# Rethinking Innateness: A Connectionist Perspective on Development

Jeffrey L. Elman, Elizabeth A. Bates,
Mark H. Johnson, Annette Karmiloff-Smith,
Domenico Parisi and Kim Plunkett

The MIT Press

*From The MIT Press*

MITCogNet

First MIT Press paperback edition, 1998


Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Rethinking innateness: a connectionist perspective on development / Jeffrey L. Elman . . . [et al.]
p. cm.—(Neural network modeling and connectionism : X)
"A Bradford book."
Includes bibliographical references and index.
ISBN 0-262-05052-8 (hb : alk. paper), 0-262-55030-X (pb)
ISBN-13 978-0-262-05052-4 (hb : alk. paper)—
ISBN-13 978-0-262-55030-7 (pb)
1. Nature and nurture. 2. Connectionism. 3. Nativism (Psychology) I. Elman, Jeffrey L. II. Series: Neural network modeling and connectionism : 10.
BF341.R35 1996
155.7—dc20 96–15522
CIP

10 9 8

CHAPTER 2

# Why connectionism?

## A conversation

*B: A little bird told me you're writing a book about connectionism and development. I didn't believe it. You must be losing it! What on earth is a nice girl like you doing in such bad company? You must know that connectionism is nothing more than associationism in high tech clothing. I thought you believed in constructivism, interactionism, epigenesis and all that murky stuff.*

*A: Oh, I'm still a believer! But the connectionist framework allows me to come up with a much more precise notion of what all that stuff really means. It can inspire a truly interactive theory about developmental change. Associationist models rested on assumptions of linearity. The multi-layer nets connectionists use are nonlinear dynamical systems, and nonlinear systems can learn relationships of considerable complexity. They can produce surprising, nonlinear forms of change. They've made me completely rethink the notion of stages.*

*B: I don't believe my ears! Connectionist nets are simply reactive, they just respond to statistical regularities in the environment.*

*A: No, that may have been true of the earliest models, but the more complex ones develop internal representations that go well beyond surface regularities to capture abstract structural relationships.*

*B: Yeah, but how will you account for the rule-governed behavior of intelligent humans. Networks may be okay at the implementation level, but they can't represent rules of grammar and things like that, and that's what human intelligence is all about.*

*A: Another misconception! The transformations that occur during processing in networks do the same work as rules in classical systems. But what's interesting is that these rules and representations take a radically different form from the explicit symbols and algorithms of serial digital computers. The representations and rules embodied in connectionist nets are implicit and highly distributed. They capture our intuitions about the status of rules in infants' and young children's knowledge. Part of the challenge of modern research on neural networks is to understand exactly what a net has learned after it has reached some criterion of performance.*

*B: From the sublime to the ridiculous! The next thing you'll say is that connectionism is compatible with nativist ideas too! Come on, it's tabula rasa personified! Anyway, though they claim to build nothing in, the connectionists sneak in the solutions by fixing the weights and connections or laying out the solution in the way they represent the input.*

*A: Wrong again! First there are lots of different kinds of connectionism and we're trying to make the case for a biologically- and developmentally-inspired connectionism. Second, connectionism's not incompatible with innately specified predispositions—just depends how you define the predispositions. You're gonna have to read the book! You're right that many early simulations assumed something like a tabula rasa in the first stages of learning (e.g., a random "seeding" of weights among fully-connected units before learning begins). This has proven to be a useful simplifying assumption, in order to learn something about the amount and type of structure that does have to be assumed for a given type of learning to go through. But there is no logical incompatibility between connectionism and nativism. The problem with current nativist theories is that they offer no serious account of what it might mean in biological terms for something to be innate. In neural networks, it is possible to actually explore various avenues for building in innate predispositions, including minor biases that have major structural consequences across a range of environmental conditions. As for sneaking in the solutions, we do a lot less of that than classical systems. Don't forget that connectionist networks are self-organizing systems that learn how to solve a problem. As the art is currently practiced, the only one who fiddles with the weights is the sys-*

*tem itself in the process of learning. In fact in a simulation of any interesting level of complexity, it would be virtually impossible to reach a solution by "hand-tweaking" of the weights. As for the issue of "sneaking the solution into the input," don't forget that there are several examples of simulations in which the Experimenter did indeed try to make the input as explicit as possible—and yet the system stubbornly found a different way to solve the problem! Connectionist systems have a mind of their own and very often surprise their modelers.*

*B: Jesus, I need a drink! A mind of their own! What next?*

*A: Okay, I set that one up, but seriously, the way networks learn is surprisingly simple, yet the learning yields surprisingly complex results. We think that complexity is an emergent property of simple interacting systems—you see this throughout the physical and biological world.*

*B: So the next thing I'm gonna hear is that connectionism has some biological plausibility! Lip service to neurons! Spare us, please!*

*A: Well, connectionists work at many different levels between brain and behavior. In current simulations of higher cognitive processes, you're right, the architecture is "brain-like" only in a very indirect sense. The typical 100-neuron connectionist toy is "brain-like" only in comparison with the serial digital computer (and don't forget, old boy, that serial digital computers are wildly unlike nervous systems of any known kind). There are two real questions: First, is there anything of interest that can be learned from simulations in simplified systems, and second, can connectionists "add in" constraints from real neural systems in a series of systematic steps, approaching something like a realistic theory of mind and brain? You know, there are many researchers in the connectionist movement who are trying to bring these systems closer to neural reality. Some are exploring analogues to synaptogenesis and synaptic pruning in neural nets. Others are looking into the computational analogues of neural transmitters within a fixed network structure. The current hope is that work at all these different levels will prove to be compatible, and that a unified theory of the mind and brain will someday emerge. Don't tell me we are a long way off, that's obvious. But most connectionist researchers are really committed to ulti-*

*mate neural plausibility, which is more than you can say for most other approaches. Anyway, what's really exciting is that it has launched a new spirit of interdisciplinary research in cognitive neuroscience, and why I got excited is that it has really crucial implications for getting a developmental perspective into connectionism and a connectionist perspective into developmental theorizing.*

*B: Well, you've got a lot of arguing to do to convince me about that!*

*A: As I said just now, you'll have to read the book! So where's that gin and tonic?*

## *Nuts and bolts (or nodes and weights)*

The first thing to be said about connectionist networks is that most are really quite simple, but their behaviors are not. That is part of what makes them so fascinating. We will describe some of these behaviors in detail in this chapter, particularly those which are relevant to the developmental issues we are concerned with in this book (e.g., innateness, modularity, domain specificity, etc.). Before doing that, however, we need to have some working knowledge of the nuts and bolts of the framework. We therefore begin with an overview of basic concepts which characterize most connectionist models.

### Basic concepts

There is considerable diversity among connectionist models, but all models are built up of the same basic components: Simple processing elements and weighted connections between those elements. The processing elements are usually called *nodes* or *units*, and are likened to simple artificial neurons. As is true of neurons, they collect input from a variety of sources. Some nodes receive and send input only to other nodes. Other nodes act like sensory receptors and receive input from the world. And still other nodes may act as effectors, and send activation outward. (Some nodes may even do

all three things.) Figure 2.1 illustrates several architectures. Here, we use the convention of a filled circle to represent a node, lines between nodes to indicate communication channels (conceptually, roughly similar to dendrites and axons), and arrows to indicate the direction of information flow.

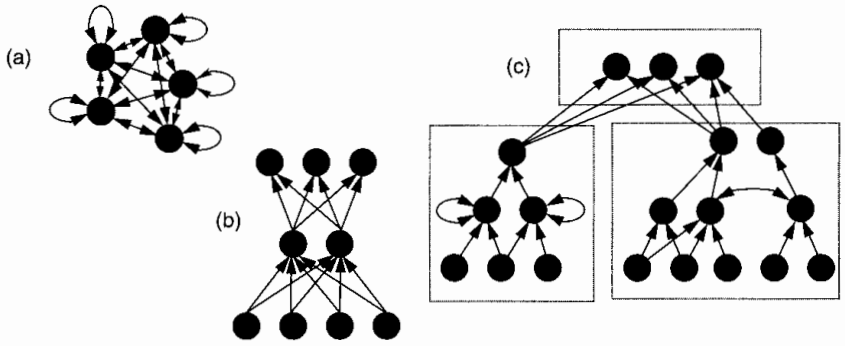


**FIGURE 2.1** Various types of connectionist networks. (a) A fully recurrent network; (b) a three-layer feedforward network; (c) a complex network consisting of several modules. Arrows indicate direction of flow of excitation/inhibition.

We can expand this basic picture in a bit more detail by considering the dynamics of processing. A node receives input from other nodes to which it is connected. In some networks these connections are unidirectional (as in Figure 2.1b). In other cases, connections may be bidirectional (Figure 2.1a). The connections between nodes have *weights*. It is in the weights that knowledge is progressively built up in a network. These are usually real-valued numbers, e.g., $1.234$, $-4.284$. These weights serve to multiply the *output* of the sending nodes. Thus if node $a$ has an output of $0.5$ and has a connection to node $b$ with a weight of $-2.0$, then node $b$ will receive a signal of $-1.0$ $(0.5 \times -2.0)$; in this case the signal would be considered *inhibitory* rather than *excitatory*.

A given node may receive input from a variety of sources. This input is often simply added up to determine the *net input* to the node (although other types of connections have been proposed in

which products of inputs are taken, as in *sigma-pi* units). It is useful to develop a formalism for describing the manner in which the activity of a node is computed from other sources of activity in the network. Let us suppose that the single input coming from some node $j$ is the product of that node $j$'s activation (we'll call this number $a_j$) and the weight on the connection between node $j$ and node $i$ (we'll designate this weight $w_{ij}$ where the first subscript denotes the receiving node and the second subscript denotes the sending node—a convention adopted from matrix notation in linear algebra). Then the single input from node $j$ is the product $w_{ij}a_j$. The total input over all incoming lines is just the sum of all of those products (we'll use the symbol $\Sigma_j$ to indicate summation from all node sources $j$). More compactly, we can write the net input to node $i$ as

$$net_i = \sum_j w_{ij}a_j \qquad \textbf{(EQ 2.1)}$$

This is the total input received by a node. But like neurons, the *response* of the node is not necessarily the same as its input. As is true of neurons, some inputs may be insufficient to cause the node to "do" very much. What a node "does" is captured by the node's activation value. So we want to know what the *response function* of a node is: For any given input, what is the corresponding output (or activation)?

In the simplest case, if the node is a *linear unit*, the output activation is in fact just the same as its net input. Or the node may have a slightly more complex output function, emitting a 1.0 just in case the net input exceeds some threshold, and outputting a 0.0 otherwise. The Perceptron and the McCulloch-Pitts neuron both had this characteristic; they are examples of *linear threshold units.* (Note that, despite their name, they have an important nonlinearity in their response.)

A more useful response function is the logistic function

$$a_i = \frac{1}{1 + e^{-net_i}} \qquad \textbf{(EQ 2.2)}$$

(where $a_i$ refers to the activation (output) of node $i$, $net_i$ is the net input to node $i$, and $e$ is the exponential). We have graphed the activation of a node with this activation function in Figure 2.2.

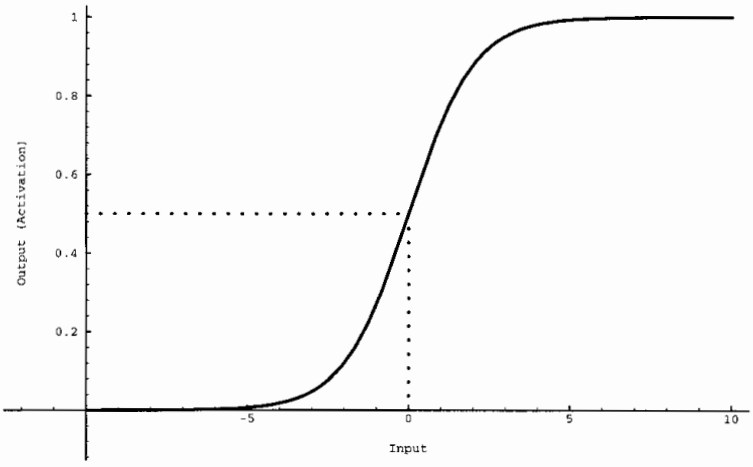


**FIGURE 2.2** The sigmoid activation function often used for units in neural networks. Outputs (along the ordinate) are shown for a range of possible inputs (abscissa). Units with this sort of activation function exhibit an all or nothing response given very positive or very negative inputs; but they are very sensitive to small differences within a narrow range around 0. With an absence of input, the nodes output 0.5, which is in the middle of their response range.

This figure tells us what the output is (the vertical axis) for any given net input (the horizontal axis). For some ranges of inputs (large positive or negative ranges along the horizontal) these units exhibit an all or none response (i.e., they output a 0.0 or 1.0). This sort of response lets the units act in a categorical, rule-like manner. For other ranges of inputs, however, (close to 0.5) the nodes are very sensitive and have a more graded response. In such cases, the nodes are able to make subtle distinctions and even categorize along dimensions which may be continuous in nature. ***The nonlinear response of such units lies at the heart of much of the behavior which makes such networks interesting.***

Note that in the absence of input (0.0 on the horizontal axis), the node's output is 0.5, which is right in the middle of its possible response range. Often this is a reasonable response, because it means that with no input, a node's response is equivocal. But sometimes times it is useful for nodes to have a default value other than 0.5, so that in the absence of input, they might be either "off" (output 0.0), "on" (output 1.0), or perhaps take some other intermediate value. This notion of different defaults is similar to the idea of varying thresholds, and can be accomplished by giving each node one additional input from a pseudo-node, called the *bias node*, which is always on. The weight on the connection from the bias node may be different for different units. Since each unit will always receive input from the bias unit, this provides a kind of threshold or default setting, in much the same way as humans exhibit default reactions to stimuli in the absence of additional data.

It is not difficult to see how simple networks of this kind might compute logical functions. For example, logical AND could be implemented by a network with two input units and a single output unit. The output unit would be "on" (have a value close to 1.0) when both inputs were 1.0; otherwise it would be off (close to 0.0). If we have a large negative weight from the bias unit to the output, it will by default (in the absence of external input) be off. The weights from the input nodes to output can then be made sufficiently large that if both inputs are present, the net input is great enough to turn the output on; but neither input by itself would be large enough to overcome the negative bias.

As this example makes apparent, part of what a network knows lies in its architecture. A network with the wrong or inappropriate input channels, with inputs which don't connect to the output, or with the wrong number of output units, etc., cannot do the desired job. Another part of a network's knowledge lies in the weights which are assigned to the connections. As the AND example shows, the weights are what allow the correct input/output relation to be achieved.

How do we know what architectures to choose, and what weights? In much of the connectionist work done in the early 1980's, and in an approach still pursued by many researchers today,

networks are designed by hand and reflect theoretical claims on the part of the modeler. For example, in the word perception model of McClelland and Rumelhart (1981; Rumelhart & McClelland, 1982), there are separate layers of nodes which are dedicated to processing information at the *word, letter,* and *orthographic feature* levels (see Figure 2.3). The connections between nodes within and across layers reflect the non-arbitrary relationships between the concepts represented by the nodes. Thus, the node for the word "trap" receives positively weighted input from the letter nodes "t", "r", "a", and "p"; and it is inhibited by other word nodes (since only one word may be reasonably present at once).

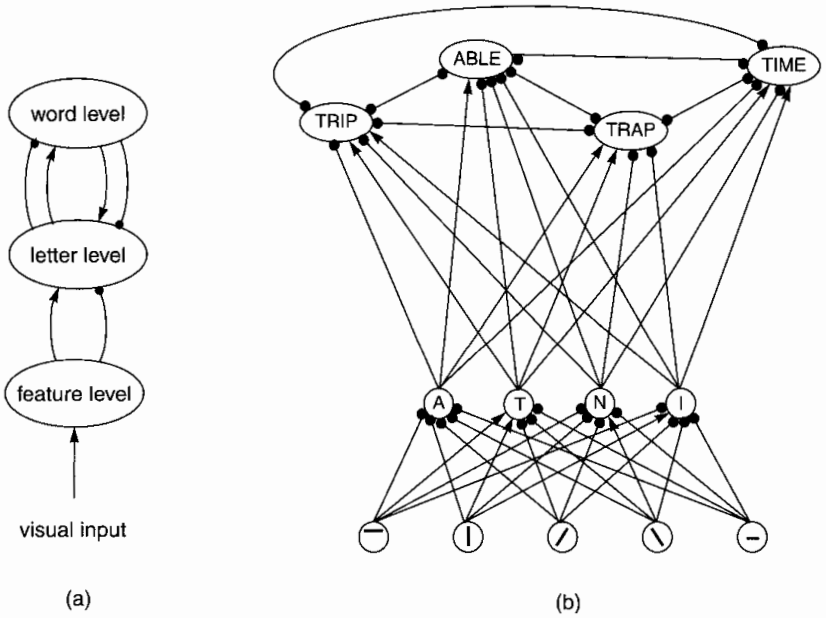


**FIGURE 2.3** (a) Global view of word perception model of McClelland & Rumelhart. (b) Detailed fragment of the model. Connections ending in arrows are excitatory; connectionist ending in filled circles are inhibitory. (Adapted from McClelland & Rumelhart (1981).

This model instantiates a theory which McClelland and Rumelhart developed in an attempt to account for a range of experimental data. Even though the model was hand-crafted and relatively simple, it exhibits a number of behaviors which are not obvious simply from inspecting the model. As is often the case with these systems, the nonlinearities and high degree of interaction give rise to phenomena that were not themselves deliberately programmed in. (It would be wrong to say they were not programmed in, since they obviously do result from the specifics of the way the model has been designed. What is relevant here is just that they do not result from the modeler's intentional efforts to produce the behaviors.) Indeed, some of these behaviors are quite unexpected and make predictions about human behavior which can then be tested experimentally.

## *Learning*

There are other significant aspects of models such as the McClelland and Rumelhart model about which we shall have more to say when we talk about representation. For now we note that the hand-wiring which is required may be problematic. As the degrees of freedom (these include, among other things, the number of nodes and connections) grow, so too does the number of ways of constructing networks. How do we know which is the best way? Or, equivalently, how do we know we have the right theory? There may be cases where we feel we have a good idea about the inputs and outputs which are relevant to some behavior—these can be observed directly. But what we want is to use the model to help *develop* a theory about the internal processing which gives rise to this behavior, rather than just *implementing* a theory we already hold. The question then becomes, is there some way by which networks can configure themselves so that they have the appropriate connections for a task? Can we use the networks for theory development?

## Hebbian learning

One of the earliest ideas about how networks might learn came from Donald Hebb. Speaking of real nervous systems, Hebb suggested that

> *When an axion of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.* (Hebb 1949; p. 62)

There are various ways of capturing this notion mathematically, but Hebb is essentially proposing here that learning is driven by correlated activity. Thus, in an artificial system in which changes in connection strength plays the role of changes in synaptic potentiation, the rule for weight change might be

$$\Delta w_{ij} = \eta a_i a_j \qquad \textbf{(EQ 2.3)}$$

(where $\Delta w_{ij}$ represents the change in the weight on the connection from sending node $j$ to receiving node $i$, $\eta$ is a small constant of proportionality (the learning rate), and $a_i$ and $a_j$ are the activations of the two nodes). The Hebb rule has been studied extensively and is widely used in modeling today. It has at least two attractive virtues: there are known biological mechanisms which might plausibly implement it (e.g., Long Term Potentiation, Bliss & Lømo, 1973); and it provides a reasonable answer to the question, "Where does the teacher information for the learning process come from?" It is easy to believe that there is a great deal of useful information which is implicit in correlated activity, and the system need not know anything in advance. It simply is looking for patterns.

However, there is a serious limitation to Hebbian learning, which is that *all* it can learn is pair-wise correlations. There are cases where it is necessary to learn to associate patterns with desired behaviors even when the patterns are not pair-wise correlated, or may exhibit higher-order correlations. The Hebb rule cannot learn to form such associations.

## The Perceptron Convergence Procedure

One solution to this dilemma is to base learning on the difference between a unit's *actual* output and its *desired* output. Two early techniques were proposed by Rosenblatt (1959, 1962) and Widrow and Hoff (1960). The approaches were very similar, and we will focus here on Rosenblatt's.

Rosenblatt proposed what he called the Perceptron Convergence Procedure (PCP; Rosenblatt, 1962). This is a technique which makes it possible to start with a network of units whose connections are initialized with random weights. The procedure specifies a way to take a target set of input/output patterns and adjust the weights automatically so that at the conclusion of training the weights will yield the correct outputs for any input. Ideally, the network will even generalize to produce the correct output for input patterns it has not seen during training.

The way that learning works is that the input pattern set is presented to the network, one pattern at a time. For each input pattern, the network's actual output is compared with the target output for that pattern. The discrepancy (or error) is then used as a basis for changing weights to inputs, and also changing the output node's threshold for firing. How much a given weight is changed depends both on the error produced and the activation coming along a weight from a given input.[1] The underlying goal is to minimize error on an output unit by apportioning credit and blame to the inputs (the intuition being that if you have made a mistake, you should probably play more attention to the people who were shouting loudest at you to go ahead and make that mistake).

This learning procedure (as well as the Widrow-Hoff rule) addresses some of the limitations of Hebbian learning, but it has its own limitations. It only works for simple two-layer networks of input units and output units. This turns out to be an unfortunate limitation. As Minsky and Papert (1969) showed, there are classes of problems which can not be solved by two-layer networks of percep-

1. There are actually several variations; in some the weight and thresholds change by a constant.

trons. The best known of these is the exclusive-OR (XOR) problem, but the basic problem is a very deep one which occurs in many situations. Two-layer networks are rather like S-R pairs in classical psychology. What is required is something between input and output that allows for internal (and abstract) representations.

## *Similarity in neural networks—a strength and a weakness*

One of the basic principles which drives learning in neural networks is *similarity.* Similar inputs tend to yield similar outputs. Thus, if a network has learned to classify a pattern, say 11110000, in a certain way then it will tend to classify a novel pattern, e.g., 11110001, in a like fashion. Neural networks are thus a kind of analogy engine. The principle of similarity is what lets networks generalize their behaviors beyond the cases they have encountered during training.

In general, the similarity metric is a good rule-of-thumb. Lacking other information, it is reasonable that networks (and people) should behave in similar ways given similar situations. On the other hand, there are clearly cases where we encounter two patterns which may resemble each other superficially, but which should be treated differently. A child learning English must learn that although "make" and "bake" sound almost the same, they form the past tense in different ways. Even more extreme is the case where two words may be identical at the surface level. For instance, the word "did" can either be a modal (as in "I did go") or a main verb (as in "I did badly"). Despite their surface identity, we must learn to treat them differently. More prosaically, kittens and tigers may share many physical features but we do not want to treat them in the same way. The problem thus is that although similarity is often a good starting point for making generalizations, and lacking other information we do well to rely on similarity, there are many circumstances in which similarity of physical form leads us astray. We might still wish to invoke the notion of similarity, but it is an

abstract or functional kind of similarity. Kitty cats, dogs, fish, and parakeets are dissimilar at the level of appearance but share the abstract feature "domestic animal"; tigers, rhinoceri, wild boar, and cobras do not look at all alike but are similar at the more abstract level of being "wild animals."

The neural network learning procedures we have described so far (Hebbian learning and the PCP with two-layer networks) have the important limitation that they can only learn to generalize on the basis of physical similarity. (To make this more precise, by physical similarity we mean here similarity of *input pattern*.) In fact, the PCP is actually guaranteed to discover a set of weights which give the right response to novel inputs, provided the correct response can be framed in terms of physical similarity (defined in a certain way). Conversely, when the correct response cannot be defined in terms of similar inputs yielding similar outputs, the PCP is guaranteed to fail!

One of the best known problems which illustrates the limitation of the PCP and simple two-layer networks is the Exclusive Or problem (XOR). XOR is a what is known as a Boolean function. (A Boolean function just means that we have to take some set of inputs, usually 1s and 0s, and decide whether a given input falls into a positive or a negative category. These categories are often denoted "true" or "false," or if we are dealing with networks, produce output node activations of 1 or 0.) In the case of XOR, the inputs consist of a pair of numbers (either one of which can be a 1 or a 0), and the task is to decide whether the pair falls into the "true" category or the "false" category. For a network this would mean taking inputs of the form shown in Table 2.1 and producing the appropriate output value.

**TABLE 2.1**

| INPUT | | OUTPUT |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

Why can't a two-layer perceptron network solve this problem? Let's imagine we have a network that looks like the one shown in Figure 2.4. It is not hard to see why this problem cannot be solved

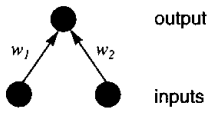


**FIGURE 2.4** two-layer network that cannot learn to solve the XOR problem.

by such a network. Consider what we would be asking of the two weights ($w_1$ and $w_2$) in this network in order to produce the correct input/output mappings.

First, remember that these weights are multipliers on the inputs, so when we want an output to be "true" (i.e., 1) at least one of the inputs must be a 1, and at least one of the weights must be big enough so that when the two numbers are multiplied, the output node turns on. Now, the first two patterns (0,0 and 1,1) have to produce a 0 on the output. For that to happen, the weights connecting the inputs to the output node be sufficiently small that the output node will remain off even if both input nodes are on. That's easily enough obtained: just set both weights to 0. That way, even if both inputs are on (1,1), the output unit will remain off.

But this is at odds with what is required for the third and fourth patterns (01 and 10). In these cases, we need the weights from *either* input to be sufficiently big such that one input alone will activate the output. Thus we have contradictory requirements; there are no set of weights which will allow the output to come on just in case either of the inputs is on, but which will keep it off if both are on.

There is another way of thinking about this problem, and it's worth introducing now because it allows us to introduce a framework for thinking about representations in networks which we have found very useful. It is also a bit more intuitively understandable than simply thinking about weights and such.

We have referred to patterns such as 0,0 and 1,1 as vectors. A vector can be thought of either as a collection of numbers, or as a point in space. If our vectors consists of two numbers, as in the case of our inputs in this example, then the vector points in a two dimensional space, with each number indicating how far out along the dimension the point is located. We could thus visualize the four input patterns that make up the XOR problem in the two dimensional space shown in Figure 2.5.

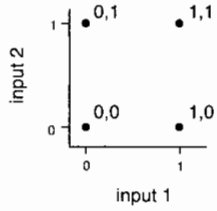


**FIGURE 2.5** Spatial representation of the four input patterns used in XOR.

Thinking of these patterns as points in space now gives us a way of defining a bit more precisely what we mean by similarity. It is simple: we judge the similarity of two vectors by their (Euclidean) distance in space. As a first pass at understanding why XOR is difficult, notice that the pairs of patterns which are furthest apart—and therefore most dissimilar—such as 0,0 and 1,1 are those which need to be grouped together by the function. Nearby patterns such as 0,1 and 0,0, on the other hand, are to be placed in different groups. This goes against the grain.

The problem is actually a bit worse. Technically, the difficulty is that the input/output weights impose a linear decision bound on the input space; patterns which fall on one side of this decision line are classified differently than those patterns which fall on the other side. When groups of inputs cannot be separated by a line (or more generally, a hyperplane) then there is no way for a unit to discriminate between categories. This is shown in shown in Figure 2.6. Such problems are called *nonlinearly separable* (for the simple reason that the categories cannot be separated by a line).

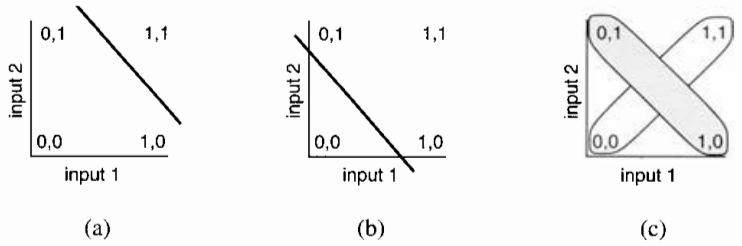


**FIGURE 2.6** Geometric representation of the XOR problem. If the four input patterns are represented as vectors in a two-dimensional space, the problem is to find a set of weights which implements a linear decision boundary for the output unit. In (a), the boundary implements logical AND. In (b), it implements logical OR. There is no linear function which will simultaneously place 00 and 11 on one side of the boundary, and 01 and 10 on the other, as required for (c).

Although this example may seem a bit arcane, it is useful because it illustrates the idea of vectors as patterns in space, of similarity as distance in space, and of the problem that can arise when patterns which are distant in space (and therefore inherently dissimilar) must nonetheless be treated as though they were similar.

The problem, then, is how to take advantage of the desirable property of networks which allows similarity to have a role in generalization, while still making it possible to escape the tyranny of similarity when it plays us false. We want to be able to define similarity at an abstract and not only form-based level. (Or, to put it another way, we want to have our cake and eat it too.)

## Solving the problem: Allowing internal representations

It turns out that this problem is relatively easy to solve. With an extra node or two interposed between the input and output (see Figure 2.7) the XOR problem can be solved.[2] These additional nodes

2. A two-layer network can actually solve XOR if the output is not a perceptron-like unit. More precisely, the output cannot have an activation function which is a monotonically increasing function of its input. A cosine activation function, for instance, will suffice.

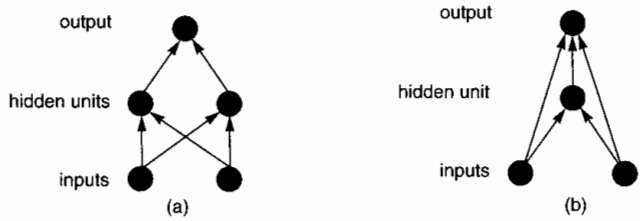


**FIGURE 2.7** Two architectures for solving the XOR problem. In both cases, at least one internal ("hidden") unit is required.

are often called "hidden units" because they are hidden from the world. They are equivalent to the internal representations we invoke in psychological theorizing. The inputs and outputs to hidden units are confined to other parts of the network. Hidden units are extraordinarily powerful; they make it possible for networks to have internal representations of inputs which capture the more abstract, functional relationships between them. Indeed, it is tempting, and not entirely unwarranted, to think of input units as analogous to sensors, output units as motor effectors, and hidden units as interneurons. However, while this metaphor captures an important aspect of what units do, we stress that there are occasions in which inputs and outputs may be interpreted in very different ways and not correspond to sensory/motor maps at all. Input similarity still plays an important role, and all things being equal, the physical resemblance of inputs will exert a strong pressure to induce similar responses. But hidden units make it possible for the network to treat physically similar inputs as different, as the need arises. The way they do this is by transforming the input representations to a more abstract kind of representation.

We can use the example of XOR to illustrate this point. Once again the spatial interpretation of patterns proves useful. In Figure 2.8 we see in (a) what the inputs "look like" to the network. The spatial distribution of these patterns constitutes the intrinsic similarity structure of the inputs. The vectors representing these points in space are then passed through (multiplied) the weights between inputs and hidden units. This process corresponds to the

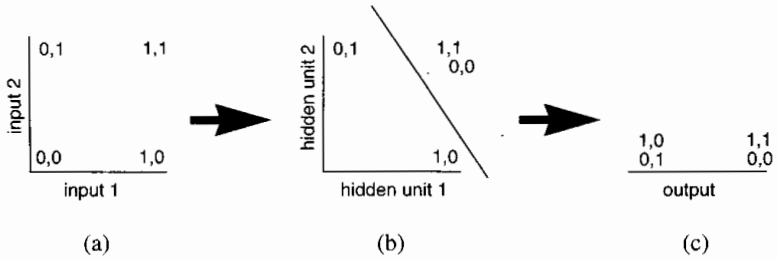


**FIGURE 2.8** Transformation in representation of the four input patterns for XOR. In (a) the similarity structure (spatial position) of the inputs is determined by the form of the inputs. In (b) the hidden units "fold" this representational space such that two of the originally dissimilar inputs (1,1 and 0,0) are now close in the hidden unit space; this makes it possible to linearly separate the two categories. In (c) we see the output unit's final categorization (because there is a single output unit, the inputs are represented in a one-dimensional space). Arrows from (a) to (b) and from (b) to (c) represent the transformation effected by input-to-hidden weights, and hidden-to-output weights, respectively.

first arrow. The weights from input to hidden units have the effect of transforming this input space; the input space is "folded," such that the two most distant patterns (0,0 and 1,1) are now close in the hidden unit space. (Remember that hidden unit activation patterns are vectors too, so we can graph the location of the activations produced on the hidden layer by any set of inputs.) This reorganization sets things up so that the weights to the output unit (second arrow, between (b) and (c)) can impose a linear decision bound, the line cutting across the space in (b), and resulting in the classification shown in (c).

Having said that hidden units can be used to construct internal representations of the external world which solve difficult problems (in fact, Hornik, Stinchcombe, and White, 1989, have proven that a single layer of hidden units gives networks the power to solve essentially any problem), all might seem rosy. But there's still a problem. It is one thing for a network to be able to *solve* a problem, in principle; this merely says that there exists some set of weights which enable the network to produce the right output for any input. It is another thing for the network to be able to *learn* these weights.

So in fact, although it had long been known that more complex multilayered networks could solve problems such as XOR, the real challenge (which Minsky and Paper suggested probably could not be solved) was how to *train* such networks. The PCP works with two layer networks, but not when there are additional hidden layers.

Fortunately, there are several solutions to this problem. One of the best known is called backpropagation of error (Rumelhart, Hinton, & Williams, 1986). Because many of the simulations we describe in this book use this learning procedure, we shall spend a bit of time describing how it works, at least at a general level.

## Backpropagation of error (informal account)

"Backprop," as it has come to be known, works very much like the PCP (or the Widrow-Hoff rule, which is also very similar; Widrow & Hoff, 1960). Recall that the approach of the PCP is to adjust the weights from input unit(s) to output unit(s) in such a way as to decrease the discrepancy between the network's actual output and its desired output. This works fine for the weights leading to outputs, because we have a target for the outputs and can therefore calculate the weight changes. When we have hidden units, however, the question arises: How shall we change the weights from inputs to hidden units? The strategy of credit/blame assignment requires that we know how much error is already apparent at the level of the hidden units—even before the output unit is activated. Unfortunately, we do not have a predefined target for the hidden units, only for the output units. So we cannot say what their activation levels should be, and therefore cannot specify an error at this level of the network.

Backprop solves this problem in a clever but entirely reasonable way. The first step is to figure out what the error is at the level of output units (just as in the PCP). This error is simply the difference between the activation we observe on a given output unit, and the activation it is supposed to have (commonly called the *target* or *teacher* value).

The second step is to adjust the weights leading into that unit so that in the future it is either more or less activated, whichever is

needed in order to decrease the error. We can do this on all the weights leading into the output unit. And if there are more than one output units, we simply repeat the same two steps (error calculation; weight change calculation) for each one, using each unit's own target to determine the error.

Now we come to the question of how to change the weights leading into the hidden units. The procedure we used for the hidden to output weights worked because we knew the target values for output units. How do we determine target values for hidden units? We assume that each hidden unit, because it is the thing which excites (or inhibits) the output units, bears some responsibility for the output units' errors. If a certain output unit is very far from its target value, and has been highly activated by a certain hidden unit, then it is reasonable to apportion some of the blame for that on the hidden unit. More precisely, we can infer the shared blame on the basis of (a) what the errors are on the output units a hidden unit is activating; and (b) the strength of the connection between the hidden unit and each output unit it connects to. Hence the name of the algorithm: We propagate the error information (often called the error signal) backwards in the network from output units to hidden units. Notice that this same procedure will work iteratively. If we have multiple layers of hidden units, we simply calculate the lower hidden layers' error based on the backpropagated error we have collected for the higher levels of hidden layers.

## Formal account

This informal account of backprop can be made somewhat more explicit and precise. In the remainder of this section we will attempt to provide a more rigorous treatment of the way the backpropagation learning works. Although not necessary for understanding how backprop works at an intuitive level, the formal account is worth at least perusing. The notation may seem daunting to those unfamiliar with mathematical formalism, but it is really just a shorthand to more compactly represent the concepts we have just discussed.

We said that the first step in the learning algorithm was to calculate the errors on the output units. We do this a unit at a time. Since the procedure is normally the same for all units, for the purposes of generality we will refer to an output unit with the index $i$. We will call the error for that unit $e_i$. The observed output for that same unit will be represented as $o_i$ and the output unit's target value will be $t_i$. Computing the error is simple. It is just

$$e_i = t_i - o_i \qquad \textbf{(EQ 2.4)}$$

Now that we know the error that is being produced for a unit, we want to adjust the weights going into that unit so that in the future it will be more or less activated, but in a way that reduces the error we just calculated. Since weights connect two units, we will use the subscripts $i$ to represent the unit on the receiving end of the connection, and the subscript $j$ to represent the sending unit. The weight itself can be referred to as $w_{ij}$—so the first subscript always refers to the receiving unit and the second subscript to the sending unit. The change in the weight, which is what we want to calculate, is $\Delta w_{ij}$.

The weight adjustment should be such that the error will be decreased as the weights are changed. This notion is captured by a construct called a partial derivative, which tells us how changes in one quantity (in our case, network error) are related to changes in another quantity (here, change in weights). Therefore, letting network error be symbolized as $E$, and the symbols $\frac{\partial E}{\partial w_{ij}}$ to represent the partial derivative of the error with respect to the weight, then the equation for the weight change can be formalized as

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \qquad \textbf{(EQ 2.5)}$$

(where $\eta$ is a small constant of proportionality called the learning rate).

We will not give the actual computations which can be performed on the partial derivative in order to convert it to a more usable form (see Rumelhart, Hinton, & Williams, 1986); we simply

report the result that the right-hand side of Equation 2.5 can be calculated as

$$\Delta w_{ij} = \eta \delta_{ip} o_{jp} \quad \textbf{(EQ 2.6)}$$

The quantities on the right-hand side of Equation 2.6 are now in a form which we can identify and begin to use in training a network.

This equation says that we should change the weights in accord with three things (corresponding to the three symbols on the right-hand side). First, we are trying to find a set of weights which will work for many different patterns. So we want to be cautious and not change the weights too much on any given trial. We therefore scale the calculated weight change by a small number. This is the term $\eta$ and is called the learning rate. Skipping to the third term, $o_{jp}$, we also make our weight change on the connection from the sender unit $j$ to receiver unit $i$ be proportional to $j$'s activation. That makes sense; after all, if the sender unit hasn't contributed any activation to unit $i$, it won't have contributed to $i$'s error. The second subscript $p$ in the expression $o_{jp}$ indicates that we are only considering the activation of unit $j$ in response to input pattern $p$. Different input patterns will produce different activations on unit $j$.

Finally, the middle term, $\delta_{ip}$, reflects the error on unit $i$ for input pattern $p$. We have spoken previously of the error as being simply the difference between target and output; the term $\delta_{ip}$ includes this but is a bit more complicated. The reason for that just has to do with the calculation of the partial derivative in Equation 2.5. Although we will not attempt to explain the derivation here, there is one interesting consequence to the definition of these $\delta_{ip}$ terms which we comment on below.

For an output unit ($i$), this term is defined as

$$\delta_i = (t_i - o_i) f'(net_i) \quad \textbf{(EQ 2.7)}$$

which says that our error is, straightforwardly, the difference between the target value for unit $i$ on this pattern and the actual output. This discrepancy is modulated by the derivative of the unit's current activation. As we said, we will not attempt here to

justify the presence of this derivative; it simply has to do with the way Equation 2.5 is worked out. However, this term has an important effect on learning.

The derivative of a function is its slope at a given point. The derivative measures how fast (or slowly) the function is changing at that point. If we look back at Figure 2.2, which shows the activation function for sigmoidal units, we see that the slope is steepest in the mid-range and decreases as the unit's activation approaches either extreme. This has important consequences for learning which we shall return to several times in this book. When networks are first created, they are typically assigned random weights values clustered around 0.0. This means that during early learning, activations tend to be in the mid-range, 0.5 (because no matter what the inputs are, they are being multiplied by weights which are close to 0.0; hence the net input to a unit is close to 0.0, and the activation function maps these into the mid-range of the receiving unit's activation). There is a further consequence: When weight changes are computed, the error term is modulated by the derivative of the unit (see Equation 2.7). This derivative is greatest in the mid-range. Therefore, all things being equal, the weights will be most malleable at early points in learning. As learning proceeds, the impact of any particular error declines. That's because once learning begins, weights will deviate more and more from 0.0 and the net input to a unit will tend to produce its minimum or maximum activation levels.

This has both good and bad consequences. If a network has learned the appropriate function, occasional outlier examples will not perturb it much. But by the same token, it may be increasingly difficult for a network to correct a wrong conclusion. Ossification sets in. The interesting thing about this phenomenon, from a developmental viewpoint, is that it suggests that ***the ability to learn may change over time—not as a function of any explicit change in the mechanism, but rather as an intrinsic consequence of learning itself.*** The network learns to learn, just as children do.

We have just seen how the learning algorithm works for weights from hidden to output units. How do we compute the $\delta_{ip}$ for hidden units? We cannot use the difference between the target

value and actual output, because we don't know what the targets for hidden units should be. But as we pointed out earlier, we can calculate a hidden unit's error in an indirect fashion from the error of the output units, because each hidden unit bears some responsibility for those errors. So for any hidden unit $i$, we collect the error of the $k$ output units to which it projects, weighted by the connections between them, and use that sum:

$$\delta_i = f'(net_i)\sum_k \delta_k w_{ki} \qquad \textbf{(EQ 2.8)}$$

This procedure can be used for networks with arbitrarily many levels of hidden units.

In summary, backprop is an extremely powerful learning tool and has been applied over a very wide range of domains. One of the attractions is that it actually provides a very general framework for learning. The method implements a gradient descent search in the space of possible network weights in order to minimize network error; but what counts as error is up to the modeler. This is most often the squared difference between target and actual output (as described here), but in fact, *any* quantity which is affected by weights may be minimized.

## Learning as gradient descent in weight space

Because the idea of gradient descent in weight space figures importantly in later chapters, we wish to be clear about what this means. A graphical representation is helpful here. Imagine a very simple network of the sort shown in Figure 2.9a.

There are two trainable weights in the network shown in Figure 2.9a. Let us suppose that we have a data set which we wish to train the network on. We might systematically vary the two weights through their possible range of values. For each combination of weights, we could pass the training data through the network and see how well it performs. We could then graph the resulting error as a function of all possible combinations of the two weights. A hypothetical version of such a graph is shown in Figure 2.9b. Regions of the surface which are low along the $z$ axis

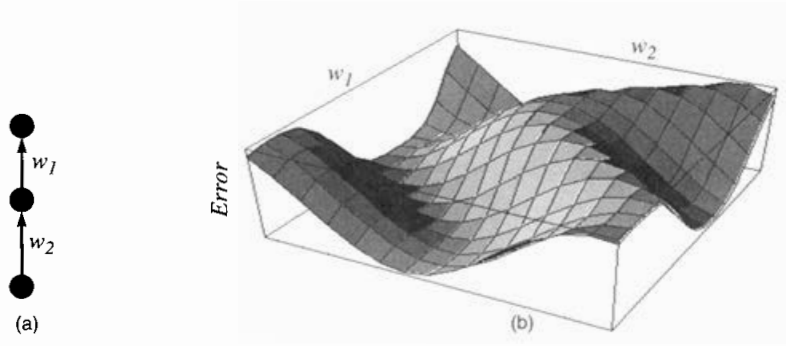


**FIGURE 2.9** (a) Simple network with 2 trainable weights. (b) A hypothetical graph depicting the error produced for some imaginary set of input/output training patterns, for all possible values of weights $w_1$ and $w_2$. Low regions on the surface correspond to low error.

tell us that the combination of weights ($x$ and $y$ axes) produce low error and so these are weight settings which are desirable.

This same technique might be used, in principle as a form of learning. We could discover good regions of weight space empirically by sampling the entire space. But this would be cumbersome and in networks of any complexity, quite impractical. What backpropagation (and many other neural network learning algorithms) provides is a technique for starting at some random point on this surface and following the downward gradient. If we are fortunate, there will exist some combination of weights which solves the problem, and we will find it.

Is success guaranteed? Not at all. We know from theoretical arguments that any problem *can* be solved with a three-layer network (Hornik, Stinchcombe, & White, 1989) but we do not know *a priori* the precise architecture needed to solve that problem (how many hidden units, what pattern of connectivity, etc.). Furthermore, if in the process of following the gradient we take very big steps as we change our weights we might overshoot a good region of weight space. Or we might find ourselves trapped in a region which is locally good but not perfect (this is called a *local minimum*). Since the gradient at this spot points up—all weight changes lead to poorer performance—we may be stuck. In fact, one of the hypothe-

ses of this book is that ***evolution produces a developmental profile which interacts with learning in just such a way as to cleverly help us avoid such local minima.***

### Other architectures and learning algorithms

Backpropagation and Hebbian learning are perhaps the best known and most widely used connectionist learning algorithms. There are a number of other alternative techniques for training networks, however. We chose not to delve into these largely because to do so would tax the patience of our readers, and there are many fine texts which give a more comprehensive presentation of network learning (see, for example, Hertz, Krogh, & Palmer, 1991 for an excellent mathematical presentation; and Bechtel & Abrahamsen, 1991, for a more cognitively/philosophically oriented review). Furthermore, the majority of the work which we will discuss employs either Hebbian or backpropagation learning, and so it is most important to us that readers be acquainted with these approaches. We simply wish to emphasize that the connectionist paradigm embraces a multitude of approaches to learning, and the interested reader should not imagine what we have presented here exhausts the range of possibilities.

## *Issues in connectionist models*

As exciting as the early successes in connectionist modeling have been, there remain a number of important challenges. We would like now to turn to some of the issues which the field is currently focussing on.

### Representing time

The networks we described in talking about backpropagation (e.g., the networks shown in Figure 2.4 and Figure 2.7) are called "feed-forward networks." The flow of activation proceeds from input

through successive layers and culminating in output units. The flow of information is unidirectional, and at the conclusion of processing an input, all activations are wiped clean and the next input is received.

Processing in such networks is atemporal, in the sense that the activations of units reflects only the current input. The only sense in which time figures in is if the network is still undergoing training, in which case the weight changes implicitly reflect the time course of learning. We might think of these changes in connection strength as implementing semantic long-term memory, in much the same way that Hebb envisioned it.

Early models attempted to circumvent the lack of temporal history in processing through an ingenious trick. Input units would be conceptualized as being divided into groups, with each group processing one input in a series. In the simplest case, for example, 10 temporally ordered inputs might be presented having the first input unit respond to the first event, the second input unit respond to the second event, etc. All 10 events/inputs would then be processed simultaneously.

Although this approach has been used to good effect in a number of models (e.g., McClelland & Rumelhart 1981; McClelland & Elman, 1986; Sejnowski & Rosenberg, 1986), it has some very basic defects (see Elman, 1990, for review). This has led researchers to investigate other ways of representing time. An enormous part of the behavior of humans and other animals is clearly time-dependent, and so the problem is a serious one. What would seem to be lacking in the feedforward network is an analog to short-term or working memory.

Recurrent networks implement short-term memory by allowing connections from nodes back to themselves, either directly or indirectly, as in Figure 2.10. In this way, the network's activity at any point in time can reflect whatever external input is presented to it, plus its own prior internal state (where that state corresponds to activations at prior points in time). There exist a variety of algorithms and architectures which make such recurrence possible (e.g., Elman, 1990; Jordan, 1986; Pearlmutter, 1989; Pineda, 1989; Rumel-

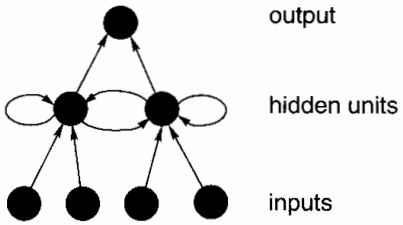


**FIGURE 2.10** A recurrent network. The network has fully interconnected hidden units; each hidden unit activates itself and the other hidden unit. As a result, the current state (activation) of a hidden unit reflects not only new input from the input units, but the hidden units' prior activation states.

hart, Hinton, & Williams, 1986; Stornetta, Hogg, & Huberman, 1988; Tank & Hopfield, 1987; Williams & Zipser, 1989).

## Scaling and modularity

The examples we have used so far have involved networks which are rather simple in design. The networks contain relatively few units; they tend to have uniform internal structure (e.g., they are organized into layers in which each layer is fully connected to the next layer); and they are trained on tasks which are usually very simplified versions of real-world situations.

No apologies are needed for such simplifications. In many cases the goal is to abstract away from irrelevant complexity in order to focus on the heart of a problem. In other cases, the simplification may be justified by the immaturity of the technology. We are still at a point where we are trying to understand basic principles. Much of the research is empirical, and it is useful to work with systems which are tractable.

There are also potential hazards associated with these simple architectures. Let us mention two.

First, there is the problem of scaling up to bigger and more realistic problems. For complex and large-scale problems, it is not clear that networks with uniform architecture are optimal. As a network grows in size (as it must, for larger-scale problems), the space of

potential free parameters—weights—grows exponentially, whereas the size of the training data typically grows more slowly. Techniques such as gradient descent typically do not do well in searching out good combinations of weights when the weight space is very large relative to the training data available. In Figure 2.9 we showed a very simple network and its associated (hypothetical) weight space. Gradient descent relies on the information provided by training examples to find the set of weights which yield minimum error. Imagine how much more complicated that task may be in a search space containing hundreds of weights, particularly if we have a relatively small number of examples. We are likely to get "trapped" in regions of weight space which work for a few of the examples but not for others.

One way to address the problem of determining optimal network architectures involves the use of what are called "constructive algorithms." These are procedures which allow a network to be dynamically reconfigured during training through the addition or deletion of nodes and weights. One of the best known techniques was proposed by Scott Fahlman, and is called Cascade Correlation (Fahlman & Lebiere, 1990). We'll see a developmental example of this in Chapter 3. Omitting details, this procedure works with a network that initially contains no hidden units. It then adds new hidden units gradually in order to reduce error. A somewhat different technique has been suggested by Steve Hanson (Hanson, 1990). In Hanson's scheme, connections weights are not discrete, but instead have a mean and variance. As training progresses, if a weight's variance grows too large the connection may split in two (hence the name, "meiosis networks"). Dynamic configuration through pruning has also been studied. Rumelhart (1987) and Weigend (1991) found that in many cases, a network's ability to generalize its performance to novel stimuli was improved by gradually eliminating weights, as long as this did not lead to increased error.

In addition to scaling, there is a second risk which can arise with overly simple networks. This problem is subtler. Here the difficultly is not just that there are two few or two many units in the network. It is that they are not connected in the best way. For example, sometimes a problem which appears hard can be solved if it is first

decomposed into smaller pieces. Then, rather than have a single network attempt to learn the entire problem as an undifferentiated whole, we might more efficiently use a network architecture which reflects the problem's functional decomposition. This is the insight which underlies a proposal by Jacobs and his colleagues (Jacobs, 1990; Jacobs, Jordan, & Barto, 1991; Jacobs, Jordan, Nowlan, & Hinton, 1991).

As an example of a difficult function which can be made easier if broken into pieces, Jacobs and colleagues point to the absolute value function. The absolute value of a number is simply its "positive value"; if the number is already positive, then that is also its absolute value. If the number is negative, then the absolute value negates that, making it positive. Formally, the function is defined as

$$f(x) = \begin{cases} -x & \text{if } (x < 0) \\ x & \text{if } (x \geq 0) \end{cases} \qquad \textbf{(EQ 2.9)}$$

This is a nonlinear function (because of the "bend" at 0), and can be learned by a single network which has at least one hidden unit. On the other hand, the function can also be learned by a network which has two modules (e.g., such as shown in Figure 2.11). Each module consists of a single linear unit, plus a simple gating network which decides which output to use, depending on whether the input is less than or greater than 0. Such a network should find it easier to learn the task because each subcomponent is linear and there are no hidden units.

Jacobs, Jordan and Barto (1991) have trained a network with such an architecture to do the "what/where" task. In this task, the network has to determine what an object is, and where it is in the visual field. They found that this modular architecture facilitated learning, compared with nonmodular networks. Furthermore, if one of the expert networks is composed of units with linear activation functions, that module always learns to carry out the "where" task.

We like this result, because it shows how it is possible for task assignment to be innately determined, not on the basis of the task

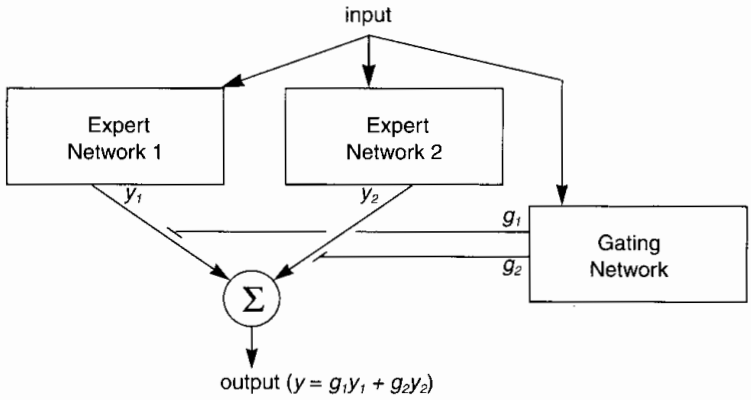


**FIGURE 2.11** Modular network proposed by Jacobs, Jordan, and Barto (1991). Input is sent to both expert networks, each of which is specialized for one aspect of the task, and also to a gating network. Both expert networks generate output; the expert network decides the appropriate mixture, given the specific input.

*per se*, but rather the match between a task's requirements and the computational properties of network architectures. It is not necessary that a piece of network be committed to a task in an explicit and hard-wired fashion, with one set of "where" units and another set of "what" units. Instead, the intrinsic capabilities of the module simply select those tasks for which it happens to be suited. Thus, the way in which the task/architecture mapping is specified may be through biasing and not rigid assignment. The important lesson is that some problems have natural good solutions; they have computational requirements which impose their own constraints on how the problem can be solved. ***Nature does not always need to provide the solution; it often suffices to make available the appropriate tools which can then be recruited to solve the problems as they arise.***

## Where does the teacher come from? Supervised vs. unsupervised learning

Learning algorithms such as backprop depend crucially on a prior notion of what good performance is. The training environment consists of input/output patterns in which the output is a target or teacher for the network. The error is defined as the discrepancy between the network's output and the target output (supplied by the teacher pattern) which goes with the input. Is this assumption of a teacher pattern psychologically reasonable?

In some cases, it is. For example, let us say we carry out an experiment in which a subject is given some input, performs some action, and then gets feedback. This scenario clearly resembles the training regime which a network undergoes. The feedback is exactly equivalent to the teacher for the network.

But it is not necessary to be quite so literal-minded. There is no reason not to take a somewhat more abstract view of what a network model is capturing. For instance, consider the case of learning the various forms associated with different tenses, person, number, and mood of verbs. There is no obvious teacher which is provided to children (i.e., their learning experience typically consists simply of hearing correctly inflected forms). On the other hand, suppose we conceptualize the learner's task as one of *binding forms*. The child has to learn to associate a set of morphologically related forms with each other. Failure to learn the correct associations would lead the child to anticipate forms which are not confirmed by the actual input; these failures would then constitute a kind of indirect teacher. The teacher signal in this scenario is internally generated. In fact, Nolfi, Parisi, and colleagues (Nolfi & Parisi, 1993, 1994, 1995) have used an evolutionary approach in order to develop networks which provide their own internal teacher in just such a manner.

Nonetheless, forms of training which require a teacher of any sort—called "supervised learning"—clearly have a fairly restricted domain within which they can be plausibly applied. In many circumstances it is not reasonable to suppose that the kind of detailed feedback which is required is available, from any source. One important goal of connectionist modeling has been to find ways to

overcome this limitation, while hopefully retaining some of the attractive aspects of gradient descent learning.

There are several ways in which the supervised learning paradigm may be altered to make it more realistic. One form of training involves what is called "auto-association." In this task, a network is given an input and is trained to reproduce the same input pattern on the output layer. What makes this a non-trivial problem is that such networks (such as the one shown in Figure 2.12) contain a narrow "waist" in the middle. This means that the network is forced to

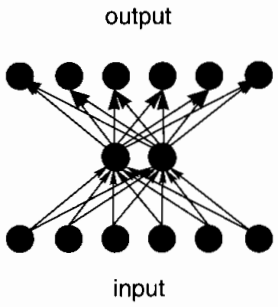


**FIGURE 2.12** An autoassociator network. The network is trained to reproduce the input pattern on the output layer. The lower-dimensionality of the hidden units requires that the network find a more efficient encoding of the input patterns, and can be used for feature discovery.

find a lower-dimensional representation of the inputs. Often these internal representations capture interesting features of the inputs. For example, Elman and Zipser (1988) trained autoassociators to reproduce speech sounds, and found that hidden units learned to respond to different classes of sounds (e.g., vowels, consonants, and certain stops). This form of training also addresses the question of where the teacher comes from, since in autoassociation, the teacher is nothing more than the input itself. Minimally, all that is required is a short-term memory.

Another task which is similar in spirit to autoassociation is the task of predicting the future. The network shown in Figure 2.13 is what has been called a simple recurrent network (Elman, 1990), or SRN. (The network is simple in the sense that the error derivatives

are propagated only one time step back into the past; this does not prevent the SRN from storing information in the distant past, but learning longer distance temporal dependencies may be difficult.). An SRN contains recurrent connections from the hidden units to a layer of context units. These context units store the hidden unit activations for one time step, and then feed them back to the hidden units on the next time step. The hidden units thus have some record of their prior activation, which means that they are able to carry out tasks which extend over time. (Note that the hidden units may continue to recycle information over multiple time steps, and also will find abstract representations of the time. So this sort of network is not merely a tape-recording of the past.)

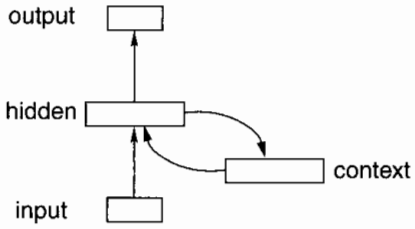


**FIGURE 2.13** Simple recurrent network (SRN). Layers of nodes are shown as rectangles.

Insofar as there are interesting sequential dependencies in the training data, the SRN is often capable of discovering them through learning to predict. The task of prediction has a certain ecological validity, since there is evidence that anticipation plays a role in early learning in many domains. There are also biological mechanisms which are plausibly implicated in mediating learning through prediction (Cole & Robbins, 1992; Morrison & Magistretti, 1983). As is true for autoassociation, the prediction task requires no special teacher, since the target output is simply the next input. All that is required to be psychologically plausible is to assume that this processing can lag a few steps behind the actual input. These forms of learning might be called "self-supervised learning."

Finally, there exists a form of supervised training called "reinforcement learning" (Barto & Andanan, 1985; Barto, Sutton, &

Anderson, 1983; Sutton, 1984) in which the teacher takes a simpler form. Rather than being instructed on exactly which aspect of an output was right or wrong (remember that in backprop, each output unit gets an error signal), the network is simply given a scalar value reflecting its overall performance—a little like being told you're getting warmer (closer to the solution) or colder (further from the solution). The problem, of course, is figuring which part of the output is contributing to the error, and so it is not surprising that reinforcement learning is much slower than fully supervised learning. But it is certainly easier to believe that this sort of situation—in which we are only told how well we did, and not why—is a psychologically plausible one.

In addition to these weakened forms of supervised learning, there are training regimes which involve "unsupervised learning." Hebbian learning is a paradigm example of this. Here, the network weights are changed according to the correlated activity between nodes which are receiving input from the environment. None of that input need be instructive in any direct sense; the network may be thought of as more or less passively experiencing the environment and striving to discover correlations in the input. Many other forms of unsupervised learning have been proposed, including competitive learning (Grossberg, 1976; Rumelhart & Zipser, 1986), feature mapping and vector quantization (Kohonen, 1982; Nasrabadi & Feng, 1988), and adaptive resonance theory (Carpenter & Grossberg, 1987, 1988). These approaches are particularly relevant when the goal is to uncover latent features or categories in a set of stimuli.

Note, however, that although these training regimes may reasonably be called unsupervised, it is not exactly the case that these approaches are entirely theory-neutral or non-parametric. Each learning regime attempts to optimize some quantity, whether it be correlations, or mutual information, or harmony, etc. So there is still supervision; it's just folded into the learning rule itself rather than into the environment being learned. We point this out only as a reminder that no learning rule can be entirely devoid of theoretical content nor can the *tabula* ever be completely *rasa*.

Another promising approach which shares the goal of unsupervised learning has been proposed by Becker and Hinton (1992) In their multiple maps scheme, Becker and Hinton replace the external teacher with internally generated teaching signals. As they put it, "these signals are generated by using the assumption that different parts of the perceptual input have common causes in the external world" (Becker & Hinton, 1992; p. 372). Thus, *the principle of coherence functions as the teacher*. Practically, what this involves is having separate parts of the network receive different (but related) parts of the perceptual input. Each module then learns to produce outputs which provides maximum information about the way in which the other will respond to various inputs. Becker and Hinton have shown that such a scheme can be applied to the problem of extracting depth from random dot stereograms. In principle, one can imagine that the approach could be used to discover correspondences across modalities as well. This could be very useful in modeling infants' capacity for cross-modal imitation, for example. Other related approaches (related in terms of goals, while using different algorithms) have been proposed by Kohonen (1982) and Zemel and Hinton (1993).

A final note on Hebbian learning: Earlier, we talked about the computational limitations in what can be learned solely on the basis of correlated activity. Despite these limitations, we emphasize that Hebbian learning plays an extremely important role in many models, particularly those which are concerned with biologically plausibility. There are similarities between Hebbian learning and long-term potentiation (LTP; for example, in hippocampal neurons), according to some authors (McNaughton & Nadel, 1990; Rolls, 1989). Furthermore, despite the limitations, a great deal can be learned through correlations. For example, as we shall see, Hebbian learning provides a plausible model of how the visual cortex might self-organize to produce visual ocular dominance columns (Miller, Keller, & Stryker, 1989; see discussion below), orientation selective cells (Linsker, 1986, 1990), and to detect dilation and rotation (Sereno & Sereno, 1991). Hebbian learning also plays an important role in the models of object detection (O'Reilly & Johnson, 1994; O'Reilly & McClelland, 1992) and of synaptic pruning (Kerszberg, Dehaene,

& Changeux, 1992; Shrager & Johnson, in press) which we discuss in Chapter 7. (See Fentress & Kline, in press, for a collection of recent work involving Hebbian learning.)

## *Finding first principles*

Connectionism comes in many flavors. In the past decade, there has been a stunning explosion of architectures, algorithms, and applications. Models have been developed of everything from inter-cellular interactions to lobster stomach muscles, expert systems, acquisition of grammatical gender in German, and plasma flow in nuclear reactions. As with any field, there are camps and factions which have very different goals. Given this diversity, is there anything shared in common? Are there any basic principles which underlie the connectionist approach, and which these different models have in common?

We think so. The differences are of course important, sometimes cut deep, and in fact represent a positive state of affairs. Such ferment and diversity are critical to the health of the field (one of our own goals in this book is to urge new directions for connectionists). But when all is said and done, we nonetheless note that there are recurring characteristics which appear in connectionist models and which are, sometimes tacitly, sometimes explicitly, valued by modelers.

We would like to make a stab at suggesting what some of these "first principles" might be. We do so not with the goal of establishing a catechism for determining who is a card-carrying connectionist, but simply because it is important to understand why the models work. To what extent does their behavior reflect superficial differences with other computational frameworks, and to what extent does it flow from underlying properties? Unless we have some notion of what these underlying properties are, such questions cannot be satisfactorily answered. There is another reason for trying to understand what makes these models tick. We have found the most useful aspect of connectionism to be the concepts it makes

available. Thinking like a connectionist need not require doing simulations, in our view. What is more important is being able to use the conceptual toolbox.

With this perspective, we would like now to focus here on four aspects of connectionist models which seem to us to be particularly relevant to developmental issues: *the problem of control, the nature of representation, nonlinear responses,* and *the time-dependent nature of learning.*

## Who's in charge? Eliminating the homunculus

One of the banes of cognitive science is the *homunculus*—the idea that our mental life is controlled by an inner being who observes the world through the retinal "screen," listens to sounds through the cochlear "earphones," and guides our actions by throwing switches that innervate our muscles. (If one were a preformationist, one would expect to find this fellow in miniature in the fertilized egg.)

The logical problem of such a scenario, with its potential for infinite regress, is self-evident. But although the homunculus is today not a popular fellow among most self-respecting cognitive scientists, he may often be found lurking in disguise in many theories. The majority of theories of planning and control, for example, presuppose some controlling entity which basically does what a homunculus does. It is easy to see why such a fellow would be useful. Without him, there's a real paradox: If something is not in control of behavior, then why is behavior not uncontrolled?

Connectionist models implement an appealing solution to this problem: Global effects arise from local interactions. Coordinated activity over the entire system thus occurs as an emergent property, rather than through the efforts of a central agency. This is not a new idea, certainly. The Belousov-Zhabotinsky reaction, in which entirely local oxidation reactions give rise to formation of waves over macroscopic distances, is a classic example of emergent behavior (see Chapter 3). Alan Turing (1952) was one of the first to show how such reaction/diffusion (RD) processes might be involved in cell patterning. More recently, RD models have been proposed for a number of developmental phenomena, such as the formation of the

tiger's stripes (or a cheetah's spots, depending on initial conditions), or the development of visual ocular dominance columns (see below). ***Connectionist models are attractive because they provide a computational framework for exploring the conditions under which such emergent properties occur.***

Computation is local in (most) connectionist models in two senses. First, nodes often have restricted patterns of connectivity; they connect only to some of the other nodes in the network. In this regard, they resemble neurons, which usually have a local area within which is there is dense interconnectivity and much sparser connections to distant areas. The activity which emerges over the entire assembly of units therefore reflects a complex process in which many local interactions, most involving relatively simple computations, yield a global state which is not the result of any single unit.

Second, many learning algorithms use only local information when changing system parameters (such as weights on connections). In the case of Hebbian learning, connections between units are modified in a way which reflects the units' joint activity. In the case of backpropagation learning, each output unit has a local sense of error and the weights into that unit are changed so as to reduce that single unit's error. (There are, to be sure, learning rules which require knowledge of global error, but such rules are often criticized for precisely this reason.) Let us give two examples of how local computation and local learning may produce globally organized behavior. The first example deals with the relatively low-level process by which visual cortex might become organized. The second example deals with the higher-level process by which the rules governing legal sequences of sounds in a language might be learned.

One of the notable characteristics of primary visual cortex in many species (such as cat, monkey, and human) is the presence of alternating patches of tissue which serve primarily one or the other eye. These ocular dominance columns are not present from birth, but in normal circumstances develop inevitably during the early part of life. Figure 2.14 shows a surface view of such tissue in the normal cat.

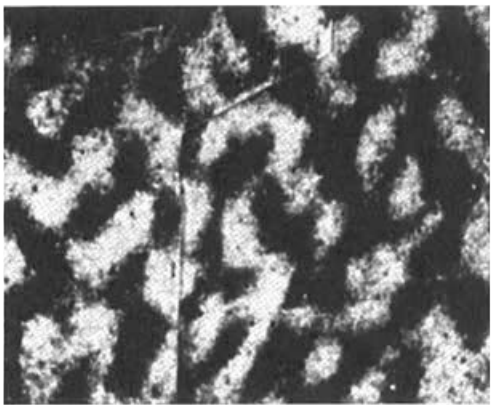

**FIGURE 2.14** Ocular dominance patches in layer IV of the cat visual cortex. The image is produced from serial autoradiograph following injection of [$^3$H]-amino acid into one eye. From LeVay, Stryker, & Shatz, 1978.

Since the majority of this tissue initially receives input from both eyes, but the eventual preferential response is clearly affected by experience (deprivation of input from one eye may disrupt the process), an important question is how might this organization arise. Miller, Keller, and Stryker (1989) and Miller, Stryker, and Keller (1988) have shown by simulation one plausible explanation. In their model, layer 4 "cortical" cells receive excitatory input from both "retinal" sources (via the lateral geniculate nucleus) in such a way as to preserve retinotopic organization; synapses from the two eyes are nearly equal in strength. Intracortical connections also exist and are modifiable by a Hebb-like learning rule.

Initially, the randomly assigned connection strengths result in a nearly uniformly innervation from both eyes, as has been observed in new-born kittens. This is shown in the top left panel of Figure 2.15, marked T=0. As time progresses, there is a progressive differentiation of the response, leading to patches which resemble those found in the older kitten. These columns emerge as the result of naturally occurring differences in the degree of correlation between neighboring cells in each eye versus across eyes, along with competitive interactions among layer 4 cells. From their analy-

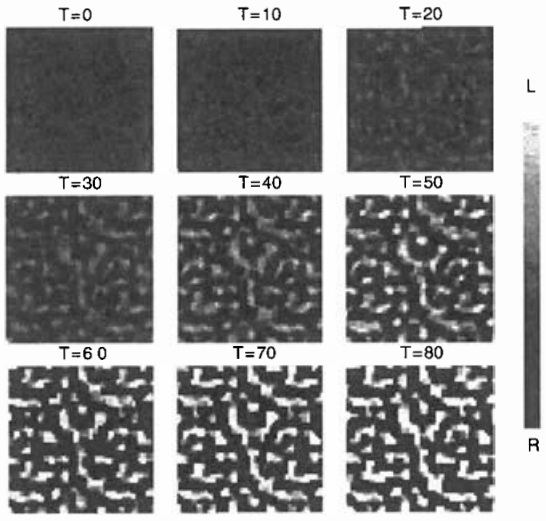


**FIGURE 2.15** Development of ocular dominance columns in model by Miller, Keller, & Stryker (1989). Degree of innervation from each eye is shown by grayscale (right eye=white; left eye=black) at various time steps during learning, from T=0 to T=80.

sis of the model, Miller and his colleagues are further able to account for the precise form of the various pathologies which occur as a result of different types of abnormal experience.

A second example illustrates emergent behavior at a much higher level of cognitive phenomenon. The TRACE model (McClelland & Elman, 1986) was developed to account for a set of experimental findings in the area of speech perception. TRACE used an interactive activation architecture similar to that of the word perception model discussed earlier (Figure 2.3). Different layers of nodes carried out processing at the level of acoustic/phonetic feature extraction, phoneme recognition, and word recognition. Many of the effects the model attempted to account for involved the important role of context in speech processing, particularly as a solution to the high degree of variability observed in the signal.

An interesting by-product of the architecture was observed, however. The word-to-phoneme connections resembled those

shown for the word-to-letter connections in Figure 2.3. This meant that the lexicon provided a very strong top-down influence on perception; that influence usefully compensated for degraded or missing input. But it also had another effect. The model could be given input corresponding to no known word. In that case, the correct set of phonemes would be activated, but of course no single word node would become active, although many partially similar words might achieve some activation. The pattern of phoneme activation was not unaffected by this activity at the word level, however, and noticeably different responses were obtained in the case of different sorts of non-word input. The sequence "bliffle," for example, was processed much better than the sequence "dliffle," in the sense that in the first case all phonemes were clearly activated; but in the second case the initial "d" and "l" phonemes were barely active.

In fact, this result is in close accord with the strong intuition English speakers have that the first sequence is better (i.e., more English-like) than the second. Linguists describe this tacit knowledge which speakers have about acceptable sound sequences with what are called phonotactic rules. English, for instance, is assumed to have a rule which marks word-initial sequences such as "dl-", "tl-", "bw-", "pw-" (among others) as ungrammatical. (Note that these rules must be language-specific and have nothing to do with articulatory difficulty, since many other languages happily tolerate sequences which are illegal in English.)

It is easy to see where the preference for grammatical sequences comes from. The non-word input activates many words which resemble it; the more word-like the input, the more word nodes first become active and then contribute top-down excitation to the phoneme level. In the case of very deviant input, few word nodes are activated and the phoneme activations depend solely on bottom-up input. (Subjectively, this is not unlike the experience of trying to identify the sounds in an unfamiliar language.) Thus what looks like rule-guided phonotactic knowledge arises simply as a result of the statistics which are present in the lexicon.[3] This is another example of emergent behavior in a connectionist network.

## Connectionist representations

Early connectionist models, and also many current ones, adopt what is called a "localist" form of representation (more recently, these have been called "structured representations"). The word recognition model and the TRACE model are examples of models which use this type of representation.

Localist representations are similar in some ways to traditional symbolic representations. Each concept is represented by a single node, which is atomic (that is, it cannot be decomposed into smaller representations). The node's semantics are assigned by the modeler and are reflected in the way the node is connected to other nodes. We can think of such nodes as hypothesis detectors. Each node's activation strength can be taken as an indicator of the strength of the concept being represented.

The advantage of localist representations are that they provide a straightforward mechanism for capturing the possibility that a system may be able to simultaneously entertain multiple propositions, each with different strength, and that the process of resolving uncertainty may be thought of as a constraint satisfaction problem in which many different pieces of information interact. Localist representations are also useful when the modeler has *a priori* knowledge about a system and wishes to design the model to reflect that knowledge in a straightforward way. Finally, the one-node/one-concept principle makes it relatively easy to analyze the behavior of models which employ localist representations.

Localist representations also have drawbacks, and these have led many modelers to explore an alternative called "distributed representations." In a distributed representation, a common pattern of units is involved in representing many different concepts. Which concept is currently active depends on the global pattern of activity across the ensemble of units.

---

3. If this account is correct, it predicts that technically ungrammatical non-word sequences might still be perceived better than other grammatical non-word sequences, just in case the ungrammatical non-words happened to almost resemble many real words. This prediction was subsequently verified experimentally with human listeners (McClelland & Elman, 1986).

Figure 2.16 gives an example of distributed representations. The same group of units is shown in Figure 2.16a and Figure 2.16b, but the units have different activations. Concepts are associated, not with individual units, but instead with the global pattern of activations across the entire ensemble. Thus, in order to know which concept is being considered at a point in time, one needs to look across the entire pattern of activation (since any single unit might have the same activation value when it participates in different patterns). The information needed to decide which concept is being represented is distributed across multiple units.


**FIGURE 2.16** Examples of distributed representations. Both (a) and (b) illustrate different patterns of activation of the same set of four units. Activation values for individual units are shown as numbers above each unit. Note that the second unit has the same activation value in both representations; in order to know which concept is represented, one has to look at the entire set of nodes.

This style of representation seems to be more consistent with the brain stores information than localist representations (e.g., Lashley, 1950). A note of caution, however: Most modelers who study higher-level cognitive processes tend to view the nodes in their models as equivalent not to single neurons but to larger populations of cells. The nodes in these models are *functional* units rather than *anatomical* units. So it is not clear how heavily to weigh what seems to be the somewhat greater biological plausibility of distributed representations.

The real advantages have to do with representational capacity. Localist representations impose a rigid framework on a model's conceptual contents. Although graded activations in a localist model allow one to capture probabilistic or partial knowledge, there is still a fixed and discrete inventory of concepts. Distributed repre-

sentations tend to be richer and more flexible. This can be seen by spatial comparison of the two types of representation.

We pointed out earlier that patterns of activation can be thought of as vectors, in which the activation of each unit is the value of an element in the vector, and the entire vector represents the values of all the units in the pattern. As vectors, these activation patterns can also be represented geometrically. Each unit corresponds to a dimension (in "activation space"), and the unit's activation indicates where along that dimension the vector is located. Thus if we had activation patterns involving three units, we could represent the patterns in a three-dimensional space, using the $x$ dimension to represent the values of the first hidden unit, the $y$ dimension to represent the second, and the $z$ dimension to represent the third.

If we use these three units in a localist fashion, then we are permitted exactly three distinct vectors, corresponding to the activation patterns $100, 010, 001$. The spatial representation of these vectors is shown in Figure 2.17a.

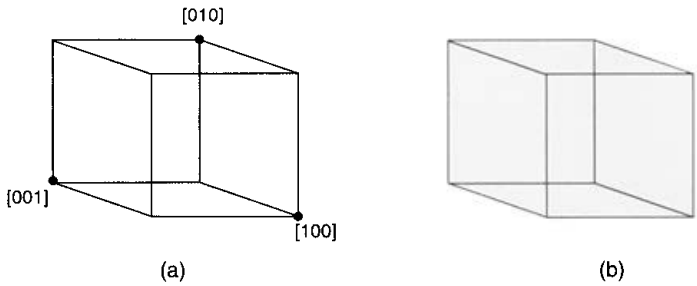


**FIGURE 2.17** Localist representations in (a) pick out just three distinct vectors. Distributed representations in (b) fill the entire space occupied by the cube.

On the other hand, if we use a distributed representation, then our activation patterns may involve any combination of values on all the units. The activation pattern $0.5, 0.5, 0.5$ picks out a point in the center of the cube. This means we have at our disposal the *entire volume of space* represented in the cube shown in Figure 2.17b. Prac-

tically speaking, of course, there are limitations on just how precisely one might be able to distinguish vectors which are very close, but it is still clear that given the same number of units, the conceptual space is much larger for distributed representations than for localist representations.

There are several other outcomes from having available the entire activation space for representation, and these interact with another important observation. When we spoke earlier of the XOR problem, we said that networks operate on the principle that "similar inputs yield similar outputs." We defined similarity in terms of the spatial proximity of activation patterns and said that activation patterns—and the concepts they represent—are similar to the degree they are close in activation space. Vectors which are close, by Euclidean distance, are similar. Patterns which are distant in space are dissimilar.

Consider the implications of this for localist representations. The vectors in Figure 2.17a are all equidistant (they are also orthogonal to one another). There may or may not be relevant similarity relationships between the concepts they represent, but there is no way to capture this in concepts' representations. On the other hand, because the entire activation space is available with the distributed representations in Figure 2.17b, one can envision a range of possibilities. Things which are close in conceptual space can be represented by activation patterns which are close in activation space; the degree of dissimilarity can be measured by their distance. The space may even be organized hierarchically. Let us give a concrete example of this, from Elman (1990).

In this task, a simple recurrent network was trained to predict successive words in sentences. Words were input one at a time, and the network's output was the prediction of the next word. After the network made its prediction, backprop learning was used to adjust the weights and then the next word was input. At the end of each sentence the first word from the next sentence was presented. This process continued over many thousands of sentences.

The words themselves were represented in a localist fashion. That is, a word such as "cat" appeared as a 31-bit vector with one bit on (1) and the remaining bits set to 0. Because of the localist rep-

resentations, words were equidistant in the activation space and there was therefore no similarity between them. This scheme was adopted quite deliberately, because it meant that the network was deprived of any clues regarding the grammatical category or meaning of the words which it might use in making predictions. Instead, the network had to rely entirely on the distributional statistics of the stimuli to carry out the task. (In much the same way, the acoustic form of a morpheme bears no intrinsic relationship to its meaning and one might imagine this as similar to the situation of the very young language learner, who has no prior knowledge of which meanings are associated with which words).

The network eventually learned to carry out the task, although not precisely as trained. For example, given the sequence of vectors corresponding to the words "the girl ate the...," the network activated *all* the output units which corresponded to the various words representing edible things, rather than predicting exactly the specific word which followed.

One might infer from this result that the network somehow developed the notion "edible." This category, and others such as noun, verb, etc., were not represented in either the input or output representations, so the only place remaining to account for the behavior would be in the hidden unit activations. The hidden units define a very high dimensional space (there were 150 hidden units, which map a 150-dimensional hypercube), and one might expect that the network would learn to represent words which "behave" in similar ways (i.e., have similar distributional properties) with vectors which are close in this internal representation space. Ideally, one would like to be able to visualize this space directly. But since the space is so highly dimensional, indirect techniques must be used. One of these involves forming a hierarchical clustering tree of the words' hidden unit activation patterns.

This is a fairly simple process. It involves first capturing the hidden unit activation pattern corresponding to each word, and then measuring the distance between each pattern and every other pattern. These inter-pattern distances are nothing more than the Euclidean distances between vectors in activation space we spoke of earlier, and we can use them to form a hierarchical clustering

tree, placing similar patterns close and low on the tree, and more distant groups on different branches. Figure 2.18 shows the tree corresponding to hierarchical clustering of the hidden unit activations obtained from the sentence prediction task.

We see that the network has learned that some inputs have very different distributional characteristics than others, and forms hidden unit representations which places these two groups in different areas of activation space. These groups correspond to what we would call nouns and verbs. In addition to grammatical differences, the network uses the spatial organization to capture semantic differences (e.g., humans vs. animals). The organization is hierarchical as well. "Dragon" occurs as a pattern in activation space which is in the region corresponding to the category animals, and also in the larger region shared by animates, and finally in the area reserved for nouns.

This spatial framework allows some categories to be distinct and disjoint, but also makes it possible to have representations which lie between category boundaries. An example of this occurs in a phenomenon called sound symbolism. This refers to the common occurrence in which sequences of sounds have loose associations with meanings. In English, for instance, words which contain a final "-rl" ("curl," "unfurl," "burl," "whirl," "twirl," etc.) often evoke the image of circularity. The association is psychologically real in the sense that given a nonce word such as "flurl," speakers will generate definitions involving some circular aspect. These associations seem to lie somewhere between the level of sound (phonemes which by definition carry no meaning) and systematic meaning (morphemes). This is the sort of phenomenon which might be exploited by distributed representations.

Another aspect of distributed representations which makes them useful is their tendency to be context-sensitive. In this way they differ fundamentally from traditional symbolic representations, which are abstract and context-insensitive.

Context-sensitivity may be encoded by the precise location of a representation in activation space. Consider again the sentence prediction task described above. If one compares the position of the vector for a word across the many instances it occurs, one notices

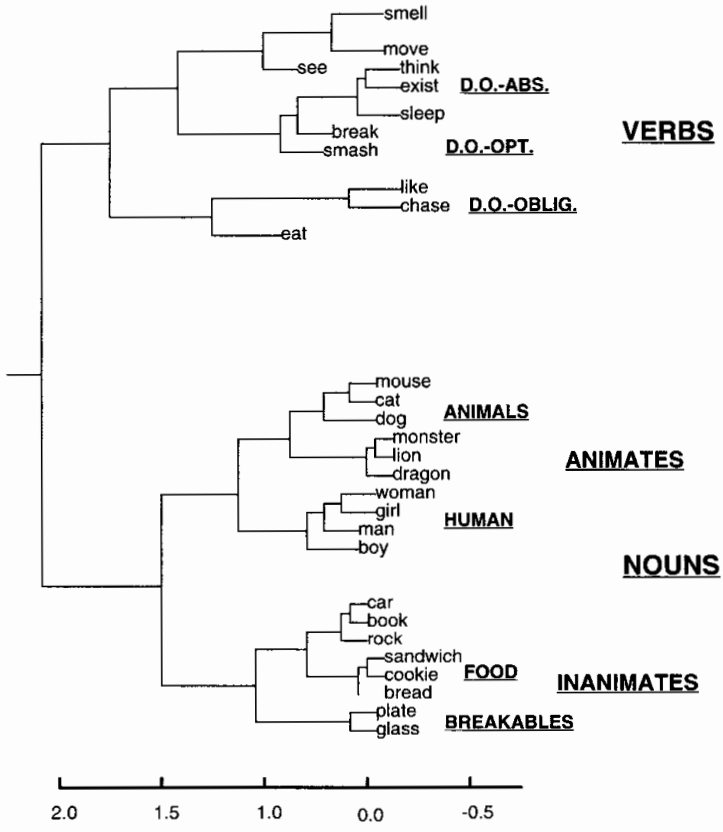


**FIGURE 2.18** Hierarchical clustering of hidden unit activation patterns from the sentence-prediction task (Elman, 1990). The network learns distributed representations for each word which reflects its similarity to other words. Words and groups of words which are similar are close in activation space, and close in position in the tree.

that the location varies. Thus there will be many different vectors corresponding (for example) to the word "dragon"; the tree shown in Figure 2.18 was actually formed using vectors which averaged across context. (The reason why the hidden unit representations varies is because the internal representations are created by activation both from the word input itself, which is invariant, and the prior internal state encoded in the context units, which is variable.)

This proliferation of multiple *tokens* for different *types* might seem to be problematic. Given a specific activation pattern, how do we know which type it belongs to? As it turns out, the answer is easy, and it's the same one used to distinguish categories such as nouns from verbs: Tokens of the same type are all spatially proximal, and closer to each other than to tokens of any other type. The fact that they inhabit a bounded region of space is what tells us they are all the same abstract word.

There's a bonus as well. The spatial distribution of tokens turns out to be non-random. Instances of "dragon" when it appears as subject in a sentence are located in a different region of the "dragon" space than "dragon"-as-object. The relative positions of these two subclasses turns out to be identical across nouns! This means that the tokenization process is actually encoding grammatically important information, in a systematic way. ***The internal representation of a word thus reflects not only its core identity, but provides a grammatically interpretable context-sensitive shading.***

## Connectionist processing: The importance of nonlinearity

Earlier we remarked on the importance of the nonlinear activation function in neural networks. We return to this issue because we view it as one of the properties which gives connectionist models great computational power.

Essentially, the fact that nodes have nonlinear responses to their inputs means that there are some conditions under which they respond in a graded and continuous manner, and other conditions where their response is abrupt, discrete, and all-or-nothing. It simply depends which region of the node's activation function is being used.

The importance of this characteristic to development should be apparent. It has long been observed that there are periods during development where behavior changes slowly and progress is incremental. Such periods may be succeeded by spurts of activity in which change is dramatic and rapid. This phenomenon is also sometimes linked to the notion of "readiness," because during the

time of rapid change it appears that the developing organism has a heightened sensitivity to inputs which previously elicited no apparent response. The organism is said to be ready to change in a way it was not before.

A reasonable interpretation of this developmental phenomenon might be that the organism has undergone a drastic internal reorganization, or that maturational factors have changed the organism in some fundamental way. Thus, dramatic—or nonlinear—changes in behavior are seen as diagnostic of dramatic, nonlinear changes in internal structure.

This interpretation is not the only one possible, and in Chapter 3 we discuss several connectionist simulations which exploit the nonlinearities in nodes' activation functions; in Chapter 4 we go into this matter of the shape of change in greater detail. The lesson will be that very small changes in internal structure may produce very big changes in observed behavior. Put another way, the same mechanism may produce discontinuous behavior over time without requiring drastic internal reorganization.

We have also briefly discussed another consequence of the nonlinearity, which we repeat here and which will figure in our account of the phenomenon we call "the importance of starting small," discussed in Chapter 6. Recall from Equation 2.6 that one of the terms which is used in computing weight changes during learning is the derivative of the activation function. This term, which measures the slope of the activation curve, modulates how much of the error signal is actually used to change the weights. Since the activation function is steepest in its midrange (see Figure 2.2), the effect of an error is greatest at this point. Because network weights are usually initialized with small random values with mean of 0.0, this means that when networks start off life they typically have net inputs close to 0.0, which is exactly the value that generates midrange outputs. The result, as we explained earlier, is that networks are more sensitive to error during early training. As training progresses there is a tendency for node activations to be pushed to their extremes, and this has the effect of slowing down learning. Whether this is good or bad, of course, depends on whether learning is successful! In any event, it provides a natural mechanism by which learning can be

self-terminated. This too is a phenomenon which has been observed in development.

## *Final words: What connectionism is, and is not*

Before we conclude this chapter, there are several issues outstanding which we wish to address. Several of these issues represent what we feel are misunderstandings about the nature of connectionist models. We will return to some of these points in later chapters, but feel it is useful to identify them clearly at the outset.

### Just how *rasa* is the *tabula*, anyway?

There is a widespread belief that connectionist models (and model*ers*) are committed to an extreme form of empiricism; and that any form of innate knowledge is to be avoided like the plague. Since a basic thesis of this book is that connectionist models provide a rich framework and a new way to think about ways that things can be innate, we obviously do not subscribe to this point of view.

To be sure, it is not difficult to understand where the belief comes from. One of the exciting lessons from the connectionist work is that relatively simple learning algorithms are capable of learning complex things. Some of the results have demonstrated very dramatically, as pointed out before, that considerably more structure is latent in the environment than one might have guessed. In the past, a common argument in favor of innate knowledge has been the claim that the input to a learner is too impoverished—or the learning algorithms too weak—for the appropriate generalizations to be induced (e.g., Gold, 1967). So one very useful function which has been served by the demonstration that learning may be possible in cases where it was thought not to be is to encourage a bit more caution in resorting to claims of poverty of the stimulus.

It is also true that our species' ability (and need) to learn seems to be unparalleled in the animal kingdom. As Gould (1977) and others have pointed out, one of the most striking characteristics of

human evolution is the dramatic increase in time spent during development. This prolonged period of dependence might be considered to be maladaptive, except that it increases the opportunity for learning and socialization. The connectionist emphasis on learning is thus highly relevant to understanding a major characteristic of our species.

Of course, the fact that something *can* be learned is not a sufficient demonstration that it *is*. The learning that takes place in a network simulation might in biological organisms occur through evolutionary mechanisms. And to the proof of the universal approximator capabilities of networks (Hornik, Stinchcombe, & White, 1989) must be counterposed another result, that when architecture is unconstrained, many classes of problems are NP-complete (i.e., for practical purposes, too hard to learn; Judd, 1990). So there are good reasons to believe that some kinds of prior constraints are necessary.

In fact, all connectionist models necessarily make some assumptions which must be regarded as constituting innate constraints. The way in which stimuli are represented, the parameters associated with learning, the architecture of the network, and the very task to be learned all provide the network with an entry point to learning. But there are other, even more interesting ways in which connectionist models allow us to take advantage of prior constraints and to understand how maturational factors may interact with learning. The net effect is to make possible highly complex interactions with the environment. To be innate need not mean to be inflexible or nonadaptive. A major goal of this book will be to explore and exploit this perspective.

## Modularity

Modularity tends to travel with innateness on the Big Issues circuit. In fact, these issues are logically separable and it is unfortunate they are so often confounded with one another. In much the same way that connectionist models have been thought to deny any role for innateness, they are often thought to be anti-modular.

It is true that many models have worked from an assumption of minimal structure. This is not an unreasonable initial working hypothesis. Furthermore, many of the early models (such as the word-recognition model) focussed on phenomena in which interaction between various knowledge sources played an important role. The sort of strict encapsulation envisioned by Fodor (1983) seemed undesirable.

But nothing intrinsic to the connectionist frame precludes modularity, and we have already made the point that some degree of organization and modular structure appears necessary if models are to be scaled up. Work by Jacobs, Nowlan, Jordan, and others shows that connectionists take the challenge of modularity very seriously.

The real questions seem to us to be, first, to what extent is the modular structure pre-existing as opposed to emergent; and, second, what are the functional contents of the modules?

Answers to these questions will vary, depending on the modules involved. The retina is a module whose structure is highly predetermined, and whose functional role is tightly coupled to a specific domain. Visual and auditory cortex, on the other hand, are modules which are partially pre-determined but in a highly indirect way. We know from results involving natural and induced pathologies (e.g., Hubel & Wiesel, 1963, 1965, 1970; Neville, 1991; Sur, Pallas, & Roe, 1990) that both the structure and content of these areas is highly dependent on appropriate input during development. To us the interesting question is not whether or not the brain is modular (it clearly is), but how and why it gets to be that way. There is a huge difference between *starting* modular and *becoming* modular. One of the important contributions of connectionist models has been in suggesting answers to these questions (e.g., Linsker, 1986, 1990; Miller, Keller, & Stryker, 1989; O'Reilly & Johnson, 1994). This will be an issue which will occupy much of our attention in the remainder of this book.

## Do connectionist models have rules?

It is sometimes claimed that connectionist models show that systems are capable of productive and systematic behavior in the absence of rules. We actually do not believe this is true—but we do have great sympathy for what often underlies the claim.

To say that a network does not have rules is factually incorrect, since networks are function approximators and functions are nothing if not rules. So arguments about whether or not networks have rules really do not make much sense.

Others have tried to distinguish between behavior which is *characterized* by rules, and behavior which is *governed* by rules. Presumably, in the first case, the behavior only accidentally conforms to a rule, whereas in the latter case the rule has causal effect. Clearly, the behavior of a network is causally connected with its topology and connection weights, so ultimately this also is not an interesting distinction.

What we take as a more interesting question is, *What do the network's rules look like?* Are they merely notational variants of the rules one sees in more traditional approaches such as production systems or linguistic analyses? Or do they make use of primitives (representations and operations) which have significantly different properties than traditional symbolic systems, and which might capture more accurately—and with more explanatory power—the behavior of learning in humans?

It is important here to distinguish between theoretical behavior, in the limit, and behavior in practice, operating in the real world with real-time constraints. In principle, connectionist networks and traditional symbolic systems may be interconvertible. This is one reading of the proof of networks as universal approximators. But systems which are instantiated in the real world with space and time constraints have different properties than their idealized counterparts. The idealized digital computer may be a Universal Turing Machine, but no such machine exists in the real world, and no real neural network is a universal function approximator. So in reality we are dealing with systems which may have very different properties when they are placed in a real world context. In practice, certain

sorts of generalizations and behaviors may be more readily captured in one system than in the other.

For instance, we know that it is possible to build a connectionist network which implements a LISP-style rule system (Touretzky & Hinton, 1985). But when it comes down to it, it is probably easier to write LISP programs in LISP. David Marr (1982) gives the example of computation with Arabic vs. Roman numerals. It is relatively easy to do numeric computation with Arabic numerals, but harder (especially multiplication) with the Roman system. In a similar manner, connectionist networks are not particularly well-suited to implementing truly recursive functions, nor to doing predicate calculus, nor to doing many basic mathematical operations. We are perplexed when people try to teach networks such things. They can be done, but at some cost and no gain.

On the other hand, connectionist networks do provide a natural formalism for carrying out many of the operations which are characteristic of biological systems. The criticism that connectionist models cannot implement strict recursion nor support syntactically composed representations (e.g., Fodor & Pylyshyn, 1988) are well-grounded, but the conclusion—that therefore connectionist networks are insufficient to capture the essence of human cognition—seems to us to reflect a profound misunderstanding about what human cognition is like. We believe that human cognition is characterized by interactive compositionality (or in van Gelder's terms, "functional compositionality," van Gelder, 1990) and that it requires exactly the kind of interactive and graded representations, and nonlinear operations which are the natural currency of connectionist models. So we believe that connectionist models do indeed implement rules. We just think those rules look very different than traditional symbolic ones.

## Is connectionist neo-Behaviorism?

One concern that has been expressed is that connectionism is simply behaviorism dressed up in modern clothing. There is some irony in this worry, since Donald Hebb, whose insights have been a

beacon for modern connectionism, was in profound disagreement with the behaviorist approach and saw connectionism (the term he used) as being distinctly anti-behaviorist.

Connectionism is apparently behaviorist insofar as connectionist models often involve inputs and outputs which play the role of Stimulus and Response. Behaviorists, however, eschewed attempts to speculate about the agency which mediated the S-R pairing. Behaviorism was both anti-physiological and anti-mentalist. Behaviorists were hostile to explanations which invoked unseen mechanisms and despaired of ever being able to relate mental function to brain function.

Connectionism, on the other hand, focuses precisely on the mechanisms which mediate behavior. Hidden units, for instance, play the role which behaviorists were unwilling to grant the brain. They allow models to form internal representations whose form and function may not be directly inferable from either input nor output. The resulting representations are abstract. Recurrent connections further enrich the system; they make it possible for the system itself to be both an input and an output of processing, and to generate activity which is not environmentally induced. Such endogenous activity is an essential component of thought.

## The importance of biology

We end with this issue because it is one which lies at the heart of the this book. The question is how seriously one should take biological constraints.

First, we wish to make clear that we think that the connectionist paradigm is interesting in its own right, and that there are valid reasons to study connectionist models regardless of whatever biological plausibility they might or might not have. There are many routes to intelligent behavior. We see no reason to focus exclusively on organically-based intelligence and neglect (for example) silicon-based intelligence. Artificial intelligence may help us better understand natural intelligence. But even if it doesn't, artificial systems are fascinating on their own terms.

Having said this, we want to make clear our own bias. We *do* believe that connectionist models resemble biological systems in important ways. We believe that connectionist models will be improved by taking seriously what is known about how computation is carried out in neural systems. We also believe that connectionist models can help clarify and bring insight into why neural systems work as they do.

Certainly there is a large gap between models and reality. For instance, there is no known evidence of any biological system which implements backpropagation learning. (Hebbian learning, on the other hand, seems much more plausible.) Sometimes this gap is unavoidable; sometimes it is even desirable. Ten years ago, for instance, there was no biological evidence for the existence of multiplicative synapses of the sort described in Feldman and Ballard (1982) and Rumelhart, Hinton, and McClelland (1986). Nonetheless, such higher-order units (often called "sigma-pi units" because they sum products of inputs) have been shown to be very useful in simplifying circuitry (e.g., Durbin & Rumelhart, 1989; Giles, Griffin, & Maxwell, 1988; Mel, 1990; Poggio & Girosi, 1990) and in principle there seemed to be no reason why synapses with these properties might not exist. And indeed, such synapses have recently been discovered in the brain. In retrospect, it would have been a mistake to have rejected out of hand models which used sigma-pi units; there is obviously a great deal which remains unknown about nervous systems and one would not want modeling to always remain several paces behind the current state of the science.

There is another reason for being willing to tolerate a gap between the model and the meat. It is often difficult to know what functional role is served by specific neural mechanisms. As David Marr has pointed out, "trying to understand perception by studying only neurons is like trying to understand bird flight by studying only feathers: It cannot be done" (Marr, 1982). Models permit a level of analysis in which the emergent properties of a complex system may be revealed. Exactly which specific details of implementation are significant and which are not cannot always be predicted in advance. One might believe that some neural systems do gradient descent learning in a way which is *functionally* similar to backprop-

agation, even if one does not believe that backpropagation is the exact mechanism.

At the same time, we take seriously the goal of trying to build models which are informed by biological research. Nature's solution may not be the only one (and as Stephen Jay Gould points out, if it were possible to rewind the evolutionary tape and start over, even Nature would be likely to find a different solution on every rerun), but it is certainly an interesting one. And it works! So on pragmatic grounds alone, there are compelling reasons to try to reverse engineer nature.

More than this, however, we are interested in understanding nature's solution. So it makes no sense to ignore nature's lessons. We are willing to tolerate a reasonable gap between our models and the reality, particularly since our own interests veer toward high-level phenomena whose neural substrates are less well understood. But we take as our goal the development of models which are informed by the biology and at least roughly consistent with it.

Our view of what this entails is perhaps somewhat broader than might be first apparent. For us, a biological perspective involves not only the narrower view which focuses on developmental neuroscience in the individual, but also the broader perspective which views the individual as embedded in an evolutionary matrix. Marr's warning about studying bird flight is appropriate here again. Trying to understand individual traits without regard for the way they interact in the whole individual is a doomed enterprise; and so is trying to understand whole individuals without regard for the way they interact in societies and evolve over time. We are very interested in ways that things can be innate, and we do not see how this can be understood unless one takes an evolutionary perspective.

Of course, this broadens our brief considerably and we run the risk of over-reaching what may be reasonably grasped. But we think the risk is worth taking, and recent work which attempts to bring together connectionist models, the study of artificial life, and the use of evolutionary mechanisms represents exactly the kind of broad biological perspective we have in mind. Let us now elaborate that view in more detail across the following chapters.