2 *The attraction of parallel distributed processing for modelling cognition*

The representation and processing of information in connectionist networks is distributed. Decisions are reached by consensus of a large number of simple computations taking place in parallel as stimulus information interacts with stored knowledge. In consequence, connectionist memories display many human characteristics: They are relatively immune to damaged components within the system or to noisy input; they allow retrieval by content; they are likely to retrieve typical instances from categories.

The last decade has seen an explosive growth in the connectionist modelling of cognitive processes, with simulation of most of the classical experimental paradigms of cognitive psychology. One reason for this enthusiasm is that, independent of their success at modelling human performance at any particular cognitive task, all connectionist models exhibit some general characteristics which are shown by human cognitive processes and distinguish them from non-biological computational systems such as computer programs: They still perform reasonably well after minor damage to components of the system; they still perform reasonably well if their input is noisy or inaccurate; they allow memory retrieval by content.

In this chapter we will look at two aspects of connectionist systems which are responsible for these characteristics. Like the principles of interneuronal communication described in chapter 1, these are based on general observations of brain structure. First, knowledge representation is *distributed* across many processing units. Second, computations take place in *parallel* across these distributed representations. The result is that conclusions are reached on the basis of a consensus of many calculations rather than depending on any particular one.

These principles put connectionist models in direct contrast to many traditional models in cognitive psychology or artificial intelligence where knowledge representation is local and computation is serial. In general, such models are not immune to

damage or resistant to noisy input. So a traditional model of, say, syllogistic reasoning, might give as good a fit to the experimental data as a connectionist model, but it would do so without exhibiting the full range of human characteristics as it performed the task.

The representation of knowledge in connectionist networks is distributed

Traditional models of cognitive processing usually assume a local representation of knowledge. That is, knowledge about different things is stored in different, independent locations. In a traditional model of reading aloud, for example, information about how to pronounce the letter string DOG is stored in one place and information about how to pronounce the string CAT in another. What could be more natural? The two pieces of information are independent and would be required at different times. So storing them independently makes obvious sense. The information storage systems we are familiar with in everyday life—dictionaries, telephone directories, computer discs—use local representation. Each discrete piece of information is stored separately. How else could it be done?

In connectionist models information storage is not local, it is *distributed*. There is no one place where a particular piece of knowledge can be located. Consider the segment of network at the bottom of figure 1.2 in chapter 1, part of a larger network which is learning to read aloud. Any input, such as the letter string DOG, would excite units and connections all over the network. Learning takes place by changing the weights of the connections leading to all output units which have an incorrect level of activity. The knowledge of how to pronounce the input DOG is distributed across many different connections in different parts of the system. It is the sum total effect of all these connections which produces the pronunciation, not any single one of them.

The concept of distributed storage may be difficult to grasp at first because it is counter to our everyday experience of information storage systems. The connections which contain the system's knowledge about how to pronounce DOG are the same as those with the knowledge about how to pronounce any other letter string. All the knowledge that the network contains is superimposed on the *same* set of connections. Intuitively this may seem entirely implausible. How can the same set of weights store independent and even contradictory pieces of information? As we will see, it can be done, and some of the emergent properties of such systems are intriguingly similar to properties of human cognitive processes. But for the moment this will have to be taken on trust. There are no familiar information storage systems which use distributed coding, so analogy to a familiar system is not possible.

Distributed representations are damage resistant and fault tolerant

When one considers the structure of the brain it is remarkable that it ever manages to come to the correct conclusion about anything. By any conventional standards neurons are an entirely unsuitable medium for computation: they die throughout the brain's life, causing random loss of stored information; they have a finite probability of firing even when they are not engaged in signal processing; the response of a neuron to any particular input is probabilistic, not fixed.

If we look at the firing pattern of a single neuron the problem that probabilistic responses cause for the system will become clear. The upper part of figure 2.1 shows the average response of a single neuron in the visual cortex to a stimulus presented to the eye. The stimulus is presented at time 0. Time after the presentation of the stimulus is shown on the horizontal axis. The vertical axis shows the neuron's firing rate. It has an average background firing rate of a few spikes per second. When the stimulus is presented a signal is superimposed on this. About 50 ms after the stimulus appears the neuron fires strongly for about 30 ms. 100 ms later it fires again, rather less intensely, for about 50 ms. This is the signal that the neuron transmits to the other neurons to which it is connected, indicating what pattern of stimulation it has received.

This seems fairly straightforward. But the histogram was obtained by summing the spike patterns over a number of presentations of the stimulus. The lower part of the figure shows 12 different occasions on which the same stimulus was presented. Each time the neuron fires there is a vertical spike. If we look at these individual trials the pattern which emerges is much less clear than that suggested by the overall average at the top. On trial 5 the initial burst was missing. On trial 10 the second burst was missing. On trial 11 the neuron did not respond at all. It is clear that the 'signal' which the histogram at the top shows is an idealised average. On any given trial the output will only approximate this, sometimes quite closely, sometimes not at all. How can the system produce a reliable response on every trial when the individual components only produce their signal on average across trials?

If the processing components in a conventional digital computer produced random spontaneous output, a different response to the same stimulus on different occasions and suffered from random component drop-out, the system would be totally unpredictable! Sometimes it would work correctly, but if a computation required access to the contents of a missing memory unit, or a burst of noise obliterated a signal, the result would be garbage. Although the components in the brain can fire and die at random, the computations performed by the brain are not unpredictable. With minor damage it becomes a little slower and less accurate, but it still produces roughly the same answer. It has to suffer serious damage before it produces nonsense.

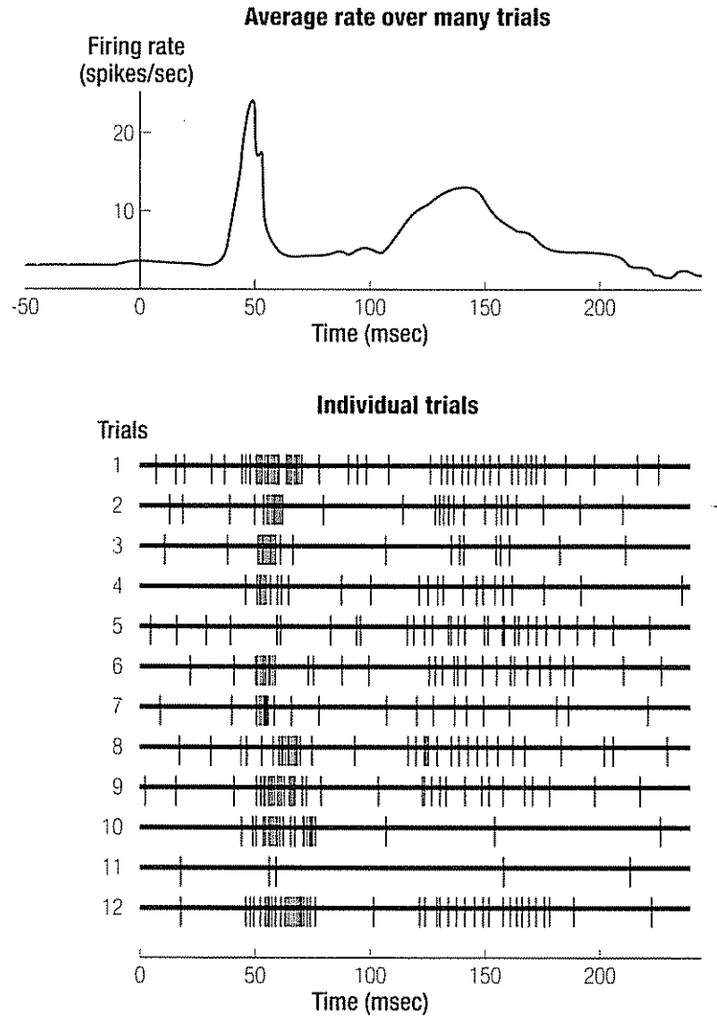


Figure 2.1 The response of a neuron in the visual cortex for 250 ms after stimulus presentation. Upper: The firing rate of the neuron summed over a number of presentations of the stimulus. Lower: The pattern on 12 of the trials which contributed to the average response shown at the top. Each vertical line represents the time at which the neuron fired. (Based on Morrell 1972.)

The brain escapes the consequences of the unpredictable behaviour of individual neurons because its computations are performed in parallel on representations which are distributed over many neurons. No one neuron plays a crucial role in processing. The overall result is the outcome of many distributed sub-computations. Even if individual components of the calculation are not accurate, the ensemble averaging

can nevertheless give an answer which is accurate enough. When the memory location required for a calculation in a localist information storage system is damaged, the result is disaster. If the first page of a dictionary is missing, there is no way of checking whether *aardvark* is really spelt like that. But, in a connectionist system, there is no such thing as 'the memory location required for a calculation'. Information and calculation are spread across the network. If one unit or connection in the network is damaged, others can make up for the missing part.

The system will, of course, be slightly less accurate if a connection is lost. But the pattern of loss is quite different in localist and distributed systems. Damage to a localist system causes some information to be lost totally while other information is unaffected. In a distributed system any damage causes partial loss of a range of information. As damage increases, the performance of the system inevitably begins to drop. But a small amount of damage may have no noticeable effect on the output of the system. The ability of brains and connectionist models to continue to produce a reasonable approximation to the correct answer following damage, rather than undergoing catastrophic failure, is an example of fault tolerance referred to as *graceful degradation*.

Connectionist networks allow memory access by content

Human memory is content-addressable. That is, you can access a memory by using some part of the information contained in the memory (the *content*) as a retrieval cue. This is unlike retrieval from familiar forms of information storage, such as dictionaries, telephone directories or computer discs. In these, the place where the information is stored has an *address*. The only way to access information for retrieval is with the address. For example, in a dictionary the address is the spelling of the word; the information stored with this address is the definition of the word. In a computer system a typical address is the name of a file; the information which can be accessed with this address is the contents of the file.

To understand the difference between accessing a memory by address and by content, contrast retrieving information from a dictionary with obtaining the same information from a person. Imagine that you want to know the name of a man-made wall, built across a valley, to contain water to build up a head of pressure to generate electricity. With a dictionary, there is no way to access the location where this information is stored and extract the missing piece, the word 'dam'. If you start with the address (DAM) you can access all the information stored at the address. Without it you can access nothing. In contrast, a person given part of the information would probably be able to retrieve the rest. Unlike a dictionary, human memory allows access via any part of the information that forms the memory. One of the reasons why connectionist models of human memory are attractive is that content addressability follows as a natural consequence of their distributed structure. Content addressability can be built into localist storage but only by adding a complex cross-referencing system.

Any information processing system which works in the brain must be fault tolerant because the signals it has to work with are seldom perfect. There is a random component to neuronal firing; speech is usually heard against a background of other noises; objects rarely present the same image on different occasions. An attractive aspect of content-addressable memory is that it is inherently fault tolerant. Imagine someone asked you to guess who they were thinking of: 'This man was a British Conservative politician. He became Prime Minister in 1978 and was Prime Minister during the Falklands War. He was ousted from office by his own party, being held responsible for the fiasco of the Poll Tax. He was eventually replaced as Prime Minister by John Major.' You could probably suggest 'Margaret Thatcher' as an answer despite the fact that some of the information is incorrect. Mrs Thatcher did not become Prime Minister until 1979, of course. With content-addressable memory the weight of evidence pointing to one answer can overcome other evidence that is inconsistent. A best fit solution can be chosen even if it is not perfect. This is unlike memory systems in which access by address is the only possibility. Any error in the address will lead to failure. A search of *Who's Who* using the address 'Margaret Patcher' would discover nothing, despite the fact that most of the search term fits an existing entry.

Retrieving information from a distributed database

To see how a distributed system with parallel processing works in practice we will look at retrieval from a simple connectionist memory described by McClelland (1981).[1] This memory demonstrates content addressability and fault tolerance. It also shows typicality effects in retrieval—if asked to retrieve a random member of a category it will produce a typical member. Considering the simplicity of the simulation it demonstrates a remarkable range of human characteristics in memory retrieval.

Imagine that you live in a neighbourhood where many of your male acquaintances belong to one of the two rival local gangs, the Jets or the Sharks. Your knowledge about these characters will come from a succession of independent episodes. One night Fred emerges from behind a bush and offers you some white powder. You hear Dave and his wife trading insults as she drives off with a car full of suitcases. Everyone in the bar is laughing because Don has been admitted to college on the basis of forged examination results. Nick hangs out with Karl whom you know to be a Shark. All these pieces of information about your neighbours are gradually

[1] The Jets and Sharks memory system is not implemented in tlearn but its properties and the examples described in the text can be explored using the iac program in McClelland and Rumelhart (1988)

Table 2.1

Address	Contents				
	Gang	Age	Education	Marital Status	Occupation
Alan	Jets	30s	JH	Married	Burglar
Art	Jets	40s	JH	Single	Pusher
Clyde	Jets	40s	JH	Single	Bookie
Dave	Sharks	30s	HS	Divorced	Pusher
Don	Sharks	30s	Col	Married	Burlar
Doug	Jes	30s	HS	Single	Bookie
Earl	Sharks	40s	HS	Married	Burglar
Fred	Jets	20s	HS	Single	Pusher
Gene	Jets	20s	Col	Single	Pusher
George	Jets	20s	JH	Divorced	Burglar
Greg	Jets	20s	HS	Married	Pusher
Ike	Sharks	30s	JH	Single	Bookie
Jim	Jets	20s	JH	Divorced	Burglar
John	Jets	20s	JH	Married	Burglar
Karl	Sharks	40s	HS	Married	Bookie
Ken	Sharks	20s	HS	Single	Burglar
Lance	Jets	20s	JH	Married	Burglar
Mike	Jets	30s	JH	Single	Bookie
Neal	Sharks	30s	HS	Single	Bookie
Ned	Sharks	30s	Col	Married	Bookie
Nick	Sharks	30s	HS	Single	Pusher
Oliver	Sharks	30s	Col	Married	Pusher
Pete	Jets	20s	HS	Single	Bookie
Phil	Sharks	30s	Col	Married	Pusher
Ralph	Jets	30s	JH	Single	Pusher
Rick	Sharks	30s	HS	Divorced	Burglar
Sam	Jets	20s	Col	Married	Bookie

(Based on McClelland 1981.)

assembled over the years. Fred is a pusher; Dave is divorced; Don went to college; Nick is a Shark.

Table 2.1 shows how this information might be stored in a conventional information storage system. The system has a set of storage locations, corresponding to a set of cards in an index file or a set of files on disc in a computer, for example, each headed by an address. The address is the name of the person. Each new piece of information relating to him is stored at the location headed by this address. In this simple simulation we imagine that we have information about the **Gang** each person belongs to (*Jets* or *Sharks*), a rough idea of his **Age** (*20s*, *30s* or *40s*), the extent of his **Education** (*Junior High*, *High School* or *College*), his **Marital Status** (*Married*, *Single* or *Divorced*), and his **Occupation** (*Bookie*, *Burglar* or *Pusher*). The format used in

table 2.1, address + contents, is a logical way of storing the information since the name *Alan* is the key which binds one set of information together, *Clyde* connects another set, and so on.

This form of storage is efficient for retrieving information in response to questions like 'Is Fred a pusher?'. The question contains the address, and the address leads directly to the place where the information which provides the answer is stored. But it is not so good for answering other enquiries. If you are asked 'Do you know the name of a pusher?', the only way is to search through the list of addresses until you find one where the information *Pusher* is stored under **Occupation**. Although the information *Pusher* is stored at many locations, it does not form part of the address. So an answer to this question cannot be extracted directly from the memory.

Admittedly, with this particular database it would not take long to find a pusher if you searched addresses at random. But a more realistic representation of knowledge of these people would include many unique pieces of information, such as the fact that Fred's grandparents came from Ballylickey. The only way to store this in a system like that shown in table 2.1 is as the fact 'grandparents came from Ballylickey' at the storage location with the address *Fred*. The question 'Whose grandparents came from Ballylickey?' could only be answered by random search of the addresses until the one containing that information was found. This might take a long time although you would get there in the end. But a human memory would not respond like that. If you could remember the information at all you would usually produce the answer reasonably quickly. This is because human memory can be accessed by *content*—any part of the knowledge base can be used to access any other part. 'Ballylickey' can be used as a cue, and will lead to *Fred*. The information storage system shown in table 2.1 is perfectly logical. Indeed, it is probably the sort of system you would use if you were asked to store the information about the Jets and Sharks. But human memory cannot be organised like this. A memory organised like table 2.1 does not allow content-addressable retrieval; human memory does.

Another way of seeing why human memory cannot be organised so that access is only possible by address is to consider how you would answer the question 'What are the Jets like?'. Table 2.1 allows easy access to information about individual Jets. But it offers no simple way to answer questions requiring generalisations across a number of entries. Human memory does allow generalisations across areas of memory. A person who knew these two gangs could probably tell you that the Jets were younger than the Sharks without having to think very hard. Although the method of information storage shown in table 2.1 seems natural, it cannot be the way that human memory is organised.

(1) Setting up a distributed database for the Jets and Sharks base. McClelland explored the consequences of storing the information about the Jets and Sharks in a distributed system. The architecture of his system is shown in the upper part of

figure 2.2. To store the information in table 2.1 we need to represent facts about **Name**, **Gang Membership**, **Age**, **Education**, **Marital Status** and **Occupation**. Within each of these areas of knowledge there is a node corresponding to the possible values that someone could have. So in the **Age** area there are nodes for *20s*, *30s* and *40s*, in **Occupation** for *Burglar*, *Bookie* and *Pusher*, in **Gang** for *Jet* and *Shark*, and so on. This model might seem to be localist rather than distributed because there are individual nodes to represent specific concepts. But, as we shall see, the underlying dynamics of the model are parallel and distributed. The result of any input is determined by interaction across the entire database. The localist coding of concepts is used to make it easy to see what the model is doing in the examples which follow.

A memory is formed by setting up a link between two nodes. If we discover that Sam is a bookie we set up a positive connection between the *Sam* node in the **Name** area and the *Bookie* node in the **Occupation** area. If we then discover that he is married we set up a positive connection between *Sam* in the **Name** area and *Married* in **Marital Status**, and between *Bookie* and *Married* (since we now know of a bookie who is married). To store all the information we know about Sam we would set up positive links between all the possible pairwise combinations of *Sam* in **Name**, *Jet* in **Gang**, *20s* in **Age**, *College* in **Education**, *Married* in **Marital Status** and *Bookie* in **Occupation**. The way that McClelland did this is shown in the middle of figure 2.2. He set up a Person node in the central region of the model and then made a positive connection between this and each fact that was related to that person. The result is similar to setting up all 15 links necessary to represent these facts individually, but requires fewer links. Person nodes were then set up for each of the people in table 2.1 and the necessary links formed to represent everything that is known about them[2].

The bottom part of figure 2.2 shows the connections between the various nodes building up as information about five of the people is added to the system. This also shows a second element of the model. There are mutually inhibitory connections between each of the exemplar nodes within a knowledge area. There must be some inhibitory links in a network like this where everything is interconnected or the result of activating any node would be that everything in the network would eventually reach its maximum activity level and no differential response to different inputs would be possible. These connections represent the fact that, for example, if someone is in his 20s, he cannot be in either his 30s or his 40s. This fact could have been implemented by making negative connections from each person node to all the things he is not. Building mutually inhibitory links between alternative instance nodes

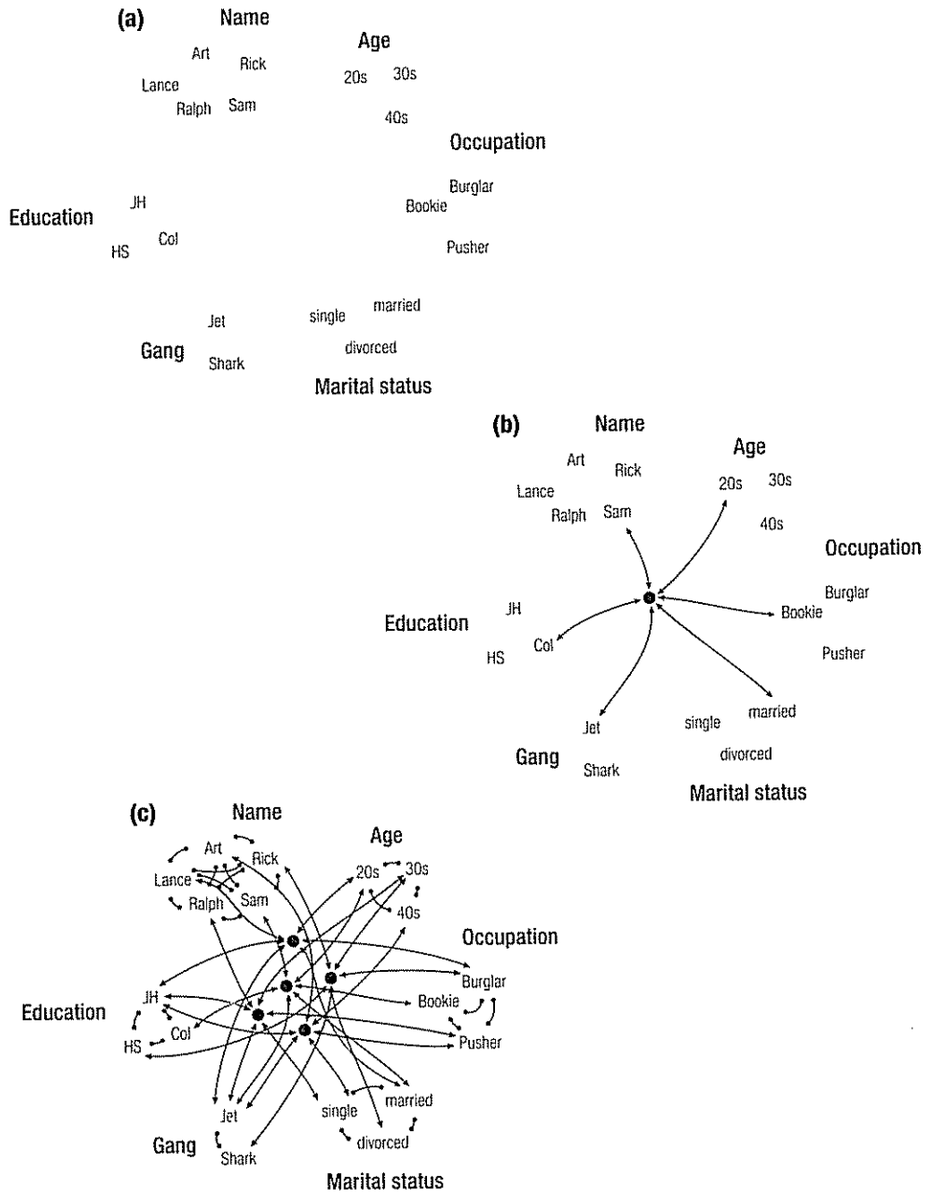
[2]It might seem that the Person node corresponds to the address for the information just as the name Sam does in table 2.1. When information is retrieved from the net the Person node cannot be accessed, so it cannot be used as a retrieval address. It is just a convenience which reduces the number of connections required to set up the model and makes the operation of the model easier to follow.

within an area has a similar effect on the performance of the net but greatly reduces the number of connections required. As we shall see, the way that inhibition is built into this network has an important role to play in the way the model runs.

(2) Running the network. The nodes are the processing units of the model. They act like the one at the bottom of figure 1.3 which was described in chapter 1. Each has an activity level associated with it. When the model runs, each node passes activity to all the other nodes to which it is connected, in the way described in equation 1.1. That is, the input to the receiving node is the product of the activity level of the sending node and the weight of the connection between them. In this simple model the weight of all positive connections between nodes is +1 and that of negative connections is −1.[3] So, when the model runs, every node which has a positive activity level tries to increase the activity level of every node to which it has a positive connection, and to reduce the activity of every node to which it has a negative connection. The net input of each node is determined by summing these negative and positive inputs (as described in equation 1.2). The net input is then converted by an activation function to an activity level. The exact form of the activation function used by McClelland was not the same as any of those shown in figure 1.4 but had the same effect as the sigmoid function in figure 1.4(c) of limiting the maximum value which the activity level could reach, and of slowing the change in activity level with changes in net input as the unit's activity level approached its maximum value. In a single processing cycle the activity level of each unit is computed by summing its inputs and converting these to an activity level with the activation function. On the next cycle these new activity levels are used to compute the new net inputs to each unit, and thus their new activity level. This is continued until the net reaches a steady state. That is, until each node in the network reaches a constant activity level.

(3) Retrieval from the database. To test the memory performance of this system we ask it a question such as 'Can you remember the name of a pusher?'. This is done by activating the *Pusher* node in **Occupation** and waiting to see which unit becomes active in the **Name** area as activity passes round the network. The activity of all nodes starts at a level of −0.1. Activity of the *Pusher* node increases the activity level of all nodes to which it has a positive connection and decreases the activity of all those to which it has a negative connection. Once the activity level of a node rises

[3]There is no gradual learning phase in Jets and Sharks. Facts are given to the model complete. Therefore the weights do not develop as knowledge is acquired as they would in a conventional connectionist model. The way that information is entered into this system is an example of Hebbian learning (which will be discussed in detail in chapter 3). If two things are mutually consistent (e.g. being in your 20s and being a burglar) a positive connection (via the appropriate Person node) is made between them. If two things are mutually inconsistent (e.g. being in your 20s and 30s) a negative connection is made between them.



above 0 it excites all the nodes to which it has a positive connection and inhibits all those to which it has a negative connection. Eventually the system reaches a steady state in which the activity level of each node is constant, either because it has reached its maximum or minimum permitted value, or because its negative and positive

Figure 2.2 The architecture of McClelland's system for storing the information about the Jets and Sharks shown in table 2.1. (a) Each cloud represents an area of knowledge about the members of the two gangs, with the nodes within a cloud representing possible instances. (b) The information about Sam is represented by setting up excitatory connections between the facts that are known about him. This is done by setting up a Person node (the black node in the central circle) and linking this to all the instance nodes which represent his properties. If any one of these nodes becomes active these links will ensure that the nodes representing his other characteristics will be activated. (c) The excitatory connections necessary to represent all the information about five members of the gang have been entered. Inhibitory connections (links with filled circles on their ends) have been set up between competing instances nodes within each area of knowledge. When the model runs, any node which has a positive activity level will inhibit any other node to which it is connected by one of these links. (Based on McClelland 1981.)

inputs are exactly balanced.[4] The **Name** node which is most strongly activated when steady state is reached is the system's answer to the question.

Does such a system behave like human memory? Apart from being able to retrieve the information it had been given directly, by answering such questions as 'What does Fred do?', anyone who knew these people would find it easy to answer questions like 'Do you know the name of a pusher?' or 'What are the Jets like?'. These are the sort of questions which it is difficult to answer with a localist memory store organised like table 2.1. Will this distributed, connectionist memory system find them any easier?

With the aid of the bottom part of figure 2.2 it is possible to get some idea of what happens when the system runs. To see what answer the system will retrieve when it is asked: 'Can you remember the name of a pusher?' the *Pusher* node is activated. This activity passes along all the connections from the *Pusher* node. So the *Ralph* and *Art* **Person** nodes become excited because there is a positive connection to them from *Pusher*. (In the real model all the **Person** nodes of pushers would be excited. For simplicity we will just follow two of them.) The *Bookie* and *Burglar* nodes become inhibited because there are negative connections to them from *Pusher*. In the next processing cycle the *Ralph* **Name**, *Jet*, *30s*, *JH*, *Single* and *Pusher* nodes are excited by the *Ralph* **Person** node, and the *Art* **Name**, *Jet*, *40s*, *JH*, *Single* and *Pusher* nodes become excited by the *Art* **Person** node. At each succeeding cycle every node which is active influences every node to which it is connected by an amount which depends on its activity level and in a direction which depends on whether the connection between them is excitatory or inhibitory. So, for example, the fact that the *30s* node is excited by the *Ralph* **Person** node will in turn cause excitation of all the **Person** nodes connected to *30s*, and inhibition of the *20s* and *40s* **Age** nodes. At the same time the excitation of the *40s* **Age** node by the *Art* **Person** node may be sufficient to overcome

[4] In McClelland's model the activity of each unit also decays on each cycle by an amount proportional to its activity level. This affects the dynamic behaviour of the net but to understand why the net reaches a steady state it can be considered as another negative input contributing to the balance between positive and negative inputs to each unit.

this and cause excitation of all **Person** nodes connected to *40s* and the inhibition of the *30s* and *20s* **Age** nodes. After several processing cycles the activity level of every node in the system is being influenced by a mixture of positive and negative inputs. Obviously it soon becomes impossible to keep track of the patterns of excitation and inhibition and predict whether the system will reach a stable state, and if so, what will be excited and what depressed. The only way to find out what the system will do in response to stimulation of any of its nodes is to run a computer simulation of the whole system.

It should now be clear that a connectionist memory system is totally unlike a conventional memory such as a computer filing system. In a conventional system independent pieces of information are stored separately. When a specific piece of information is accessed, it and it alone is retrieved. But in a distributed connectionist system, an attempt to extract any information from the system leads to a flow of excitation and inhibition throughout the system to everything which has any relation to this information. This results in many different nodes becoming active. What is retrieved is the information which corresponds to the most active node(s) once this flow has stabilised. If one is accustomed to information retrieval from a conventional, non-connectionist system such as a telephone directory, this might seem very odd. If you looked up Tom Brown's number in a connectionist telephone directory the number retrieved would be influenced by the entries of everyone with a similar name or a similar number to Tom's. This would not be useful. You do not want the number retrieved to be influenced by the fact that there happens to be someone called Tim Brown who has a telephone number quite unlike Tom's. You want the information stored at the location with the address 'Tom Brown' and nothing else. But the interference which a connectionist system allows during retrieval between related items of stored information turns out to have some interesting and useful properties.

(4) Content-addressable memory in Jets and Sharks. To see whether this memory system allows access by content, we can ask it the question 'Do you know the name of a pusher?'. To do this we activate the *Pusher* node, leave it on, and see which **Name** nodes becomes activated. Figure 2.3 shows the activity level of three of the **Name** nodes as a function of the number of processing cycles for which the system has been allowed to run. All the pushers names initially become activated. Most of them, like *Oliver*, quickly return to their resting level. But *Fred* and *Nick* both become increasingly activated. After about 50 cycles *Fred* starts to dominate and soon the system enters a stable state with *Fred* activated and all the other names back at their resting level. The system answers the question with the reply: 'Err... Fred.' So, unlike the storage system of table 2.1, this system *does* allow information to be retrieved when it has been accessed by content rather than address.

The relative activity of the name *Fred* compared to the name *Nick* over the last 50

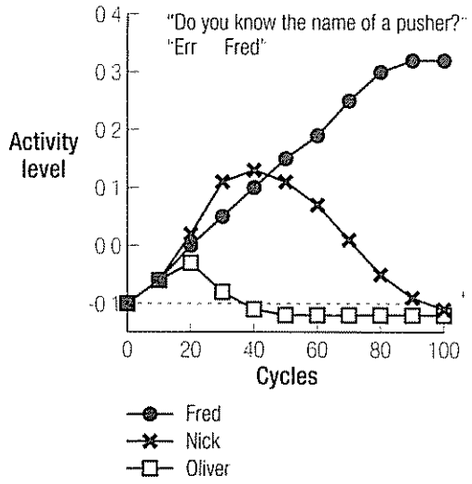


Figure 2.3 To see how the system responds to the question, the *Pusher* node in the **Occupation** area is held On and activity flows from here through the system. The initial activity level of all nodes is set to −0.1. The activity level of three of the nodes in the **Name** area is plotted as a function of the number of cycles of activity passing around the system.

cycles demonstrates an important characteristic of models with mutual inhibition between competing responses. When alternatives are equally activated they inhibit each other equally and everything is balanced. But once one gets ahead it inhibits the others more than they inhibit it. This reduces their activity and thus the extent to which they inhibit the one which is ahead. So it becomes more active and inhibits the others yet more. This rapidly results in the one that is a little more active consolidating its position in the lead and completely inhibiting the alternatives. The effect is sometimes referred to as 'the rich get richer' or 'winner takes all'.

Building positive feedback into the system in this way makes it likely that the system will quickly come to a definite conclusion, even if the difference between the evidence favouring one alternative rather than the other is small. But it makes the decision process vulnerable to noise. A random disturbance may be magnified and treated as a signal. This would generally be considered a drawback in a decision making system but it has one useful consequence. Figure 2.3 suggests that the system would always answer the question 'Do you know a pusher?' with the reply 'Fred'. If so, it would be an indifferent model of human memory. People would give a different answer to this question on different occasions, or if asked for an alternative answer, could provide the names of other pushers. It is straightforward to achieve this response variability with the model. If random noise is added to the starting activity levels the system will produce a different answer. Positive feedback ensures that a node that gets ahead is likely to consolidate its advantage. So a small change in starting conditions, or during processing, can make a radical difference to the outcome. Figure 2.4 shows the result of setting the initial activity level of Nick's **Person** node to −0.07 rather than −0.1 before activating the *Pusher* node. Now the system answers the question 'Do you know a pusher?' with the reply 'Err...Nick'.

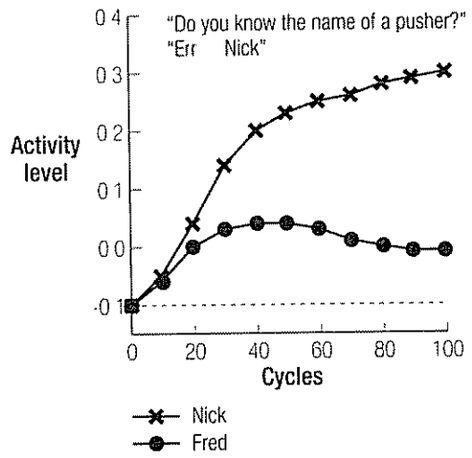


Figure 2.4 The effect of adding noise to the activity levels before asking the question. The initial activity of Nick's **Person** node is set to −0.07 rather than −0.1.

Given that we know the brain is a noisy system, it is appropriate to model it with an inherently noisy network. The result is the human characteristic of variability in response to the same input.

(5) Typicality effects in memory retrieval. If the information about the Jets and Sharks was stored in the manner of table 2.1 it would not be possible to retrieve the names of pushers directly, because pusher is not part of the address. But the information could be extracted by sampling addresses at random and giving the names of ones that turned out to contain the information *Pusher* under **Occupation**. In that case any pusher would have an equal chance of being produced as an example. But this is not how the network behaves when asked to name pushers. As figure 2.3 shows, it is more likely to retrieve the names of some pushers than others. It might retrieve Fred or Nick, but it is unlikely to produce Oliver's name.

Human memory retrieval has the same characteristic (see, for example, Rosch 1975). If you ask people to produce a list of birds (the equivalent of asking the network 'Tell me the names of all the pushers you know') most people will include Robin in their list but fewer will include Chicken. Some information is more easily available for retrieval from memory than other information from the same category. Since the answers which are most likely to be given to this question are the names of birds which would be rated as *typical* examples of the category, this result is called, unsurprisingly, a typicality effect.

The reason why the network is likely to retrieve Fred rather than Oliver in reply to the request for the name of a pusher can be seen in table 2.2. Pushers are more likely to be a Jet than a Shark, they tend to be in their 30s, to have been educated to High School level and to be single. So the **Person** nodes which get excited when *Pusher* is activated send more activity to the *Jet*, *30s*, *HS* and *Single* nodes than to other nodes.

Table 2.2 The pushers

Gang	*Jets*	*Sharks*	
	5	4	
Age	*20s*	*30s*	*40s*
	3	5	1
Education	*JH*	*HS*	*College*
	2	4	3
Marital status	*Single*	*Married*	*Divorced*
	5	3	1

The prototypical pusher:	*Jet*	*30s*	*HS*	*Single*
Fred	Jet	20s	HS	Single
Nick	Shark	30s	HS	Single
Oliver	Shark	30s	Col	Married

The mutual inhibition between alternative instance nodes with an area means that *Single*, *HS* and *Jet* become more activated, and *Married*, *College* and *Shark* become less activated. This in turn means that the **Person** nodes connected to *Single*, *HS* and *Jet* get supported, and the **Name** nodes connected to these become activated but the **Person** nodes and hence the **Name** nodes connected to *Married*, *College* and *Shark* do not. Fred is a single, High School educated, Jet; Oliver is a married, College educated, Shark. So, as a result of Fred's similarity to a *prototypical* pusher, his **Person** and **Name** nodes become more and more active as processing continues. Oliver's dissimilarity to the prototypical pusher means that his become less and less active. The result is that when the system is asked to think of a pusher, Fred's name is retrieved but Oliver's is not. If asked to generate instances from a category, distributed connectionist nets automatically generate typical instances, just like people.

Constraint satisfaction in connectionist networks

When activity flows through a connectionist network in response to an input, each unit influences the state of all the units to which it is connected. If the connection weight is positive the sending unit tries to put the receiving unit into the same state of activity as itself; if the connection weight is negative it tries to put it into the opposite state. Since all activity changes are determined by these influences, each input can be seen as setting constraints on the final state (i.e. set of unit activities) which the system can settle into. When the system runs, the activities of individual units will change in a way which increases the number of these constraints which are satisfied. Thus connectionist networks are said to work by *constraint satisfaction*.

The ideal final state would be a set of activities for the individual units where all the constraints were satisfied. The network would then be stable because no unit would be trying to change the state of any of the units to which it was connected. This ideal solution is unlikely to exist because most units are connected to some units which are trying to increase its activity and others which are trying to reduce it. There is no way of satisfying both. But if the system can find a state in which any change in the activity levels of the units reduces the overall number of satisfied constraints, it will stop changing activities. That is, it will have found a stable state. For most networks there will be many possible stable states with a different pattern of activity, each one of which will be reached from a different pattern of input activity. The realisation that these stable states could be viewed as the network's memories—that is, the set of possible states that it could reach in response to different inputs—was an important step in the history of connectionism which will be discussed further in chapter 15.

In a conventional connectionist network where knowledge is distributed across many units, it is difficult to follow constraint satisfaction at work because it is difficult to see what role any particular unit is playing. In Jets and Sharks it is easy, because the concept coding is localist rather than distributed. Each node stands for an identifiable concept. The constraints on the system are the various facts in table 2.1, each one of which is represented by one of the links in the network. The fact that Clyde is a Jet in his 40s means that if either of the nodes representing these concepts is activated it will try to activate the other, and inhibit the *Shark*, *30s* and *20s* nodes. The fact that there are also both Jets and Sharks in their 30s and 20s means that a whole set of other, mutually contradictory constraints are influencing the way that the system changes the activity level of the units in response to any particular pattern of input. The key point is that the changes in activity on each cycle will increase the number of constraints which are satisfied.

A system which works by constraint satisfaction has a number of desirable characteristics for modelling human cognition. The main one is that it allows a decision to be reached by a consensus of evidence, a reasonable fit between input and memory, rather than requiring an exact match. We have already seen this as a virtue in any model of human cognition because the nature of the nervous system requires a degree of fault tolerance in the information processing system (remember figure 2.1). It is also desirable because of the nature of the input which the cognitive system has to work with in the real world. Consider what happens when you listen to one particular speaker in a crowded room. The signals arriving at your ear contain the sounds made by the person you are listening to, but superimposed on these are a jumble of sounds from different speakers, the whole thing obliterated from time to time by bursts of laughter and other noises. And yet, most of the time, what you perceive is words. The signal you receive bears *some* relationship to a prototypical

representation of the word that you perceive but will be far from an exact match. The fact that you *perceive* words shows that the word recognition system must be looking for a *best fit* to the word patterns it has stored rather than for an exact match. This effect has been studied in the laboratory with an experimental paradigm called 'phoneme restoration'. In the original study by Warren (1970) the sound /s/ was removed from the word '*legislature*' and replaced with a cough. People then listened to a sentence containing the word '*legi*<cough>*lature*' and were asked what they heard. People reported the sentence correctly, adding that there was a cough before or after the word 'legislature'. In other words, the perceptual system does not necessarily give a veridical account of the stimulus, it gives a plausible *interpretation* of the input, given its knowledge of English words.

The same effect can be seen in the Jets and Sharks system. Figure 2.5 shows what happens when the system is probed with a variety of retrieval cues. The filled circles show what happens when the net is asked: 'Do you know a Shark, in his 20s, who went to High School, who is single and a burglar?' (i.e. the *Shark*, *20s*, *HS*, *Single* and *Burglar* nodes are all switched On). The circles joined by solid lines show the activity of Ken's **Name** node and the circles joined by dashed lines the activity of the next most activated **Name** node. Not surprisingly, the net answers 'Ken' quickly.

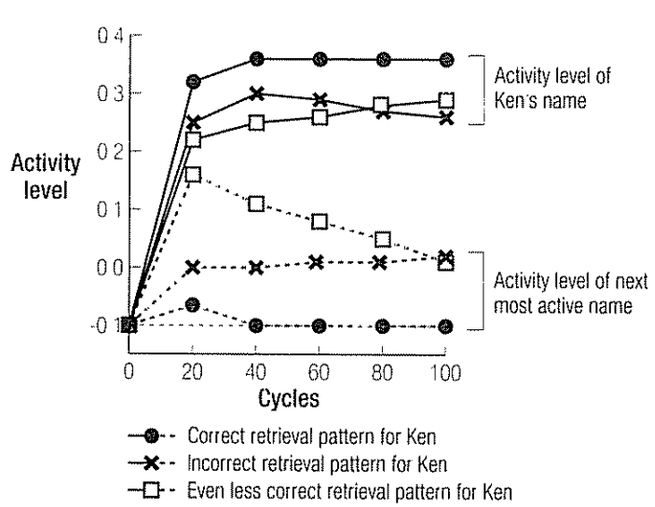


Figure 2.5 Constraint satisfaction in operation Given a correct description of Ken as a retrieval cue, the system retrieves Ken's name. Given progressively less accurate retrieval cues, which are, nevertheless, closer to a description of him than to anyone else in the database, it still produces his name Input activity levels:
(●) *Shark* = 1; *20s* = 1; *HS* = 1; *Single* = 1; *Burglar* = 1.
(✕) *Shark* = 1; *20s* = 1; *JH* = 1; *Single* = 1; *Burglar* = 1.
(□) *Shark* = 1; *20s* = 1; *JH* = 1; *Single* = 1; *Burglar* = 0.5; *Pusher* = 0.25; *Bookie* = 0.25.

This is the equivalent of presenting a listener with a clear and unambiguous example of the word 'legislature' and asking her what word she heard. The crosses show what happens when *High School* is changed to *Junior High* in the input activity pattern. This input is no longer an accurate description of anyone in the database (the equivalent of presenting a listener with '*legi*<cough>*lature*'). A system which tried to find an exact match to the input would fail. There is no person who matches that input pattern in the database. But the network has no problem. It takes slightly longer to respond (i.e. Ken's name takes more time to become activated, and does not reach such a high level) but *Ken* still comes out as the clearly preferred item retrieved from memory. The squares show what happens with an even more ambiguous input. In this the *Bookie* and *Pusher* nodes have all been activated as well as *Burglar*. As long as the input pattern is closer to a description of Ken than to any alternative, the system makes a clear decision in favour of the response 'Ken'[5].

The fact that connectionist systems work by constraint satisfaction is the reason why they exhibit fault tolerance. No part of the input uniquely determines the outcome. The network's response is the best fit it can make between the current input and information it has acquired in the past. It would be possible to devise a system, based on address + contents information storage like table 2.1, which, if given an input that failed to match any stored information, could compute a best fit. However, this would be a time consuming process once possible inputs achieved any degree of complexity. The parallel distributed computation in the Jets and Sharks database automatically computes a best fit between input and stored information. It would continue to do so whatever the degree of complexity of the patterns describing the individual entries without taking any more time.

There is no distinction between 'memory' and 'processing' in connectionist models

One general point to note about distributed representations is that they blur the distinction between memory and processing. Traditional models of cognitive processes often distinguish between 'memory', a store of learnt information, and 'processing', operations which enable the system to interpret incoming information. The processing operations may use information from memory, but the conceptual distinction is clear. Indeed, in many models this is made explicit with separate parts of the model labelled 'memory' and 'processor'. Such models exploit the analogy to conventional digital computers where there are independent systems for storing information and for processing it.

There is no such distinction in connectionist models. All the information which the network has—its memory—is stored in the weights of the connections between units. All the processing that the net can do is determined by the same set of weights.

[5]The interaction of information in the Jets and Sharks system produces some strange and unpredictable results which can only be appreciated by playing with the iac model. For example, figure 2.5 shows that if you wait long enough the *less* accurate description of Ken produces a stronger preference for his name than the more accurate description.

Problems for distributed representations

We have emphasised the advantage of distributed representations over localist representations in allowing the system some degree of resistance to damage and tolerance of noisy inputs. Given the unreliable nature of the matter which the brain uses for computation, it seems inevitable that it would use distributed representations. However, there are two properties of human memory which would not seem to be easy to account for with connectionist models but which would be expected with localist information storage: First, the addition of new information does not necessarily cause the loss of old. Second, learning can be immediate.

In a distributed system, any new information has to be added to the connections which already carry the system's current store of knowledge. To add the new information the strength of connections must be changed. If this is done in a single trial, addition of new information is likely to lead to some loss of old information. In a localist system, in contrast, the addition of new information is no problem. It is simply added to new storage locations and does not affect old information. At this point, however, we will just suggest that at an intuitive level there are different sorts of human learning.

At one extreme it seems clear that some sorts of knowledge can be acquired immediately, without interfering with other information. All young chess players are shown the smothered mate sequence with a queen sacrificed to a rook on g1 followed by mate of the king on h1 by a knight moving from h3 to f2. If you understand chess you only have to see this sequence once to remember it for ever, despite the fact that you may never have an opportunity to use it in a game. Similarily, if you knew anything about the British ex-Prime Minister Mrs Thatcher, and were told that her nickname at school was 'Bossy Roberts', you would be unlikely to forget it. A novel piece of information which you find interesting or amusing, in a domain where you already have sufficient knowledge to understand its significance, is likely to be remembered after a single presentation. And it can be retrieved as a specific item of information in future, independent of any other facts in the database. It is difficult to believe that such acquisition is accompanied by the loss of any other information. Quick, cost-free, addition of new information to existing databases characterises certain sorts of human knowledge acquisition. It is natural with localist representation of knowledge—you just add another entry to the database. But it is difficult to

see how it can happen with a distributed system. (That connectionist models *can* perform one trial learning will be shown in chapter 13 which demonstrates a model of the role of the hippocampus in episodic memory formation.)

At the other extreme there are many areas of knowledge acquisition, such as learning to play tennis or learning to talk, where acquisition of new knowledge is gradual, and accompanied by the modification or loss of previous patterns. As your tennis serve improves or you learn to pronounce the language correctly you *want* to lose some aspects of your old response patterns because they were inaccurate. Later it is difficult to recall when a specific piece of information was added to the database. Such a pattern, where new information is inextricably interwoven with old, occurs naturally with a distributed system, but not with a localist one. Distinctions between different sorts of knowledge representation occur in many models of the cognitive system. Most of the connectionist learning algorithms we will look at are more appropriate for modelling the latter sort of acquisition and representation than the former.

3 *Pattern association*

This chapter will introduce the architecture and properties of one specific kind of network, a pattern associator, and the operation of one particular learning rule, the Hebb rule. During training a pattern associator is presented with pairs of patterns. If learning is successful then the network will subsequently recall one of the patterns at output when the other is presented at input. After training, a pattern associator can also respond to novel inputs, generalising from its experience with similar patterns. Pattern associators are tolerant of noisy input and resistant to internal damage. They are capable of extracting the central tendency or prototype from a set of similar examples.

The first two chapters introduced general principles which are shared by all connectionist networks. In this and the next four chapters we will look in detail at the structure and properties of a variety of specific network architectures: pattern associators, autoassociators, competitive nets and recurrent nets. Networks can be trained in a variety of ways. In this chapter we will look at one particular learning rule, the Hebb rule. In chapter 4 we will look at the Delta rule and in chapter 5 at backpropagation.

The examples and the networks in these early chapters have deliberately been kept very simple so that the principles involved in their operation can be seen at work. It may seem that the problems they solve are so trivial that they have little to do with human cognition. Don't worry. In chapters 8–14 we will look at networks which have been scaled up to the point where they can simulate realistic aspects of human behaviour. For example, in chapter 8 we will look at a model with roughly 1000 processing units and 200 000 connections which learns to pronounce all the monosyllabic words in the English language. In chapter 13 we will look at a model of episodic memory formation in the hippocampus with about 4000 processing units and about half a million connections. Exactly the same principles will be at work in these networks as in the examples which follow.

The architecture and operation of a pattern associator

A fundamental task for the nervous system is to discover the structure of the world by finding what is correlated with what. That is, to learn to associate one stimulus