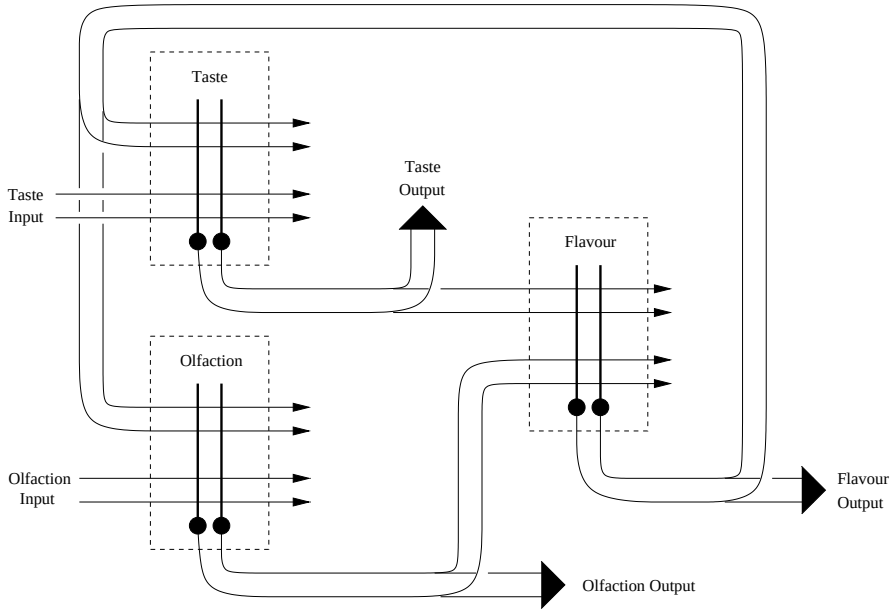


Fig. B.19 A two-layer set of competitive nets in which feedback from layer 2 can influence the categories formed in layer 1. Layer 2 could be a higher cortical visual area with convergence from earlier cortical visual areas (see Chapter 4). In the example, taste and olfactory inputs are received by separate competitive nets in layer 1, and converge into a single competitive net in layer 2. The categories formed in layer 2 (which may be described as representing 'flavour') may be dominated by the relatively orthogonal set of a few tastes that are received by the net. When these layer 2 categories are fed back to layer 1, they may produce in layer 1 categories in, for example, the olfactory network that reflect to some extent the flavour categories of layer 2, and are different from the categories that would otherwise be formed to a large set of rather correlated olfactory inputs. A similar principle may operate in any multilayer hierarchical cortical processing system, such as the ventral visual system, in that the categories that can be formed only at later stages of processing may help earlier stages to form categories relevant to what can be identified at later stages.

from a convergent net in layer 2. The difference from the previous architecture is that we now have a two-layer net, with unimodal or separate networks in layer 1, each feeding forward to a single competitive network in layer 2. The categories formed in layer 2 reflect the co-occurrence of a particular taste with particular odours (which together form flavour in layer 2). Layer 2 then provides feedback connections to both the networks in layer 1. It can be shown in such a network that the categories formed in, for example, the olfactory net in layer 1 are influenced by the tastes with which the odours are paired. The feedback signal is built only in layer 2, after there has been convergence between the different modalities. This architecture captures some of the properties of sensory systems, in which there are unimodal processing cortical areas followed by multimodal cortical areas. The multimodal cortical areas can build representations that represent the unimodal inputs that tend to co-occur, and the higher level representations may in turn, by the highly developed cortico-cortical backprojections, be able to influence sensory categorization in earlier cortical processing areas (Rolls 1989a). Another such example might be the effect by which the phonemes heard are influenced by the visual inputs produced by seeing mouth movements (cf. McGurk and MacDonald (1976)). This could be implemented by auditory inputs coming together in the cortex in the superior temporal sulcus onto neurons activated by the sight of the lips moving (recorded during experiments of Baylis, Rolls and Leonard (1987), and Hasselmo, Rolls, Baylis and Nalwa (1989b)), using

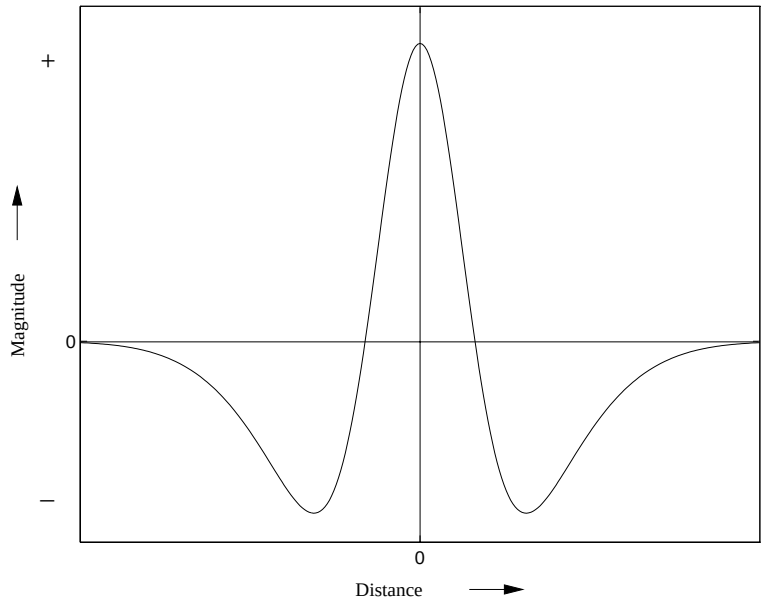


Fig. B.20 Mexican hat lateral spatial interaction profile.

Hebbian learning with co-active inputs. Backprojections from such multimodal areas to the early auditory cortical areas could then influence the responses of auditory cortex neurons to auditory inputs (see Section B.9 and cf. Calvert, Bullmore, Brammer, Campbell, Williams, McGuire, Woodruff, Iversen and David (1997)).

A similar principle may operate in any multilayer hierarchical cortical processing system, such as the ventral visual system, in that the categories that can be formed only at later stages of processing may help earlier stages to form categories relevant to what can be identified at the later stages as a result of the operation of backprojections (Rolls 1989a). The idea that the statistical correlation between the inputs received by neighbouring processing streams can be used to guide unsupervised learning within each stream has also been developed by Becker and Hinton (1992) and others (see Phillips, Kay and Smyth (1995)). The networks considered by these authors self-organize under the influence of collateral connections, such as may be implemented by cortico-cortical connections between parallel processing systems in the brain. They use learning rules that, although somewhat complex, are still local in nature, and tend to optimize specific objective functions. The locality of the learning rule, and the simulations performed so far, raise some hope that, once the operation of these types of networks is better understood, they might achieve similar computational capabilities to backpropagation networks (see Section B.12) while retaining biological plausibility.

B.4.6 Topographic map formation

A simple modification to the competitive networks described so far enables them to develop topological maps. In such maps, the closeness in the map reflects the similarity (correlation) between the features in the inputs. The modification that allows such maps to self-organize is to add short-range excitation and long-range inhibition between the neurons. The function to be implemented has a spatial profile that is described as having a Mexican hat shape (see Fig. B.20). The effect of this connectivity between neurons, which need not be modifiable, is to encourage neurons that are close together to respond to similar features in the input

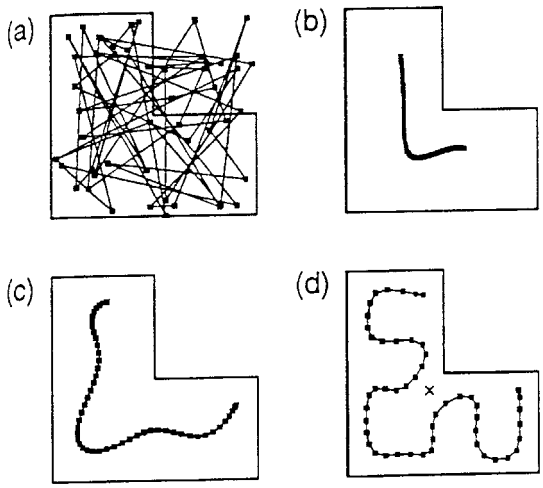


Fig. B.21 Kohonen feature mapping from a two-dimensional L-shaped region to a linear array of 50 units. Each unit has 2 inputs. The input patterns are the X,Y coordinates of points within the L-shape shown. In the diagrams, each point shows the position of a weight vector. Lines connect adjacent units in the 1 D (linear) array of 50 neurons. The weights were initialized to random values within the unit square (a). During feature mapping training, the weights evolved through stages (b) and (c) to (d). By stage (d) the weights have formed so that the positions in the original input space are mapped to a 1 D vector in which adjacent points in the input space activate neighbouring units in the linear array of output units. (Reproduced with permission from Hertz, Krogh and Palmer 1991, Fig. 9.13.)

space, and to encourage neurons that are far apart to respond to different features in the input space. When these response tendencies are present during learning, the feature analyzers that are built by modifying the synapses from the input onto the activated neurons tend to be similar if they are close together, and different if far apart. This is illustrated in Figs. B.21 and B.22. Feature maps built in this way were described by von der Malsburg (1973) and Willshaw and von der Malsburg (1976). It should be noted that the learning rule needed is simply the modified Hebb rule described above for competitive networks, and is thus local and biologically plausible. (For computational convenience, the algorithm that Kohonen (Kohonen 1982, Kohonen 1989, Kohonen 1995) has mainly used does not use Mexican hat connectivity between the neurons, but instead arranges that when the weights to a winning neuron are updated, so to a smaller extent are those of its neighbours – see further Hertz, Krogh and Palmer (1991).)

A very common characteristic of connectivity in the brain, found for example throughout the neocortex, consists of short-range excitatory connections between neurons, with inhibition mediated via inhibitory interneurons. The density of the excitatory connectivity even falls gradually as a function of distance from a neuron, extending typically a distance in the order of 1 mm from the neuron (Braitenberg and Schuz 1991), contributing to a spatial function quite like that of a Mexican hat. (Longer-range inhibitory influences would form the negative part of the spatial response profile.) This supports the idea that topological maps, though in some cases probably seeded by chemoaffinity, could develop in the brain with the assistance of the processes just described. It is noted that some cortico-cortical connections even within an area may be longer, skipping past some intermediate neurons, and then making connections after some distance with a further group of neurons. Such longer-range connections are found

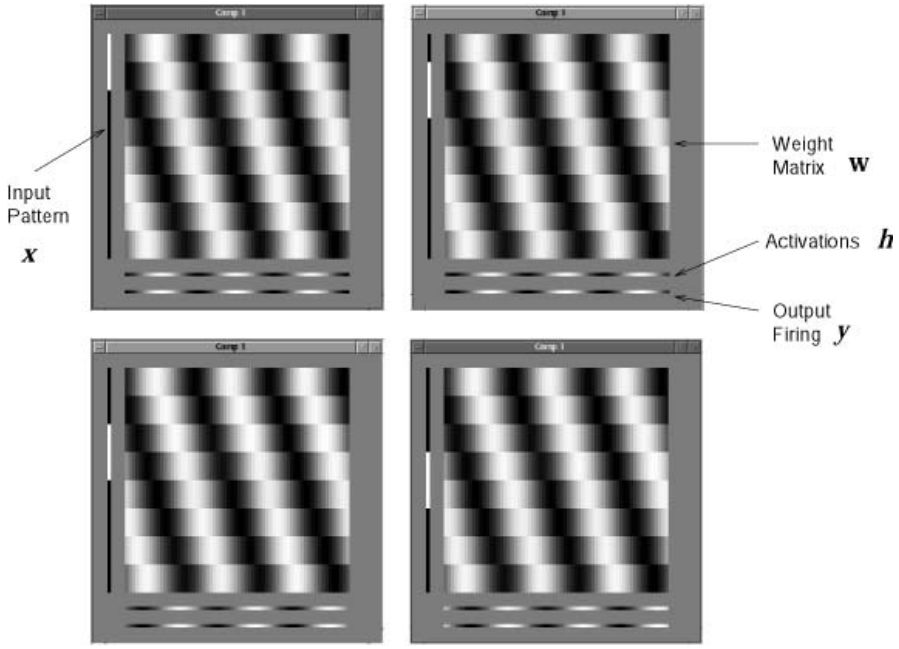


Fig. B.22 Example of a one-dimensional topological map that self-organized from inputs in a low-dimensional space. The network has 64 neurons (vertical elements in the diagram) and 64 inputs per neuron (horizontal elements in the diagram). The four different diagrams represent the net tested with different input patterns. The input patterns $\mathbf{x}$ are displayed at the left of each diagram, with white representing firing and black not firing for each of the 64 inputs. The central square of each diagram represents the synaptic weights of the neurons, with white representing a strong weight. The row vector below each weight matrix represents the activations of the 64 output neurons, and the bottom row vector the output firing $\mathbf{y}$. The network was trained with a set of 8 binary input patterns, each of which overlapped in 8 of its 16 'on' elements with the next pattern. The diagram shows that as one moves through correlations in the input space (top left to top right to bottom left to bottom right), so the output neurons activated move steadily across the output array of neurons. Closely correlated inputs are represented close together in the output array of neurons. The way in which this occurs can be seen by inspection of the weight matrix. The network architecture was the same as for a competitive net, except that the activations were converted linearly into output firings, and then each neuron excited its neighbours and inhibited neurons further away. This lateral inhibition was implemented for the simulation by a spatial filter operating on the output firings with the following filter weights (cf. Fig. B.20): 5, 5, 5, 5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 10, 10, 10, 5, 5, 5, 5, 5, 5, 5, 5, 5 which operated on the 64-element firing rate vector.

for example between different columns with similar orientation selectivity in the primary visual cortex. The longer range connections may play a part in stabilizing maps, and again in the exchange of information between neurons performing related computations, in this case about features with the same orientations.

If a low-dimensional space, for example the orientation sensitivity of cortical neurons in the primary visual cortex (which is essentially one-dimensional, the dimension being angle), is mapped to a two-dimensional space such as the surface of the cortex, then the resulting map can have long spatial runs where the value along the dimension (in this case orientation tuning) alters gradually, and continuously. Such self-organization can account for many aspects of the mapping of orientation tuning, and of ocular dominance columns, in V1 (Miller 1994, Harris, Ermentrout and Small 1997). If a high-dimensional information space is mapped to the two-dimensional cortex, then there will be only short runs of groups of neurons with similar feature

responsiveness, and then the map must fracture, with a different type of feature mapped for a short distance after the discontinuity. This is exactly what Rolls suggests is the type of topology found in the anterior inferior temporal visual cortex, with the individual groupings representing what can be self-organized by competitive networks combined with a trace rule as described in Section 4.5. Here, visual stimuli are not represented with reference to their position on the retina, because here the neurons are relatively translation invariant. Instead, when recording here, small clumps of neurons with similar responses may be encountered close together, and then one moves into a group of neurons with quite different feature selectivity (personal observations). This topology will arise naturally, given the anatomical connectivity of the cortex with its short-range excitatory connections, because there are very many different objects in the world and different types of features that describe objects, with no special continuity between the different combinations of features possible.

Rolls' hypothesis contrasts with the view of Tanaka (1996), who has claimed that the inferior temporal cortex provides an alphabet of visual features arranged in discrete modules. The type of mapping found in higher cortical visual areas as proposed by Rolls implies that topological self-organization is an important way in which maps in the brain are formed, for it seems most unlikely that the locations in the map of different types of object seen in an environment could be specified genetically (Rolls and Stringer 2000). Consistent with this, Tsao, Freiwald, Tootell and Livingstone (2006) described with macaque fMRI 'anterior face patches' at A15 to A22. A15 might correspond to where we have analysed face-selective neurons (it might translate to 3 mm posterior to our sphenoid reference, see Section 4.2), and at this level there are separate regions specialized for face identity in areas TEa and TEm on the ventral lip of the superior temporal sulcus and the adjacent gyrus, and for face expression and movement in the cortex deep in the superior temporal sulcus (Hasselmo, Rolls and Baylis 1989a, Baylis, Rolls and Leonard 1987, Rolls 2007i). The 'middle face patch' of Tsao, Freiwald, Tootell and Livingstone (2006) was at A6, which is probably part of the posterior inferior temporal cortex, and, again consistent with self-organizing map principles, has a high concentration of face-selective neurons within the patch.

The biological utility of developing such topology-preserving feature maps may be that if the computation requires neurons with similar types of response to exchange information more than neurons involved in different computations (which is more than reasonable), then the total length of the connections between the neurons is minimized if the neurons that need to exchange information are close together (cf. Cowey (1979), Durbin and Mitchison (1990)). Examples of this include the separation of colour constancy processing in V4 from global motion processing in MT as follows (see Chapter 3 of Rolls and Deco (2002)). In V4, to compute colour constancy, an estimate of the illuminating wavelengths can be obtained by summing the outputs of the pyramidal cells in the inhibitory interneurons over several degrees of visual space, and subtracting this from the excitatory central ON colour-tuned region of the receptive field by (subtractive) feedback inhibition. This enables the cells to discount the illuminating wavelength, and thus compute colour constancy. For this computation, no inputs from motion-selective cells (which in the dorsal stream are colour insensitive) are needed. In MT, to compute global motion (e.g. the motion produced by the average flow of local motion elements, exemplified for example by falling snow), the computation can be performed by averaging in the larger (several degrees) receptive fields of MT the local motion inputs received by neurons in earlier cortical areas (V1 and V2) with small receptive fields (see Chapter 3 of Rolls and Deco (2002) and Rolls and Stringer (2007)). For this computation, no input from colour cells is useful. Having separate areas (V4 and MT) for these different computations minimizes the wiring lengths, for having intermingled colour and motion cells in a single cortical area would increase the average connection length between the neurons that need to be connected for the computations being performed. Minimizing the total connection length

between neurons in the brain is very important in order to keep the size of the brain relatively small.

Placing close to each other neurons that need to exchange information, or that need to receive information from the same source, or that need to project towards the same destination, may also help to minimize the complexity of the rules required to specify cortical (and indeed brain) connectivity (Rolls and Stringer 2000). For example, in the case of V4 and MT, the connectivity rules can be simpler (e.g. connect to neurons in the vicinity, rather than look for colour-, or motion-marked cells, and connect only to the cells with the correct genetically specified label specifying that the cell is either part of motion or of colour processing). Further, the V4 and MT example shows that how the neurons are connected can be specified quite simply, but of course it needs to be specified to be different for different computations. Specifying a general rule for the classes of neurons in a given area also provides a useful simplification to the genetic rules needed to specify the functional architecture of a given cortical area (Rolls and Stringer 2000). In our V4 and MT example, the genetic rules would need to specify the rules separately for different populations of inhibitory interneurons if the computations performed by V4 and MT were performed with intermixed neurons in a single brain area. Together, these two principles, of minimization of wiring length, and allowing simple genetic specification of wiring rules, may underlie the separation of cortical visual information processing into different (e.g. ventral and dorsal) processing streams. The same two principles operating within each brain processing stream may underlie (taken together with the need for hierarchical processing to enable the computations to be biologically plausible in terms of the number of connections per neuron, and the need for local learning rules, see Section B.13) much of the overall architecture of visual cortical processing, and of information processing and its modular architecture throughout the cortex more generally.

The rules of information exchange just described could also tend to produce more gross topography in cortical regions. For example, neurons that respond to animate objects may have certain visual feature requirements in common, and may need to exchange information about these features. Other neurons that respond to inanimate objects might have somewhat different visual feature requirements for their inputs, and might need to exchange information strongly. (For example, selection of whether an object is a chisel or a screwdriver may require competition by mutual (lateral) inhibition to produce the contrast enhancement necessary to result in unambiguous neuronal responses.) The rules just described would account for neurons with responsiveness to inanimate and animate objects tending to be grouped in separate parts of a cortical map or representation, and thus separately susceptible to brain damage (see e.g. Farah (1990), Farah (2000)).

B.4.7 Invariance learning by competitive networks

In conventional competitive learning, the weight vector of a neuron can be thought of as moving towards the centre of a cluster of similar overlapping input stimuli (Rumelhart and Zipser 1985, Hertz, Krogh and Palmer 1991, Rolls and Treves 1998, Rolls and Deco 2002, Perry, Rolls and Stringer 2007). The weight vector points towards the centre of the set of stimuli in the category. The different training stimuli that are placed into the same category (i.e. activate the same neuron) are typically overlapping in that the pattern vectors are correlated with each other. Figure B.23a illustrates this.

For the formation of invariant representations, there are multiple occurrences of the object at different positions in the space. The object at each position represents a different transform (whether in position, size, view etc.) of the object. The different transforms may be uncorrelated with each other, as would be the case for example with an object translated so far in the space that there would be no active afferents in common between the two transforms.

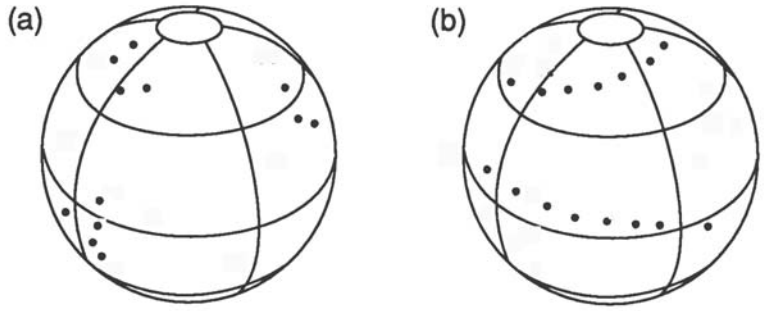


Fig. B.23 (a) Conventional competitive learning. A cluster of overlapping input patterns is categorized as being similar, and this is implemented by a weight vector of an output neuron pointing towards the centre of the cluster. Three clusters are shown, and each cluster might after training have a weight vector pointing towards it. (b) Invariant representation learning. The different transforms of an object may span an elongated region of the space, and the transforms at the far ends of the space may have no overlap (correlation), yet the network must learn to categorize them as similar. The different transforms of two different objects are represented.

Yet we need these two orthogonal patterns to be mapped to the same output. It may be a very elongated part of the input space that has to be mapped to the same output in invariance learning. These concepts are illustrated in Fig. B.23b.

Objects in the world have temporal and spatial continuity. That is, the statistics of the world are such that we tend to look at one object for some time, during which it may be transforming continuously from one view to another. The temporal continuity property is used in trace rule invariance training, in which a short-term memory in the associative learning rule normally used to train competitive networks is used to help build representations that reflect the continuity in time that characterizes the different transforms of each object, as described in Chapter 4. The transforms of an object also show spatial continuity, and this can also be used in invariance training in what is termed continuous spatial transform learning, described in Section 4.5.11.

In conventional competitive learning the overall weight vector points to the prototypical representation of the object. The only sense in which after normal competitive training (without translations etc) the network generalizes is with respect to the dot product similarity of any input vector compared to the central vector from the training set that the network learns. Continuous spatial transformation learning works by providing a set of training vectors that overlap, and between them cover the whole space over which an invariant transform of the object must be learned. Indeed, it is important for continuous spatial transformation learning that the different exemplars of an object are sufficiently close that the similarity of adjacent training exemplars is sufficient to ensure that the same postsynaptic neuron learns to bridge the continuous space spanned by the whole set of training exemplars of a given object (Stringer, Perry, Rolls and Proske 2006, Perry, Rolls and Stringer 2006, Perry, Rolls and Stringer 2007). This will enable the postsynaptic neuron to span a very elongated space of the different transforms of an object, as described in Section 4.5.11.

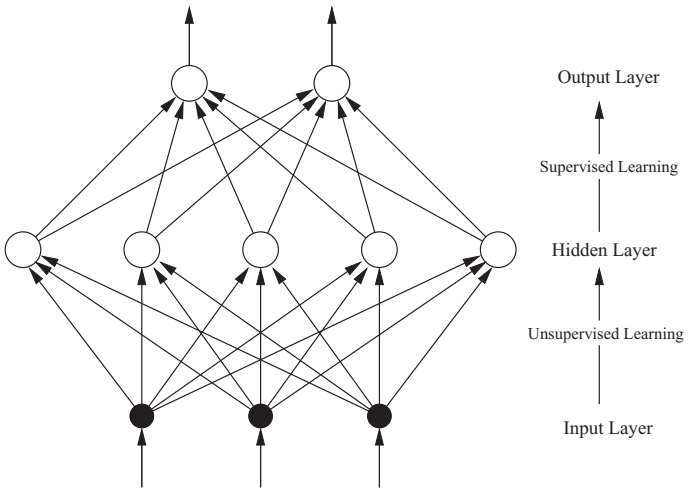


Fig. B.24 A hybrid network, in which for example unsupervised learning rapidly builds relatively orthogonal representations based on input differences, and this is followed by a one-layer supervised network (taught for example by the delta rule) that learns to classify the inputs based on the categorizations formed in the hidden/intermediate layer.

B.4.8 Radial Basis Function networks

As noted above, a competitive network can act as a useful preprocessor for other networks. In the neural examples above, competitive networks were useful preprocessors for associative networks. Competitive networks are also used as preprocessors in artificial neural networks, for example in hybrid two-layer networks such as that illustrated in Fig. B.24. The competitive network is advantageous in this hybrid scheme, because as an unsupervised network, it can relatively quickly (with a few presentations of each stimulus) discover the main features in the input space, and code for them. This leaves the second layer of the network to act as a supervised network (taught for example by the delta rule, see Section B.11), which learns to map the features found by the first layer into the output required. This learning scheme is very much faster than that of a (two-layer) backpropagation network, which learns very slowly because it takes it a long time to perform the credit assignment to build useful feature analyzers in layer one (the hidden layer) (see Section B.12).

The general scheme shown in Fig. B.24 is used in radial basis function (RBF) neural networks. The main difference from what has been described is that in an RBF network, the hidden neurons do not use a winner-take-all function (as in some competitive networks), but instead use a normalized activation function in which the measure of distance from a weight vector of the neural input is (instead of the dot product $\mathbf{x} \cdot \mathbf{w}_i$ used for most of the networks described in this book), a Gaussian measure of distance:

$$y_i = \frac{\exp[-(\mathbf{x} - \mathbf{w}_i)^2/2\sigma_i^2]}{\sum_k \exp[-(\mathbf{x} - \mathbf{w}_k)^2/2\sigma_k^2]}. \tag{B.21}$$

The effect is that the response y_i of neuron i is a maximum if the input stimulus vector $\mathbf{x}$ is centred at $\mathbf{w}_i$, the weight vector of neuron i (this is the upper term in equation B.21). The magnitude is normalized by dividing by the sum of the activations of all the k neurons in the network. If the input vector $\mathbf{x}$ is not at the centre of the receptive field of the neuron, then the response is decreased according to how far the input vector is from the weight vector $\mathbf{w}_i$ of the neuron, with the weighting decreasing as a Gaussian function with a standard deviation

of σ. The idea is like that implemented with soft competition, in that the relative response of different neurons provides an indication of where the input pattern is in relation to the weight vectors of the different neurons. The rapidity with which the response falls off in a Gaussian radial basis function neuron is set by σ_i, which is adjusted so that for any given input pattern vector, a number of RBF neurons are activated. The positions in which the RBF neurons are located (i.e. the directions of their weight vectors, $\mathbf{w}$) are determined usually by unsupervised learning, e.g. the vector quantization that is produced by the normal competitive learning algorithm. The first layer of an RBF network is not different in principle from a network with soft competition, and it is not clear how biologically a Gaussian activation function would be implemented, so the treatment is not developed further here (see Hertz, Krogh and Palmer (1991), Poggio and Girosi (1990a), and Poggio and Girosi (1990b) for further details).

B.4.9 Further details of the algorithms used in competitive networks

B.4.9.1 Normalization of the inputs

Normalization is useful because in step 1 of the training algorithm described in Section B.4.2.2, the neuronal activations, formed by the inner product of the pattern and the normalized weight vector on each neuron, are scaled in such a way that they have a maximum value of 1.0. This helps different input patterns to be equally effective in the learning process. A way in which this normalization could be achieved by a layer of input neurons is given by Grossberg (1976a). In the brain, a number of factors may contribute to normalization of the inputs. One factor is that a set of input axons to a neuron will come from another network in which the firing is controlled by inhibitory feedback, and if the numbers of axons involved is large (hundreds or thousands), then the inputs will be in a reasonable range. Second, there is increasing evidence that the different classes of input to a neuron may activate different types of inhibitory interneuron (e.g. Buhl, Halasy and Somogyi (1994)), which terminate on separate parts of the dendrite, usually close to the site of termination of the corresponding excitatory afferents. This may allow separate feedforward inhibition for the different classes of input. In addition, the feedback inhibitory interneurons also have characteristic termination sites, often on or close to the cell body, where they may be particularly effective in controlling firing of the neuron by shunting (divisive) inhibition, rather than by scaling a class of input (see Section B.6).

B.4.9.2 Normalization of the length of the synaptic weight vector on each dendrite

This is necessary to ensure that one or a few neurons do not always win the competition. (If the weights on one neuron were increased by simple Hebbian learning, and there was no normalization of the weights on the neuron, then it would tend to respond strongly in the future to patterns with some overlap with patterns to which that neuron has previously learned, and gradually that neuron would capture a large number of patterns.) A biologically plausible way to achieve this weight adjustment is to use a modified Hebb rule:

$$\delta w_{ij} = \alpha y_i (x_j - w_{ij}) \tag{B.22}$$

where α is a constant, and x_j and w_{ij} are in appropriate units. In vector notation,

$$\delta \mathbf{w}_i = \alpha y_i (\mathbf{x} - \mathbf{w}_i) \tag{B.23}$$

where $\mathbf{w}_i$ is the synaptic weight vector on neuron i. This implements a Hebb rule that increases synaptic strength according to conjunctive pre- and post-synaptic activity, and also allows the strength of each synapse to decrease in proportion to the firing rate of the postsynaptic neuron

(as well as in proportion to the existing synaptic strength). This results in a decrease in synaptic strength for synapses from weakly active presynaptic neurons onto strongly active postsynaptic neurons. Such a modification in synaptic strength is termed heterosynaptic long-term depression in the neurophysiological literature, referring to the fact that the synapses that weaken are other than those that activate the neuron.

This is an important computational use of heterosynaptic long-term depression (LTD). In that the amount of decrease of the synaptic strength depends on how strong the synapse is already, the rule is compatible with what is frequently reported in studies of LTD (see Section 1.5). This rule can maintain the sums of the synaptic weights on each dendrite to be very similar without any need for explicit normalization of the synaptic strengths, and is useful in competitive nets. This rule was used by Willshaw and von der Malsburg (1976). As is made clear with the vector notation above, the modified Hebb rule moves the direction of the weight vector $\mathbf{w}_i$ towards the current input pattern vector $\mathbf{x}$ in proportion to the difference between these two vectors and the firing rate y_i of neuron i.

If explicit weight (vector length) normalization is needed, the appropriate form of the modified Hebb rule is:

$$\delta w_{ij} = \alpha y_i (x_j - y_i w_{ij}). \tag{B.24}$$

This rule, formulated by Oja (1982), makes weight decay proportional to y_i^2, normalizes the synaptic weight vector (see Hertz, Krogh and Palmer (1991)), is still a local learning rule, and is known as the Oja rule.

B.4.9.3 Non-linearity in the learning rule

Non-linearity in the learning rule can assist competition (Rolls 1989b, Rolls 1996c). For example, in the brain, long-term potentiation typically occurs only when strong activation of a neuron has produced sufficient depolarization for the voltage-dependent NMDA receptors to become unblocked, allowing Ca^{2+} to enter the cell (see Section 1.5). This means that synaptic modification occurs only on neurons that are strongly activated, effectively assisting competition to select few winners. The learning rule can be written:

$$\delta w_{ij} = \alpha m_i x_j \tag{B.25}$$

where m_i is a (e.g. threshold) non-linear function of the post-synaptic firing y_i which mimics the operation of the NMDA receptors in learning. (It is noted that in associative networks the same process may result in the stored pattern being more sparse than the input pattern, and that this may be beneficial, especially given the exponential firing rate distribution of neurons, in helping to maximize the number of patterns stored in associative networks (see Sections B.2, B.3, and C.3.1).

B.4.9.4 Competition

In a simulation of a competitive network, a single winner can be selected by searching for the neuron with the maximum activation. If graded competition is required, this can be achieved by an activation function that increases greater than linearly. In some of the networks we have simulated (Rolls 1989b, Rolls 1989a, Wallis and Rolls 1997), raising the activation to a fixed power, typically in the range 2–5, and then rescaling the outputs to a fixed maximum (e.g. 1) is simple to implement. In a real neuronal network, winner-take-all competition can be implemented using mutual (lateral) inhibition between the neurons with non-linear activation functions, and self-excitation of each neuron (see e.g. Grossberg (1976a), Grossberg (1988), Hertz, Krogh and Palmer (1991)).

Another method to implement soft competition in simulations is to use the normalized exponential or 'softmax' activation function for the neurons (Bridle (1990); see Bishop (1995)):

$$y = \exp(h) / \sum_i \exp(h_i) \ . \tag{B.26}$$

This function specifies that the firing rate of each neuron is an exponential function of the activation, scaled by the whole vector of activations h_i, $i = 1, N$. The exponential function (in increasing supralinearly) implements soft competition, in that after the competition the faster firing neurons are firing relatively much faster than the slower firing neurons. In fact, the strength of the competition can be adjusted by using a 'temperature' T greater than 0 as follows:

$$y = \exp(h/T) / \sum_i \exp(h_i/T). \tag{B.27}$$

Very low temperatures increase the competition, until with $T \rightarrow 0$, the competition becomes 'winner-take-all'. At high temperatures, the competition becomes very soft. (When using the function in simulations, it may be advisable to prescale the firing rates to for example the range 0–1, both to prevent machine overflow, and to set the temperature to operate on a constant range of firing rates, as increasing the range of the inputs has an effect similar to decreasing T.)

The softmax function has the property that activations in the range $-\infty$ to $+\infty$ are mapped into the range 0 to 1.0, and the sum of the firing rates is 1.0. This facilitates interpretation of the firing rates under certain conditions as probabilities, for example that the competitive network firing rate of each neuron reflects the probability that the input vector is within the category or cluster signified by that output neuron (see Bishop (1995)).

B.4.9.5 Soft competition

The use of graded (continuous valued) output neurons in a competitive network, and soft competition rather than winner-take-all competition, has the value that the competitive net generalizes more continuously to an input vector that lies between input vectors that it has learned. Also, with soft competition, neurons with only a small amount of activation by any of the patterns being used will nevertheless learn a little, and move gradually towards the patterns that are being presented. The result is that with soft competition, the output neurons all tend to become allocated to one of the input patterns or one of the clusters of input patterns.

B.4.9.6 Untrained neurons

In competitive networks, especially with winner-take-all or finely tuned neurons, it is possible that some neurons remain unallocated to patterns. This may be useful, in case patterns in the unused part of the space occur in future. Alternatively, unallocated neurons can be made to move towards the parts of the space where patterns are occurring by allowing such losers in the competition to learn a little. Another mechanism is to subtract a bias term μ_i from y_i, and to use a 'conscience' mechanism that raises μ_i if a neuron wins frequently, and lowers μ_i if it wins infrequently (Grossberg 1976b, Bienenstock, Cooper and Munro 1982, De Sieno 1988).

B.4.9.7 Large competitive nets: further aspects

If a large neuronal network is considered, with the number of synapses on each neuron in the region of 10,000, as occurs on large pyramidal cells in some parts of the brain, then there is a potential disadvantage in using neurons with synaptic weights that can take on only positive values. This difficulty arises in the following way. Consider a set of positive normalized input firing rates and synaptic weight vectors (in which each element of the vector can take on any value between 0.0 and 1.0). Such vectors of random values will on average be more highly aligned with the direction of the central vector (1,1,1,...,1) than with any other vector. An example can be given for the particular case of vectors evenly

distributed on the positive 'quadrant' of a high-dimensional hypersphere: the average overlap (i.e. normalized dot product) between two binary random vectors with half the elements on and thus a sparseness of 0.5 (e.g. a random pattern vector and a random dendritic weight vector) will be approximately 0.5, while the average overlap between a random vector and the central vector will be approximately 0.707. A consequence of this will be that if a neuron begins to learn towards several input pattern vectors it will get drawn towards the average of these input patterns which will be closer to the 1,1,1,...,1 direction than to any one of the patterns. As a dendritic weight vector moves towards the central vector, it will become more closely aligned with more and more input patterns so that it is more rapidly drawn towards the central vector. The end result is that in large nets of this type, many of the dendritic weight vectors will point towards the central vector. This effect is not seen so much in small systems, since the fluctuations in the magnitude of the overlaps are sufficiently large that in most cases a dendritic weight vector will have an input pattern very close to it and thus will not learn towards the centre. In large systems, the fluctuations in the overlaps between random vectors become smaller by a factor of $\frac{1}{\sqrt{N}}$ so that the dendrites will not be particularly close to any of the input patterns.

One solution to this problem is to allow the elements of the synaptic weight vectors to take negative as well as positive values. This could be implemented in the brain by feedforward inhibition. A set of vectors taken with random values will then have a reduced mean correlation between any pair, and the competitive net will be able to categorize them effectively. A system with synaptic weights that can be negative as well as positive is not physiologically plausible, but we can instead imagine a system with weights lying on a hypersphere in the positive quadrant of space but with additional inhibition that results in the cumulative effects of some input lines being effectively negative. This can be achieved in a network by using positive input vectors, positive synaptic weight vectors, and thresholding the output neurons at their mean activation. A large competitive network of this general nature does categorize well, and has been described more fully elsewhere (Bennett 1990). In a large network with inhibitory feedback neurons, and principal cells with thresholds, the network could achieve at least in part an approximation to this type of thresholding useful in large competitive networks.

A second way in which nets with positive-only values of the elements could operate is by making the input vectors sparse and initializing the weight vectors to be sparse, or to have a reduced contact probability. (A measure a of neuronal population sparseness is defined (as before) in equation B.28:

$$a = \frac{(\sum_{i=1}^{N} y_i/N)^2}{\sum_{i=1}^{N} y_i^2/N} \tag{B.28}$$

where y_i is the firing rate of the ith neuron in the set of N neurons.) For relatively small net sizes simulated ($N = 100$) with patterns with a sparseness a of, for example, 0.1 or 0.2, learning onto the average vector can be avoided. However, as the net size increases, the sparseness required does become very low. In large nets, a greatly reduced contact probability between neurons (many synapses kept identically zero) would prevent learning of the average vector, thus allowing categorization to occur. Reduced contact probability will, however, prevent complete alignment of synapses with patterns, so that the performance of the network will be affected.

B.5 Continuous attractor networks

B.5.1 Introduction

Single-cell recording studies have shown that some neurons represent the current position along a continuous physical dimension or space even when no inputs are available, for example in darkness (see Chapter 2). Examples include neurons that represent the positions of the eyes (i.e. eye direction with respect to the head), the place where the animal is looking in space, head direction, and the place where the animal is located. In particular, examples of such classes of cells include head direction cells in rats (Ranck 1985, Taube, Muller and Ranck 1990a, Taube, Goodridge, Golob, Dudchenko and Stackman 1996, Muller, Ranck and Taube 1996) and primates (Robertson, Rolls, Georges-François and Panzeri 1999), which respond maximally when the animal's head is facing in a particular preferred direction; place cells in rats (O'Keefe and Dostrovsky 1971, McNaughton, Barnes and O'Keefe 1983, O'Keefe 1984, Muller, Kubie, Bostock, Taube and Quirk 1991, Markus, Qin, Leonard, Skaggs, McNaughton and Barnes 1995) that fire maximally when the animal is in a particular location; and spatial view cells in primates that respond when the monkey is looking towards a particular location in space (Rolls, Robertson and Georges-François 1997a, Georges-François, Rolls and Robertson 1999, Robertson, Rolls and Georges-François 1998). In the parietal cortex there are many spatial representations, in several different coordinate frames (see Chapter 4 of Rolls and Deco (2002) and Andersen, Batista, Snyder, Buneo and Cohen (2000)), and they have some capability to remain active during memory periods when the stimulus is no longer present. Even more than this, the dorsolateral prefrontal cortex networks to which the parietal networks project have the capability to maintain spatial representations active for many seconds or minutes during short-term memory tasks, when the stimulus is no longer present (see Section 5.1). In this section, we describe how such networks representing continuous physical space could operate. The locations of such spatial networks in the brain are the parietal areas, the prefrontal areas that implement short-term spatial memory and receive from the parietal cortex (see Section 5.1), and the hippocampal system which combines information about objects from the inferior temporal visual cortex with spatial information (see Chapter 2).

A class of network that can maintain the firing of its neurons to represent any location along a continuous physical dimension such as spatial position, head direction, etc. is a 'Continuous Attractor' neural network (CANN). It uses excitatory recurrent collateral connections between the neurons to reflect the distance between the neurons in the state space of the animal (e.g. head direction space). These networks can maintain the bubble of neural activity constant for long periods wherever it is started to represent the current state (head direction, position, etc) of the animal, and are likely to be involved in many aspects of spatial processing and memory, including spatial vision. Global inhibition is used to keep the number of neurons in a bubble or packet of actively firing neurons relatively constant, and to help to ensure that there is only one activity packet. Continuous attractor networks can be thought of as very similar to autoassociation or discrete attractor networks (described in Section B.3), and have the same architecture, as illustrated in Fig. B.25. The main difference is that the patterns stored in a CANN are continuous patterns, with each neuron having broadly tuned firing which decreases with for example a Gaussian function as the distance from the optimal firing location of the cell is varied, and with different neurons having tuning that overlaps throughout the space. Such tuning is illustrated in Fig. 2.16. For comparison, the autoassociation networks described in Section B.3 have discrete (separate) patterns (each pattern implemented by the firing of a particular subset of the neurons), with no continuous distribution of the patterns throughout the space (see Fig. 2.16). A consequent difference is that the CANN can maintain its firing at any location in the trained continuous space, whereas a discrete attractor or autoassociation

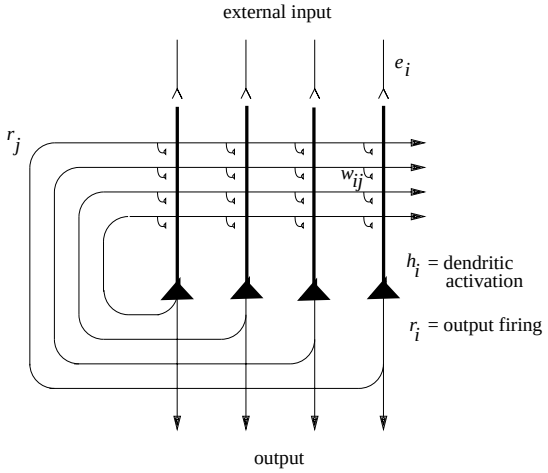


Fig. B.25 The architecture of a continuous attractor neural network (CANN).

network moves its population of active neurons towards one of the previously learned attractor states, and thus implements the recall of a particular previously learned pattern from an incomplete or noisy (distorted) version of one of the previously learned patterns. The energy landscape of a discrete attractor network (see equation B.12) has separate energy minima, each one of which corresponds to a learned pattern, whereas the energy landscape of a continuous attractor network is flat, so that the activity packet remains stable with continuous firing wherever it is started in the state space. (The state space refers to the set of possible spatial states of the animal in its environment, e.g. the set of possible head directions.)

In Section B.5.2, we first describe the operation and properties of continuous attractor networks, which have been studied by for example Amari (1977), Zhang (1996), and Taylor (1999), and then, following Stringer, Trappenberg, Rolls and De Araujo (2002b), address four key issues about the biological application of continuous attractor network models.

One key issue in such continuous attractor neural networks is how the synaptic strengths between the neurons in the continuous attractor network could be learned in biological systems (Section B.5.3).

A second key issue in such continuous attractor neural networks is how the bubble of neuronal firing representing one location in the continuous state space should be updated based on non-visual cues to represent a new location in state space (Section B.5.5). This is essentially the problem of path integration: how a system that represents a memory of where the agent is in physical space could be updated based on idiothetic (self-motion) cues such as vestibular cues (which might represent a head velocity signal), or proprioceptive cues (which might update a representation of place based on movements being made in the space, during for example walking in the dark).

A third key issue is how stability in the bubble of activity representing the current location can be maintained without much drift in darkness, when it is operating as a memory system (Section B.5.6).

A fourth key issue is considered in Section B.5.8 in which we describe networks that store both continuous patterns and discrete patterns (see Fig. 2.16), which can be used to store for example the location in (continuous, physical) space where an object (a discrete item) is present.

B.5.2 The generic model of a continuous attractor network

The generic model of a continuous attractor is as follows. (The model is described in the context of head direction cells, which represent the head direction of rats (Taube et al. 1996, Muller et al. 1996) and macaques (Robertson, Rolls, Georges-François and Panzeri 1999), and can be reset by visual inputs after gradual drift in darkness.) The model is a recurrent attractor network with global inhibition. It is different from a Hopfield attractor network primarily in that there are no discrete attractors formed by associative learning of discrete patterns. Instead there is a set of neurons that are connected to each other by synaptic weights w_{ij} that are a simple function, for example Gaussian, of the distance between the states of the agent in the physical world (e.g. head directions) represented by the neurons. Neurons that represent similar states (locations in the state space) of the agent in the physical world have strong synaptic connections, which can be set up by an associative learning rule, as described in Section B.5.3. The network updates its firing rates by the following 'leaky-integrator' dynamical equations. The continuously changing activation h_i^{HD} of each head direction cell i is governed by the equation

$$\tau \frac{\mathrm{d}h_i^{\mathrm{HD}}(t)}{\mathrm{d}t} = -h_i^{\mathrm{HD}}(t) + \frac{\phi_0}{C^{\mathrm{HD}}} \sum_j (w_{ij} - w^{\mathrm{inh}}) r_j^{\mathrm{HD}}(t) + I_i^V, \tag{B.29}$$

where r_j^{HD} is the firing rate of head direction cell j, w_{ij} is the excitatory (positive) synaptic weight from head direction cell j to cell i, w^{inh} is a global constant describing the effect of inhibitory interneurons, and τ is the time constant of the system [40]. The term $-h_i^{\mathrm{HD}}(t)$ indicates the amount by which the activation decays (in the leaky integrator neuron) at time t. (The network is updated in a typical simulation at much smaller timesteps than the time constant of the system, τ.) The next term in equation B.29 is the input from other neurons in the network r_j^{HD} weighted by the recurrent collateral synaptic connections w_{ij} (scaled by a constant ϕ_0 and C^{HD} which is the number of synaptic connections received by each head direction cell from other head direction cells in the continuous attractor). The term I_i^V represents a visual input to head direction cell i. Each term I_i^V is set to have a Gaussian response profile in most continuous attractor networks, and this sets the firing of the cells in the continuous attractor to have Gaussian response profiles as a function of where the agent is located in the state space (see e.g. Fig. 2.16 on page 65), but the Gaussian assumption is not crucial. (It is known that the firing rates of head direction cells in both rats (Taube, Goodridge, Golob, Dudchenko and Stackman 1996, Muller, Ranck and Taube 1996) and macaques (Robertson, Rolls, Georges-François and Panzeri 1999) is approximately Gaussian.) When the agent is operating without visual input, in memory mode, then the term I_i^V is set to zero. The firing rate r_i^{HD} of cell i is determined from the activation h_i^{HD} and the sigmoid function

$$r_i^{\mathrm{HD}}(t) = \frac{1}{1 + e^{-2\beta(h_i^{\mathrm{HD}}(t) - \alpha)}}, \tag{B.30}$$

where α and β are the sigmoid threshold and slope, respectively.

B.5.3 Learning the synaptic strengths between the neurons that implement a continuous attractor network

So far we have said that the neurons in the continuous attractor network are connected to each other by synaptic weights w_{ij} that are a simple function, for example Gaussian, of the

[40] Note that for this section, we use r rather than y to refer to the firing rates of the neurons in the network, remembering that, because this is a recurrently connected network (see Fig. B.13), the output from a neuron y might be the input x_j to another neuron.

distance between the states of the agent in the physical world (e.g. head directions, spatial views etc) represented by the neurons. In many simulations, the weights are set by formula to have weights with these appropriate Gaussian values. However, Stringer, Trappenberg, Rolls and De Araujo (2002b) showed how the appropriate weights could be set up by learning. They started with the fact that since the neurons have broad tuning that may be Gaussian in shape, nearby neurons in the state space will have overlapping spatial fields, and will thus be co-active to a degree that depends on the distance between them. They postulated that therefore the synaptic weights could be set up by associative learning based on the co-activity of the neurons produced by external stimuli as the animal moved in the state space. For example, head direction cells are forced to fire during learning by visual cues in the environment that produce Gaussian firing as a function of head direction from an optimal head direction for each cell. The learning rule is simply that the weights w_{ij} from head direction cell j with firing rate r_j^{HD} to head direction cell i with firing rate r_i^{HD} are updated according to an associative (Hebb) rule

$$\delta w_{ij} = k r_i^{\mathrm{HD}} r_j^{\mathrm{HD}} \tag{B.31}$$

where δw_{ij} is the change of synaptic weight and k is the learning rate constant. During the learning phase, the firing rate r_i^{HD} of each head direction cell i might be the following Gaussian function of the displacement of the head from the optimal firing direction of the cell

$$r_i^{\mathrm{HD}} = e^{-s_{\mathrm{HD}}^2/2\sigma_{\mathrm{HD}}^2}, \tag{B.32}$$

where s_{HD} is the difference between the actual head direction x (in degrees) of the agent and the optimal head direction x_i for head direction cell i, and σ_{HD} is the standard deviation.

Stringer, Trappenberg, Rolls and De Araujo (2002b) showed that after training at all head directions, the synaptic connections develop strengths that are an almost Gaussian function of the distance between the cells in head direction space, as shown in Fig. B.26 (left). Interestingly if a non-linearity is introduced into the learning rule that mimics the properties of NMDA receptors by allowing the synapses to modify only after strong postsynaptic firing is present, then the synaptic strengths are still close to a Gaussian function of the distance between the connected cells in head direction space (see Fig. B.26, left). They showed that after training, the continuous attractor network can support stable activity packets in the absence of visual inputs (see Fig. B.26, right) provided that global inhibition is used to prevent all the neurons becoming activated. (The exact stability conditions for such networks have been analyzed by Amari (1977)). Thus Stringer, Trappenberg, Rolls and De Araujo (2002b) demonstrated biologically plausible mechanisms for training the synaptic weights in a continuous attractor using a biologically plausible local learning rule.

Stringer, Trappenberg, Rolls and De Araujo (2002b) went on to show that if there was a short term memory trace built into the operation of the learning rule, then this could help to produce smooth weights in the continuous attractor if only incomplete training was available, that is if the weights were trained at only a few locations. The same rule can take advantage in training the synaptic weights of the temporal probability distributions of firing when they happen to reflect spatial proximity. For example, for head direction cells the agent will necessarily move through similar head directions before reaching quite different head directions, and so the temporal proximity with which the cells fire can be used to set up the appropriate synaptic weights. This new proposal for training continuous attractor networks can also help to produce broadly tuned spatial cells even if the driving (e.g. visual) input (I_i^V in equation B.29) during training produces rather narrowly tuned neuronal responses. The learning rule with such temporal properties is a memory trace learning rule that strengthens synaptic connections between neurons, based on the temporal probability distribution of the firing. There are many versions of such rules (Rolls and Milward 2000, Rolls

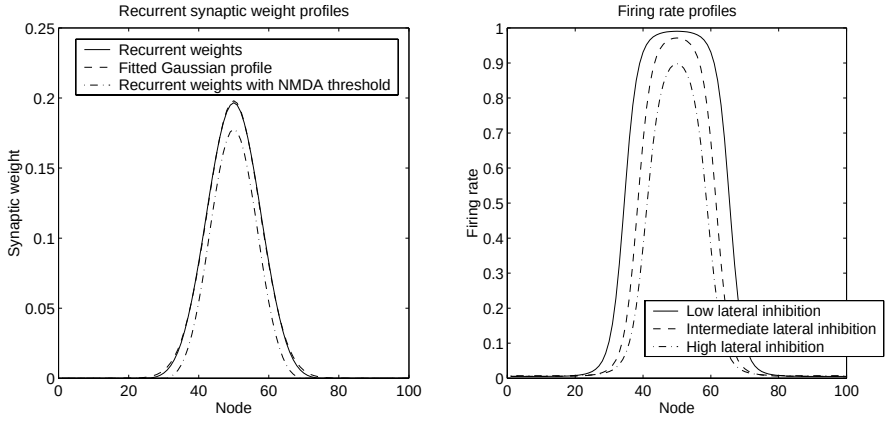


Fig. B.26 Training the weights in a continuous attractor network with an associative rule (equation B.31). Left: the trained recurrent synaptic weights from head direction cell 50 to the other head direction cells in the network arranged in head direction space (solid curve). The dashed line shows a Gaussian curve fitted to the weights shown in the solid curve. The dash-dot curve shows the recurrent synaptic weights trained with rule equation (B.31), but with a non-linearity introduced that mimics the properties of NMDA receptors by allowing the synapses to modify only after strong postsynaptic firing is present. Right: the stable firing rate profiles forming an activity packet in the continuous attractor network during the testing phase when the training (visual) inputs are no longer present. The firing rates are shown after the network has been initially stimulated by visual input to initialize an activity packet, and then allowed to settle to a stable activity profile without visual input. The three graphs show the firing rates for low, intermediate and high values of the lateral inhibition parameter w^{inh}. For both left and right plots, the 100 head direction cells are arranged according to where they fire maximally in the head direction space of the agent when visual cues are available. (After Stringer, Trappenberg, Rolls and de Araujo 2002.)

and Stringer 2001a), which are described more fully in Chapter 4, but a simple one that works adequately is

$$\delta w_{ij} = k\overline{r}_i^{\mathrm{HD}}\overline{r}_j^{\mathrm{HD}} \tag{B.33}$$

where δw_{ij} is the change of synaptic weight, and $\overline{r}^{\mathrm{HD}}$ is a local temporal average or trace value of the firing rate of a head direction cell given by

$$\overline{r}^{\mathrm{HD}}(t+\delta t) = (1-\eta)r^{\mathrm{HD}}(t+\delta t) + \eta\overline{r}^{\mathrm{HD}}(t) \tag{B.34}$$

where η is a parameter set in the interval [0,1] which determines the contribution of the current firing and the previous trace. For $\eta = 0$ the trace rule (B.33) becomes the standard Hebb rule (B.31), while for $\eta > 0$ learning rule (B.33) operates to associate together patterns of activity that occur close together in time. The rule might allow temporal associations to influence the synaptic weights that are learned over times in the order of 1 s. The memory trace required for operation of this rule might be no more complicated than the continuing firing that is an inherent property of attractor networks, but it could also be implemented by a number of biophysical mechanisms, discussed in Chapter 4. Finally, we note that some long-term depression (LTD) in the learning rule could help to maintain the weights of different neurons equally potent (see Section B.4.9.2 and equation B.22), and could compensate for irregularity during training in which the agent might be trained much more in some than other locations in the space (see Stringer, Trappenberg, Rolls and De Araujo (2002b)).

B.5.4 The capacity of a continuous attractor network: multiple charts and packets

The capacity of a continuous attractor network can be approached on the following bases. First, as there are no discrete attractor states, but instead a continuous physical space is being represented, some concept of spatial resolution must be brought to bear, that is the number of different positions in the space that can be represented. Second, the number of connections per neuron in the continuous attractor will directly influence the number of different spatial positions (locations in the state space) that can be represented. Third, the sparseness of the representation can be thought of as influencing the number of different spatial locations (in the continuous state space) that can be represented, in a way analogous to that described for discrete attractor networks in equation B.14 (Battaglia and Treves 1998b). That is, if the tuning of the neurons is very broad, then fewer locations in the state space may be represented. Fourth, and very interestingly, if representations of different continuous state spaces, for example maps or charts of different environments, are stored in the same network, there may be little cost of adding extra maps or charts. The reason for this is that the large part of the interference between the different memories stored in such a network arises from the correlations between the different positions in any one map, which are typically relatively high because quite broad tuning of individual cells is common. In contrast, there are in general low correlations between the representations of places in different maps or charts, and therefore many different maps can be simultaneously stored in a continuous attractor network (Battaglia and Treves 1998b).

For a similar reason, it is even possible to have the activity packets that operate in different spaces simultaneously active in a single continuous attractor network of neurons, and to move independently of each other in their respective spaces or charts (Stringer, Rolls and Trappenberg 2004).

B.5.5 Continuous attractor models: path integration

So far, we have considered how spatial representations could be stored in continuous attractor networks, and how the activity can be maintained at any location in the state space in a form of short-term memory when the external (e.g. visual) input is removed. However, many networks with spatial representations in the brain can be updated by internal, self-motion (i.e. idiothetic), cues even when there is no external (e.g. visual) input. Examples are head direction cells in the presubiculum of rats and macaques, place cells in the rat hippocampus, and spatial view cells in the primate hippocampus (see Chapter 2). The major question arises about how such idiothetic inputs could drive the activity packet in a continuous attractor network and, in particular, how such a system could be set up biologically by self-organizing learning.

One approach to simulating the movement of an activity packet produced by idiothetic cues (which is a form of path integration whereby the current location is calculated from recent movements) is to employ a look-up table that stores (taking head direction cells as an example), for every possible head direction and head rotational velocity input generated by the vestibular system, the corresponding new head direction (Samsonovich and McNaughton 1997). Another approach involves modulating the strengths of the recurrent synaptic weights in the continuous attractor on one but not the other side of a currently represented position, so that the stable position of the packet of activity, which requires symmetric connections in different directions from each node, is lost, and the packet moves in the direction of the temporarily increased weights, although no possible biological implementation was proposed of how the appropriate dynamic synaptic weight changes might be achieved (Zhang 1996). Another mechanism (for head direction cells) (Skaggs, Knierim, Kudrimoti and McNaughton 1995) relies on

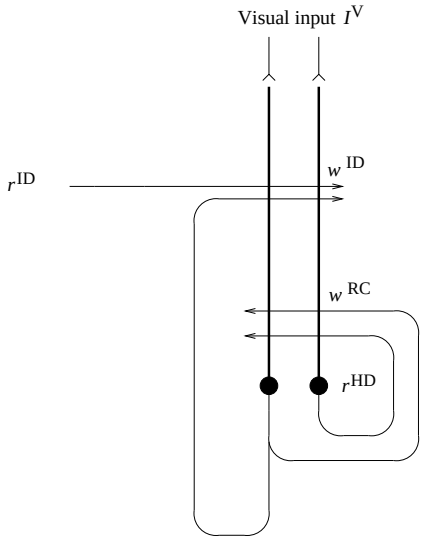


Fig. B.27 General network architecture for a one-dimensional continuous attractor model of head direction cells which can be updated by idiothetic inputs produced by head rotation cell firing r^{ID}. The head direction cell firing is r^{HD}, the continuous attractor synaptic weights are w^{RC}, the idiothetic synaptic weights are w^{ID}, and the external visual input is I^{V}.

a set of cells, termed (head) rotation cells, which are co-activated by head direction cells and vestibular cells and drive the activity of the attractor network by anatomically distinct connections for clockwise and counter-clockwise rotation cells, in what is effectively a look-up table. However, no proposal was made about how this could be achieved by a biologically plausible learning process, and this has been the case until recently for most approaches to path integration in continuous attractor networks, which rely heavily on rather artificial pre-set synaptic connectivities.

Stringer, Trappenberg, Rolls and De Araujo (2002b) introduced a proposal with more biological plausibility about how the synaptic connections from idiothetic inputs to a continuous attractor network can be learned by a self-organizing learning process. The essence of the hypothesis is described with Fig. B.27. The continuous attractor synaptic weights w^{RC} are set up under the influence of the external visual inputs I^{V} as described in Section B.5.3. At the same time, the idiothetic synaptic weights w^{ID} (in which the ID refers to the fact that they are in this case produced by idiothetic inputs, produced by cells that fire to represent the velocity of clockwise and anticlockwise head rotation), are set up by associating the change of head direction cell firing that has just occurred (detected by a trace memory mechanism described below) with the current firing of the head rotation cells r^{ID}. For example, when the trace memory mechanism incorporated into the idiothetic synapses w^{ID} detects that the head direction cell firing is at a given location (indicated by the firing r^{HD}) and is moving clockwise (produced by the altering visual inputs I^{V}), and there is simultaneous clockwise head rotation cell firing, the synapses w^{ID} learn the association, so that when that rotation cell firing occurs later without visual input, it takes the current head direction firing in the continuous attractor into account, and moves the location of the head direction attractor in the appropriate direction.

For the learning to operate, the idiothetic synapses onto head direction cell i with firing r_i^{HD} need two inputs: the memory traced term from other head direction cells $\overline{r}_j^{\mathrm{HD}}$ (given by equation B.34), and the head rotation cell input with firing r_k^{ID}; and the learning rule can be

written

$$\delta w_{ijk}^{\mathrm{ID}} = \tilde{k}\, r_i^{\mathrm{HD}}\, \overline{r}_j^{\mathrm{HD}}\, r_k^{\mathrm{ID}}, \tag{B.35}$$

where $\tilde{k}$ is the learning rate associated with this type of synaptic connection. The head rotation cell firing (r_k^{ID}) could be as simple as one set of cells that fire for clockwise head rotation (for which k might be 1), and a second set of cells that fire for anticlockwise head rotation (for which k might be 2).

After learning, the firing of the head direction cells would be updated in the dark (when $I_i^V = 0$) by idiothetic head rotation cell firing r_k^{ID} as follows

$$\begin{aligned}\tau \frac{\mathrm{d}h_i^{\mathrm{HD}}(t)}{\mathrm{d}t} &= -h_i^{\mathrm{HD}}(t) + \frac{\phi_0}{C^{\mathrm{HD}}} \sum_j (w_{ij} - w^{inh}) r_j^{\mathrm{HD}}(t) + I_i^V \\ &+ \phi_1 \left(\frac{1}{C^{\mathrm{HD}\times\mathrm{ID}}} \sum_{j,k} w_{ijk}^{\mathrm{ID}}\, r_j^{\mathrm{HD}}\, r_k^{\mathrm{ID}}\right).\end{aligned} \tag{B.36}$$

Equation B.36 is similar to equation B.29, except for the last term, which introduces the effects of the idiothetic synaptic weights w_{ijk}^{ID}, which effectively specify that the current firing of head direction cell i, r_i^{HD}, must be updated by the previously learned combination of the particular head rotation now occurring indicated by r_k^{ID}, and the current head direction indicated by the firings of the other head direction cells r_j^{HD} indexed through j[41]. This makes it clear that the idiothetic synapses operate using combinations of inputs, in this case of two inputs. Neurons that sum the effects of such local products are termed Sigma-Pi neurons (see Section A.2.3). Although such synapses are more complicated than the two-term synapses used throughout the rest of this book, such three-term synapses appear to be useful to solve the computational problem of updating representations based on idiothetic inputs in the way described. Synapses that operate according to Sigma-Pi rules might be implemented in the brain by a number of mechanisms described by Koch (1999) (Section 21.1.1), Jonas and Kaczmarek (1999), and Stringer, Trappenberg, Rolls and De Araujo (2002b), including having two inputs close together on a thin dendrite, so that local synaptic interactions would be emphasized.

Simulations demonstrating the operation of this self-organizing learning to produce movement of the location being represented in a continuous attractor network were described by Stringer, Trappenberg, Rolls and De Araujo (2002b), and one example of the operation is shown in Fig. B.28. They also showed that, after training with just one value of the head rotation cell firing, the network showed the desirable property of moving the head direction being represented in the continuous attractor by an amount that was proportional to the value of the head rotation cell firing. Stringer, Trappenberg, Rolls and De Araujo (2002b) also describe a related model of the idiothetic cell update of the location represented in a continuous attractor, in which the rotation cell firing directly modulates in a multiplicative way the strength of the recurrent connections in the continuous attractor in such a way that clockwise rotation cells modulate the strength of the synaptic connections in the clockwise direction in the continuous attractor, and vice versa.

It should be emphasized that although the cells are organized in Fig. B.28 according to the spatial position being represented, there is no need for cells in continuous attractors that represent nearby locations in the state space to be close together, as the distance in the state space between any two neurons is represented by the strength of the connection between them,

[41] The term $\phi_1 / C^{\mathrm{HD}\times\mathrm{ID}}$ is a scaling factor that reflects the number $C^{\mathrm{HD}\times\mathrm{ID}}$ of inputs to these synapses, and enables the overall magnitude of the idiothetic input to each head direction cell to remain approximately the same as the number of idiothetic connections received by each head direction cell is varied.

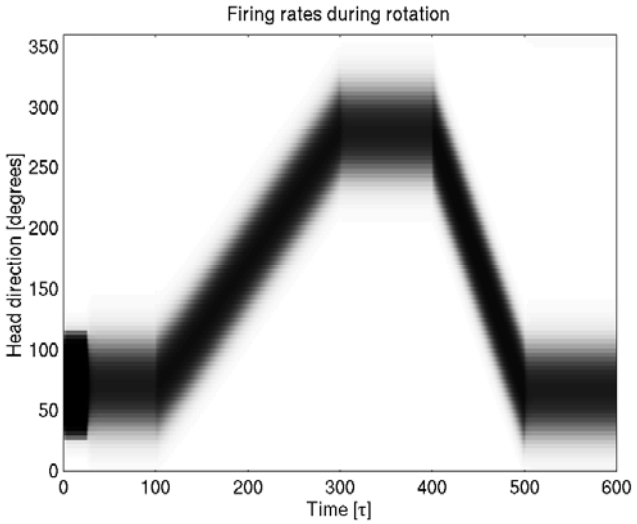


Fig. B.28 Idiothetic update of the location represented in a continuous attractor network. The firing rate of the cells with optima at different head directions (organized according to head direction on the ordinate) is shown by the blackness of the plot, as a function of time. The activity packet was initialized to a head direction of 75 degrees, and the packet was allowed to settle without visual input. For timestep=0 to 100 there was no rotation cell input, and the activity packet in the continuous attractor remained stable at 75 degrees. For timestep=100 to 300 the clockwise rotation cells were active with a firing rate of 0.15 to represent a moderate angular velocity, and the activity packet moved clockwise. For timestep=300 to 400 there was no rotation cell firing, and the activity packet immediately stopped, and remained still. For timestep=400 to 500 the anti-clockwise rotation cells had a high firing rate of 0.3 to represent a high velocity, and the activity packet moved anticlockwise with a greater velocity. For timestep=500 to 600 there was no rotation cell firing, and the activity packet immediately stopped.

not by where the neurons are physically located. This enables continuous attractor networks to represent spaces with arbitrary topologies, as the topology is represented in the connection strengths (Stringer, Trappenberg, Rolls and De Araujo 2002b, Stringer, Rolls, Trappenberg and De Araujo 2002a, Stringer, Rolls and Trappenberg 2005, Stringer and Rolls 2002). Indeed, it is this that enables many different charts each with its own topology to be represented in a single continuous attractor network (Battaglia and Treves 1998b).

In the network described so far, self-organization occurs, but one set of synapses is Sigma-Pi. We have gone on to show that the Sigma-Pi synapses are not necessary, and can be replaced by a competitive network that learns to respond to combinations of the spatial position and the idiothetic velocity, as illustrated in Fig. B.29 (Stringer and Rolls 2006).

B.5.6 Stabilization of the activity packet within the continuous attractor network when the agent is stationary

With irregular learning conditions (in which identical training with high precision of every node cannot be guaranteed), the recurrent synaptic weights between nodes in the continuous attractor will not be of the perfectly regular and symmetric form normally required in a continuous attractor neural network. This can lead to drift of the activity packet within the continuous attractor network of e.g. head direction cells when no visual cues are present, even when the agent is not moving. This drift is a common property of the short-term memories of spatial position implemented in the brain, which emphasizes the computational problems

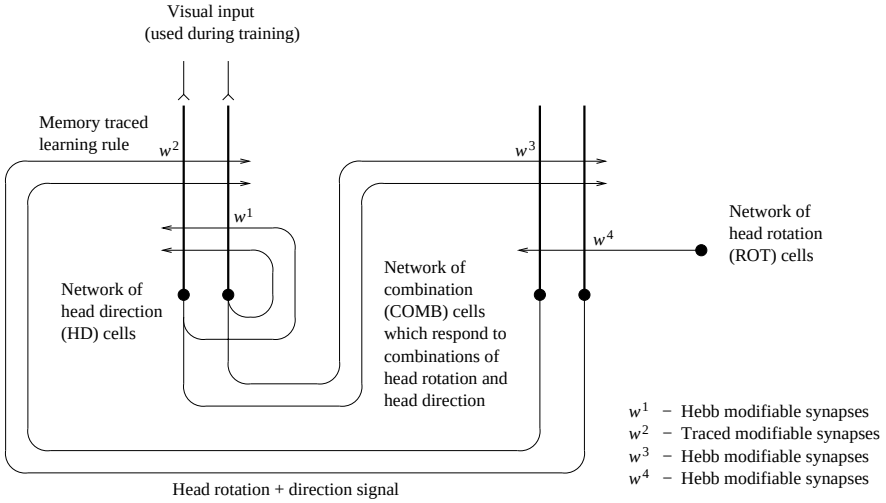


Fig. B.29 Network architecture for a two-layer self-organizing neural network model of the head direction system. The network architecture contains a layer of head direction (HD) cells representing the head direction of the agent, a layer of combination (COMB) cells representing a combination of head direction and rotational velocity, and a layer of rotation (ROT) cells which become active when the agent rotates. There are four types of synaptic connection in the network, which operate as follows. The w^1_{ij} synapses are Hebb-modifiable recurrent connections between head direction cells. These connections help to support stable packets of activity within this continuous attractor layer of head direction cells in the absence of visual input. The combination cells receive inputs from the head direction cells through the Hebb-modifiable w^3_{ij} synapses, and inputs from the rotation cells through the Hebb-modifiable w^4_{ij} synapses. These synaptic inputs encourage combination cells by competitive learning to respond to particular combinations of head direction and rotational velocity. In particular, the combination cells only become significantly active when the agent is rotating. The head direction cells receive inputs from the combination cells through the w^2_{ij} synapses. The $w^2_{i,j}$ synapses are trained using a 'trace' learning rule, which incorporates a temporal trace of recent combination cell activity. This rule introduces an asymmetry into the w^2_{ij} weights, which plays an important role in shifting the activity packet through the layer of head direction cells during path integration in the dark. (After Stringer and Rolls 2006.)

that can arise in continuous attractor networks if the weights between nodes are not balanced in different directions in the space being represented. An approach to stabilizing the activity packet when it should not be drifting in real nervous systems, which does help to minimize the drift that can occur, is now described.

The activity packet may be stabilized within a continuous attractor network, when the agent is stationary and the network is operating in memory mode without an external stabilizing input, by enhancing the firing of those cells that are already firing. In biological systems this might be achieved through mechanisms for short-term synaptic enhancement (Koch 1999). Another way is to take advantage of the non-linearity of the activation function of neurons with NMDA receptors, which only contribute to neuronal firing once the neuron is sufficiently depolarized (Wang 1999). The effect is to enhance the firing of neurons that are already reasonably well activated. The effect has been utilized in a model of a network with recurrent excitatory synapses that can maintain active an arbitrary set of neurons that are initially sufficiently strongly activated by an external stimulus (see Lisman, Fellous and Wang (1998), and, for a discussion on whether these networks could be used to implement short-term memories, see Kesner and Rolls (2001)).

We have incorporated this non-linearity into a model of a head direction continuous

attractor network by adjusting the sigmoid threshold α_i (see equation B.30) for each head direction cell i as follows (Stringer, Trappenberg, Rolls and De Araujo 2002b). If the head direction cell firing rate r_i^{HD} is lower than a threshold value, γ, then the sigmoid threshold α_i is set to a relatively high value α^{high}. Otherwise, if the head direction cell firing rate r_i^{HD} is greater than or equal to the threshold value, γ, then the sigmoid threshold α_i is set to a relatively low value α^{low}. It was shown that this procedure has the effect of enhancing the current position of the activity packet within the continuous attractor network, and so prevents the activity packet drifting erratically due to the noise in the recurrent synaptic weights produced for example by irregular learning. An advantage of using the non-linearity in the activation function of a neuron (produced for example by the operation of NMDA receptors) is that this tends to enable packets of activity to be kept active without drift even when the packet is not in one of the energy minima that can result from irregular learning (or from diluted connectivity in the continuous attractor as described below). Thus use of this non-linearity increases the number of locations in the continuous physical state space at which a stable activity packet can be maintained (Stringer, Trappenberg, Rolls and De Araujo 2002b).

The same process might help to stabilize the activity packet against drift caused by the probabilistic spiking of neurons in the network (cf. Chapter 7).

B.5.7 Continuous attractor networks in two or more dimensions

Some types of spatial representation used by the brain are of spaces that exist in two or more dimensions. Examples are the two- (or three-) dimensional space representing where one is looking at in a spatial scene. Another is the two- (or three-) dimensional space representing where one is located. It is possible to extend continuous attractor networks to operate in higher dimensional spaces than the one-dimensional spaces considered so far (Taylor 1999, Stringer, Rolls, Trappenberg and De Araujo 2002a). Indeed, it is also possible to extend the analyses of how idiothetic inputs could be used to update two-dimensional state spaces, such as the locations represented by place cells in rats (Stringer, Rolls, Trappenberg and De Araujo 2002a) and the location at which one is looking represented by primate spatial view cells (Stringer, Rolls and Trappenberg 2005, Stringer and Rolls 2002). Interestingly, the number of terms in the synapses implementing idiothetic update do not need to increase beyond three (as in Sigma-Pi synapses) even when higher dimensional state spaces are being considered (Stringer, Rolls, Trappenberg and De Araujo 2002a). Also interestingly, a continuous attractor network can in fact represent the properties of very high dimensional spaces, because the properties of the spaces are captured by the connections between the neurons of the continuous attractor, and these connections are of course, as in the world of discrete attractor networks, capable of representing high dimensional spaces (Stringer, Rolls, Trappenberg and De Araujo 2002a). With these approaches, continuous attractor networks have been developed of the two-dimensional representation of rat hippocampal place cells with idiothetic update by movements in the environment (Stringer, Rolls, Trappenberg and De Araujo 2002a), and of primate hippocampal spatial view cells with idiothetic update by eye and head movements (Stringer, Rolls and Trappenberg 2005, Rolls and Stringer 2005, Stringer and Rolls 2002). Continuous attractor models with some similar properties have also been applied to understanding motor control, for example the generation of a continuous movement in space (Stringer and Rolls 2007a, Stringer, Rolls and Taylor 2007a).

B.5.8 Mixed continuous and discrete attractor networks

It has now been shown that attractor networks can store both continuous patterns and discrete patterns, and can thus be used to store for example the location in (continuous, physical) space

where an object (a discrete item) is present (see Fig. 2.16 and Rolls, Stringer and Trappenberg (2002)). In this network, when events are stored that have both discrete (object) and continuous (spatial) aspects, then the whole place can be retrieved later by the object, and the object can be retrieved by using the place as a retrieval cue. Such networks are likely to be present in parts of the brain that receive and combine inputs both from systems that contain representations of continuous (physical) space, and from brain systems that contain representations of discrete objects, such as the inferior temporal visual cortex. One such brain system is the hippocampus, which appears to combine and store such representations in a mixed attractor network in the CA3 region, which thus is able to implement episodic memories which typically have a spatial component, for example where an item such as a key is located (see Chapter 2). This network thus shows that in brain regions where the spatial and object processing streams are brought together, then a single network can represent and learn associations between both types of input. Indeed, in brain regions such as the hippocampal system, it is essential that the spatial and object processing streams are brought together in a single network, for it is only when both types of information are in the same network that spatial information can be retrieved from object information, and vice versa, which is a fundamental property of episodic memory (see Chapter 2). It may also be the case that in the prefrontal cortex, attractor networks can store both spatial and discrete (e.g. object-based) types of information in short-term memory (see Section 5.1).

B.6 Network dynamics: the integrate-and-fire approach

The concept that attractor (autoassociation) networks can operate very rapidly if implemented with neurons that operate dynamically in continuous time was introduced in Section B.3.3.5. The result described was that the principal factor affecting the speed of retrieval is the time constant of the synapses between the neurons that form the attractor ((Treves 1993, Rolls and Treves 1998, Battaglia and Treves 1998a, Panzeri, Rolls, Battaglia and Lavis 2001). This was shown analytically by Treves (1993), and described by Rolls and Treves (1998) Appendix 5. We now describe in more detail the approaches that produce these results, and the actual results found on the speed of processing.

The networks described so far in this chapter, and analyzed in Appendices 3 and 4 of Rolls and Treves (1998), were described in terms of the steady-state activation of networks of neuron-like units. Those may be referred to as 'static' properties, in the sense that they do not involve the time dimension. In order to address 'dynamical' questions, the time dimension has to be reintroduced into the formal models used, and the adequacy of the models themselves has to be reconsidered in view of the specific properties to be discussed.

Consider for example a real network whose operation has been described by an autoassociative formal model that acquires, with learning, a given attractor structure. How does the state of the network approach, in real time during a retrieval operation, one of those attractors? How long does it take? How does the amount of information that can be read off the network's activity evolve with time? Also, which of the potential steady states is indeed a stable state that can be reached asymptotically by the net? How is the stability of different states modulated by external agents? These are examples of dynamical properties, which to be studied require the use of models endowed with some dynamics.

B.6.1 From discrete to continuous time

Already at the level of simple models in which each unit is described by an input–output relation, one may introduce equally simple 'dynamical' rules, in order both to fully specify

the model, and to simulate it on computers. These rules are generally formulated in terms of 'updatings': time is considered to be discrete, a succession of time steps, and at each time step the output of one or more of the units is set, or updated, to the value corresponding to its input variable. The input variable may reflect the outputs of other units in the net as updated at the previous time step or, if delays are considered, the outputs as they were at a prescribed number of time steps in the past. If all units in the net are updated together, the dynamics is referred to as parallel; if instead only one unit is updated at each time step, the dynamics is sequential. (One main difference between the Hopfield (1982) model of an autoassociator and a similar model considered earlier by Little (1974) is that the latter was based on parallel rather than sequential dynamics.) Many intermediate possibilities obviously exist, involving the updating of groups of units at a time. The order in which sequential updatings are performed may for instance be chosen at random at the beginning and then left the same in successive cycles across all units in the net; or it may be chosen anew at each cycle; yet a third alternative is to select at each time step a unit, at random, with the possibility that a particular unit may be selected several times before some of the other ones are ever updated. The updating may also be made probabilistic, with the output being set to its new value only with a certain probability, and otherwise remaining at the current value.

Variants of these dynamical rules have been used for decades in the analysis and computer simulation of physical systems in statistical mechanics (and field theory). They can reproduce in simple but effective ways the stochastic nature of transitions among discrete quantum states, and they have been subsequently considered appropriate also in the simulation of neural network models in which units have outputs that take discrete values, implying that a change from one value to another can only occur in a sudden jump. To some extent, different rules are equivalent, in that they lead, in the evolution of the activity of the net along successive steps and cycles, to the same set of possible steady states. For example, it is easy to realize that when no delays are introduced, states that are stable under parallel updating are also stable under sequential updating. The reverse is not necessarily true, but on the other hand states that are stable when updating one unit at a time are stable irrespective of the updating order. Therefore, static properties, which can be deduced from an analysis of stable states, are to some extent robust against differences in the details of the dynamics assigned to the model. (This is a reason for using these dynamical rules in the study of the thermodynamics of physical systems.) Such rules, however, bear no relation to the actual dynamical processes by which the activity of real neurons evolves in time, and are therefore inadequate for the discussion of dynamical issues in neural networks.

A first step towards realism in the dynamics is the substitution of discrete time with continuous time. This somewhat parallels the substitution of the discrete output variables of the most rudimentary models with continuous variables representing firing rates. Although continuous output variables may evolve also in discrete time, and as far as static properties are concerned differences are minimal, with the move from discrete to continuous outputs the main raison d'etre for a dynamics in terms of sudden updatings ceases to exist, since continuous variables can change continuously in continuous time. A paradox arises immediately, however, if a continuous time dynamics is assigned to firing rates. The paradox is that firing rates, although in principle continuous if computed with a generic time-kernel, tend to vary in jumps as new spikes – essentially discrete events – come to be included in the kernel. To avoid this paradox, a continuous time dynamics can be assigned, instead, to instantaneous continuous variables such as membrane potentials. Hopfield (1984), among others, has introduced a model of an autoassociator in which the output variables represent membrane potentials and evolve continuously in time, and has suggested that under certain conditions the stable states attainable by such a network are essentially the same as for a network of binary units evolving in discrete time. If neurons in the central nervous system communicated with each

other via the transmission of graded membrane potentials, as they do in some peripheral and invertebrate neural systems, this model could be an excellent starting point. The fact that, centrally, transmission is primarily via the emission of discrete spikes makes a model based on membrane potentials as output variables inadequate to correctly represent spiking dynamics.

B.6.2 Continuous dynamics with discontinuities

In principle, a solution would be to keep the membrane potential as the basic dynamical variable, evolving in continuous time, and to use as the output variable the spike emission times, as determined by the rapid variation in membrane potential corresponding to each spike. A point-like neuron can generate spikes by altering the membrane potential V according to continuous equations of the Hodgkin–Huxley type:

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = g_0(V_{\text{rest}} - V) + g_{\text{Na}}mh^3(V_{\text{Na}} - V) + g_{\text{K}}n^4(V_{\text{K}} - V) + I \tag{B.37}$$

$$\tau_m\frac{\mathrm{d}m}{\mathrm{d}t} = m_\infty(V) - m \tag{B.38}$$

$$\tau_h\frac{\mathrm{d}h}{\mathrm{d}t} = h_\infty(V) - h \tag{B.39}$$

$$\tau_n\frac{\mathrm{d}n}{\mathrm{d}t} = n_\infty(V) - n. \tag{B.40}$$

The changes in the membrane potential, driven by the input current I, interact with the opening and closing of intrinsic conductances (here a sodium conductance, whose channels are gated by the 'particles' m and h, and a potassium conductance, whose channels are gated by n; Hodgkin and Huxley (1952)). These equations provide an effective description, phenomenological but broadly based on physical principles, of the conductance changes underlying action potentials, and they are treated in any standard neurobiology text.

From the point of view of formal models of neural networks, this level of description is too complicated to be the basis for an analytic understanding of the operation of networks, and it must be simplified. The most widely used simplification is the so-called integrate-and-fire model (see for example MacGregor (1987) and Brunel and Wang (2001)), which is legitimized by the observation that (sodium) action potentials are typically brief and self-similar events. If, in particular, the only relevant variable associated with the spike is its time of emission (at the soma, or axon hillock), which essentially coincides with the time the potential V reaches a certain threshold level V_{thr}, then the conductance changes underlying the rest of the spike can be omitted from the description, and substituted with the ad hoc prescription that (i) a spike is emitted, with its effect on receiving units and on the unit itself; and (ii) after a brief time corresponding to the duration of the spike plus a refractory period, the membrane potential is reset and resumes its integration of the input current I. After a spike the membrane potential is taken to be reset to a value V_{reset}. This type of simplified dynamics of the membrane potentials is thus in continuous time with added discontinuities: continuous in between spikes, with discontinuities occurring at different times for each neuron in a population, every time a neuron emits a spike.

A leaky integrate-and-fire neuron can be modelled as follows. The model describes the depolarization of the membrane potential V (which typically is dynamically changing as a result of the synaptic effects described below between approximately –70 and –50 mV) until threshold V_{thr} (typically –50 mV) is reached when a spike is emitted and the potential is reset to V_{reset} (typically –55 mV). The membrane time constant τ_{m} is set by the membrane capacitance C_{m} and the membrane leakage conductance g_{m} where $\tau_{\text{m}} = C_{\text{m}}/g_{\text{m}}$. V_{L} denotes

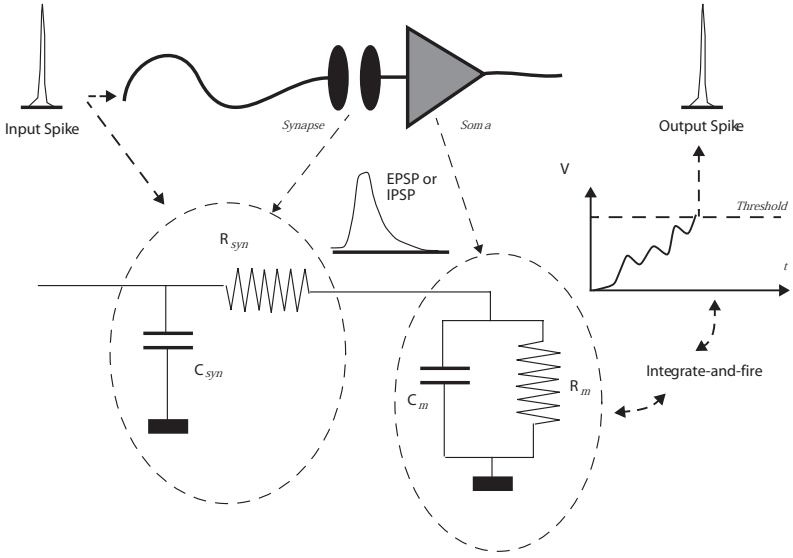


Fig. B.30 Integrate-and-fire neuron. The basic circuit of an integrate-and-fire model consists of the neuron's membrane capacitance C_m in parallel with the membrane's resistance R_m (the reciprocal of the membrane conductance $g_{\rm m}$) driven by a synaptic current with a conductance and time constant determined by the synaptic resistance $R_{\rm syn}$ (the reciprocal of the synaptic conductance g_j) and capacitance $C_{\rm syn}$ shown in the Figure. These effects produce excitatory or inhibitory post-synaptic potentials, EPSPs or IPSPs. These potentials are integrated by the cell, and if a threshold $V_{\rm thr}$ is reached a δ-pulse (spike) is fired and transmitted to other neurons, and the membrane potential is reset. (After Deco and Rolls 2003.)

the resting potential of the cell, typically –70 mV. Changes in the membrane potential are defined by the following equation

$$C_{\rm m}\frac{{\rm d}V(t)}{{\rm d}t} = g_{\rm m}\left(V_{\rm L} - V\left(t\right)\right) + \sum_j g_j\left(V_{\rm AMPA} - V\left(t\right)\right) + \sum_j g_j\left(V_{\rm GABA} - V\left(t\right)\right). \quad \text{(B.41)}$$

The first term on the right of the equation describes how the membrane potential decays back towards the resting potential of the cell depending on how far the cell potential V is from the resting potential $V_{\rm L}$, and the membrane leak conductance $g_{\rm m}$. The second term on the right represents the excitatory synaptic current that could be injected through AMPA receptors. This is a sum over all AMPA synapses indexed by j on the cell. At each synapse, the current is driven into the cell by the difference between the membrane potential V and the reversal potential $V_{\rm AMPA}$ of the channels opened by AMPA receptors, weighted by the synaptic conductance g_j. This synaptic conductance changes dynamically as a function of the time since a spike reached the synapse, as shown below. Due to their reversal potential $V_{\rm AMPA}$ of typically 0 mV, these currents will tend to depolarize the cell, that is to move the membrane potential towards the firing threshold. The third term on the right represents the inhibitory synaptic current that could be injected through GABA receptors. Due to their reversal potential $V_{\rm GABA}$ of typically –70 mV, these currents will tend to hyperpolarize the cell, that is to move the membrane potential away from the firing threshold. There may be other types of receptor on a cell, for example NMDA receptors, and these operate analogously though with interesting differences, as described in Section B.6.3.

The opening of each synaptic conductance g_j is driven by the arrival of spikes at the presynaptic terminal j, and its closing can often be described as a simple exponential process. A simplified equation for the dynamics of $g_j(t)$ is then

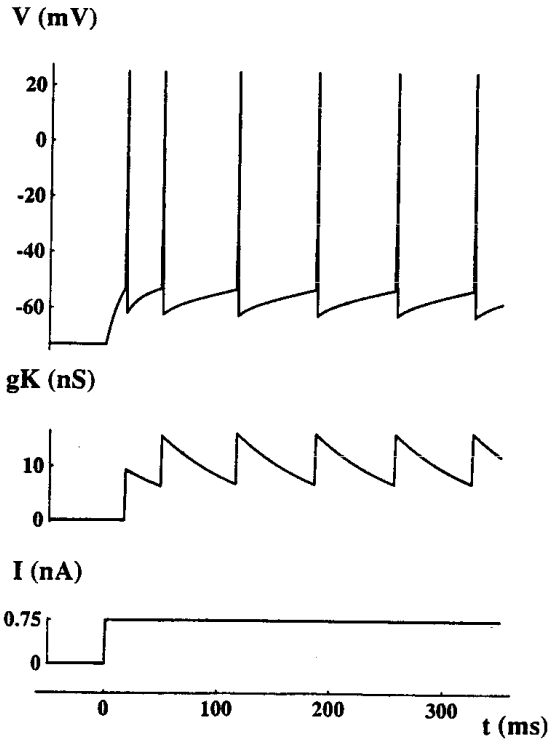


Fig. B.31 Model behaviour of an integrate-and-fire neuron: the membrane potential and adaptation-producing potassium conductance in response to a step of injected current. The spikes were added to the graph by hand, as they do not emerge from the simplified voltage equation. (From Treves 1993.)

$$\frac{\mathrm{d}g_j(t)}{\mathrm{d}t} = -\frac{g_j(t)}{\tau} + \Delta g_j \sum_l \delta(t - \Delta t - t_l). \tag{B.42}$$

According to the above equation, each synaptic conductance opens instantaneously by a fixed amount Δg_j a time Δt after the emission of the presynaptic spike at t_l. Δt summarizes delays (axonal, synaptic, and dendritic), and each opening superimposes linearly, without saturating, on previous openings. The value of τ in this equation will be different for AMPA receptors (typically 2–10 ms), NMDA receptors (typically 100 ms), and GABA receptors (typically 10 ms).

In order to model the phenomenon of adaptation in the firing rate, prominent especially with pyramidal cells, it is possible to include a time-varying intrinsic (potassium-like) conductance in the cell membrane (Brown, Gähwiler, Griffith and Halliwell 1990a) (shown as g_{K} in equations B.43 and B.44). This can be done by specifying that this conductance, which if open tends to shunt the membrane and thus to prevent firing, opens by a fixed amount with the potential excursion associated with each spike, and then relaxes exponentially to its closed state. In this manner sustained firing driven by a constant input current occurs at lower rates after the first few spikes, in a way similar, if the relevant parameters are set appropriately, to the behaviour observed in vitro of many pyramidal cells (for example, Lanthorn, Storn and Andersen (1984), Mason and Larkman (1990)).

The equations for the dynamics of each neuron with adaptation are then

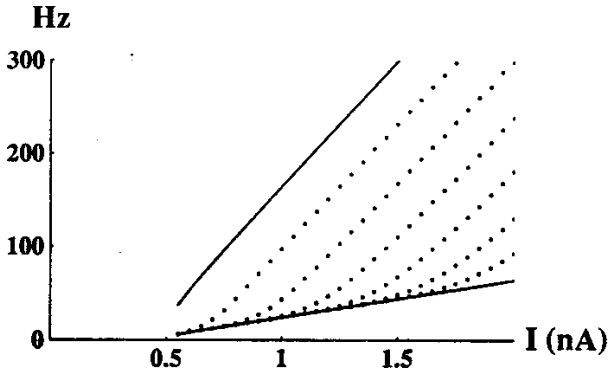


Fig. B.32 Current-to-frequency transduction in a pyramidal cell modelled as an integrate-and-fire neuron. The top solid curve is the firing frequency in the absence of adaptation, $\Delta g_{\mathrm{K}} = 0$. The dotted curves are the instantaneous frequencies computed as the inverse of the ith interspike interval (top to bottom, $i = 1, ..., 6$). The bottom solid curve is the adapted firing curve ($i \to \infty$). With or without adaptation, the input–output transform is close to threshold–linear. (From Treves 1993.)

$$C_{\mathrm{m}}\frac{\mathrm{d}V(t)}{\mathrm{d}t} = g_{\mathrm{m}}\left(V_{\mathrm{L}} - V(t)\right) + \sum_j g_j\left(V_{\mathrm{AMPA}} - V(t)\right) + \sum_j g_j\left(V_{\mathrm{GABA}} - V(t)\right) + g_{\mathrm{K}}(t)\left(V_{\mathrm{K}} - V(t)\right) \tag{B.43}$$

and

$$\frac{\mathrm{d}g_{\mathrm{K}}(t)}{\mathrm{d}t} = -\frac{g_{\mathrm{K}}(t)}{\tau_{\mathrm{K}}} + \sum_k \Delta g_{\mathrm{K}}\delta(t - t_k) \tag{B.44}$$

supplemented by the prescription that when at time $t = t_{k+1}$ the potential reaches the level V_{thr}, a spike is emitted, and hence included also in the sum of equation B.44, and the potential resumes its evolution according to equation B.43 from the reset level V_{reset}. The resulting behaviour is exemplified in Fig. B.31, while Fig. B.32 shows the input–output transform (current to frequency transduction) operated by an integrate-and-fire unit of this type with firing rate adaptation. One should compare this with the transduction operated by real cells, as exemplified for example in Fig. 1.4.

It should be noted that synaptic conductance dynamics is not always included in integrate-and-fire models: sometimes it is substituted with current dynamics, which essentially amounts to neglecting non-linearities due to the appearance of the membrane potential in the driving force for synaptic action (see for example, Amit and Tsodyks (1991), Gerstner (1995)); and sometimes it is simplified altogether by assuming that the membrane potential undergoes small sudden jumps when it receives instantaneous pulses of synaptic current (see the review in Gerstner (1995)). The latter simplification is quite drastic and changes the character of the dynamics markedly; whereas the former can be a reasonable simplification in some circumstances, but it produces serious distortions in the description of inhibitory $\mathrm{GABA_A}$ currents, which, having an equilibrium ($\mathrm{Cl^-}$) synaptic potential close to the operating range of the membrane potential, are quite sensitive to the instantaneous value of the membrane potential itself.

B.6.3 An integrate-and-fire implementation

In this subsection the mathematical equations that describe the spiking activity and synapse dynamics in some of the integrate-and-fire simulations performed by Deco and Rolls (Deco and Rolls 2003, Deco, Rolls and Horwitz 2004, Deco and Rolls 2005d, Deco and Rolls 2005c, Deco and Rolls 2005b, Deco and Rolls 2006) are set out, in order to show in more detail how an integrate-and-fire simulation is implemented. The simulation is that of Deco, Rolls and Horwitz (2004), and follows in general the formulation described by Brunel and Wang (2001).

Each neuron is described by an integrate-and-fire model. The subthreshold membrane potential $V(t)$ of each neuron evolves according to the following equation:

$$C_{\mathrm{m}}\frac{\mathrm{d}V(t)}{\mathrm{d}t} = -g_{\mathrm{m}}(V(t) - V_{\mathrm{L}}) - I_{\mathrm{syn}}(t) \tag{B.45}$$

where $I_{\mathrm{syn}}(t)$ is the total synaptic current flow into the cell, V_{L} is the resting potential, C_{m} is the membrane capacitance, and g_{m} is the membrane conductance. When the membrane potential $V(t)$ reaches the threshold V_{thr} a spike is generated, and the membrane potential is reset to V_{reset}. The neuron is unable to spike during the first τ_{ref} which is the absolute refractory period.

The total synaptic current is given by the sum of glutamatergic excitatory components (NMDA and AMPA) and inhibitory components (GABA). The external excitatory contributions (ext) from outside the network are produced through AMPA receptors ($I_{\mathrm{AMPA,ext}}$), while the excitatory recurrent synapses (rec) within the network act through AMPA and NMDA receptors ($I_{\mathrm{AMPA,rec}}$ and $I_{\mathrm{NMDA,rec}}$). The total synaptic current is therefore given by:

$$I_{\mathrm{syn}}(t) = I_{\mathrm{AMPA,ext}}(t) + I_{\mathrm{AMPA,rec}}(t) + I_{\mathrm{NMDA,rec}}(t) + I_{\mathrm{GABA}}(t) \tag{B.46}$$

where

$$I_{\mathrm{AMPA,ext}}(t) = g_{\mathrm{AMPA,ext}}(V(t) - V_{\mathrm{E}})\sum_{j=1}^{N_{\mathrm{ext}}} s_j^{\mathrm{AMPA,ext}}(t) \tag{B.47}$$

$$I_{\mathrm{AMPA,rec}}(t) = g_{\mathrm{AMPA,rec}}(V(t) - V_{\mathrm{E}})\sum_{j=1}^{N_E} w_j s_j^{\mathrm{AMPA,rec}}(t) \tag{B.48}$$

$$I_{\mathrm{NMDA,rec}}(t) = \frac{g_{\mathrm{NMDA,rec}}(V(t) - V_{\mathrm{E}})}{(1 + [\mathrm{Mg}^{++}]\exp(-0.062V(t))/3.57)}\sum_{j=1}^{N_E} w_j s_j^{\mathrm{NMDA,rec}}(t) \tag{B.49}$$

$$I_{\mathrm{GABA}}(t) = g_{\mathrm{GABA}}(V(t) - V_{\mathrm{I}})\sum_{j=1}^{N_I} s_j^{\mathrm{GABA}}(t) \tag{B.50}$$

In the preceding equations the reversal potential of the excitatory synaptic currents $V_{\mathrm{E}} = 0$ mV and of the inhibitory synaptic currents $V_{\mathrm{I}} = -70$ mV. The different form for the NMDA receptor-activated channels implements the voltage-dependence of NMDA receptors. This voltage-dependency, and the long time constant of the NMDA receptors, are important in the effects produced through NMDA receptors (Brunel and Wang 2001, Wang 1999). The synaptic strengths w_j are specified in the papers by Deco and Rolls (Deco and Rolls 2003, Deco, Rolls and Horwitz 2004, Deco and Rolls 2005d, Deco and Rolls 2005c, Deco and Rolls 2005b, Deco

and Rolls 2006), and depend on the architecture being simulated. The fractions of open channels s are given by:

$$\frac{ds_j^{\mathrm{AMPA,ext}}(t)}{dt} = -\frac{s_j^{\mathrm{AMPA,ext}}(t)}{\tau_{\mathrm{AMPA}}} + \sum_k \delta(t - t_j^k) \tag{B.51}$$

$$\frac{ds_j^{\mathrm{AMPA,rec}}(t)}{dt} = -\frac{s_j^{\mathrm{AMPA,rec}}(t)}{\tau_{\mathrm{AMPA}}} + \sum_k \delta(t - t_j^k) \tag{B.52}$$

$$\frac{ds_j^{\mathrm{NMDA,rec}}(t)}{dt} = -\frac{s_j^{\mathrm{NMDA,rec}}(t)}{\tau_{\mathrm{NMDA,decay}}} + \alpha x_j(t)(1 - s_j^{\mathrm{NMDA,rec}}(t)) \tag{B.53}$$

$$\frac{dx_j(t)}{dt} = -\frac{x_j(t)}{\tau_{\mathrm{NMDA,rise}}} + \sum_k \delta(t - t_j^k) \tag{B.54}$$

$$\frac{ds_j^{\mathrm{GABA}}(t)}{dt} = -\frac{s_j^{\mathrm{GABA}}(t)}{\tau_{\mathrm{GABA}}} + \sum_k \delta(t - t_j^k) \tag{B.55}$$

where the sums over k represent a sum over spikes emitted by presynaptic neuron j at time t_j^k. The value of α = 0.5 ms^{-1}.

Typical values of the conductances for pyramidal neurons are: $g_{\mathrm{AMPA,ext}}$=2.08, $g_{\mathrm{AMPA,rec}}$=0.052, $g_{\mathrm{NMDA,rec}}$=0.164, and g_{GABA}=0.67 nS; and for interneurons: $g_{\mathrm{AMPA,ext}}$=1.62, $g_{\mathrm{AMPA,rec}}$=0.0405, $g_{\mathrm{NMDA,rec}}$=0.129 and g_{GABA}=0.49 nS.

Further details of an integrate-and-fire simulation performed to analyse decision-making (Deco and Rolls 2006) are given in Section 7.8, and a full list of the values of the parameters used is given in Section 7.10.

B.6.4 Simulation of fMRI signals: hemodynamic convolution of synaptic activity

The links between neural and synaptic activity, and fMRI measurements, are still not fully understood. The fMRI signal is unfortunately strongly filtered and perturbed by the hemodynamic delay inherent in the blood oxygen level-dependent (BOLD) contrast mechanism (Buxton and Frank 1997). The fMRI signal is only a secondary consequence of neuronal activity, and yields therefore a blurred distortion of the temporal development of the underlying brain processes. Regionally, increased oxidative metabolism causes a transient decrease in oxyhemoglobin and increase in deoxyhemoglobin, as well as an increase in CO_2 and NO. This provokes over several seconds a local dilatation and increased blood flow in the affected regions that leads by overcompensation to a relative decrease in the concentration of deoxyhemoglobin in the venules draining the activated region, and the alteration of deoxyhemoglobin, which is paramagnetic, can be detected by changes in $T2$ or $T2^*$ in the MRI signal as a result of the decreased susceptibility and thus decreased local inhomogeneity which increases the MR intensity value (Glover 1999, Buxton and Frank 1997, Buxton, Wong and Frank 1998).

The functional magnetic resonance neuroimaging (fMRI) BOLD (blood oxygen level-dependent) signal is likely to reflect the total synaptic activity in an area (as ions need to be pumped back across the cell membrane) rather than the spiking neuronal activity (Logothetis et al. 2001), and is spatially and temporally filtered. The filtering reflects the inherent spatial resolution with which the blood flow changes, as well as the resolution of the scanner, and filtering which may be applied for statistical purposes, and the slow temporal response of the

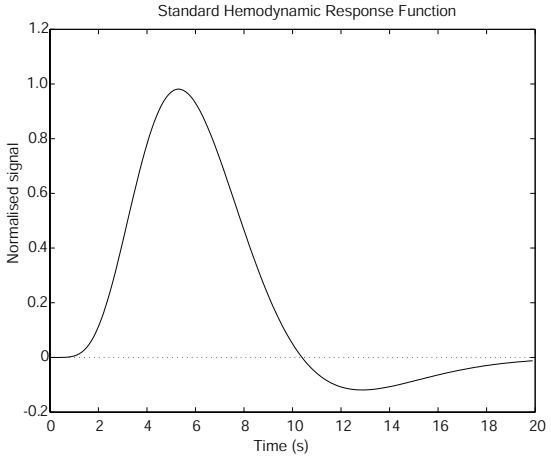


Fig. B.33 The standard hemodynamic response function $h(t)$ (see text).

blood flow changes (Glover 1999, Buxton and Frank 1997, Buxton et al. 1998). Glover (1999) demonstrated that a good fitting of the hemodynamical response $h(t)$ can be achieved by the following analytic function:

$$h(t) = c_1 t^{n_1} e^{-\frac{t}{t_1}} - a_2 c_2 t^{n_2} e^{-\frac{t}{t_2}}$$

$$c_i = \max(t^{n_i} e^{-\frac{t}{t_i}})$$

where t is the time, and c_1, c_2, a_2, n_1, and n_2 are parameters that are adjusted to fit the experimentally measured hemodynamical response. Figure B.33 plots the hemodynamic standard response $h(t)$ for a biologically realistic set of parameters (see Deco, Rolls and Horwitz (2004)).

The temporal evolution of fMRI signals can be simulated from an integrate-and-fire population of neurons by convolving the total synaptic activity in the simulated population of neurons with the standard hemodynamic response formulation of Glover (1999) presented above (Deco, Rolls and Horwitz 2004, Horwitz and Tagamets 1999). The rationale for this is that the major metabolic expenditure in neural activity is the energy required to pump back against the electrochemical gradient the ions that have entered neurons as a result of the ion channels opened by synaptic activity, and that mechanisms have evolved to increase local neuronal blood flow in order to help this increased metabolic demand to be met. (In fact, the increased blood flow overcompensates, and the blood oxygenation level-dependent (BOLD) signal by reflecting the consequent alteration of deoxyhaemoglobin which is paramagnetic reflects this.) The total synaptic current (I_{syn}) is given by the sum of the absolute values of the glutamatergic excitatory components (implemented through NMDA and AMPA receptors) and inhibitory components (GABA) (Tagamets and Horwitz 1998, Horwitz, Tagamets and McIntosh 1999, Rolls and Deco 2002, Deco et al. 2004). In our integrate-and-fire simulations the external excitatory contributions are produced through AMPA receptors ($I_{\mathrm{AMPA,ext}}$), while the excitatory recurrent synaptic currents are produced through AMPA and NMDA receptors ($I_{\mathrm{AMPA,rec}}$ and $I_{\mathrm{NMDA,rec}}$). The GABA inhibitory currents are denoted by I_{GABA}. Consequently, the simulated fMRI signal activity S_{fMRI} is calculated by the following convolution equation:

$$S_{\mathrm{fMRI}}(t) = \int_0^\infty h(t - t') I_{\mathrm{syn}}(t')\, dt'.$$

It is noted here that it could be useful to weight the corresponding currents by the electrochemical gradients against which each ion that passes through the different channels must be pumped back.

Deco, Rolls and Horwitz (2004) applied this approach to predicting fMRI BOLD signals based on activity simulated at the integrate-and-fire level of neuronal activity in the dorsolateral prefrontal cortex. They showed that differences in the fMRI BOLD signal from the dorsal as compared to the ventral prefrontal cortex in working memory tasks may reflect a higher level of inhibition in the dorsolateral prefrontal cortex, as described in Section 5.6. In their simulation the convolution was calculated numerically by sampling the total synaptic activity every 0.1 s and introducing a cut-off at a delay of 25 s. The parameters utilized for the hemodynamic standard response $h(t)$ were taken from the paper of Glover (1999), and were: $n_1 = 6.0$, $t_1 = 0.9$s, $n_2 = 12.0$, $t_2 = 0.9$s, and $a_2 = 0.2$.

B.6.5 The speed of processing of one-layer attractor networks with integrate-and-fire neurons

Given that the analytic approach to the rapidity of the dynamics of attractor networks with integrate-and-fire dynamics (Treves 1993, Rolls and Treves 1998) applies mainly when the state is close to the attractor basin, it is of interest to check the performance of such networks by simulation when the completion of partial patterns that may be towards the edge of the attractor basin can be tested. Simmen, Rolls and Treves (1996a) and Treves, Rolls and Simmen (1997) made a start with this, and showed that retrieval could indeed be fast, within 1–2 time constants of the synapses. However, they found that they could not load the systems they simulated with many patterns, and the firing rates during the retrieval process tended to be unstable. The cause of this turned out to be that the inhibition they used to maintain the activity level during retrieval was subtractive, and it turns out that divisive (shunting) inhibition is much more effective in such networks, as described by Rolls and Treves (1998) in Appendix 5. Divisive inhibition is likely to be organized by inhibitory inputs that synapse close to the cell body (where the reversal potential is close to that of the channels opened by GABA receptors), in contrast to synapses on dendrites, where the different potentials result in opening of the same channels producing hyperpolarization (that is an effectively subtractive influence with respect to the depolarizing currents induced by excitatory (glutamate-releasing) terminals). Battaglia and Treves (1998a) therefore went on to study networks with neurons where the inhibitory neurons could be made to be divisive by having them synapse close to the cell body in neurons modelled with multiple (ten) dendritic compartments. The excitatory inputs terminated on the compartments more distant from the cell body in the model. They found that with this divisive inhibition, the neuronal firing during retrieval was kept under much better control, and the number of patterns that could be successfully stored and retrieved was much higher. Some details of their simulation follow.

Battaglia and Treves (1998a) simulated a network of 800 excitatory and 200 inhibitory cells in its retrieval of one of a number of memory patterns stored in the synaptic weights representing the excitatory-to-excitatory recurrent connections. The memory patterns were assigned at random, drawing the value of each unit in each of the patterns from a binary distribution with sparseness 0.1, that is a probability of 1 in 10 for the unit to be active in the pattern. No baseline excitatory weight was included, but the modifiable weights were instead constrained to remain positive (by clipping at zero synaptic modifications that would make a synaptic weight negative), and a simple exponential decay of the weight with successive modifications was also applied, to prevent runaway synaptic modifications within a rudimentary model of forgetting. Both excitation on inhibitory units and inhibition were mediated by non-modifiable uniform synaptic weights, with values chosen so as to satisfy stability

conditions of the type shown in equations A5.13 of Rolls and Treves (1998). Both inhibitory and excitatory neurons were of the general integrate-and-fire type, but excitatory units had in addition an extended dendritic cable, and they received excitatory inputs only at the more distal end of the cable, and inhibitory inputs spread along the cable. In this way, inhibitory inputs reached the soma of excitatory cells with variable delays, and in any case earlier than synchronous excitatory inputs, and at the same time they could shunt the excitatory inputs, resulting in a largely multiplicative form of inhibition (Abbott 1991). The uniform connectivity was not complete, but rather each type of unit could contact units of the other type with a probability of 0.5, and the same was true for inhibitory-to-inhibitory connections.

After 100 (simulated) ms of activity evoked by external inputs uncorrelated with any of the stored patterns, a cue was provided that consisted of the external input becoming correlated with one of the patterns, at various levels of correlation, for 300 ms. After that, external inputs were removed, but when retrieval operated successfully the activity of the units remained strongly correlated with the memory pattern, or even reached a higher level of correlation if a rather corrupted cue had been used, so that, if during the 300 ms the network had stabilized into a state rather distant from the memory pattern, it got much closer to it once the cue was removed. All correlations were quantified using information measures (see Appendix C)), in terms of mutual information between the firing rate pattern across units and the particular memory pattern being retrieved, or in terms of mutual information between the firing rate of one unit and the set of patterns, or, finally, in terms of mutual information between the decoded firing rates of a subpopulation of 10 excitatory cells and the set of memory patterns. The same algorithms were used to extract information measures as were used for example by Rolls, Treves, Tovee and Panzeri (1997d) with real neuronal data from inferior temporal cortex neurons. The firing rates were measured over sliding windows of 30 ms, after checking that shorter windows produced noisier measures. The effect of using a relatively long window, 30 ms, for measuring rates is an apparent linear early rise in information values with time. Nevertheless, in the real system the activity of these cells is 'read' by other cells receiving inputs from them, and that in turn have their own membrane capacitance-determined characteristic time for integrating input activity, a time broadly in the order of 30 ms. Using such a time window for integrating firing rates did not therefore artificially slow down the read-out process.

It was shown that the time course of different information measures did not depend significantly on the firing rates prevailing during the retrieval state, nor on the resistance–capacitance-determined membrane time constants of the units. Figure B.34 shows that the rise in information after providing the cue at time = 100 ms followed a roughly exponential approach to its steady-state value, which continued until the steady state switched to a new value when the retrieval cue was removed at time = 400 ms. The time constant of the approach to the first steady state was a linear function, as shown in Fig. B.34, of the time constant for excitatory conductances, as predicted by the analysis. (The proportionality factor in the Figure is 2.5, or a collective time constant 2.5 times longer than the synaptic time constant.) The approach to the second steady-state value was more rapid, and the early apparent linear rise prevented the detection of a consistent exponential mode. Therefore, it appears that the cue leads to the basin of attraction of the correct retrieval state by activating transient modes, whose time constant is set by that of excitatory conductances; once the network is in the correct basin, its subsequent reaching the 'very bottom' of the basin after the removal of the cue is not accompanied by any prominent transient mode (see further Battaglia and Treves (1998a)).

Overall, these simulations confirm that recurrent networks, in which excitation is mediated mainly by fast (AMPA, see Section 1.5) channels, can reach asynchronous steady firing states very rapidly, over a few tens of milliseconds, and the rapid approach to steady state is reflected

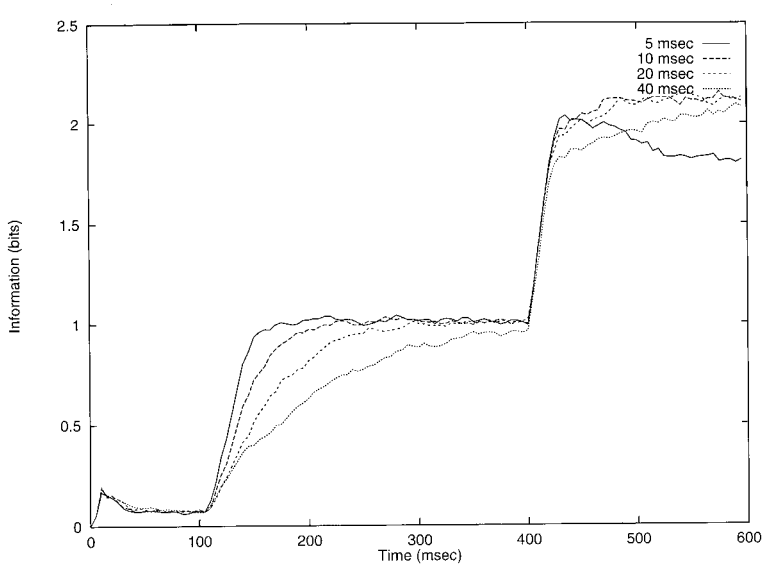


Fig. B.34 Time course of the transinformation about which memory pattern had been selected, as decoded from the firing rates of 10 randomly selected excitatory units. Excitatory conductances closed exponentially with time constants of 5, 10, 20 and 40 ms (curves from top to bottom). A cue of correlation 0.2 with the memory pattern was presented from 100 to 400 ms, uncorrelated external inputs with the same mean strength and sparseness as the cue were applied at earlier times, and no external inputs were applied at later times. (After Battaglia and Treves 1998b.)

in the relatively rapid rise of information quantities that measure the speed of the operation in functional terms.

An analysis based on integrate-and-fire model units thus indicates that recurrent dynamics can be so fast as to be practically indistinguishable from purely feedforward dynamics, in contradiction to what simple intuitive arguments would suggest. This makes it hazardous to draw conclusions on the underlying circuitry on the basis of the experimentally observed speed with which selective neuronal responses arise, as attempted by Thorpe and Imbert (1989). The results also show that networks that implement feedback processing can settle into a global retrieval state very rapidly, and that rapid processing is not just a feature of feedforward networks.

We return to the intuitive understanding of this rapid processing. The way in which networks with continuous dynamics (such as networks made of real neurons in the brain, and networks modelled with integrate-and-fire neurons) can be conceptualized as settling so fast into their attractor states is that spontaneous activity in the network ensures that some neurons are close to their firing threshold when the retrieval cue is presented, so that the firing of these neurons is influenced within 1–2 ms by the retrieval cue. These neurons then influence other neurons within milliseconds (given the point that some other neurons will be close to threshold) through the modified recurrent collateral synapses that store the information. In this way, the neurons in networks with continuous dynamics can influence each other within a fraction of the synaptic time constant, and retrieval can be very rapid.

B.6.6 The speed of processing of a four-layer hierarchical network with integrate-and-fire attractor dynamics in each layer

Given that the visual system has a whole series of cortical areas organized predominantly hierarchically (e.g. V1 to V2 to V4 to inferior temporal cortex), the issue arises of whether the rapid information processing that can be performed for object recognition is predominantly feedforward, or whether there is sufficient time for feedback processing within each cortical area implemented by the local recurrent collaterals to contribute to the visual information processing being performed. Some of the constraints are as follows.

An analysis of response latencies indicates that there is sufficient time for only 10–20 ms per processing stage in the visual system. In the primate cortical ventral visual system the response latency difference between neurons in layer 4Cβ of V1 and inferior temporal cortical cells is approximately 60 ms (Bullier and Nowak 1995, Nowak and Bullier 1997, Schmolesky, Wang, Hanes, Thompson, Leutgeb, Schall and Leventhal 1998). For example, the latency of the responses of neurons in V1 is approximately 30–40 ms (Celebrini, Thorpe, Trotter and Imbert 1993), and in the temporal cortex visual areas approximately 80–110 ms (Baylis, Rolls and Leonard 1987, Sugase, Yamane, Ueno and Kawano 1999). Given that there are 4–6 stages of processing in the ventral visual system from V1 to the anterior inferior temporal cortex, the difference in latencies between each ventral cortical stage is on this basis approximately 10 ms (Rolls 1992a, Oram and Perrett 1994). Information theoretic analyses of the responses of single visual cortical cells in primates reveal that much of the information that can be extracted from neuronal spike trains is often found to be present in periods as short as 20–30 ms (Tovee, Rolls, Treves and Bellis 1993, Tovee and Rolls 1995, Heller, Hertz, Kjaer and Richmond 1995, Rolls, Tovee and Panzeri 1999b). Backward masking experiments indicate that each cortical area needs to fire for only 20–30 ms to pass information to the next stage (Rolls and Tovee 1994, Rolls, Tovee, Purcell, Stewart and Azzopardi 1994b, Kovacs, Vogels and Orban 1995, Rolls, Tovee and Panzeri 1999b, Rolls 2003) (see Section C.3.4). Rapid serial visual presentation of image sequences shows that cells in the temporal visual cortex are still face selective when faces are presented at the rate of 14 ms/image (Keysers, Xiao, Foldiak and Perrett 2001). Finally, event-related potential studies in humans provide strong evidence that the visual system is able to complete some analyses of complex scenes in less than 150 ms (Thorpe, Fize and Marlot 1996).

To investigate whether feedback processing within each layer could contribute to information processing in such a multilayer system in times as short as 10–20 ms per layer, Panzeri, Rolls, Battaglia and Lavis (2001) simulated a four-layer network with attractor networks in each layer. The network architecture is shown schematically in Fig. B.35. All the neurons realized integrate-and-fire dynamics, and indeed the individual layers and neurons were implemented very similarly to the implementation used by Battaglia and Treves (1998a). In particular, the current flowing from each compartment of the multicompartment neurons to the external medium was expressed as:

$$I(t) = g_{\text{leak}}(V(t) - V^0) + \sum_j g_j(t)(V(t) - V_j), \tag{B.56}$$

where g_{leak} is a constant passive leakage conductance, V^0 the membrane resting potential, $g_j(t)$ the value of the jth synapse conductance at time t, and V_j the reversal potential of the jth synapse. $V(t)$ is the potential in the compartment at time t. The most important parameter in the simulation, the AMPA inactivation time constant, was set to 10 ms. The recurrent collateral (RC) integration time constant of the membrane of excitatory cells was 20 ms long for the simulations presented. The synaptic conductances decayed exponentially in time, obeying the equation

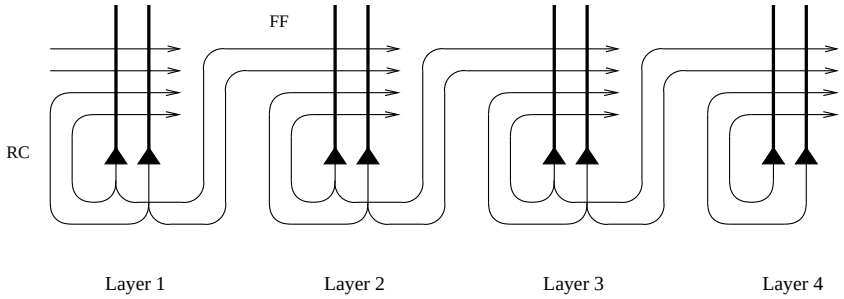


Fig. B.35 The structure of the excitatory connections in the network. There are feedforward (FF) connections between each layer and the next, and excitatory recurrent collaterals (RC) in each layer. Inhibitory connections are also present within each layer, but they are not shown in this Figure. (After Panzeri, Rolls, Battaglia and Lavis 2001.)

$$\frac{\mathrm{d}g_j}{\mathrm{d}t} = -\frac{g_j}{\tau_j} + \Delta g_j \sum_k \delta(t - \Delta t - t_k^j), \tag{B.57}$$

where τ_j is the synaptic decay time constant, Δt is a delay term summarizing axonal and synaptic delays, and Δg_j is the amount that the conductance is increased when the presynaptic unit fires a spike. Δg_j thus represents the (unidirectional) coupling strength between the pre-synaptic and the post-synaptic cell. t_k^j is the time at which the pre-synaptic unit fires its kth spike.

An example of the rapid information processing of the system is shown in Fig. B.36, obtained under conditions in which the local recurrent collaterals can contribute to correct performance because the feedforward (FF) inputs from the previous stage are noisy. (The noise implemented in these simulations was some imperfection in the FF signals produced by some alterations to the FF synaptic weights.) Figure B.36 shows that, when the FF carry an incomplete signal, some information is still transmitted successfully in the 'No RC' condition (in which the recurrent collateral connections in each layer are switched off), and with a relatively short latency. However, the noise term in the FF synaptic strengths makes the retrieval fail more and more layer by layer. When in contrast the recurrent collaterals (RC) are present and operating after Hebbian training, the amount of information retrieved is now much higher, because the RC are able to correct a good part of the erroneous information injected into the neurons by the noisy FF synapses. In Layer 4, 66 ms after cue injection in Layer 1, the information in the Hebbian RC case is 0.2 bits higher than that provided by the FF connections in the 'No RC' condition. This shows that the RC are able to retrieve information in Layer 4 that is not available by any other purely FF mechanism after only roughly 50–55 ms from the time when Layer 1 responds. (This corresponds to 17–18 ms per layer.)

A direct comparison of the latency differences in layers 1–4 of the integrate-and-fire network simulated by Panzeri, Rolls, Battaglia and Lavis (2001) is shown in Fig. B.37. The results are shown for the Hebbian condition illustrated in Fig. B.36, and separate curves are shown for each of the layers 1–4. The Figure shows that, with the time constant of the synapses set to 10 ms, the network can operate with full utilization of and benefit from recurrent processing within each layer in a time which enables the signal to propagate through the 4-layer system with a time course of approximately 17 ms per layer.

The overall results of Panzeri, Rolls, Battaglia and Lavis (2001) were as follows. Through the implementation of continuous dynamics, latency differences were found in information retrieval of only 5 ms per layer when local excitation was absent and processing was purely feedforward. However, information latency differences increased significantly when non-associative local excitation (simulating spontaneous firing or unrelated inputs present in the

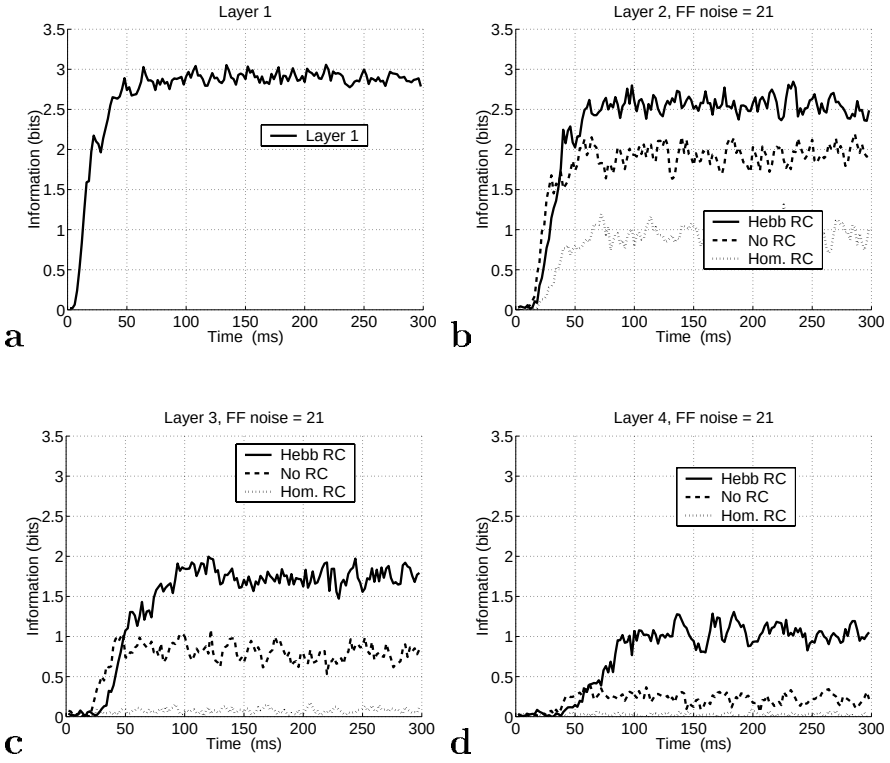


Fig. B.36 The speed of information processing in a 4-layer network with integrate-and-fire neurons. The information time course of the average information carried by the responses of a population of 30 excitatory neurons in each layer. In the simulations considered here, there is noise in the feedforward (FF) synapses. Layer 1 was tested in just one condition. Layers 2–4 are tested in three different conditions: No RC (in which the recurrent collateral synaptic effects do not operate), Hebbian RC (in which the recurrent collaterals have been trained by as associative rule and can help pattern retrieval in each layer), and a control condition named Homogeneous RC (in which the recurrent collaterals could inject current into the neurons, but no useful information was provided by them because they were all set to the same strength). (After Panzeri, Rolls, Battaglia and Lavis 2001.)

brain) was included. It was also found that local recurrent excitation through associatively modified synapses can contribute significantly to processing in as little as 15 ms per layer, including the feedforward and local feedback processing. Moreover, and in contrast to purely feed-forward processing, the contribution of local recurrent feedback was useful and approximately this rapid even when retrieval was made difficult by noise. These findings provide evidence that cortical information processing can be very fast even when local recurrent circuits are critically involved. The time cost of this recurrent processing is minimal when compared with a feedforward system with spontaneous firing or unrelated inputs already present, and the performance is better than that of a purely feedforward system when noise is present.

It is concluded that local feedback loops within each cortical area can contribute to fast visual processing and cognition.

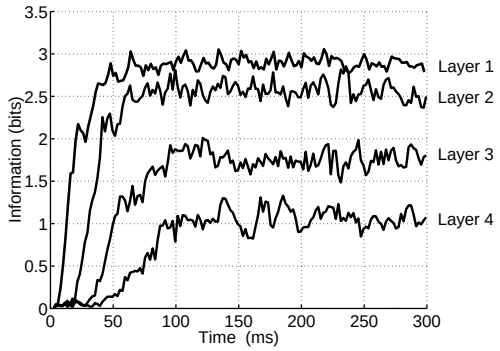


Fig. B.37 The speed of information processing in a 4-layer network with integrate-and-fire neurons. The information time course of the average information carried by the responses of a population of 30 excitatory neurons in each layer. The results are shown for the Hebbian condition illustrated in Fig. B.36, and separate curves are shown for each of the layers 1–4. The Figure shows that, with the time constant of the synapses set to 10 ms, the network can operate with full utilization of and benefit from recurrent processing within each layer in a time in the order of 17 ms per layer. (After Panzeri, Rolls, Battaglia and Lavis 2001.)

B.6.7 Spike response model

In this section, we describe another mathematical model that models the activity of single spiking neurons. This model captures the principal effects of real neurons in a realistic way and is simple enough to permit analytical calculations (Gerstner, Ritz and Van Hemmen 1993). In contrast to some integrate-and-fire models (Tuckwell 1988), which are essentially given by differential equations, the spike-response model is based on response kernels that describe the integrated effect of spike reception or emission on the membrane potential. In this model, spikes are generated by a threshold process (i.e. the firing time t' is given by the condition that the membrane potential reaches the firing threshold θ; that is, $h(t') = \theta$). Figure B.38 (bottom) shows schematically the spike-generating mechanism.

The membrane potential is given by the integration of the input signal weighted by a kernel defined by the equations

$$h(t') = h^{\text{refr}}(t') + h^{\text{syn}}(t') \tag{B.58}$$

$$h^{\text{refr}}(t') = \int_0^\infty \eta^{\text{refr}}(z)\delta(t' - z - t'_{\text{last}})dz \tag{B.59}$$

$$h^{\text{syn}}(t') = \sum_j J_j \int_0^\infty \Lambda(z', t' - t'_{\text{last}})s(t' - z')dz' \,. \tag{B.60}$$

The kernel $\eta^{\text{refr}}(z)$ is the refractory function. If we consider only absolute refractoriness, $\eta^{\text{refr}}(z)$ is given by:

$$\eta^{\text{refr}}(z) = \begin{cases} -\infty \text{ for } 0 < z \leq \tau^{\text{refr}} \\ 0 \quad \text{for } z \geq \tau^{\text{refr}} \end{cases} \tag{B.61}$$

where τ^{refr} is the absolute refractory time. The time t'_{last} corresponds to the last postsynaptic spike (i.e. the most recent firing of the particular neuron). The second response function is the synaptic kernel $\Lambda(z', t' - t'_{\text{last}})$. It describes the effect of an incoming spike on the membrane potential at the soma of the postsynaptic neuron, and it eventually includes also the dependence on the state of the receiving neuron through the difference $t' - t'_{\text{last}}$ (i.e. through the time that has passed since the last postsynaptic spike). The input spike train yields

$s(t'-z') = \sum_i \delta(t'-z'-t_{ij})$, t_{ij} being the ith spike at presynaptic input j. In order to simplify the discussion and without losing generality, let us consider only a single synaptic input, and therefore we can remove the subindex j. In addition, we assume that the synaptic strength J is positive (i.e. excitatory). Integrating equations B.59 and B.60, we obtain

$$h(t') = \eta^{\text{refr}}(t'-t'_{\text{last}}) + J\sum_i \Lambda(t'-t_i, t'-t'_{\text{last}}) \tag{B.62}$$

Synaptic kernels are of the form

$$\Lambda(t'-t_i, t'-t'_{\text{last}}) = \mathrm{H}(t'-t_i)\mathrm{H}(t_i-t'_{\text{last}})\Psi(t'-t_i) \tag{B.63}$$

where $\mathrm{H}(s)$ is the step (or Heaviside) function which vanishes for $s \leq 0$ and takes a value of 1 for $s > 0$. After firing, the membrane potential is reset according to the renewal hypothesis.

This spike-response model is not used in the models described in this book, but is presented to show alternative approaches to modelling the dynamics of network activity.

B.7 Network dynamics: introduction to the mean-field approach

Units whose potential and conductances follow the integrate-and-fire equations in Section B.6 can be assembled together in a model network of any composition and architecture. It is convenient to imagine that units are grouped into classes, such that the parameters quantifying the electrophysiological properties of the units are uniform, or nearly uniform, within each class, while the parameters assigned to synaptic connections are uniform or nearly uniform for all connections from a given presynaptic class to another given postsynaptic class. The parameters that have to be set in a model at this level of description are quite numerous, as listed in Tables B.3 and B.4.

In the limit in which the parameters are constant within each class or pair of classes, a mean-field treatment can be applied to analyze a model network, by summing equations that describe the dynamics of individual units to obtain a more limited number of equations that describe the dynamical behaviour of groups of units (Frolov and Medvedev 1986). The treatment is exact in the further limit in which very many units belong to each class, and is an approximation if each class includes just a few units. Suppose that N_C is the number of classes defined. Summing equations B.43 and B.44 across units of the same class results in N_C functional equations describing the evolution in time of the fraction of cells of a particular class that at a given instant have a given membrane potential. In other words, from a treatment in which the evolution of the variables associated with each unit is followed separately, one moves to a treatment based on density functions, in which the common behaviour of units of the same class is followed together, keeping track solely of the portion of units at any given value of the membrane potential. Summing equation B.42 across connections with the same class of origin and destination results in $N_C \times N_C$ equations describing the dynamics of the overall summed conductance opened on the membrane of a cell of a particular class by all the cells of another given class. A more explicit derivation of mean-field equations is given by Treves (1993) and in Section B.8.

The system of mean-field equations can have many types of asymptotic solutions for long times, including chaotic, periodic, and stationary ones. The stationary solutions are stationary in the sense of the mean fields, but in fact correspond to the units of each class firing tonically at a certain rate. They are of particular interest as the dynamical equivalent of the steady

Table B.3 Cellular parameters (chosen according to the class of each unit)

V_{rest}	Resting potential
V_{thr}	Threshold potential
V_{ahp}	Reset potential
V_{K}	Potassium conductance equilibrium potential
C	Membrane capacitance
τ_{K}	Potassium conductance time constant
g_0	Leak conductance
Δg_{K}	Extra potassium conductance following a spike
Δt	Overall transmission delay

Table B.4 Synaptic parameters (chosen according to the classes of presynaptic and postsynaptic units)

V_α	Synapse equilibrium potential
τ_α	Synaptic conductance time constant
Δg_α	Conductance opened by one presynaptic spike
Δt_α	Delay of the connection

states analyzed by using non-dynamical model networks. In fact, since the neuronal current-to-frequency transfer function resulting from the dynamical equations is rather similar to a threshold linear function (see Fig. B.32), and since each synaptic conductance is constant in time, the stationary solutions are essentially the same as the states described using model networks made up of threshold linear, non-dynamical units. Thus the dynamical formulation reduces to the simpler formulation in terms of steady-state rates when applied to asymptotic stationary solutions; but, among simple rate models, it is equivalent only to those that allow description of the continuous nature of neuronal output, and not to those, for example based on binary units, that do not reproduce this fundamental aspect. The advantages of the dynamical formulation are that (i) it enables one to describe the character and prevalence of other types of asymptotic solutions, and (ii) it enables one to understand how the network reaches, in time, the asymptotic behaviour.

The development of this mean-field approach, and the foundations for its application to models of cortical visual processing and attention, are described in Section B.8.

B.8 Mean-field based neurodynamics

A model of brain functions requires the choice of an appropriate theoretical framework, which permits the investigation and simulation of large-scale biologically realistic neural networks. Starting from the mathematical models of biologically realistic single neurons (i.e. spiking neurons), one can derive models that describe the joint activity of pools of equivalent neurons. This kind of neurodynamical model at the neuronal assembly level is motivated by the experimental observation that cortical neurons of the same type that are near to each other tend to receive similar inputs. As described in the previous section, it is convenient in this simplified approach to neural dynamics to consider all neurons of the same type in a small cortical volume as a computational unit of a neural network. This computational unit is called a neuronal pool or assembly. The mathematical description of the dynamical evolution of neuronal pool activity in multimodular networks, associated with different cortical areas, establishes the roots of the dynamical approach that are used in Chapters 6 and 7, and in Rolls and Deco (2002)

Chapters 9–11. In this Section (B.8), we introduce the mathematical fundamentals utilized for a neurodynamical description of pool activity (see also Section 7.9). Beginning at the microscopic level and using single spiking neurons to form the pools of a network, we derive the mathematical formulation of the neurodynamics of cell assemblies. Further, we introduce the basic architecture of neuronal pool networks that fulfil the basic mechanisms consistent with the biased competition hypothesis. Each of these networks corresponds to cortical areas that also communicate with each other. We describe therefore the dynamical interaction between different modules or networks, which will be the basis for the implementation of attentional top-down bias.

B.8.1 Population activity

We now introduce thoroughly the concept of a neuronal pool and the differential equations representing the neurodynamics of pool activity.

Starting from individual spiking neurons one can derive a differential equation that describes the dynamical evolution of the averaged activity of a pool of extensively many equivalent neurons. Several areas of the brain contain groups of neurons that are organized in populations of units with (somewhat) similar properties (though in practice the neurons convey independent information, as described in Appendix C). These groups for mean-field modelling purposes are usually called pools of neurons and are constituted by a large and similar population of identical spiking neurons that receive similar external inputs and are mutually coupled by synapses of similar strength. Assemblies of motor neurons (Kandel, Schwartz and Jessel 2000) and the columnar organization in the visual and somatosensory cortex (Hubel and Wiesel 1962, Mountcastle 1957) are examples of these pools. Each single cell in a pool can be described by a spiking model, e.g. the spike response model presented in Section B.6.7. Due to the fact that for large-scale cortical modelling, neuronal pools form a relevant computational unit, we adopt a population code. We take the activity level of each pool of neurons as the relevant dependent variable rather than the spiking activity of individual neurons. We therefore derive a dynamical model for the mean activity of a neural population. In a population of M neurons, the mean activity $A(t)$ is determined by the proportion of active neurons by counting the number of spikes $n_{\text{spikes}}(t, t+\Delta t)$ in a small time interval Δt and dividing by M and by Δt (Gerstner 2000), i.e. formally

$$A(t) = \lim_{\Delta t \to 0} \frac{n_{\text{spikes}}(t, t+\Delta t)}{M \Delta t} \,. \tag{B.64}$$

As indicated by Gerstner (2000), and as depicted in Fig. B.38, the concept of pool activity is quite different from the definition of the average firing rate of a single neuron. Contrary to the concept of temporal averaging over many spikes of a single cell, which requires that the input is slowly varying compared with the size of the temporal averaging window, a coding scheme based on pool activity allows rapid adaptation to real-world situations with quickly changing inputs. It is possible to derive dynamical equations for pool activity levels by utilizing the mean-field approximation (Wilson and Cowan 1972, Abbott 1991, Amit and Tsodyks 1991). The mean-field approximation consists of replacing the temporally averaged discharge rate of a cell with an equivalent momentary activity of a neural population (ensemble average) that corresponds to the assumption of ergodicity. According to this approximation, we categorize each cell assembly by means of its activity $A(t)$. A pool of excitatory neurons without external input can be described by the dynamics of the pool activity given by

$$\tau \frac{\partial A(t)}{\partial t} = -A(t) + q\mathrm{F}(A(t)) \tag{B.65}$$

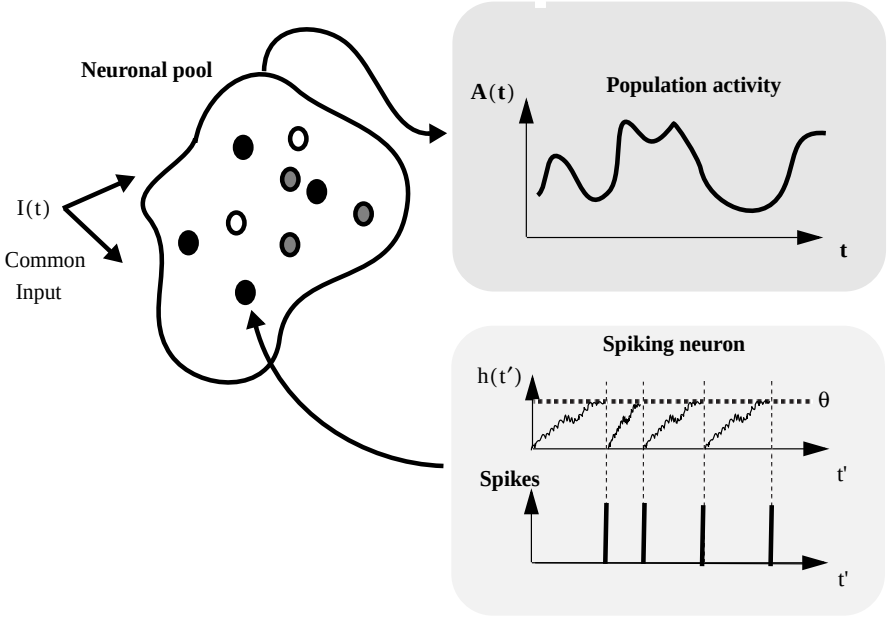


Fig. B.38 Population averaged rate of a neuronal pool of spiking neurons (top) and the action potential generating mechanism of single neurons (bottom). In a neuronal pool, the mean activity $A(t)$ is determined by the proportion of active neurons by counting the number of spikes in a small time interval Δt and dividing by the number of neurons in the pool and by Δt. Spikes are generated by a threshold process. The firing time t' is given by the condition that the membrane potential $h(t')$ reaches the firing threshold θ. The membrane potential $h(t')$ is given by the integration of the input signal weighted by a given kernel (see text for details). (After Rolls and Deco 2002.)

where the first term on the right hand side is a decay term and the second term takes into account the excitatory stimulation between the neurons in the pool. In the previous equation, the non-linearity

$$\mathrm{F}(x) = \frac{1}{T_r - \tau \log(1 - \frac{1}{\tau x})} \tag{B.66}$$

is the response function (transforming current into discharge rate) for a spiking neuron with deterministic input, membrane time constant τ, and absolute refractory time T_r. Equation B.65 was derived by Gerstner (2000) assuming adiabatic conditions. Gerstner (2000) has shown that the population activity in a homogeneous population of neurons can be described by an integral equation. A systematic reduction of the integral equation to a single differential equation of the form B.65 always supposes that the activity changes only slowly compared with the typical interval length. In other words, the mean-field approach described in the above equations and utilized in parts of Chapters 4, 6, 7 and 8 generates a dynamics that neglects fast, transient behaviour. This means that we are assuming that rapid oscillations (and synchronization) do not play a computational role at least for the brain functions that we will consider. Rapid oscillations of neural activity could have a relevant functional role, namely of dynamical cooperation between pools in the same or different brain areas. It is well known in the theory of dynamical systems that the synchronization of oscillators is a cooperative phenomenon. Cooperative mechanisms might complement the competitive mechanisms on which our computational cortical model is based.

An example of the application of the mean field approach, in a model of decision-making (Deco and Rolls 2006), is provided in Section 7.9.

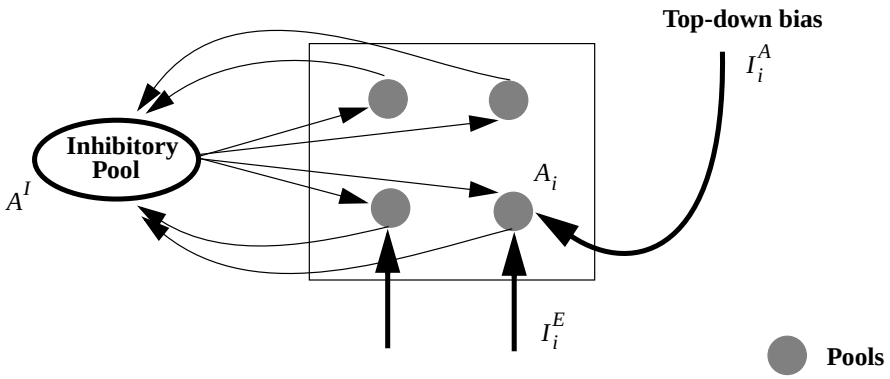


Fig. B.39 Basic computational module for biased competition: a competitive network with external top-down bias. Excitatory pools with activity A_i for the ith pool are connected with a common inhibitory pool with activity A^I in order to implement a competition mechanism. I_i^E is the external sensory input to the cells in pool i, and I_i^A attentional top-down bias, an external input coming from higher modules. The external top-down bias can shift the competition in favour of a specific pool or group of pools. This architecture is similar to that shown in Fig. B.18, but with competition between pools of similar neurons. (After Rolls and Deco 2002.)

B.8.2 A basic computational module based on biased competition

We are interested in the neurodynamics of modules composed of several pools that implement a competitive mechanism[42]. This can be achieved by connecting the pools of a given module with a common inhibitory pool, as is schematically shown in Fig. B.39.

In this way, the more pools of the module that are active, the more active the common inhibitory pool will be, and consequently, the more feedback inhibition will affect the pools in the module, such that only the most excited group of pools will survive the competition. On the other hand, external top-down bias could shift the competition in favour of a specific group of pools. This basic computational module implements therefore the biased competition hypothesis described in Chapter 6. Let us assume that there are m pools in a given module. The system of differential equations describing the dynamics of such a module is given by two differential equations, both of the type of equation B.65. The first differential equation describes the dynamics of the activity level of the excitatory pools (pyramidal neurons) and is mathematically expressed by

$$\tau \frac{\partial A_i(t)}{\partial t} = -A_i(t) + a\mathrm{F}(A_i(t)) - b\mathrm{F}(A^I(t)) + I_0 + I_i^E(t) + I_i^A(t) + \nu \quad ; \qquad \text{for } i = 1, ..., m \tag{B.67}$$

and the second one describes the dynamics of the activity level of the common inhibitory pool for each feature dimension (stellate neurons)

$$\tau_p \frac{\partial A^I(t)}{\partial t} = -A^I(t) + c\sum_{i=1}^{m} \mathrm{F}(A_i(t)) - d\mathrm{F}(A^I(t)) \tag{B.68}$$

where $A_i(t)$ is the activity for pool i, $A^I(t)$ is the activity in the inhibitory pool, I_0 is a diffuse spontaneous background input, $I^E(t)$ is the external sensory input to the cells in pool i, and ν is additive Gaussian noise. The attentional top-down bias $I_i^A(t)$ is defined as an external input coming from higher modules that is not explicitly modelled.

[42] These neurodynamics are used in Chapter 6.

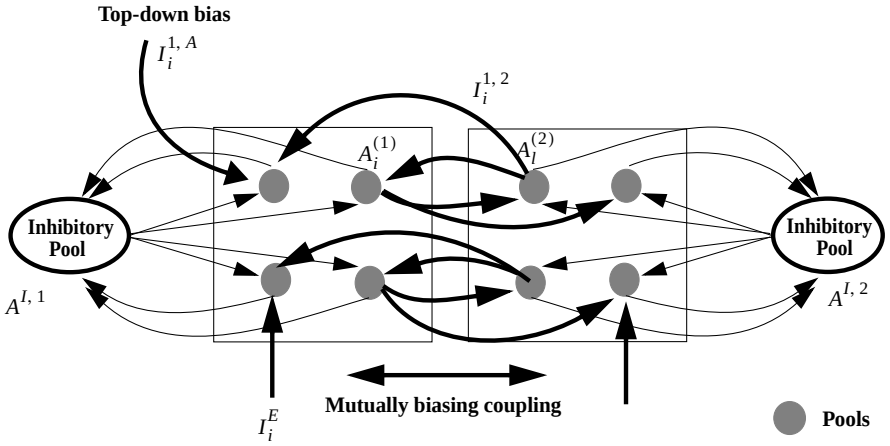


Fig. B.40 Two competitive networks mutually biased through intermodular connections. The activity $A_i^{(1)}$ of the ith excitatory pool in module 1 (on the left) and of the lth excitatory pool in module 2 (on the right) are connected by the mutually biasing coupling $I_i^{1,2}$. The architecture could implement top-down feedback originating from the interaction between brain areas that are explicitly modelled in the system. (Module 2 might be the higher module.) The external top-down bias $I_i^{1,A}$ corresponds to the coupling to pool i of module 1 from brain area A that is not explicitly modelled in the system. (After Rolls and Deco 2002.)

A qualitative description of the main fixed point attractors of the system of differential equations B.67 and B.68 was provided by Usher and Niebur (1996). Basically, we will be interested in the fixed points corresponding to zero activity and larger activation. The parameters will therefore be fixed such that the dynamics evolves to these attractors.

B.8.3 Multimodular neurodynamical architectures

In order to model complex psychophysically and neuropsychologically relevant brain functions such as visual search or object recognition (see e.g. Chapter 6), we must take into account the computational role of individual brain areas and their mutual interaction. The macroscopic phenomenological behaviour will therefore be the result of the mutual interaction of several computational modules.

The dynamical coupling of different basic modules in a multimodular architecture can be described, in our neurodynamical framework, by allowing mutual interaction between pools belonging to different modules. Figure B.40 shows this idea schematically. The system of differential equations describing the global dynamics of such a multimodular system is given by a set of equations of the type of equation B.65.

The excitatory pools belonging to a module obey the following equations:

$$\tau \frac{\partial A_i^{(j)}(t)}{\partial t} = -A_i^{(j)}(t) + a\mathrm{F}(A_i^{(j)}(t)) - b\mathrm{F}(A^{I,j}(t)) + I_i^{j,k} + I_0 + I_i^E(t) + I_i^A(t) + \nu \qquad \text{for } i = 1, ..., m \qquad \text{(B.69)}$$

and the corresponding inhibitory pools evolve according to

$$\tau_p \frac{\partial A^{I,j}(t)}{\partial t} = -A^{I,j}(t) + c\sum_{i=1}^{m} \mathrm{F}(A_i^{(j)}(t)) - d\mathrm{F}(A^{I,j}(t)). \qquad \text{(B.70)}$$

The mutual coupling $I_i^{j,k}$ between module (j) and (k) is given by

$$I_i^{j,k} = \sum_l W_{il} \mathrm{F}(I_l^{(k)}(t)) \tag{B.71}$$

where W_{il} is the synaptic coupling strength between the pool i of module (j) and pool l of module (k). This mutual coupling term can be interpreted as a top-down bias originating from the interaction between brain areas that is explicitly modelled in our system. On the other hand, the external top-down bias I_i^A corresponds to the coupling with brain areas A that are not explicitly modelled in our system.

Additionally, it is interesting to note that the top-down bias in this kind of architecture modulates the response of the pool activity in a multiplicative manner. Responses of neurons in parietal area 7a are modulated by combined eye and head movement, exhibiting a multiplicative gain modulation that modifies the amplitude of the neural responses to retinal input but does not change the preferred retinal location of a cell, nor in general the width of the receptive field (Brotchie, Andersen, Snyder and Goodman 1995). It has also been suggested that multiplicative gain modulation might play a role in translation invariant object representation (Salinas and Abbott 1997). We will use this multiplicative effect for formulating an architecture for attentional gain modulation, which can contribute to correct translation invariant object recognition in ways to be described in Chapter 6.

We show now that multiplicative-like responses can arise from the top-down biased mutual interaction between pools. Another alternative architecture that can also perform product operations on additive synaptic inputs was proposed by Salinas and Abbott (1996). Our basic architecture for showing this multiplicative effect is presented schematically in Fig. B.41a.

Two pools are mutually connected via fixed weights. The first pool or unit receives a bottom-up visual input I_1^V, modelled by the response of a vertically oriented complex cell. The second pool receives a top-down attentional bias $I_2^A = B$. The two pools are mutually coupled with unity weight. The equations describing the activities of the two pools are given by:

$$\tau \frac{\partial A_1(t)}{\partial t} = -A_1(t) + \alpha \mathrm{F}(A_1(t)) + I_o + I_1^V + \nu \tag{B.72}$$

$$\tau \frac{\partial A_2(t)}{\partial t} = -A_2(t) + \alpha \mathrm{F}(A_2(t)) + I_o + I_2^A + \nu \tag{B.73}$$

where A_1 and A_2 are the activities of pool 1 and pool 2 respectively, $\alpha = 0.95$ is the coefficient of recurrent self-excitation of the pool, ν is the noise input to the pool drawn from a normal distribution $N(\mu = 0, \sigma = 0.02)$, and $I_o = 0.025$ is a direct current biasing input to the pool.

Simulation of the above dynamical equations produces the results shown in Fig. B.41b. The orientation tuning curve of unit (or pool) 1 was modulated by a top-down bias B introduced to unit (pool) 2. The gain modulation was transmitted through the coupling from pool 2 to pool 1 after a few steps of evolution of the dynamical equations. Without the feedback from unit 2, unit 1 exhibits the orientation tuning curve shown as ($B = 0$). As B increased, the increase in pool 1's response to the vertical bar was significantly greater than the increase in its response to the horizontal bar. Therefore, the attentional gain modulation produced in pool 1 through the mutual coupling was not a simple additive effect, but had a strong multiplicative component. The net effect was that the width of the orientation tuning curve of the cell was roughly preserved under attentional modulation. This was due to the non-linearity in the activation function.

This finding is basically consistent with the effect of attention on the orientation tuning curves of neurons in V4 (McAdams and Maunsell 1999).

Summarizing, the neurodynamics of the competitive mechanisms between neuronal pools, and their mutual gain modulation, are the two main ingredients used for proposing a cortical

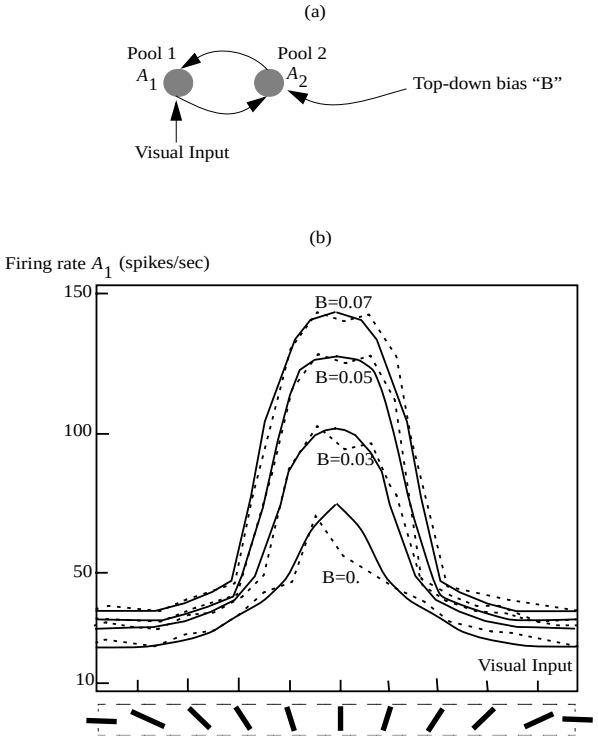


Fig. B.41 (a) The basic building block of the top-down attentional system utilizes non-specific competition between pools within the same module and specific mutual facilitation between pools in different modules. Excitatory neuronal pools within the same module compete with each other through one or more inhibitory neuronal pool(s) I_I with activity A^{I_I}. Excitatory pool 1 (with activity A_1) receives a bottom-up (visual) input I_1^V, and excitatory pool 2 receives a top-down ('attentional') bias input I_2^A. Excitatory neuronal pools in the two different modules can excite each other via mutually biased coupling. (b) The effect of altering the bias input B to pool 2 on the responses or activity of pool 1 to its orientation-tuned visual input (see text). (After Rolls and Deco 2002.)

architecture that models attention and different kinds of object search in a visual scene (see Chapter 6 and Section 6.13, and Rolls and Deco (2002) Chapters 9–11).

B.9 Interacting attractor networks

It is prototypical of the cerebral neocortical areas that there are recurrent collateral connections between the neurons within an area or module, and forward connections to the next cortical area in the hierarchy, which in turn sends backprojections (see Section 1.11). This architecture, made explicit in Fig. 5.3, immediately suggests, given that the recurrent connections within a module, and the forward and backward connections, are likely to be associatively modifiable, that the operation incorporates at least to some extent, interactions between coupled attractor (autoassociation) networks. For these reasons, it is important to analyze the rules that govern the interactions between coupled attractor networks. This has been done using the formal type of model described in Section 5.3.

One boundary condition is when the coupling between the networks is so weak that there is effectively no interaction. This holds when the coupling parameter g between the networks is less than approximately 0.002, where the coupling parameter indicates the relative strength

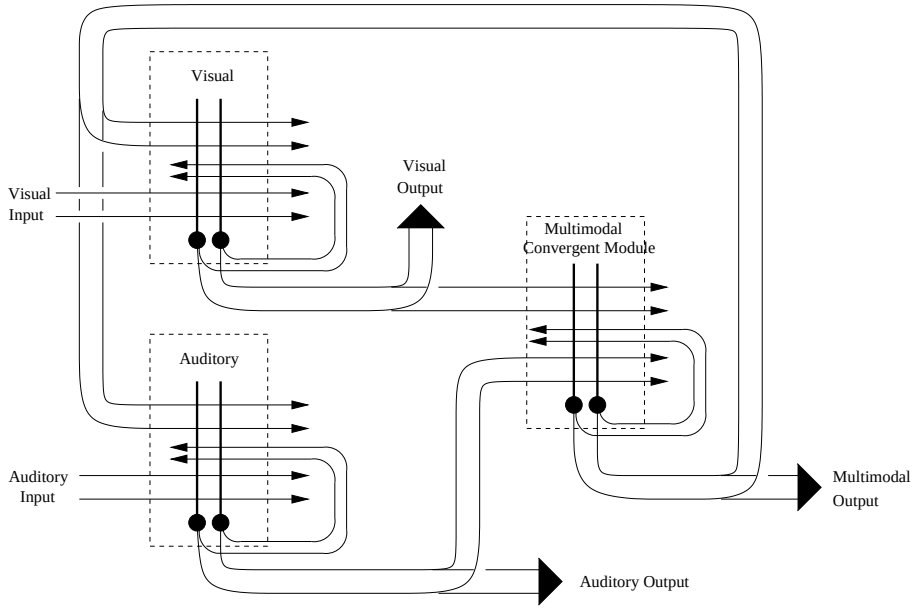


Fig. B.42 A two-layer set of attractor nets in which feedback from layer 2 can influence the states reached in layer 1. Layer 2 could be a higher cortical visual area with convergence from earlier cortical visual areas (see Chapter 4). Layer 2 could also be a multimodal area receiving inputs from unimodal visual and auditory cortical areas, as labelled. Each of the 3 modules has recurrent collateral synapses that are trained by an associative synaptic learning rule, and also inter-modular synaptic connections in the forward and backward direction that are also associatively trained. Attractors are formed within modules, the different modules interact, and attractors are also formed by the forward and backward inter-modular connections. The higher area may not only affect the states reached during attractor settling in the input layers, but may also, as a result of this, influence the representations that are learned in earlier cortical areas. A similar principle may operate in any multilayer hierarchical cortical processing system, such as the ventral visual system, in that the categories that can be formed only at later stages of processing may help earlier stages to form categories relevant to what can be diagnosed at later stages.

of the inter-modular to the intra-modular connections, and measures effectively the relative strengths of the currents injected into the neurons by the inter-modular relative to the intra-modular (recurrent collateral) connections (Renart, Parga and Rolls 1999b). At the other extreme, if the coupling parameter is strong, all the networks will operate as a single attractor network, together able to represent only one state (Renart, Parga and Rolls 1999b). This critical value of the coupling parameter (at least for reciprocally connected networks with symmetric synaptic strengths) is relatively low, in the region of 0.024 (Renart, Parga and Rolls 1999b). This is one reason why cortico-cortical backprojections are predicted to be quantitatively relatively weak, and for this reason it is suggested end on the apical parts of the dendrites of cortical pyramidal cells (see Section 1.11). In the strongly coupled regime when the system of networks operates as a single attractor, the total storage capacity (the number of patterns that can be stored and correctly retrieved) of all the networks will be set just by the number of synaptic connections onto any single neuron received from other neurons in the network, a number in the order of a few thousand (see Section B.3). This is one reason why connected cortical networks are thought not to act in the strongly coupled regime, because the total number of memories that could be represented in the whole of the cerebral cortex would be so small, in the order of a few thousand, depending on the sparseness of the patterns (see equation B.15) (O'Kane and Treves 1992).

Between these boundary conditions, that is in the region where the inter-modular coupling parameter g is in the range 0.002–0.024, it has been shown that interesting interactions can occur (Renart, Parga and Rolls 1999b, Renart, Parga and Rolls 1999a). In a bimodular architecture, with forward and backward connections between the modules, the capacity of one module can be increased, and an attractor is more likely to be found under noisy conditions, if there is a consistent pattern in the coupled attractor. By consistent we mean a pattern that during training was linked associatively by the forward and backward connections, with the pattern being retrieved in the first module. This provides a quantitative model for understanding some of the effects that backprojections can produce by supporting particular states in earlier cortical areas (Renart, Parga and Rolls 1999b). The total storage capacity of the two networks is however, in line with O'Kane and Treves (1992), not a great deal greater than the storage capacity of one of the modules alone. Thus the help provided by the attractors in falling into a mutually compatible global retrieval state (in e.g. the scenario of a hierarchical system) is where the utility of such coupled attractor networks must lie. Another interesting application of such weakly coupled attractor networks is in coupled perceptual and short-term memory systems in the brain, described in Section 5.1 and Chapter 6.

In a trimodular attractor architecture shown in Fig. B.42 (which is similar to the architecture of the multilayer competitive net illustrated in Fig. B.19 but has recurrent collateral connections within each module), further interesting interactions occur that account for effects such as the McGurk effect, in which what is seen affects what is heard (Renart, Parga and Rolls 1999a). The effect was originally demonstrated with the perception of auditory syllables, which were influenced by what is seen (McGurk and MacDonald 1976). The trimodular architecture (studied using similar methods to those used by Renart, Parga and Rolls (1999b) and frequently utilizing scenarios in which first a stimulus was presented to a module, then removed during a memory delay period in which stimuli were applied to other modules) showed a phase with $g < 0.005$ in which the modules operated in an isolated way. With g in the range 0.005–0.012, an 'independent' regime existed in which each module could be in a separate state to the others, but in which interactions between the modules occurred, which could assist or hinder retrieval in a module depending on whether the states in the other modules were consistent or inconsistent. It is in this 'independent' regime that a module can be in a continuing attractor that can provide other modules with a persistent external modulatory input that is helpful for tasks such as making comparisons between stimuli processed sequentially (as in delayed match-to-sample tasks and visual search tasks) (see Section 5.1). In this regime, if the modules are initially quiescent, then application of a stimulus to one input module propagates to the central module, and from it to the non-stimulated input module as well (see Fig. B.42). When g grows beyond 0.012, the picture changes and the independence between the modules is lost. The delay activity states found in this region (of the phase space) *always* involve the three modules in attractors correlated with consistent features associated in the synaptic connections. Also, since g is now larger, changes in the properties of the external stimuli have more impact on the delay activity states. The general trend seen in this phase under the change of stimulus after a previous consistent attractor has been reached is that, first, if the second stimulus is not effective enough (it is weak or brief), it is unable to move any of the modules from their current delay activity states. If the stimulus is made more effective, then as soon as it is able to change the state of the stimulated input module, the internal and non-stimulated input modules follow, and the whole network moves into the new consistent attractor selected by the second stimulus. In this case, the interaction between the modules is so large that it does not allow contradictory local delay activity states to coexist, and the network is described as being in a 'locked' state.

The conclusion is that the most interesting scenario for coupled attractor networks is when they are weakly coupled (in the trimodular architecture $0.005 < g < 0.012$), for then

interactions occur whereby how well one module responds to its own inputs can be influenced by the states of the other modules, but it can retain partly independent representations. This emphasizes the importance of weak interactions between coupled modules in the brain (Renart, Parga and Rolls 1999b, Renart, Parga and Rolls 1999a, Renart, Parga and Rolls 2000).

These generally useful interactions between coupled attractor networks can be useful in implementing top-down constraint satisfaction (see Section 1.11) and short-term memory (see Section 5.1). One type of constraint satisfaction in which they are also probably important is cross-modal constraint satisfaction, which occurs for example when the sight of the lips moving assists the hearing of syllables. If the experimenter mismatches the visual and auditory inputs, then auditory misperception can occur, as in the McGurk effect. In such experiments (McGurk and MacDonald 1976) the subject receives one stimulus through the auditory pathway (e.g. the syllables *ga-ga*) and a *different* stimulus through the visual pathway (e.g. the lips of a person performing the movements corresponding to the syllables *ba-ba* on a video monitor). These stimuli are such that their acoustic waveforms as well as the lip motions needed to pronounce them are rather different. One can then assume that although they share the same vowel 'a', the internal representation of the syllables is dominated by the consonant, so that the representations of the syllables *ga-ga* and *ba-ba* are not correlated either in the primary visual cortical areas or in the primary auditory ones. At the end of the experiment, the subject is asked to repeat what he heard. When this procedure is repeated with many subjects, it is found that roughly 50% of them claim to have heard either the auditory stimulus (*ga-ga*), or the visual one (*ba-ba*). The rest of the subjects report to have heard neither the auditory nor the visual stimuli, but actually a combination of the two (e.g. *gabga*) or even something else including phonemes not presented auditorially or visually (e.g. *gagla*).

Renart, Parga and Rolls (1999a) were able to show that the McGurk effect can be accounted for by the operation of coupled attractor networks of the form shown in Fig. B.42. One input module is for the auditory input, the second is for the visual input, and both converge into a higher area which represents the syllable formed on the evidence of combination of the two inputs. There are backprojections from the convergent module back to the input modules. Persistent (continuing) inputs were applied to both the inputs, and during associative training of all the weights the visual and auditory inputs corresponded to the same syllable. When tested with inconsistent visual and auditory inputs, it was found for g between ~ 0.10 and ~ 0.11, the convergent module can either remain in a symmetric state in which it represents a mixture of the two inputs, or choose between one of the inputs, with either situation being stable. For lower g the convergent module always settles into a state corresponding to the input in one of the input modules. It is the random fluctuations produced during the convergence to the attractor that determine the pattern selected by the convergent module. When the convergent module becomes correlated with *one* of its stored patterns, the signal back-projected to the input module stimulated with the feature associated with that pattern becomes stronger and the overlap in this module is increased. Thus, with low values of the inter-module coupling parameter g, situations are found in which sometimes the input to one module dominates, and sometimes the input to the other module dominates what is represented in the convergent module, and sometimes mixture states are stable in the convergent module. This model can thus account for the influences that visual inputs can have on what is heard, in for example the McGurk effect.

The interactions between coupled attractor networks can lead to the following effects. Facilitation can occur in a module if its external input is matched by an input from another module, whereas suppression in a module of its response to an external input can occur if the two inputs mismatch. This type of interaction can be used in imaging studies to identify brain regions where different signals interact with each other. One example is to locate brain regions where multimodal inputs converge. If the inputs in two sensory modalities

are consistent based on previous experience, then facilitation will occur, whereas if they are inconsistent, suppression of the activity in a module can occur. This is one of the effects described in the bimodular and trimodular architectures investigated by Renart, Parga and Rolls (1999b), Renart, Parga and Rolls (1999a) and Rolls and Stringer (2001b), and found in architectures such as that illustrated in Fig. B.42.

If a multimodular architecture is trained with each of many patterns (which might be visual stimuli) in one module associated with one of a few patterns (which might be mood states) in a connected module, then interesting effects due to this asymmetry are found, as described in Section 3.11 and by Rolls and Stringer (2001b).

An interesting issue that arises is how rapidly a system of interacting attractor networks such as that illustrated in Fig. B.42 settles into a stable state. Is it sufficiently rapid for the interacting attractor effects described to contribute to cortical information processing? It is likely that the settling of the whole system is quite rapid, if it is implemented (as it is in the brain) with synapses and neurons that operate with continuous dynamics, where the time constant of the synapses dominates the retrieval speed, and is in the order of 15 ms for each module, as described in Section B.6 and by Panzeri, Rolls, Battaglia and Lavis (2001). In that Section, it is shown that a multimodular attractor network architecture can process information in approximately 15 ms per module (assuming an inactivation time constant for the synapses of 10 ms), and similarly fast settling may be expected of a system of the type shown in Fig. B.42.

B.10 Sequence memory implemented by adaptation in an attractor network

Sequence memory can be implemented by using synaptic adaptation to effectively encode the order of the items in a sequence, as described in Section 9.3 (Deco and Rolls 2005c). Whenever the attractor system is quenched into inactivity, the next member of the sequence emerges out of the spontaneous activity, because the least recently activated member of the sequence has the least synaptic or neuronal adaptation. This mechanism could be implemented in recurrent networks such as the hippocampal CA3 or the prefrontal cortex.

B.11 Perceptrons and one-layer error correction networks

The networks described in the next two Sections (B.11 and B.12) are capable of mapping a set of inputs to a set of required outputs using correction when errors are made. Although some of the networks are very powerful in the types of mapping they can perform, the power is obtained at the cost of learning algorithms that do not use local learning rules. A local learning rule specifies that synaptic strengths should be altered on the basis of information available locally at the synapse, for example the activity of the presynaptic and the post-synaptic neurons. Because the networks described here do not use local learning rules, their biological plausibility remains at present uncertain. One of the aims of future research must be to determine whether comparably difficult problems to those solved by the networks described in Sections B.11 and B.12 can be solved by biologically plausible neuronal networks.

We now describe one-layer networks taught by an error correction algorithm. The term *perceptron* refers strictly to networks with binary threshold activation functions. The outputs might take the values only 1 or 0 for example. The term perceptron arose from networks designed originally to solve perceptual problems (Rosenblatt 1961, Minsky and Papert 1969), and these networks are referred to briefly below. If the output neurons have continuous-valued

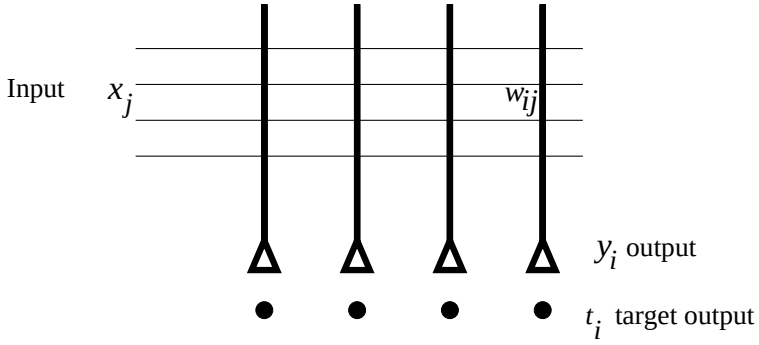


Fig. B.43 One-layer perceptron.

firing rates, then a more general error-correcting rule called the delta rule is used, and is introduced in this Section (B.11). For such networks, the activation function may be linear, or it may be non-linear but monotonically increasing, without a sharp threshold, as in the sigmoid activation function (see Fig. 1.3).

B.11.1 Architecture and general description

The one-layer error-correcting network has a set of inputs that it is desired to map or classify into a set of outputs (see Fig. B.43). During learning, an input pattern is selected, and produces output firing by activating the output neurons through modifiable synapses, which then fire as a function of their typically non-linear activation function. The output of each neuron is then compared with a target output for that neuron given that input pattern, an error between the actual output and the desired output is determined, and the synaptic weights on that neuron are then adjusted to minimize the error. This process is then repeated for all patterns until the average error across patterns has reached a minimum. A one-layer error-correcting network can thus produce output firing for each pattern in a way that has similarities to a pattern associator. It can perform more powerful mappings than a pattern associator, but requires an error to be computed for each neuron, and for that error to affect the synaptic strength in a way that is not altogether local. A more detailed description follows.

These one-layer networks have a target for each output neuron (for each input pattern). They are thus an example of a supervised network. With the one-layer networks taught with the delta rule or perceptron learning rule described next, there is a separate teacher for each output neuron, as shown in Fig. B.43.

B.11.2 Generic algorithm for a one-layer error correction network)

For each input pattern and desired target output:

1. Apply an input pattern to produce input firing $\mathbf{x}$, and obtain the activation of each neuron in the standard way by computing the dot product of the input pattern and the synaptic weight vector. The synaptic weight vector can be initially zero, or have random values.

$$h_i = \sum_j x_j w_{ij} \tag{B.74}$$

where $\sum_j$ indicates that the sum is over the C input axons (or connections) indexed by j to each neuron.

2. Apply an activation function to produce the output firing y_i:

$$y_i = \mathrm{f}(h_i) \ . \tag{B.75}$$

This activation function f may be sigmoid, linear, binary threshold, linear threshold, etc. If the activation function is non-linear, this helps to classify the inputs into distinct output patterns, but a linear activation function may be used if an optimal linear mapping is desired (see Adaline and Madaline, below).

3. Calculate the difference for each cell i between the target output t_i and the actual output y_i produced by the input, which is the error Δ_i

$$\Delta_i = t_i - y_i. \tag{B.76}$$

4. Apply the following learning rule, which corrects the (continuously variable) weights according to the error and the input firing x_j

$$\delta w_{ij} = k(t_i - y_i)x_j \tag{B.77}$$

where k is a constant that determines the learning rate. This is often called the delta rule, the Widrow–Hoff rule, or the LMS (least mean squares) rule (see below).

5. Repeat steps 1–4 for all input pattern – output target pairs until the root mean square error becomes zero or reaches a minimum.

In general, networks taught by the delta rule may have linear, binary threshold, or non-linear but monotonically increasing (e.g. sigmoid) activation functions, and may be taught with binary or continuous input patterns (see Rolls and Treves (1998), Chapter 5). The properties of these variations are made clear next.

B.11.3 Capability and limitations of single-layer error-correcting networks

Perceptrons perform pattern classification. That is, each neuron classifies the input patterns it receives into classes determined by the teacher. This is thus an example of a supervised network, with a separate teacher for each output neuron. The classification is most clearly understood if the output neurons are binary, or are strongly non-linear, but the network will still try to obtain an optimal mapping with linear or near-linear output neurons.

When each neuron operates as a binary classifier, we can consider how many input patterns p can be classified by each neuron, and the classes of pattern that can be correctly classified. The result is that the maximum number of patterns that can be correctly classified by a neuron with C inputs is

$$p_{\max} = 2C \tag{B.78}$$

when the inputs have random continuous-valued inputs, but the patterns must be linearly separable (see Section A.2.1, and Hertz, Krogh and Palmer (1991)). For a one-layer network, no set of weights can be found that will perform the XOR (exclusive OR), or any other non-linearly separable function (see Appendix A).

Although the inability of one-layer networks with binary neurons to solve non-linearly separable problems is a limitation, it is not in practice a major limitation on the processing that can be performed in a neural network for a number of reasons. First, if the inputs can take continuous values, then if the patterns are drawn from a random distribution, the

one-layer network can map up to $2C$ of them. Second, as described for pattern associators, the perceptron could be preceded by an expansion recoding network such as a competitive network with more output than input neurons. This effectively provides a two-layer network for solving the problem, and multilayer networks are in general capable of solving arbitrary mapping problems. Ways in which such multilayer networks might be trained are discussed later in this Appendix.

We now return to the issue of the capacity of one-layer perceptrons, that is, how many patterns p can be correctly mapped to correct binary outputs if the input patterns are linearly separable.

B.11.3.1 Output neurons with continuous values, and random patterns

Before treating this case, we note that if the inputs are orthogonal, then just as in the pattern associator, C patterns can be correctly classified, where there are C inputs, x_j, $(j = 1, C)$, per neuron. The argument is the same as for a pattern associator.

We consider next the capacity of a one-layer error-correcting network that learns patterns drawn from a random distribution. For neurons with continuous output values, whether the activation function is linear or not, the capacity (for fully distributed inputs) is set by the criterion that the set of input patterns must be linearly independent (see Hertz, Krogh and Palmer (1991)). (Three patterns are linearly independent if any one cannot be formed by addition (with scaling allowed) of the other two patterns – see Appendix A.) Given that there can be a maximum of C linearly independent patterns in a C-dimensional space (see Appendix A), the capacity of the perceptron with such patterns is C patterns. If we choose p random patterns with continuous values, then they will be linearly independent for $p \leq C$ (except for cases with very low probability when the randomly chosen values may not produce linearly independent patterns). (With random continuous values for the input patterns, it is very unlikely that the addition of any two, with scaling allowed, will produce a third pattern in the set.) Thus with continuous valued input patterns,

$$p_{\max} = C. \tag{B.79}$$

If the inputs are not linearly independent, networks trained with the delta rule produce a least mean squares (LMS) error (optimal) solution (see below).

B.11.3.2 Output neurons with binary threshold activation functions

Let us consider here strictly defined perceptrons, that is, networks with (binary) threshold output neurons, and taught by the perceptron learning procedure.

Capacity with fully distributed output patterns

The condition here for correct classification is that described in Appendix A for the AND and XOR functions, that the patterns must be linearly separable. If we consider random continuous-valued inputs, then the capacity is

$$p_{\max} = 2C \tag{B.80}$$

(see Cover (1965), Hertz, Krogh and Palmer (1991); this capacity is the case with C large, and the number of output neurons small). The interesting point to note here is that, even with fully distributed inputs, a perceptron is capable of learning more (fully distributed) patterns than there are inputs per neuron. This formula is in general valid for large C, but happens to hold also for the AND function illustrated in Appendix A.2.1.

Sparse encoding of the patterns

If the output patterns $\mathbf{y}$ are sparse (but still distributed), then just as with the pattern associator, it is possible to map many more than C patterns to correct outputs. Indeed, the number of different patterns or prototypes p that can be stored is

$$p \approx C/a \tag{B.81}$$

where a is the sparseness of the target pattern $\mathbf{t}$. p can in this situation be much larger than C (cf. Rolls and Treves (1990), and Rolls and Treves (1998) Appendix A3).

Perceptron convergence theorem

It can be proved that such networks will learn the desired mapping in a finite number of steps (Block 1962, Minsky and Papert 1969, Hertz, Krogh and Palmer 1991). (This of course depends on there being such a mapping, the condition for this being that the input patterns are linearly separable.) This is important, for it shows that single-layer networks can be proved to be capable of solving certain classes of problem.

As a matter of history, Minsky and Papert (1969) went on to emphasize the point that no one-layer network can correctly classify non-linearly separable patterns. Although it was clear that multilayer networks can solve such mapping problems, Minsky and Papert were pessimistic that an algorithm for training such a multilayer network would be found. Their emphasis that neural networks might not be able to solve general problems in computation, such as computing the XOR, which is a non-linearly separable mapping, resulted in a decline in research activity in neural networks. In retrospect, this was unfortunate, for humans are rather poor at solving parity problems such as the XOR (Thorpe, O'Regan and Pouget 1989), yet can perform many other useful neural network operations very quickly. Algorithms for training multilayer perceptrons were gradually discovered by a number of different investigators, and became widely known after the publication of the algorithm described by Rumelhart, Hinton and Williams (1986b), and Rumelhart, Hinton and Williams (1986a). Even before this, interest in neural network pattern associators, autoassociators and competitive networks was developing (see Hinton and Anderson (1981), Kohonen (1977), Kohonen (1988)), but the acceptance of the algorithm for training multilayer perceptrons led to a great rise in interest in neural networks, partly for use in connectionist models of cognitive function (McClelland and Rumelhart 1986, McLeod, Plunkett and Rolls 1998), and partly for use in applications (see Bishop (1995)).

In that perceptrons can correctly classify patterns provided only that they are linearly separable, but pattern associators are more restricted, perceptrons are more powerful learning devices than Hebbian pattern associators.

B.11.3.3 Gradient descent for neurons with continuous-valued outputs

We now consider networks trained by the delta (error correction) rule B.77, and having continuous-valued outputs. The activation function may be linear or non-linear, but provided that it is differentiable (in practice, does not include a sharp threshold), the network can be thought of as gradually decreasing the error on every learning trial, that is as performing some type of gradient descent down a continuous error function. The concept of gradient descent arises from defining an error ϵ for a neuron as

$$\epsilon = \sum_{\mu} (t^{\mu} - y^{\mu})^2 \tag{B.82}$$

where μ indexes the patterns learned by the neuron. The error function for a neuron in the direction of a particular weight would have the form shown in Fig. B.44. The delta rule can

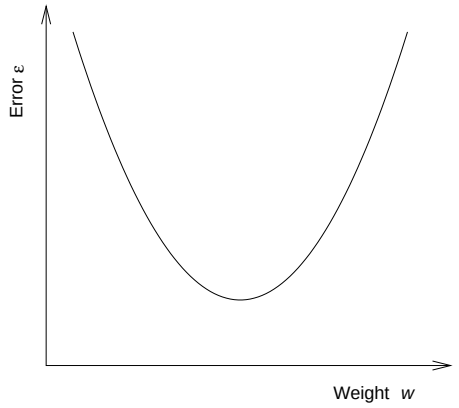


Fig. B.44 The error function ϵ for a neuron in the direction of a particular weight w.

be conceptualized as performing gradient descent of this error function, in that for the jth synaptic weight on the neuron

$$\delta w_j = -k\partial\epsilon/\partial w_j \tag{B.83}$$

where $\partial\epsilon/\partial w_j$ is just the slope of the error curve in the direction of w_j in Fig. B.44. This will decrease the weight if the slope is positive and increase the weight if the slope is negative. Given equation B.76, and recalling that $h = \sum_j x_j w_j$, equation B.44 becomes

$$\begin{aligned} \delta w_j &= -k\partial/\partial w_j \sum_\mu [(t^\mu - \mathrm{f}(h^\mu))^2] \\ &= 2k \sum_\mu [(t^\mu - y^\mu)]\mathrm{f}'(h)x_j \end{aligned} \tag{B.84}$$

where $\mathrm{f}'(h)$ is the derivative of the activation function. Provided that the activation function is monotonically increasing, its derivative will be positive, and the sign of the weight change will only depend on the mean sign of the error. Equation B.85 thus shows one way in which, from a gradient descent conceptualization, equation B.77 can be derived.

With linear output neurons, this gradient descent is proved to reach the correct mapping (see Hertz, Krogh and Palmer (1991)). (As with all single-layer networks with continuous-valued output neurons, a perfect solution is only found if the input patterns are linearly independent. If they are not, an optimal mapping is achieved, in which the sum of the squares of the errors is a minimum.) With non-linear output neurons (for example with a sigmoid activation function), the error surface may have local minima, and is not guaranteed to reach the optimal solution, although typically a near-optimal solution is achieved. Part of the power of this gradient descent conceptualization is that it can be applied to multilayer networks with neurons with non-linear but differentiable activation functions, for example with sigmoid activation functions (see Hertz, Krogh and Palmer (1991)).

B.11.4 Properties

The properties of single-layer networks trained with a delta rule (and of perceptrons) are similar to those of pattern associators trained with a Hebbian rule in many respects (see Section B.2). In particular, the properties of generalization and graceful degradation are similar, provided that (for both types of network) distributed representations are used. The

main differences are in the types of pattern that can be separated correctly, the learning speed (in that delta-rule networks can take advantage of many training trials to learn to separate patterns that could not be learned by Hebbian pattern associators), and in that the delta-rule network needs an error term to be supplied for each neuron, whereas an error term does not have to be supplied for a pattern associator, just an unconditioned or forcing stimulus. Given these overall similarities and differences, the properties of one-layer delta-rule networks are considered here briefly.

B.11.4.1 Generalization

During recall, delta-rule one-layer networks with non-linear output neurons produce appropriate outputs if a recall cue vector $\mathbf{x}_{\mathrm{r}}$ is similar to a vector that has been learned already. This occurs because the recall operation involves computing the dot (inner) product of the input pattern vector $\mathbf{x}_{\mathrm{r}}$ with the synaptic weight vector $\mathbf{w}_i$, so that the firing produced, y_i, reflects the similarity of the current input to the previously learned input pattern $\mathbf{x}$. Distributed representations are needed for this property. If two patterns that a delta-rule network has learned to separate are very similar, then the weights of the network will have been adjusted to force the different outputs to occur correctly. At the same time, this will mean that the way in which the network generalizes in the space between these two vectors will be very sharply defined. (Small changes in the input vector will force it to be classified one way or the other.)

B.11.4.2 Graceful degradation or fault tolerance

One-layer delta-rule networks show graceful degradation provided that the input patterns $\mathbf{x}$ are distributed.

B.11.4.3 Prototype extraction, extraction of central tendency, and noise reduction

These occur as for pattern autoassociators.

B.11.4.4 Speed

Recall is very fast in a one-layer pattern associator or perceptron, because it is a feedforward network (with no recurrent or lateral connections). Recall is also fast if the neuron has cell-like properties, because the stimulus input firings x_j ($j = 1, C$ axons) can be applied simultaneously to the synapses w_{ij}, and the activation h_i can be accumulated in one or two time constants of the synapses and dendrite (e.g. 10–20 ms) (see Section B.6.6). Whenever the threshold of the cell is exceeded, it fires. Thus, in effectively one time step, which takes the brain no more than 10–20 ms, all the output neurons of the delta-rule network can be firing with rates that reflect the input firing of every axon.

Learning is as fast ('one-shot') in perceptrons as in pattern associators if the input patterns are orthogonal. If the patterns are not orthogonal, so that the error correction rule has to work in order to separate patterns, then the network may take many trials to achieve the best solution (which will be perfect under the conditions described above).

B.11.4.5 Non-local learning rule

The learning rule is not truly local, as it is in pattern associators, autoassociators, and competitive networks, in that with one-layer delta-rule networks, the information required to change each synaptic weight is not available in the presynaptic terminal (reflecting the presynaptic rate) and the postsynaptic activation. Instead, an error for the neuron must be computed, possibly by another neuron, and then this error must be conveyed back to the postsynaptic neuron to provide the postsynaptic error term, which together with the presynaptic rate determines how much the synapse should change, as in equation B.77,

$$\delta w_{ij} = k(t_i - y_i)x_j$$

where $(t_i - y_i)$ is the error.

A rather special architecture would be required if the brain were to utilize delta-rule error-correcting learning. One such architecture might require each output neuron to be supplied with its own error signal by another neuron. The possibility (Albus 1971) that this is implemented in one part of the brain, the cerebellum, is described in Rolls and Treves (1998) Chapter 9. Another functional architecture would require each neuron to compute its own error by subtracting its current activation by its $\mathbf{x}$ inputs from another set of afferents providing the target activation for that neuron. A neurophysiological architecture and mechanism for this is not currently known.

B.11.4.6 Interference

Interference is less of a property of single-layer delta rule networks than of pattern autoassociators and autoassociators, in that delta rule networks can learn to separate patterns even when they are highly correlated. However, if patterns are not linearly independent, then the delta rule will learn a least mean squares solution, and interference can be said to occur.

B.11.4.7 Expansion recoding

As with pattern associators and autoassociators, expansion recoding can separate input patterns into a form that makes them learnable, or that makes learning more rapid with only a few trials needed, by delta rule networks. It has been suggested that this is the role of the granule cells in the cerebellum, which provide for expansion recoding by 1,000:1 of the mossy fibre inputs before they are presented by the parallel fibres to the cerebellar Purkinje cells (Marr 1969, Albus 1971, Rolls and Treves 1998).

B.11.4.8 Utility of single-layer error-correcting networks in information processing by the brain

In the cerebellum, each output cell, a Purkinje cell, has its own climbing fibre, that distributes from its inferior olive cell its terminals throughout the dendritic tree of the Purkinje cell. It is this climbing fibre that controls whether learning of the $\mathbf{x}$ inputs supplied by the parallel fibres onto the Purkinje cell occurs, and it has been suggested that the function of this architecture is for the climbing fibre to bring the error term to every part of the postsynaptic neuron (see Rolls and Treves (1998) Chapter 9). This rather special arrangement with each output cell apparently having its own teacher is probably unique in the brain, and shows the lengths to which the brain might need to go to implement a teacher for each output neuron. The requirement for error-correction learning is to have the neuron forced during a learning phase into a state that reflects its error while presynaptic afferents are still active, and rather special arrangements are needed for this.

B.12 Multilayer perceptrons: backpropagation of error networks

B.12.1 Introduction

So far, we have considered how error can be used to train a one-layer network using a delta rule. Minsky and Papert (1969) emphasized the fact that one-layer networks cannot solve certain classes of input–output mapping problems (as described above). It was clear then that these restrictions would not apply to the problems that can be solved by feedforward

multilayer networks, if they could be trained. A multilayer feedforward network has two or more connected layers, in which connections allow activity to be projected forward from one layer to the next, and in which there are no lateral connections within a layer. Such a multilayer network has an output layer (which can be trained with a standard delta rule using an error provided for each output neuron), and one or more hidden layers, in which the neurons do not receive separate error signals from an external teacher. (Because they do not provide the outputs of the network directly, and do not directly receive their own teaching error signal, these layers are described as hidden.) To solve an arbitrary mapping problem (in which the inputs are not linearly separable), a multilayer network could have a set of hidden neurons that would remap the inputs in such a way that the output layer can be provided with a linearly separable problem to solve using training of its weights with the delta rule. The problem was: how could the synaptic weights into the hidden neurons be trained in such a way that they would provide an appropriate representation? Minsky and Papert (1969) were pessimistic that such a solution would be found and, partly because of this, interest in computations in neural networks declined for many years. Although some work in neural networks continued in the following years (e.g. (Marr 1969, Marr 1970, Marr 1971, Willshaw and Longuet-Higgins 1969, Willshaw 1981, Malsburg 1973, Grossberg 1976a, Grossberg 1976b, Arbib 1964, Amari 1982, Amari, Yoshida and Kanatani 1977)), widespread interest in neural networks was revived by the type of approach to associative memory and its relation to human memory taken by the work described in the volume edited by Hinton and Anderson (1981), and by Kohonen (Kohonen 1977, Kohonen 1989). Soon after this, a solution to training a multilayer perceptron using backpropagation of error became widely known (Rumelhart, Hinton and Williams 1986b, Rumelhart, Hinton and Williams 1986a) (although earlier solutions had been found), and very great interest in neural networks and also in neural network approaches to cognitive processing (connectionism) developed (Rumelhart and McClelland 1986, McClelland and Rumelhart 1986, McLeod, Plunkett and Rolls 1998).

B.12.2 Architecture and algorithm

An introduction to the way in which a multilayer network can be trained by backpropagation of error is described next. Then we consider whether such a training algorithm is biologically plausible. A more formal account of the training algorithm for multilayer perceptrons (sometimes abbreviated MLP) is given by Rumelhart, Hinton and Williams (1986b), Rumelhart, Hinton and Williams (1986a), and Hertz, Krogh and Palmer (1991).

Consider the two-layer network shown in Fig. B.45. Inputs to the hidden neurons in layer A feed forward activity to the output neurons in layer B. The neurons in the network have a sigmoid activation function. One reason for such an activation function is that it is non-linear, and non-linearity is needed to enable multilayer networks to solve difficult (non-linearly separable) problems. (If the neurons were linear, the multilayer network would be equivalent to a one-layer network, which cannot solve such problems.) Neurons B1 and B2 of the output layer, B, are each trained using a delta rule and an error computed for each output neuron from the target output for that neuron when a given input pattern is being applied to the network. Consider now the error that needs to be used to train neuron A1 by a delta rule. This error clearly influences the error of neuron B1 in a way that depends on the magnitude of the synaptic weight from neuron A1 to B1; and on the error of neuron B2 in a way that depends on the magnitude of the synaptic weight from neuron A1 to B2. In other words, the error for neuron A1 depends on:

the weight from A1 to B1 (w_{11}) · error of neuron B1
\+ the weight from A1 to B2 (w_{21}) · error of neuron B2.

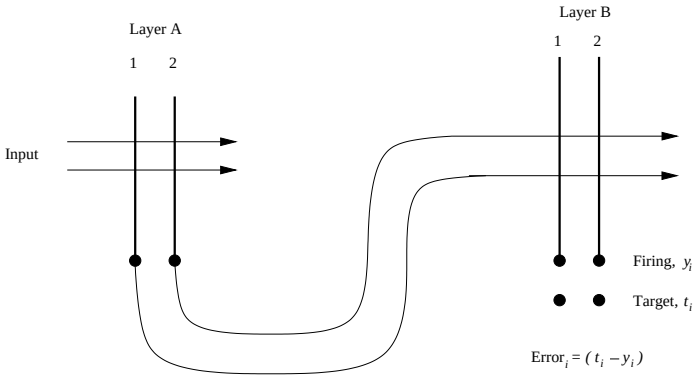


Fig. B.45 A two-layer perceptron. Inputs are applied to layer A through modifiable synapses. The outputs from layer A are applied through modifiable synapses to layer B. Layer B can be trained using a delta rule to produce firing y_i which will approach the target t_i. It is more difficult to modify the weights in layer A, because appropriate error signals must be backpropagated from layer B.

In this way, the error calculation can be propagated backwards through the network to any neuron in any hidden layer, so that each neuron in the hidden layer can be trained, once its error is computed, by a delta rule (which uses the computed error for the neuron and the presynaptic firing at the synapse to correct each synaptic weight). For this to work, the way in which each neuron is activated and sends a signal forward must be continuous (not binary), so that the extent to which there is an error in, for example, neuron B1 can be related back in a graded way to provide a continuously variable correction signal to previous stages. This is one of the requirements for enabling the network to descend a continuous error surface. The activation function must be non-linear (e.g. sigmoid) for the network to learn more than could be learned by a single-layer network. (Remember that a multilayer linear network can always be made equivalent to a single-layer linear network, and that there are some problems that cannot be solved by single-layer networks.) For the way in which the error of each output neuron should be taken into account to be specified in the error correction rule, the position at which the output neuron is operating on its activation function must also be taken into account. For this, the slope of the activation function is needed, and because the slope is needed, the activation function must be differentiable. Although we indicated use of a sigmoid activation function, other activation functions that are non-linear and monotonically increasing (and differentiable) can be used. (For further details, see Rumelhart, Hinton and Williams (1986b), Rumelhart, Hinton and Williams (1986a), and Hertz, Krogh and Palmer (1991)).

B.12.3 Properties of multilayer networks trained by error backpropagation

B.12.3.1 Arbitrary mappings

Arbitrary mappings of non-linearly separable patterns can be achieved. For example, such networks can solve the XOR problem, and parity problems in general of which XOR is a special case. (The parity problem is to determine whether the sum of the (binary) bits in a vector is odd or even.) Multilayer feedforward backpropagation of error networks are not guaranteed to converge to the best solution, and may become stuck in local minima in the error surface. However, they generally perform very well.

B.12.3.2 Fast operation

The network operates as a feedforward network, without any recurrent or feedback processing. Thus (once it has learned) the network operates very quickly, with a time proportional to the number of layers.

B.12.3.3 Learning speed

The learning speed can be very slow, taking many thousands of trials. The network learns to gradually approximate the correct input–output mapping required, but the learning is slow because of the credit assignment problem for neurons in the hidden layers. The credit assignment problem refers to the issue of how much to correct the weights of each neuron in the hidden layer. As the example above shows, the error for a hidden neuron could influence the errors of many neurons in the output layers, and the error of each output neuron reflects the error from many hidden neurons. It is thus difficult to assign credit (or blame) on any single trial to any particular hidden neuron, so an error must be estimated, and the net run until the weights of the crucial hidden neurons have become altered sufficiently to allow good performance of the network. Another factor that can slow learning is that if a neuron operates close to a horizontal part of its activation function, then the output of the neuron will depend rather little on its activation, and correspondingly the error computed to backpropagate will depend rather little on the activation of that neuron, so learning will be slow.

More general approaches to this issue suggest that the number of training trials for such a network will (with a suitable training set) be of the same order of magnitude as the number of synapses in the network (see Cortes, Jaeckel, Solla, Vapnik and Denker (1996)).

B.12.3.4 Number of hidden neurons and generalization

Backpropagation networks are generally intended to discover regular mappings between the input and output, that is mappings in which generalization will occur usefully. If there were one hidden neuron for every combination of inputs that had to be mapped to an output, then this would constitute a look-up table, and no generalization between similar inputs (or inputs not yet received) would occur. The best way to ensure that a backpropagation network learns the structure of the problem space is to set the number of neurons in the hidden layers close to the minimum that will allow the mapping to be implemented. This forces the network not to operate as a look-up table. A problem is that there is no general rule about how many hidden neurons are appropriate, given that this depends on the types of mappings required. In practice, these networks are sometimes trained with different numbers of hidden neurons, until the minimum number required to perform the required mapping has been approximated.

B.13 Biologically plausible networks

Given that the error for a hidden neuron in an error backpropagation network is calculated by propagating backwards information based on the errors of all the output neurons to which a hidden neuron is connected, and all the relevant synaptic weights, and the activations of the output neurons to define the part of the activation function on which they are operating, it is implausible to suppose that the correct information to provide the appropriate error for each hidden neuron is propagated backwards between real neurons. A hidden neuron would have to 'know', or receive information about, the errors of all the neurons to which it is connected, and its synaptic weights to them, and their current activations. If there were more than one hidden layer, this would be even more difficult.

To expand on the difficulties: first, there would have to be a mechanism in the brain for providing an appropriate error signal to each output neuron in the network. With the possible

exception of the cerebellum, an architecture where a separate error signal could be provided for each output neurons is difficult to identify in the brain. Second, any retrograde passing of messages across multiple-layer forward-transmitting pathways in the brain that could be used for backpropagation seems highly implausible, not only because of the difficulty of getting the correct signal to be backpropagated, but also because retrograde signals passed by axonal transport in a multilayer net would take days to arrive, long after the end of any feedback given in the environment indicating a particular error. Third, as noted in Section 1.11, the backprojection pathways that are present in the cortex seem suited to perform recall, and this would make it difficult for them also to have the correct strength to carry the correct error signal.

A problem with the backpropagation of error approach in a biological context is thus that in order to achieve their competence, backpropagation networks use what is almost certainly a learning rule that is much more powerful than those that could be implemented biologically, and achieve their excellent performance by performing the mapping though a minimal number of hidden neurons. In contrast, real neuronal networks in the brain probably use much less powerful learning rules, in which errors are not propagated backwards, and at the same time have very large numbers of hidden neurons, without the bottleneck that helps to provide backpropagation networks with their good performance. A consequence of these differences between backpropagation and biologically plausible networks may be that the way in which biological networks solve difficult problems may be rather different from the way in which backpropagation networks find mappings. Thus the solutions found by connectionist systems may not always be excellent guides to how biologically plausible networks may perform on similar problems. Part of the challenge for future work is to discover how more biologically plausible networks than backpropagation networks can solve comparably hard problems, and then to examine the properties of these networks, as a perhaps more accurate guide to brain computation.

As stated above, it is a major challenge for brain research to discover whether there are algorithms that will solve comparably difficult problems to backpropagation, but with a local learning rule. Such algorithms may be expected to require many more hidden neurons than backpropagation networks, in that the brain does not appear to use information bottlenecks to help it solve difficult problems. The issue here is that much of the power of backpropagation algorithms arises because there is a minimal number of hidden neurons to perform the required mapping using a final one-layer delta-rule network. Useful generalization arises in such networks because with a minimal number of hidden neurons, the net sets the representation they provide to enable appropriate generalization. The danger with more hidden neurons is that the network becomes a look-up table, with one hidden neuron for every required output, and generalization when the inputs vary becomes poor. The challenge is to find a more biologically plausible type of network that operates with large numbers of neurons, and yet that still provides useful generalization. An example of such an approach is described in Chapter 4.

B.14 Contrastive Hebbian learning: the Boltzmann machine

In a move towards a learning rule that is more local than in backpropagation networks, yet that can solve similar mapping problems in a multilayer architecture, we describe briefly contrastive Hebbian learning. The multilayer architecture has forward connections through the network to the output layer, and a set of matching backprojections from the output layer through each of the hidden layers to the input layer. The forward connection strength between

any pair of neurons has the same value as the backward connection strength between the same two neurons, resulting in a symmetric set of forward and backward connection strengths. An input pattern is applied to the multilayer network, and an output is computed using normal feedforward activation processing with neurons with a sigmoid (non-linear and monotonically increasing) activation function. The output firing then via the backprojections is used to create firing of the input neurons. This process is repeated until the firing rates settle down, in an iterative way (which is similar to the settling of the autoassociative nets described in Section B.3). After settling, the correlations between any two neurons are remembered, for this type of unclamped operation, in which the output neurons fire at the rates that the process just described produces. The correlations reflect the normal presynaptic and postsynaptic terms used in the Hebb rule, e.g. $(x_j y_i)^{\mathrm{uc}}$, where 'uc' refers to the unclamped condition, and as usual x_j is the firing rate of the input neuron, and y_i is the activity of the receiving neuron. The output neurons are then clamped to their target values, and the iterative process just described is repeated, to produce for every pair of synapses in the network $(x_j y_i)^{\mathrm{c}}$, where the c refers now to the clamped condition. An error correction term for each synapse is then computed from the difference between the remembered correlation of the unclamped and the clamped conditions, to produce a synaptic weight correction term as follows:

$$\delta w_{ij} = k[(x_j y_i)^{\mathrm{c}} - (x_j y_i)^{\mathrm{uc}}], \tag{B.85}$$

where k is a learning rate constant. This process is then repeated for each input pattern to output pattern to be learned. The whole process is then repeated many times with all patterns until the output neurons fire similarly in the clamped and unclamped conditions, that is until the errors have become small. Further details are provided by Hinton and Sejnowski (1986). The version described above is the mean field (or deterministic) Boltzmann machine (Peterson and Anderson 1987, Hinton 1989). More traditionally, a Boltzmann machine updates one randomly chosen neuron at a time, and each neuron fires with a probability that depends on its activation (Ackley, Hinton and Sejnowski 1985, Hinton and Sejnowski 1986). The latter version makes fewer theoretical assumptions, while the former may operate an order of magnitude faster (Hertz, Krogh and Palmer 1991).

In terms of biological plausibility, it certainly is the case that there are backprojections between adjacent cortical areas (see Chapter 1). Indeed, there are as many backprojections between adjacent cortical areas as there are forward projections. The backward projections seem to be more diffuse than the forward projections, in that they connect to a wider region of the preceding cortical area than the region that sends the forward projections. If the backward and the forward synapses in such an architecture were Hebb-modifiable, then there is a possibility that the backward connections would be symmetric with the forward connections. Indeed, such a connection scheme would be useful to implement top-down recall, as summarized in Chapter 2 and described by Rolls and Treves (1998) in their Chapter 6. What seems less biologically plausible is that after an unclamped phase of operation, the correlations between all pairs of neurons would be remembered, there would then be a clamped phase of operation with each output neuron clamped to the required rate for that particular input pattern, and then the synapses would be corrected by an error correction rule that would require a comparison of the correlations between the neuronal firing of every pair of neurons in the unclamped and clamped conditions.

Although this algorithm has the disadvantages that it is not very biologically plausible, and does not operate as well as standard backpropagation, it has been made use of by O'Reilly and Munakata (2000) in approaches to connectionist modelling in cognitive neuroscience.

B.15 Reinforcement learning

In supervised networks, an error signal is provided for each output neuron in the network, and whenever an input to the network is provided, the error signals specify the magnitude and direction of the error in the output produced by each neuron. These error signals are then used to correct the synaptic weights in the network in such a way that the output errors for each input pattern to be learned gradually diminish over trials (see Sections B.11 and B.12). These networks have an architecture that might be similar to that of the pattern associator shown in Fig. B.1, except that instead of an unconditioned stimulus, there is an error correction signal provided for each output neuron. Such a network trained by an error correcting (or delta) rule is known as a one-layer perceptron. The architecture is not very plausible for most brain regions, in that it is not clear how an individual error signal could be computed for each of thousands of neurons in a network, and fed into each neuron as its error signal and then used in a delta rule synaptic correction (see Section B.11).

The architecture can be generalized to a multilayer feedforward architecture with many layers between the input and output (Rumelhart, Hinton and Williams 1986a), but the learning is very non-local and rather biologically implausible (see Section B.12), in that an error term (magnitude and direction) for each neuron in the network must be computed from the errors and synaptic weights of all subsequent neurons in the network that any neuron influences, usually on a trial-by-trial basis, by a process known as error backpropagation. Thus although computationally powerful, an issue with perceptrons and multilayer perceptrons that makes them generally biologically implausible for many brain regions is that a separate error signal must be supplied for each output neuron, and that with multilayer perceptrons, computed error backpropagation must occur.

When operating in an environment, usually a simple binary or scalar signal representing success or failure of the whole network or organism is received. This is usually action-dependent feedback that provides a single evaluative measure of the success or failure. Evaluative feedback tells the learner whether or not, and possibly by how much, its behaviour has improved; or it provides a measure of the 'goodness' of the behaviour. Evaluative feedback does not directly tell the learner what it should have done, and although it may provide an index of the degree (i.e. magnitude) of success, it does not include directional information telling the learner how to change its behaviour towards a target, as does error-correction learning (see Barto (1995)). Partly for this reason, there has been some interest in networks that can be taught with such a single reinforcement signal. In this Section (B.15), approaches to such networks are described. It is noted that such networks are classified as reinforcement networks in which there is a single teacher, and that these networks attempt to perform an optimal mapping between an input vector and an output neuron or set of neurons. They thus solve the same class of problems as single layer and multilayer perceptrons. They should be distinguished from pattern-association networks in the brain, which might learn associations between previously neutral stimuli and primary reinforcers such as taste (signals which might be interpreted appropriately by a subsequent part of the brain), but do not attempt to produce arbitrary mappings between an input and an output, using a single reinforcement signal.

A class of problems to which such reinforcement networks might be applied are motor-control problems. It was to such a problem that Barto and Sutton (Barto 1985, Sutton and Barto 1981) applied a reinforcement learning algorithm, the associative reward–penalty algorithm described next. The algorithm can in principle be applied to multilayer networks, and the learning is relatively slow. The algorithm is summarized in Section B.15.1 and by Hertz, Krogh and Palmer (1991). More recent developments in reinforcement learning (Sections B.15.2 and B.15.3) are described by Sutton and Barto (1998) and reviewed by Dayan and Abbott (2001).

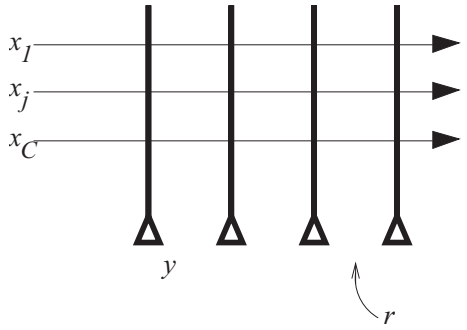


Fig. B.46 A network trained by a single reinforcement input r. The inputs to each neuron are $x_j, j = 1, C$; and y is the output of one of the output neurons.

B.15.1 Associative reward–penalty algorithm of Barto and Sutton

The terminology of Barto and Sutton is followed here (see Barto (1985)).

B.15.1.1 Architecture

The architecture, shown in Fig. B.46, uses a single reinforcement signal, r, = +1 for reward, and –1 for penalty. The inputs x_j take real (continuous) values. The output of a neuron, y, is binary, +1 or –1. The weights on the output neuron are designated w_j.

B.15.1.2 Operation

1. An input vector is applied to the network, and produces activation, h, in the normal way as follows:

$$h = \sum_{j=1}^{C} x_j w_j \tag{B.86}$$

where $\sum_{j=1}^{C}$ indicates that the sum is over the C input axons (or connections) indexed by j to each neuron.

2. The output y is calculated from the activation with a noise term η included. The principle of the network is that if the added noise on a particular trial helps performance, then whatever change it leads to should be incorporated into the synaptic weights, in such a way that the next time that input occurs, the performance is improved.

$$y = \begin{cases} +1 & \text{if } h + \eta \geq 0, \\ -1 & \text{else.} \end{cases} \tag{B.87}$$

where η = the noise added on each trial.

3. Learning rule. The weights are changed as follows:

$$\delta w_j = \begin{cases} \rho(y - \mathrm{E}[y|h])x_j & \text{if } r = +1, \\ \rho\lambda(-y - \mathrm{E}[y|h])x_j & \text{if } r = -1. \end{cases} \tag{B.88}$$

ρ and λ are learning-rate constants. (They are set so that the learning rate is higher when positive reinforcement is received than when negative reinforcement is received.) $\mathrm{E}[y|h]$ is the expectation of y given h (usually a sigmoidal function of h with the range ± 1). $\mathrm{E}[y|h]$ is a (continuously varying) indication of how the neuron usually responds to the current input

pattern, i.e. if the actual output y is larger than normally expected, by computing $h = \sum w_j x_j$, because of the noise term, and the reinforcement is +1, increase the weight from x_j; and vice versa. The expectation could be the prediction generated before the noise term is incorporated.

This network combines an associative capacity with its properties of generalization and graceful degradation, with a single 'critic' or error signal for the whole network (Barto 1985). [The term $y - \mathrm{E}[y|h]$ in Equation B.88 can be thought of as an error for the output of the neuron: it is the difference between what occurred, and what was expected to occur. The synaptic weight is adjusted according to the sign and magnitude of the error of the postsynaptic firing, multiplied by the presynaptic firing, and depending on the reinforcement r received. The rule is similar to a Hebb synaptic modification rule (Equation B.2), except that the postsynaptic term is an error instead of the postsynaptic firing rate, and the learning is modulated by the reinforcement.] The network can solve difficult problems (such as balancing a pole by moving a trolley that supports the pole from side to side, as the pole starts to topple). Although described for single-layer networks, the algorithm can be applied to multilayer networks. The learning rate is very slow, for there is a single reinforcement signal on each trial for the whole network, not a separate error signal for each neuron in the network as is the case in a perceptron trained with an error rule (see Section B.11).

This associative reward–penalty reinforcement-learning algorithm is certainly a move towards biological relevance, in that learning with a single reinforcer can be achieved. That single reinforcer might be broadcast throughout the system by a general projection system. It is not clear yet how a biological system might store the expected output $\mathrm{E}[y|h]$ for comparison with the actual output when noise has been added, and might take into account the sign and magnitude of this difference. Nevertheless, this is an interesting algorithm, which is related to the temporal difference reinforcement learning algorithm described in Section B.15.3.

B.15.2 Reward prediction error or delta rule learning, and classical conditioning

In classical or Pavlovian associative learning, a number of different types of association may be learned (see Section 3.2.1). This type of associative learning may be performed by networks with the general architecture and properties of pattern associators (see Section B.2 and Fig. B.1). However, the time course of the acquisition and extinction of these associations can be expressed concisely by a modified type of learning rule in which an error correction term is used (introduced in Section B.15.1), rather than the postsynaptic firing y itself as in Equation B.2. Use of this modified, error correction, type of learning also enables some of the properties of classical conditioning to be explained (see Dayan and Abbott (2001) for review), and this type of learning is therefore described briefly here. The rule is known in learning theory as the Rescorla–Wagner rule, after Rescorla and Wagner (1972).

The Rescorla–Wagner rule is a version of error correction or delta-rule learning (see Section B.11), and is based on a simple linear prediction of the expected reward value, denoted by v, associated with a stimulus representation x ($x = 1$ if the stimulus is present, and $x = 0$ if the stimulus is absent). The **expected reward value** v is expressed as the input stimulus variable x multiplied by a weight w

$$v = wx. \tag{B.89}$$

The **reward prediction error** is the difference between the expected reward value v and the actual **reward outcome** r obtained, i.e.

$$\Delta = r - v \tag{B.90}$$

where Δ is the reward prediction error. The value of the weight w is learned by a rule designed to minimize the expected squared error $< (r - v)^2 >$ between the actual reward outcome r and the predicted reward value v. The angle brackets indicate an average over the presentations of the stimulus and reward. The delta rule will perform the required type of learning:

$$\delta w = k(r - v)x \tag{B.91}$$

where δw is the change of synaptic weight, k is a constant that determines the learning rate, and the term $(r-v)$ is the reward prediction error Δ (equivalent to the error in the postsynaptic firing, rather than the postsynaptic firing y itself as in Equation B.2). Application of this rule during conditioning with the stimulus x presented on every trial results in the weight w approaching the asymptotic limit $w = r$ exponentially over trials as the error Δ becomes zero. In extinction, when $r = 0$, the weight (and thus the output of the system) exponentially decays to $w = 0$. This rule thus helps to capture the time course over trials of the acquisition and extinction of conditioning. The rule also helps to account for a number of properties of classical conditioning, including blocking, inhibitory conditioning, and overshadowing (see Dayan and Abbott (2001)).

How this functionality is implemented in the brain is not yet clear. We consider one suggestion (Schultz et al. 1995b, Schultz 2004, Schultz 2006) after we introduce a further sophistication of reinforcement learning which allows the time course of events within a trial to be taken into account.

B.15.3 Temporal Difference (TD) learning

An important advance in the area of reinforcement learning was the introduction of algorithms that allow for learning to occur when the reinforcement is delayed or received over a number of time steps, and which allow effects within a trial to be taken into account (Sutton and Barto 1998, Sutton and Barto 1990). A solution to these problems is the addition of an adaptive critic that learns through a time difference (TD) algorithm how to predict the future value of the reinforcer. The time difference algorithm takes into account not only the current reinforcement just received, but also a temporally weighted average of errors in predicting future reinforcements. The temporal difference error is the error by which any two temporally adjacent error predictions are inconsistent (see Barto (1995)). The output of the critic is used as an effective reinforcer instead of the instantaneous reinforcement being received (see Sutton and Barto (1998), Sutton and Barto (1990), and Barto (1995)). This is a solution to the temporal credit assignment problem, and enables future rewards to be predicted. Summaries are provided by Doya (1999), Schultz, Dayan and Montague (1997), and Dayan and Abbott (2001).

In reinforcement learning, a learning agent takes an *action* $\mathbf{u}(t)$ in response to the *state* $\mathbf{x}(t)$ of the environment, which results in the change of the state

$$\mathbf{x}(t+1) = \mathrm{F}(\mathbf{x}(t), \mathbf{u}(t)), \tag{B.92}$$

and the delivery of the reinforcement signal, or *reward*

$$r(t+1) = \mathrm{R}(\mathbf{x}(t), \mathbf{u}(t)). \tag{B.93}$$

In the above equations, $\mathbf{x}$ is a vector representation of inputs x_j, and equation B.92 indicates that the next state $\mathbf{x}(t+1)$ at time $(t+1)$ is a function F of the state at the previous time step of the inputs and actions at that time step in a closed system. In equation B.93 the reward at the next time step is determined by a reward function R which uses the current sensory inputs and action taken. The time t may refer to time within a trial.

The goal is to find a *policy* function G which maps sensory inputs **x** to actions

$$\mathbf{u}(t) = \mathrm{G}(\mathbf{x}(t)) \tag{B.94}$$

which maximizes the cumulative sum of the rewards based on the sensory inputs.

The current action $\mathbf{u}(t)$ affects all future states and accordingly all future rewards. The maximization is realized by the use of the *value function* V of the states to predict, given the sensory inputs **x**, the cumulative sum (possibly discounted as a function of time) of all future rewards $\mathrm{V}(\mathbf{x})$ (possibly within a learning trial) as follows:

$$\mathrm{V}(\mathbf{x}) = E[r(t+1) + \gamma r(t+2) + \gamma^2 r(t+3) + ...] \tag{B.95}$$

where $r(t)$ is the reward at time t, and $E[\cdot]$ denotes the expected value of the sum of future rewards up to the end of the trial. $0 \leq \gamma \leq 1$ is a discount factor that makes rewards that arrive sooner more important than rewards that arrive later, according to an exponential decay function. (If $\gamma = 1$ there is no discounting.) It is assumed that the presentation of future cues and rewards depends only on the current sensory cues and not the past sensory cues. The right-hand side of equation B.95 is evaluated for the dynamics in equations B.92–B.94 with the initial condition $\mathbf{x}(t) = \mathbf{x}$. The two basic ingredients in reinforcement learning are the estimation (which we term $\hat{\mathrm{V}}$) of the value function V, and then the improvement of the policy or action **u** using the value function (Sutton and Barto 1998).

The basic algorithm for learning the value function is to minimize the *temporal difference* (TD) *error* $\Delta(t)$ for time t within a trial, and this is computed by a 'critic' for the estimated value predictions $\hat{\mathrm{V}}(\mathbf{x}(t))$ at successive time steps as

$$\Delta(t) = [r(t) + \gamma \hat{\mathrm{V}}(\mathbf{x}(t))] - \hat{\mathrm{V}}(\mathbf{x}(t-1)) \tag{B.96}$$

where $\hat{\mathrm{V}}(\mathbf{x}(t)) - \hat{\mathrm{V}}(\mathbf{x}(t-1))$ is the difference in the reward value prediction at two successive time steps, giving rise to the terminology temporal difference learning. If we introduce the term $\hat{v}$ as the estimate of the cumulated reward by the end of the trial, we can define it as a function $\hat{\mathrm{V}}$ of the current sensory input $\mathbf{x}(t)$, i.e. $\hat{v} = \hat{\mathrm{V}}(\mathbf{x})$, and we can also write equation B.96 as

$$\Delta(t) = r(t) + \gamma \hat{v}(t) - \hat{v}(t-1) \tag{B.97}$$

which draws out the fact that it is differences at successive timesteps in the reward value predictions $\hat{v}$ that are used to calculate Δ.

$\Delta(t)$ is used to improve the estimates $\hat{v}(t)$ by the 'critic', and can also be used (by an 'actor') to learn appropriate actions.

For example, when the value function is represented (in the critic) as

$$\hat{\mathrm{V}}(\mathbf{x}(t)) = \sum_{j=1}^{n} w_j^{\mathrm{C}} x_j(t) \tag{B.98}$$

the learning algorithm for the (value) weight w_j^{C} in the critic is given by

$$\delta w_j^{\mathrm{C}} = k_c \Delta(t) x_j(t-1) \tag{B.99}$$

where δw_j^{C} is the change of synaptic weight, k_c is a constant that determines the learning rate for the sensory input x_j, and $\Delta(t)$ is the Temporal Difference error at time t. Under certain conditions this learning rule will cause the estimate $\hat{v}$ to converge to the true value (Dayan and Sejnowski 1994).

A simple way of improving the policy of the actor is to take a stochastic action

$$u_i(t) = \mathrm{g}(\sum_{j=1}^{n} w_{ij}^{\mathrm{A}} x_j(t) + \mu_i(t)), \tag{B.100}$$

where g() is a scalar version of the policy function G, w_{ij}^{A} is a weight in the actor, and $\mu_i(t)$ is a noise term. The TD error $\Delta(t)$ as defined in equation B.96 then signals the unexpected delivery of the reward $r(t)$ or the increase in the state value $\hat{\mathrm{V}}(\mathbf{x}(t))$ above expectation, possibly due to the previous choice of action $u_i(t-1)$. The learning algorithm for the action weight w_{ij}^{A} in the actor is given by

$$\delta w_{ij}^{\mathrm{A}} = k_a \Delta(t)(u_i(t-1) - < u_i >) x_j(t-1), \tag{B.101}$$

where $< u_i >$ is the average level of the action output, and k_a is a learning rate constant in the actor.

Thus, the TD error $\Delta(t)$, which signals the error in the reward prediction at time t, works as the main teaching signal in both learning the value function (implemented in the critic), and the selection of actions (implemented in the actor). The usefulness of a separate critic is that it enables the TD error to be calculated based on the difference in reward value predictions at two successive time steps as shown in equation B.96.

The algorithm has been applied to modelling the time course of classical conditioning (Sutton and Barto 1990). The algorithm effectively allows the future reinforcement predicted from past history to influence the responses made, and in this sense allows behaviour to be guided not just by immediate reinforcement, but also by 'anticipated' reinforcements. Different types of temporal difference learning are described by Sutton and Barto (1998). An application is to the analysis of decisions when future rewards are discounted with respect to immediate rewards (Dayan and Abbott 2001, Tanaka, Doya, Okada, Ueda, Okamoto and Yamawaki 2004). Another application is to the learning of sequences of actions to take within a trial (Suri and Schultz 1998).

The possibility that dopamine neuron firing may provide an error signal useful in training neuronal systems to predict reward has been discussed in Section 3.10.2.5. It has been proposed that the firing of the dopamine neurons can be thought of as an error signal about reward prediction, in that the firing occurs in a task when a reward is given, but then moves forward in time within a trial to the time when a stimulus is presented that can be used to predict when the taste reward will be obtained (Schultz et al. 1995b) (see Fig. 3.63). The argument is that there is no prediction error when the taste reward is obtained if it has been signalled by a preceding conditioned stimulus, and that is why the dopamine midbrain neurons do not respond at the time of taste reward delivery, but instead, at least during training, to the onset of the conditioned stimulus (Waelti, Dickinson and Schultz 2001). If a different conditioned stimulus is shown that normally predicts that no taste reward will be given, there is no firing of the dopamine neurons to the onset of that conditioned stimulus.

This hypothesis has been built into models of learning in which the error signal is used to train synaptic connections in dopamine pathway recipient regions (such as presumably the striatum and orbitofrontal cortex) (Houk, Adams and Barto 1995, Schultz 2004, Schultz, Dayan and Montague 1997, Waelti, Dickinson and Schultz 2001, Dayan and Abbott 2001). Some difficulties with the hypothesis are discussed in Section 3.10.2.5 on page 243. The difficulties include the fact that dopamine is released in large quantities by aversive stimuli (see Section 3.10.2.5); that error computations for differences between the expected reward and the actual reward received on a trial are computed in the primate orbitofrontal cortex, where expected reward, actual reward, and error neurons are all found, and lesions of which

impair the ability to use changes in reward contingencies to reverse behaviour (see Section 3.6.5.5); that the tonic, sustained, firing of the dopamine neurons in the delay period of a task with probabilistic rewards may reflect reward uncertainty, and not the expected reward, nor the magnitude of the prediction error (see Section 3.10.2.5 and Shizgal and Arvanitogiannis (2003)); and that reinforcement learning is suited to setting up connections that might be required in fixed tasks such as motor habit or sequence learning, for reinforcement learning algorithms seek to set weights correctly in an 'actor', but are not suited to tasks where rules must be altered flexibly, as in rapid one-trial reversal, for which a very different type of mechanism is described in Chapter 9.

The temporal difference approach to reinforcement learning has a weakness that although it can be used to predict internal signals during reinforcement learning in some tasks, it does not directly address learning with respect to actions that are not taken. Q-learning can be considered as an extension of TD learning which adds additional terms to take into account signals from actions that are not taken, for example information gained by observation of others (Montague, King-Casas and Cohen 2006).

Overall, reinforcement learning algorithms are certainly a move towards biological relevance, in that learning with a single reinforcer can be achieved in systems that might learn motor habits or fixed sequences. Whether a single prediction error is broadcast throughout a neural system by a general projection system, such as the dopamine pathways in the brain, which distribute to large parts of the striatum and the prefrontal cortex, remains to be clearly established (see further Chapters 3 and 10).

B.16 Forgetting in associative neural networks and in the brain, and memory reconsolidation

Forgetting is an important feature of associative neural networks and the brain, and is important in their successful operation. There are a number of different mechanisms for forgetting, and a number of different reasons why forgetting is important in particular classes of network.

Consider attractor, that is autoassociation, networks, which are used for short-term memory, episodic memory, etc. These networks have a critical storage capacity, as described in Section B.3, and if this is exceeded, most of the memories in the network become unretrievable. It is therefore crucial to have a mechanism for forgetting in these networks.

One mechanism is decay of synaptic strength. The simple forgetting mechanism is just an exponential decay of the synaptic value back to its baseline, which may be exponential in time or in the number of learning changes incurred (Nadal, Toulouse, Changeux and Dehaene 1986). This form of forgetting does not require keeping track of each individual change and preserves linear superposition, that is each memory is added linearly to previous memories, as provided for by equation B.9 on page 561. In calculating the storage capacity of pattern associators and of autoassociators, the inclusion or exclusion of simple exponential decay does not change significantly the calculation of capacity, and only results in a different prefactor (one 2.7 times the other) for the maximum number of associations that can be stored. Therefore a forgetting mechanism as simple as exponential decay is normally omitted, and one has just to remember that its inclusion would reduce the critical capacity obtained to roughly 0.37 of that without the decay. This type of memory has been called a palimpsest.

Another form of forgetting, which is potentially interesting in terms of biological plausibility, is implemented by setting limits to the range allowed for each synaptic strength or weight (Parisi 1986). As a particular synapse hits the upper or lower limit on its range, it is taken to be unable to further modify in the direction that would take it beyond the limit. Only after modifications in the opposite direction have taken it away from the limit does the

synapse regain its full plasticity. A forgetting scenario of this sort requires a slightly more complicated formal analysis (since it violates linear superposition of different memories), but it effectively results in a progressive, exponential degradation of older memories similar to that produced by straight exponential decay of synaptic weights.

A combined forgetting rule that may be particularly attractive in the context of modelling synapses between pyramidal cells is implemented by setting a lower limit, that is, zero, on the excitatory weight (just requiring that the associated conductance be a non-negative quantity!), and allowing exponential decay of the value of the weight with time. Again, this type of combined forgetting rule places demands on the analytical techniques that have to be used to calculate the storage capacity, but leads to functionally similar effects (Rolls and Treves 1998).

Two conditions under which synaptic strengths may decrease are described in Fig. 1.6. Heterosynaptic long-term depression, which can occur for inactive presynaptic terminals on active postsynaptic neurons, can be useful computationally in the following ways. First, it is a useful way to subtract the effect of the mean presynaptic firing rate of each neuron in a pattern associator, which removes the effect of the mean firing rate in increasing the correlation between different input patterns used in the network, as is made evident in equation B.7. This orthogonalizing effect helps to maximize the storage capacity of pattern associators. The same situation applies to autoassociation (attractor) networks (see equation B.13). This decrease of firing rate for inactive inputs may of course result in some loss of memories previously stored in these association networks, if a particular synapse had been strengthened as part of a previous memory. Homosynaptic long-term depression might contribute to a similar function.

Heterosynaptic long-term depression (LTD) is also useful in competitive networks, for it provides a way for the synaptic weight vectors of different neurons to be kept of approximately equal length, and this is important to ensure that the different categories of input patterns find different output neurons to activate. In the case of competitive networks, the appropriate effect is achieved if the subtractive term in the presynaptic component depends on the existing strength of the synapse, as shown in equation B.22. The fact that in studies of LTD it is sometimes remarked that LTD is easier to demonstrate after LTP has been induced lends support to the likelihood that LTD of the form indicated in equation B.22 that depends on the existing synaptic strength is implemented in the brain. Because of the computational significance of LTD that depends on the existing strength of synapses for competitive networks, it would be useful to see further experimental exploration of this.

Forgetting in attractor networks takes two forms that can be clearly distinguished. One is that the current attractor state implemented by the continuing firing of one set of neurons in the network is labile, and may be interrupted by a strong new input which forces the network into a new attractor state, or may be interrupted by quenching effects through non-specific effects implemented through for example inhibitory neurons (Chapter 9). Both these effects could be facilitated by synaptic or neuronal adaptation, as described in Chapter 9.

The second form of forgetting in attractor networks is of the strengthened synaptic connections that specify each of the different attractor memories in the network. If the network is to be used for large numbers of different short-term memories, then these synaptic weights must decay or be overwritten by LTD as described above. This is likely to be required for short-term memory networks in the prefrontal cortex which must adapt themselves to be capable of storing the particular stimuli and actions that may be required in particular tasks (see e.g. Chapter 8), in short-term memory networks that implement the visuo-spatial scratchpad, etc. We may note that because short-term memory networks have these two different aspects, once an attractor set of synaptic connections has been imprinted in a network by synaptic modification, then no further synaptic modification is necessary to use that network repeatedly for holding the neurons in one of the stored attractors active to implement the short-term memory in a delay period (Kesner and Rolls 2001).

Forgetting may be less important in semantic networks. (Semantic networks store structured information with appropriate associative links and hierarchical structure, for example a family tree, or one's geographical knowledge (McClelland and Rumelhart 1986).) We may note that because any one associative memory network has a memory capacity that is related to the number of associative synapses onto each neuron from other neurons in the network, semantic memory is likely to involve connections between modules in the cortex, where each module might be defined by a 1–3 mm region of cortex with high local connectivity between the neurons. For this type of memory, forgetting is not so much the requirement as incorporating new semantic knowledge, that is making new appropriate links, and perhaps weakening existing links. In this scenario, the fact that when a memory is retrieved, as would occur when a sematic memory is being updated or extended, then it may need to be reconsolidated, suggests a possible useful function for memory reconsolidation (see below), as it could facilitate the restructuring of a semantic memory. In contrast, such restructuring would not be a very useful property of an episodic memory, in which each episode must be clearly distinguished from others.

Reconsolidation refers to a process in which after a memory has been stored, it may be weakened or lost if recall is performed during the presence of a protein synthesis inhibitor (Debiec, LeDoux and Nader 2002, Debiec, Doyere, Nader and LeDoux 2006). The implication that has been drawn is that whenever a memory is recalled, some reconsolidation process requiring protein synthesis may be needed.

One possible function of reconsolidation is that it may allow some restructuring of a memory, as described above, though this might be useful more in semantic than episodic memory systems.

A second possible computational function is that reconsolidation might be useful as a mechanism to ensure that whenever a memory is retrieved, additional LTP (long-term potentiation of synaptic strength) is not added to the existing LTP. This could be achieved if during the recall process the memory strength is reset to a low value from which it is then strengthened. Indeed, a potential problem with memory systems is what separates storage from recall, in that whenever recall occurs, pre- and post-synaptic activity is present at the relevant synapses for the memory, and thus one might expect another round of synaptic strengthening to occur. Reconsolidation, by effectively resetting the baseline of synaptic strength during recall, might then provide for the restrengthened synapses not to be stronger than they were before the memory recall. A relevant point here is that in associative memories, the amount of information stored and retrieved from any one synapse is quite low, in the order of 0.2–0.3 bits for autoassociators (Rolls and Treves 1998, Treves and Rolls 1991) and a little higher for pattern associators (Rolls and Treves 1998), so that in any case having synaptic strengths that could be repeatedly strengthened by superposition of different memories with distributed representations and with precision maintained at each strengthening would not appear to be a necessary property of the synapses in such memory systems. Under these circumstances, allowing during recall a weakening of a memory, and then its reconsolidation from a relatively fixed baseline might not lead to loss of useful information, and might be a possible solution to continually strengthening synapses every time a memory is recalled.

A third possible computational function of reconsolidation is that it could enable the selective retention of 'useful' memories (or in fact memories being used), and the forgetting of memories not being used, as follows. Consider a memory system in which there is slow exponential decay of synaptic strength with time, a not altogether unlikely scenario given the properties of a biological system. In this situation memories will gradually be lost, perhaps with a different time course in different memory systems, which might be in the order of days, weeks, months or years. In this scenario, if a piece of information was actually recalled

because an environmental situation occurred in which for example there was a retrieval cue for a memory, then that memory (i.e. the synaptic strengths) would by reconsolidation be strengthened back to near its initial value. That memory would then be strong and available for future use, compared to other memories not recalled that would be passively decaying. The passive decay of memories not being recalled and reset by reconsolidation would be useful in cleaning out the memory stores so that any critical capacity was not reached, and at the same time in minimizing interference (due to generalization to similar patterns) between memories in store. An example might be the number of one's hotel room, which while it is being repeatedly recalled for use while in that hotel and thus restored, would then decay passively and gradually be lost when it was not longer being actively recalled and hence reconsolidated. One could propose that in some memory systems the passive decay might be relatively rapid, occurring within hours or days. An example might be the dorsolateral prefrontal cortex, where depending on the requirements of the short-term memory or planning tasks being performed, synapses might by reconsolidation keep representations used in attractor networks available while a given task was being performed. However, when that task was no longer being performed, passive synaptic decay would mean that neurons allocated to that task would gradually decline, and instead new attractor landscapes (i.e. memories) could be set up for new tasks or planning, without interference from representations that were previously being used. There is some evidence at the phenomenological level that neuronal representations are made and kept relevant to whatever task is being performed (Miller et al. 2003, Everling et al. 2006), and I have just proposed a possible mechanism for this implemented by reconsolidation.

Memories stored early in life may be stored better, and later recalled better, than those stored later in life. There are a number of possible reasons for this. One is that the transmitters that generally facilitate synaptic modification, such as acetylcholine and noradrenaline (see Sections 3.10.5 and 3.10.6), may become depleted with ageing.

Another mechanism, not necessarily independent, is that new synaptic modification, as assessed by long-term potentiation (LTP), appears to be less long-lasting with ageing (Burke and Barnes 2006). Another mechanism may be that storing memories in a flat energy landscape (i.e. without much prior synaptic modification) may help these memories to stand out from those added later. While this would not be a natural property of the type of autoassociation palimpsest memory described above, it could be a property of the way in which an episodic memory stored in the hippocampus may be retrieved into the neocortex where it can be incorporated into a semantic memory (see Chapter 2), the relevant example of which in this case would be an autobiographical memory.

In semantic memories, it could be that the first stored links tend to provide the framework around which other information is structured.

Another factor in the apparent strength of early memories may be that some may be stored with an affective component, and this may not only make the memory strong by activation of the cholinergic and related systems described in Section 3.10.5, but may also mean that part at least of the memory is stored in different brain structures such as the amygdala which may have relatively more persistent and less flexible or reversible memories than other memory systems.

Another factor in the importance and stability of synaptic modification early in life arises in perceptual systems, in which it is important to allow neurons to become tuned to the statistics of for example the visual environment, but once feature analyzers have been formed, stability of the feature analyzers in early cortical processing layers may be important so that later stages in the hierarchy can perform reliable object recognition which achieves stability only if the input filters to the system do not keep changing (see Chapter 4). This could be the

importance of a critical period for learning early on in perceptual development.

Sleep has been proposed as a state in which useful forgetting or consolidation of memories could occur. One suggestion was that if deep basins of attraction formed in a memory network, then this could impair performance, as the memories in the basins would tend to be recalled whatever the retrieval cue. If noise, present in the disorganized patterns of neural firing during sleep, caused these memories to be recalled, this would indicate that they were 'parasitic', and the suggestion was that associative synaptic weakening (LTD) of synapses of neurons with high firing during sleep would tend to decrease the depth of those basins of attraction, and improve the performance of the memory (Crick and Mitchison 1995). At least at the formal level of neural networks, the suggestion does have some merit as a possible way to 'clean up' associative networks, even if it is not a process implemented in the brain. Although the idea of some role of sleep in memory remains active, this remains to be fully established (Walker and Stickgold 2006).

The idea that sleep could be a time when memories are unloaded from the hippocampus to be consolidated in long term, possibly semantic, memories during sleep (Marr 1971) (allowing hippocampal episodic memories to then be overwritten by new episodic memories) continues to be explored. It has been shown for example that after hippocampal spatial representations have been altered by experience during the day, these changes are reflected in neuronal activity in the neocortex during sleep (Wilson and McNaughton 1994, Wilson 2002). The type of experience might involve repeated locomotion between two places, and the place fields of rat hippocampal neurons for those places may become associated with each other because of coactivity of the neurons representing the frequently visited places. The altered co-firing of the hippocampal neurons for those places may then be reflected in neocortical representations of those places. This could then result in altered representations in the neocortex, if LTP occurs during sleep in the neocortex. Of course, any change in neocortical neuronal activity might just reflect the altered representations in the hippocampus, which would be expected to influence the neocortical representations via hippocampo-neocortical backprojections, even without any neocortical learning (see Chapter 2).

In conclusion, we have seen in this Section that forgetting has important functions in the brain, and is a necessary property of many different types of memory system if they are to continue to function efficiently and to allow new learning in the same networks in the brain.

B.17 Brain computation compared to computation on a digital computer

To highlight some of the principles of brain computation by for example the cortex described in this Appendix and throughout this book, it is interesting to compare the principles of computation by the brain with those of a digital computer.

An item of data is retrieved from the memory of a digital computer by providing the address of the data in memory, and then the data can be manipulated (moved, compared, added to the data at another address in the computer etc.) using typically a 32 bit or 64 bit binary word of data. Pointers to memory locations are thus used extensively. In contrast, in the cortex, the data are used as the access key (in for example a pattern associator), and the neurons with synaptic weights that match the data respond. Memory in the brain is thus *content-addressable*. In one time constant of the synapses/cell membranes the brain has thus found the correct output. In contrast, on a digital computer a serial search is required, in which

the data at every address must be retrieved and compared in turn to the test data to discover if there is a match.

Cortical computation including that performed by associative memories and competitive networks operates by vector similarity – the dot product of the input and of the synaptic weight vector are compared, and the neurons with the highest dot product will be most activated. Even if an exact match is not found, some output is likely to result. In contrast, in a digital computer, logic operations (such as AND, OR, XOR) and exact mathematical operations (such as addition, subtraction, multiplication, and division) are computed. (There is no bit-wise similarity between the binary representations of 7 (0111) and 8 (1000).) The similarity computations performed by the brain may be very useful, in enabling similarities to be seen and parallels to be drawn, and this may be an interesting aspect of human creativity, realized for example in Finnegans's Wake by James Joyce. However, the lateral thinking must be controlled, to prevent bizarre similarities being found, and this is argued to be related to the symptoms of schizophrenia in Section 8.5.

Because exact computations are performed in a digital computer, there is no in-built fault tolerance or graceful degradation. If one bit of a memory has a fault, the whole memory chip must be discarded. In contrast, the brain is naturally fault tolerant, because it uses vector similarity (between its input firing rate vector and synaptic weight vectors) in its calculations, and linked to this, distributed representations. This makes the brain robust developmentally with respect to 'missing synapses', and robust with respect to losing some synapses or neurons later (see e.g. Section B.2).

To enable the vector similarity comparison to have high capacity (for example memory capacity) the 'word length' in the brain is typically long, with between 10,000 and 50,000 synapses onto every neuron being common in cortical areas. (Remember that the leading term in the factor that determines the storage capacity of an associative memory is the number of synapses per neuron – see Sections B.2 and B.3.) In contrast, the word length in typical digital computers at 32 or 64 bits is much shorter, though with the binary and exact encoding used this allows great precision in a digital computer.

To comment further on the encoding: in the cortex, the code must not be too compact, so that it can be read by neuronally plausible dot product decoding, as shown in Appendix C. In contrast, the binary encoding used in a digital computer is optimally efficient, with one bit stored and retrievable for each binary memory location.

The precision of the components in a digital computer is that every modifiable memory location must store one bit accurately. In contrast, it is of interest that synapses in the brain need not work with exact precision, with for example typically less that one bit per synapse being usable in associative memories (Treves and Rolls 1991, Rolls and Treves 1998). The precision of the encoding of information in the firing rate of a neuron is likely to be a few bits – perhaps 3 – as judged by the standard deviation and firing rate range of individual cortical neurons.

This brings us to the speed of computation. In the brain, considerable information can be read in 20 ms from the firing rate of an individual neuron (e.g. 0.2 bits, see e.g. Fig. C.15), leading to estimates of 10–30 bits/s for primate temporal cortex visual cells (Rolls, Treves and Tovee 1997b), and 2–3 bits/s for rat hippocampal cells (Skaggs, McNaughton, Gothard and Markus 1993) (see Section C.3.4). Though this is very slow compared to a digital computer, the brain does have the advantage that a single neuron receives spikes from thousands of individual neurons, and computes its output from all of these inputs within a period of approximately 10–20 ms (determined largely by the time constant of the synapses), as described in this Appendix. Moreover, each neuron, up to at least the order of tens of neurons, conveys independent information, as described in Appendix C.

Computation in a conventional digital computer is inherently serial, with a single central

processing unit that must fetch the data from a memory address, manipulate the word of data, and store it again at a memory address. In contrast, brain computation is parallel in at least three senses. First, an individual neuron in performing a dot product between its input firing rate vector and its synaptic weight vector does operate in an analog way to sum all the injected currents through the thousands of synapses to calculate the activation h_i, and fire if a threshold is reached, in a time in the order of the synaptic time constant. To implement this on a digital computer would take $2C$ operations (C multiply operations, and C add operations, where C is the number of synapses per neuron – see equation B.3). Second, each neuron in a single network (e.g. a small region of the cortex with of the order of hundreds of thousands of neurons) does this dot product computation in parallel, followed by interaction through the GABA inhibitory neurons, which again is fast. (It is in the order of the time constant of the synapses involved, operates in continuous time, and does not have to wait at all until the dot product operation of the pyramidal cells has been completed by all neurons given the spontaneous neuronal activity which allows some neurons to be influenced rapidly.) This interaction sets the threshold in associative and competitive networks, and helps to set the sparseness of the representation of the population of neurons. Third, different brain areas operate in parallel. An example is that the ventral visual stream computes object representations, while simultaneously the dorsal visual streams computes (inter alia) the types of global motion described in Section 4.5.13. Another example is that within a hierarchical system in the brain, every stage operates simultaneously, as a pipeline processor, with a good example being V1–V2–V4–IT, which can all operate simultaneously as the data are pipelined through.

We could refer to the computation that takes place in different modules, that is in networks that are relatively separate in terms of the number of connections between modules relative to those within modules, such as those in the dorsal and ventral visual streams, as being parallel computation. Within a single module or network, such as the CA3 region of the hippocampus, or inferior temporal visual cortex, we could refer to the computation as being *parallel distributed computation*, in that the closely connected neurons in the network all contribute to the result of the computation. For example, with distributed representations in an attractor network, all the neurons interact with each other directly and through the inhibitory interneurons to retrieve and then maintain a stable pattern in short term memory (Section B.3). In a competitive network involved in pattern categorization, all the neurons interact through the inhibitory interneurons to result in an active population of neurons that represents the best match between the input stimulus and what has been learned previously by the network, with neurons with a poor match being inhibited by neurons with a good match (Section B.4). In a more complicated scenario with closely connected interacting modules, such as the prefrontal cortex and the inferior temporal cortex during top-down attention tasks and more generally forward and backward connections between adjacent cortical areas, we might also use the term parallel distributed computation, as the bottom-up and top-down interactions may be important in how the whole dynamical system of interconnected networks settles (see examples in Chapters 5, 6 and 8 and Section B.9).

Digital computers do not have noise to contend with as part of the computation, as they use binary logic levels, and perform exact computation. In contrast, brain computation is inherently noisy, and this gives it a non-exact, probabilistic, character. One of the sources of noise in the brain is the spiking activity of each neuron. Each neuron must transmit information by spikes, for an all-or-none spike carried along an axon ensures that the signal arrives faithfully, and is not subject to the uncertain cable transmission line losses of analog potentials. But once a neuron needs to spike, then it turns out to be important to have spontaneous activity, so that neurons do not all have to charge up from a hyperpolarized baseline whenever a new input is received. The fact that neurons are kept near threshold, with therefore some spontaneous

spiking, in inherent to the rapid operation of for example autoassociative retrieval, as described in Section B.3.3.5. But keeping the neurons close to threshold, and the spiking activity received from other neurons, results in spontaneous spike trains that are approximately Poisson, that is randomly timed. The result of the interaction of all these randomly timed inputs is that in a network of finite size there will be statistical fluctuations, which influence which memory is recalled, which decision is taken, etc. as described in Chapter 7. Thus brain computation is inherently noisy and probabilistic.

Digital computers can perform arbitrary syntactical operations on operands, because they use different pointers to point to the address of each of the different operands required (corresponding even for example to the subject, the verb, and the object of a sentence). In contrast, as data are not accessed in the brain by pointers that can point anywhere, but instead just have neurons firing to represent a data item, a real problem arises in specifying which neurons firing represent for example the subject, the verb, and the object, and distributed representations potentially make this even more difficult. The brain thus inherently finds syntactical operations difficult. We do not know how the brain implements the syntax required for language. But we do know that the firing of neurons conveys 'meaning' based on spatial location in the brain. For example, a neuron firing in V1 indicates that a bar or edge matching the filter characteristic of the neuron is present at a particular location in space. Another neuron in V1 encodes another feature at another position in space. A neuron in the inferior temporal visual cortex indicates (with other neurons helping to form a distributed representation) that a particular object or face is present in the visual scene. Perhaps the implementation of the syntax required for language that is implemented in the brain also utilizes the spatial location of the network in the cortex to help specify what syntactical role the representation should perform. This is a suggestion I make, as it is one way that the brain could deal with the implementation of the syntax required for language.

The physical architecture (what is connected to what) of a digital computer is fixed. In contrast, the connectivity of the brain alters as a result of experience and learning, and indeed it is alterations in the strength of the synapses (which implement the connectivity) that underlies learning and memory. Indeed, self-organization in for example competitive networks has a strong influence on how the brain is matched to the statistics of the incoming signals, and of the architecture that develops. In a digital computer, every connection must be specified. In contrast, in the brain there are far too few genes (of order 30,000) for the synaptic connections in the brain (of order 10^{15}, given approximately 10^{11} neurons each with in the order of 10^4 synapses) for the genes to specify every connection[43]. The genes must therefore specify some much more general rules, such as that each CA3 neuron should make approximately 12,000 synapses with other CA3 neurons, and receive approximately 48 synapses from dentate granule cells (see Chapter 2). The actual connections made would then be made randomly within these constraints, and then strengthened or lost as a result of self-organization based on for example conjunctive pre- and post-synaptic activity. Some of the rules that may be specified genetically have been suggested on the basis of a comparison of the architecture of different brain areas (Rolls and Stringer 2000). Moreover, it has been shown that if these rules are selected by a genetic algorithm based on the fitness of the network that self-organizes and learns based on these rules, then architectures are built that solve different computational problems in one-layer networks, including pattern association learning, autoassociation memory, and competitive learning (Rolls and Stringer 2000). The architecture of the brain is thus interestingly adaptive, but guided in the long term by genetic selection of the building rules.

[43] For comparison, a computer with 1 Gb of memory has approximately 10^{10} modifiable locations, and if it had a 100 Gb disk that would have approximately 10^{12} modifiable locations.

The learning rules that are implemented in the brain that are most widely accepted are associative, as exemplified by LTP and LTD. This, and the vector similarity operations implemented by neurons, set the stage for processes such as pattern association, autoassociation, and competitive learning to occur naturally, but not for logical operations such as XOR and NAND or arithmetic operations. Of course, the non-linearity inherent in the firing threshold of neurons is important in many of the properties of associative memories and competitive learning, as described in this Appendix, and indeed are how some of the non-linearities that can be seen with attention can arise (Deco and Rolls 2005b).

Because the brain has populations of neurons that are simultaneously active (operating in parallel), but are interconnected, many properties arise naturally in dynamical neural systems, including the interactions that give rise to top-down attention (Chapter 6), the effects of mood on memory (Section 3.11) etc. Because simultaneous activity of different computational nodes does not occur in digital computers, these dynamical systems properties that arise from interacting subsystems do not occur naturally, though they can be simulated.

The cortex has recurrent excitatory connections within a cortical area, and reciprocal, forward and feedback, connections between adjacent cortical areas in the hierarchy. The excitatory connections enable cortical activity to be maintained over short periods, making short-term memory an inherent property of the cortex, and also autoassociative long-term memory with completion from a partial cue (given associative synaptic modifiability in these connections). Completion is a difficult and serial process to identify a possible correct partial match on a digital computer. The short-term memory property of the cortex is part of what makes the cortex a dynamical interacting system, with for example what is in short-term memory in for example the prefrontal cortex acting to influence memory recall, perception, and even what decision is taken, in other networks, by top-down biased competition (see Chapters 5–10). There is a price that the brain pays for this positive feedback inherent in its recurrent cortical circuitry, which is that this circuitry is inherently unstable, and requires strong control by inhibitory interneurons to minimize the risk of epilepsy.

Finally, we can note that many brain systems are organized hierarchically. A major reason for this is that this enables the connectivity to be kept within the limits of which neurons appear capable (up to 50,000 synapses per neuron), yet for global computation (such as the presence of a particular object anywhere in the visual field) to be achieved, as exemplified by VisNet (see Fig. 4.2). Another important reason is that this simplifies the learning that is required at each stage and enables it to be a local operation, in contrast to backpropagation of error networks where similar problems could in principle be solved in a two-layer network (with one hidden layer), but would require training with a non-local learning rule (see Section B.12) as well as potentially neurons with very large numbers of connections.