Chapter 2
Multilayer Perceptrons

This chapter is devoted to multilayer perceptrons—how they work, what people have said about them, and why people find them attractive. Because multilayer perceptrons are the only explicitly formulated competitors to symbol-manipulation, it is important to understand how they work, on their own terms. Readers who are already familiar with the operation of multilayer perceptrons might skip section 2.1 in this chapter (How Multilayer Perceptrons Work), but readers who are unfamiliar with how they operate are strongly encouraged to read this chapter in its entirety. For even though I ultimately argue that multilayer perceptrons do not offer an adequate basis for cognition, understanding their operation is an important step toward building alternative accounts of how cognition could be implemented in a neural substrate. So it is worth taking some time to understand them.

2.1 How Multilayer Perceptrons Work

A multilayer perceptron consists of a set of *input nodes,* one or more sets of *hidden nodes,* and a set of *output nodes,* as depicted in figure 2.1. These nodes are attached to each other through *weighted connections;* the weights of these connections are generally adjusted by some sort of *learning algorithm.*[1]

2.1.1 Nodes

Nodes are *units* that have activation values, which in turn are simply numbers like 1.0 or 0.5 (see below). *Input* and *output nodes* also have *meanings* or *labels* that are assigned by an external programmer. For example, in a well-known model presented by Rumelhart and McClelland (1986a), each input node (simplifying slightly) stands for a different sequence of three sounds—for example, one node represents the sound sequence /sli/, another /spi/, and so forth. In McClelland's (1989) model of children's abilities to solve balance-beam problems, particular nodes stand for (among other things) particular numbers of weights that could appear on a balance beam.

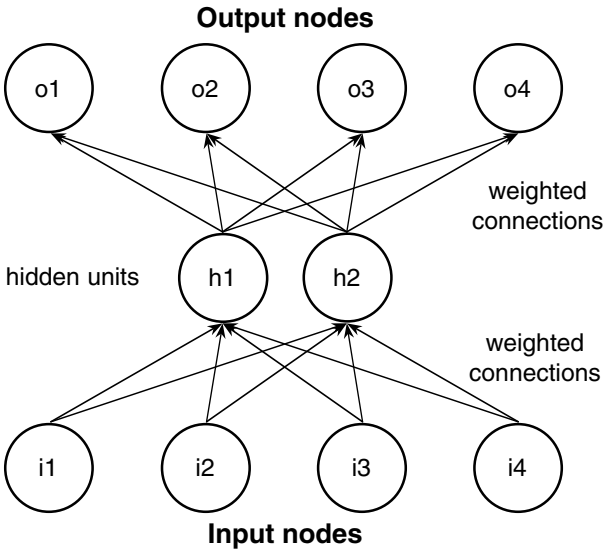

Figure 2.1
General multilayer-perceptron architecture: Input nodes, hidden nodes, and output nodes attached to each other by weighted connections.

The *meanings* of nodes (their labels) play no direct role in the computation: a network's computations depend only on the activation values of nodes and not on the labels of those nodes. But node labels do nonetheless play an important indirect role, because the nature of the input to the model depends on the labels and the output of a model depends on its input. For example, a model that encodes the word *cat* in terms of its component sounds, other things being equal, tends to treat *cat* as being similar to words that are similar in sound (such as *cab* and *chat*), whereas a model that encodes the word *cat* in terms of semantic features (+animate, +four-legged, and so on) tends to treat *cat* as being similar to words that are similar in meaning (such as *dog* and *lion*).

In addition to input nodes and output nodes, there are *hidden nodes* that represent neither the input nor the output; the purpose of these is discussed below.

2.1.2 Activation Values

The activation values of input nodes are given by the programmer. If the input to a given model is something that is furry, the programmer might "turn on" the node that stands for **furriness** (that is, set its activation to, say, 1.0), whereas if the input were something that isn't furry, the pro-

grammer would "turn off" the **furriness** node (that is, set its activation to, say, 0).

The activation values of the inputs are then multiplied by connection weights that specify how strongly any two nodes are connected to one another. In the simplest network, a single input node connects to a single output node. The activation value of the input node is multiplied by the weight of that connection to calculate the total input to the output node.

The activation value of an output node is calculated as some function of its total input. For example, an output node's activation value might simply be equal to the total activity that feeds it (a *linear activation rule*), or it might fire only if the total activity is greater than some threshold (a *binary threshold activation rule*). Models with hidden units use a more complicated *sigmoidal* activation rule, in which the activity produced by a given node ranges smoothly between 0 and 1. These possibilities are illustrated in figure 2.2.

In networks with more than one input node, the total input to a given node is calculated by taking the sum of activity fed to it by each node. For example, in a network with two input nodes (A and B) and one output node (C), the total input to the output node C would be found by adding together the input from A (calculated as the product of the activation of input node A times the weight of the connection between A and output node C) and the input from B (calculated as the product of the activation of node B times the weight of the connection between it and output node C). The total input to a given node is thus always a weighted sum of the activation values that feed it.

2.1.3 Localist and Distributed Representations

Some input (and output) representations are *localist*, and others are *distributed*. In localist representations, each input node corresponds to a specific word or concept. For example, in Elman's (1990, 1991, 1993) syntax model, each input unit corresponds to a particular word (such as *cat* or *dog*). Likewise, each output unit corresponds to a particular word. Other localist representational schemes include those in which a given node corresponds to a particular location in a retinalike visual array (Munakata, McClelland, Johnson & Siegler, 1997), a letter in a sequence (Cleeremans, Servan-Schrieber & McClelland, 1989; Elman, 1990), or a distance along a balance beam from the beam's fulcrum (Shultz, Mareschal & Schmidt, 1994).

In distributed representations, any particular input is encoded by means of a set of simultaneously activated nodes, each of which can participate in the encoding of more than one distinct input. For example, in a model of the inflection of the English past tense proposed by Hare,

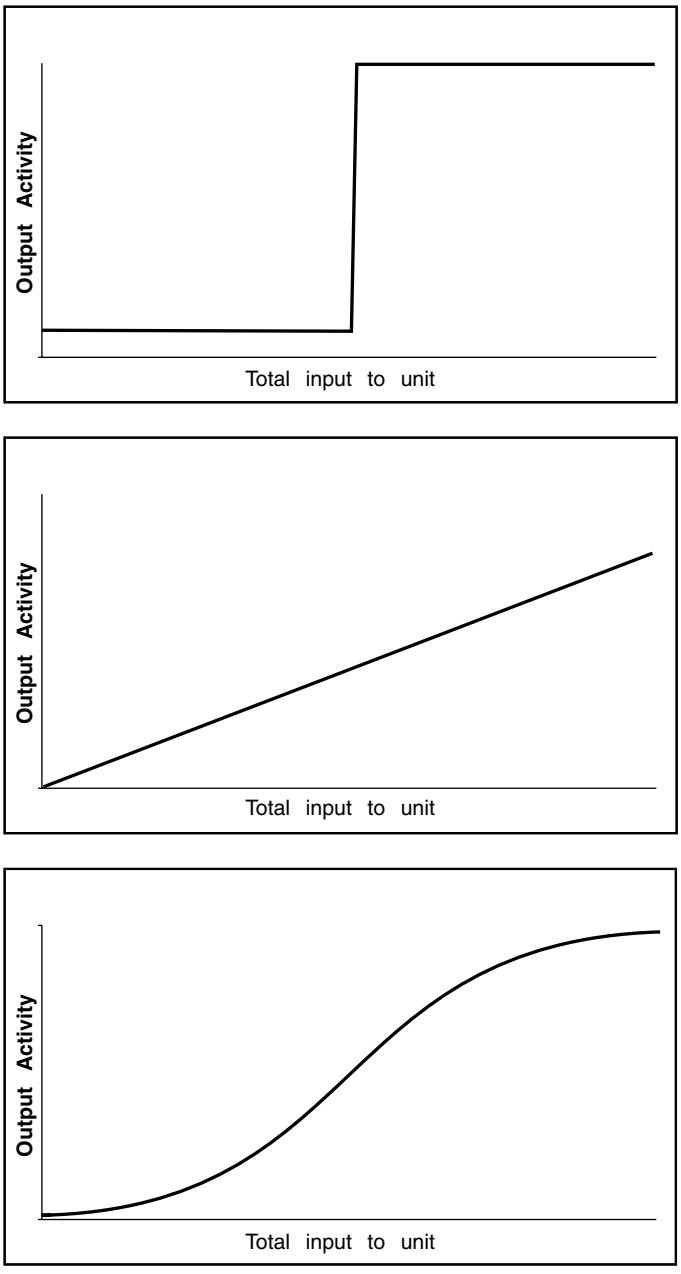

Figure 2.2
Activation functions transform the total input to a node into an activation. Left: A linear function. Middle: A binary threshold function. Right: A nonlinear sigmoidal activation function.

Elman, and Daugherty (1995), input features correspond to speech segments in particular positions: 14 input nodes correspond to 14 possible onsets (beginnings of syllables), six input nodes correspond to six possible instantiations of the nucleus (middles of syllables), and 18 input nodes correspond to 18 possible codas (ends of syllables). The word *bid* would be represented by the simultaneous activation of three nodes, the nodes corresponding to *b* in the initial position, *i* in the nucleus position, and *d* in the coda position. Each of those nodes would also participate in the encoding of other inputs. Other distributed representation schemes include those in which input nodes correspond to phonetic features like [±voiced] (Plunkett & Marchman, 1993) or semantic features like [±circle] or [±volitional] (MacWhinney & Leinbach, 1991). (As discussed in section 2.5, in some models input nodes do not correspond to anything obviously meaningful.)

2.1.4 Relations between Inputs and Outputs

Any given network architecture can represent a variety of different relationships between the input and output nodes, depending on the weights of the connections between units. Consider, for example, the very simple network shown in figure 2.3. Suppose that we wanted to use this model to represent the logical function OR, which is true if either or both of its inputs are true (or turned 'on') and false if both inputs are

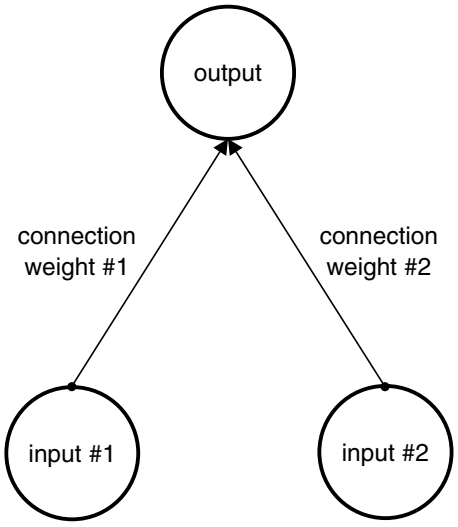

Figure 2.3
A two-layer perceptron with two input nodes and one output node.

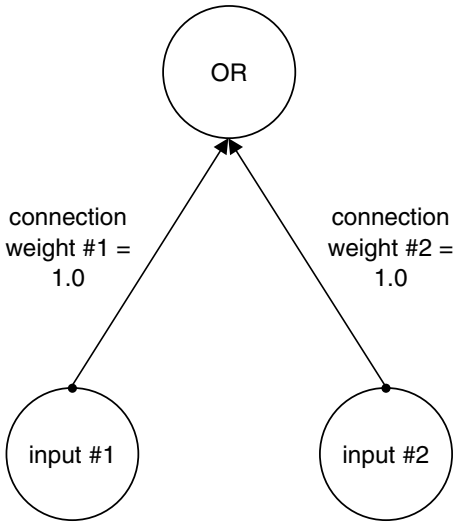

Figure 2.4
A two-layer perceptron that computes the function OR.

false. (As in *School will close if there is a blizzard or a power outage*.) Let's assume that the input units are turned on (set to 1) if they are true and turned off (set to 0) if they are false. Let's also assume that the activation function of the output node is a binary threshold, such that the node has an output activation of 1.0 any time the total input to the output node is equal to or exceeds 1 and an activation value of 0 otherwise.

The total input to the output node is calculated as the sum of (input #1 * connection weight #1) plus (input #2 * connection weight #2). Given the assumptions we have made, we can use infinitely many sets of weights. One set that works is given in figure 2.4, where the weight running from input node #1 to the output node is 1.0 and the weight running from input node #2 to the output node is (also) 1.0.

In the figure, if input node #1 is turned on and input node #2 is turned off, then the weighted sum of the inputs to the output unit is (1.0 * 1.0) + (0.0 * 1.0) = 1.0. Since 1.0 is equal to the threshold, the output unit is activated. If instead both input node #1 and input node #2 are turned on, then the weighted sum of the inputs to the output unit is (1.0 * 1.0) + (1.0 * 1.0) = 2.0, again above the activation threshold of the output node. In contrast, if both input nodes are turned off, then the weighted sum of the inputs to the output node is (0 * 1) + (0 * 1) = 0, a value that is less than the threshold for output activation. The output unit is thus turned off.

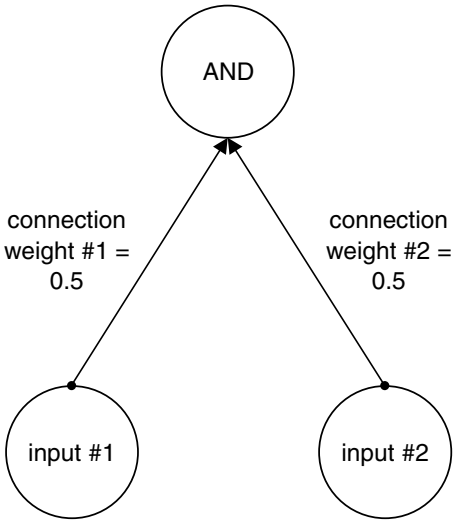

Figure 2.5
A two-layer perceptron that computes the function AND.

Using the same output activation function (a binary threshold of greater than or equal to one) but a different set of weights, such as those depicted in figure 2.5, the same network could be used to represent the logical function AND. Here, the weight running from input node #1 to the output node is 0.5, and the weight running from input node #2 to the output node is 0.5. If both input nodes #1 and #2 are turned on, then the weighted sum of the inputs to the output unit is (0.5 * 1) + (0.5 * 1) = 1.0. Since 1.0 is equal to the threshold, the output unit is activated. If, instead, only input node #1 is turned on, then the weighted sum of the input to the output node is (0.5 * 1) + (0.5 * 0) = 0.5, a value that is less than the threshold for output activation. Hence the output node is not turned on.

2.1.5 The Need for Hidden Units

Although functions like AND and OR are easily represented in simple two-layer networks, many other functions cannot be represented so easily. For example, our simple network could not represent the function of *exclusive or* (XOR), which is true only if exactly one input is true. (You can have either the cake or the ice cream but not both.)

Simple functions like logical AND and logical OR are said to be *linearly separable* because, as illustrated in figure 2.6, we can draw a straight line that divides the inputs that lead to a true output from the inputs that lead to a false output.

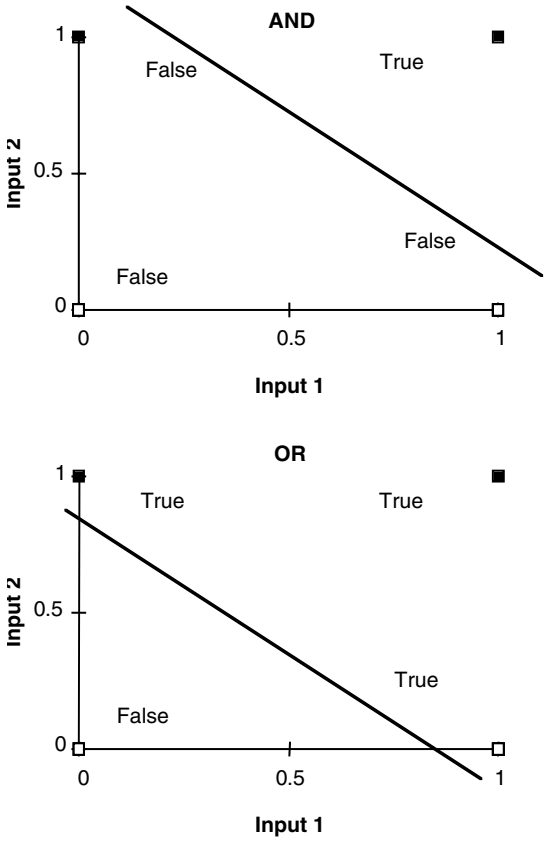

Figure 2.6
Illustrations of the logical functions OR and AND. The axes of these graphs correspond to the values of the input units; each input can be thought of as a point in that space. The labels *true* and *false* indicate the expected outputs corresponding to some sample inputs. The heavy lines show possible ways of dividing the true cases from the false cases.

But as shown in figure 2.7, if we draw the corresponding plot for XOR, no simple line will divide the true cases from the false cases. Functions in which the true cases cannot be separated from the false cases with a single straight line are not linearly separable. It turns out that in such cases no set of weights will do; we simply cannot represent functions like XOR in our simple network (Minsky & Papert, 1969).

As Minsky and Papert (1988) noted, we can get around this problem in an unsatisfying way by customizing our input nodes in ways that build in the function that we are trying to represent. Similarly, we can customize our output function in question-begging ways. For example,

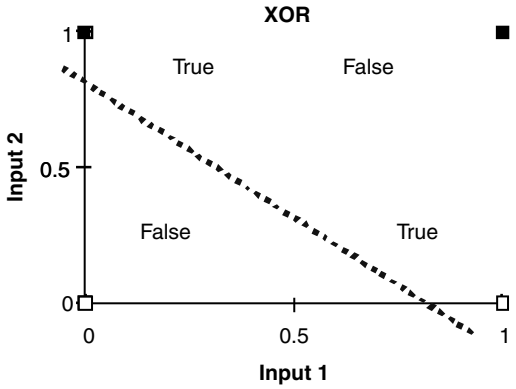

Figure 2.7
The exclusive or (XOR) function. No straight line can separate inputs that yield *true* from the inputs that yield *false*.

if we connect both inputs to the output with weights of 1, we can stipulate that the output node will turn on only if the weighted sum of its input equals exactly 1. But such an activation function—known as *nonmonotonic* because it goes up but then comes back down—essentially builds XOR into the output function. As a consequence, few if any researchers take such an explanation of XOR to be satisfying.

But there is another way of capturing functions that are not linearly separable—without having to rely on either dubious input-encoding schemes or question-begging output-activation functions. Assuming that we stick to our binary threshold of 1, we can readily represent XOR in our network—simply by incorporating hidden units. One way to do so, using two hidden units,[2] is shown in figure 2.8, with values of the hidden units and output unit, for selected input values, given in table 2.1. In effect, the hidden units, which I call *h1* and *h2*, serve as intermediate states in the computation: O = (*h1* * –1.0) + (*h2* * 1.0), where *h1* = ((0.5 * input 1) + (0.5 * input 2)) and *h2* = ((1.0 * input 1) + (1.0 * input 2)).

In our simple example, the meanings of the hidden units are easy to understand. For example, hidden unit *h1* effectively computes the logical AND of input 1 and input 2, and *h2* effectively computes the logical OR of input 1 and input 2. (The output subtracts the value of *h1*'s OR from the value of *h2*'s AND.)

In more complex models, it is sometimes transparent what a given hidden node computes. In a model in which the inputs are words, one hidden unit might be strongly connected to input words that are nouns, while another might be strongly connected to input words that are verbs. In other cases what a given hidden unit is doing may be far less

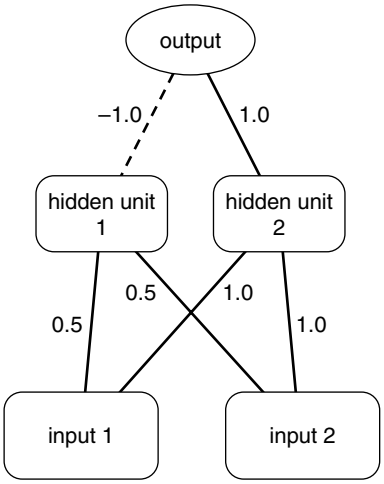

Figure 2.8
A network that represents exclusive or (XOR). All units turn on only if the weighted sum of their inputs is greater than or equal to 1. Thin solid lines indicate positive activation, dotted lines indicate negative activation, and thick lines indicate positive activation with strong absolute values.

Table 2.1
Activation values of units in an exclusive or (XOR) network (see figure 2.8).

Input 1	Input 2	Input to hidden unit 1	Output from hidden unit 1	Input to hidden unit 2	Output from hidden unit 2	Input to output unit	Output
F = 0	F = 0	0	0	0	0	0	0
F = 0	T = 1	0.5	0	1	1	1	1
T = 1	F = 0	0.5	0	1	1	1	1
T = 1	T = 1	1	1	2	1	0	0

transparent, but it is important to bear in mind that all hidden units ever do is apply their activation functions to the weighted sums of their inputs; to a first approximation, they compute (weighted) combinations of their inputs.

Sometimes hidden units are thought of as *recoding* the inputs. For example, in our exclusive or model, one hidden unit recodes the raw inputs by taking their logical AND, while the other hidden unit recodes the raw inputs by taking their logical OR. In this sense, the hidden units serve as *re-presentations* or *internal representations* of the input. Since the

output nodes are typically fed only by the hidden units, these internal representations assume a great importance. For example, in the XOR model the output units do their work by combing the ANDs and ORs produced by the hidden units rather than by directly combining the raw input. Because the behavior of hidden nodes depends on how they are connected to the input nodes, we can sometimes tell something about how a given network parcels up a particular problem by understanding what its hidden units are doing.

2.1.6 Learning

Perhaps the most interesting aspect of these models is that the connection weights need not be set by hand or fixed in advance. Most models are born with their weights initially set to random values.[3] These weights are then adjusted by a *learning algorithm* on the basis of a series of *training examples* that pair inputs with *targets*. The two most common algorithms are the *Hebbian algorithm* and a version of the *delta rule* known as *back-propagation*.

The Hebbian algorithm The Hebbian algorithm, named for a suggestion by D. O. Hebb (1949), strengthens the connection weights between input node A and output node B by a fixed amount each time both are active simultaneously, a process sometimes described by the slogan "cells that fire together, wire together." A somewhat more complex version of the Hebbian algorithm adjusts the weight of the connection between nodes A and B by an amount proportionate to their product (McClelland, Rumelhart & Hinton, 1986, p. 36). In that version, if the product of the activation of node A multiplied by the activation of node B is positive, the connection between them is strengthened, whereas if that product is negative, the connection between nodes A and B is weakened.

The delta rule The *delta rule* changes the weights of the connection between input node A and output node B in proportion to the activation of input node A multiplied by *the difference between what output node B actually produces and the target for output node B*. In a formula:

$$\Delta w_{io} = \eta^* (target_o - observed_o)\, a_i,$$

where Δw_{io} is the change in the weight of the connection that runs from input node i to output node o, η is the learning rate, $target_o$ is the target for node o, $observed_o$ is the actual activation value of node o, and a_i is the activation value for input i.

Back-propagation One cannot directly apply the delta rule to networks that have hidden layers—because the targets for hidden nodes are unknown. The *back-propagation* algorithm, introduced by Rumelhart,

Hinton, and Williams (1986), supplements the delta rule with additional machinery for estimating "targets" for the hidden units.

Back-propagation receives its name from the fact that the learning algorithm operates in a series of stages that move backward through the network. In the first stage, the algorithm adjusts the weights of connections that run from the hidden units to the output units.[4] Following the delta rule, each connection that runs from a hidden node *h* to an output node *o* is adjusted as a function of the product of the activation value of hidden node *h* and a measure of error for output node *o*, all scaled by the parameter called the *learning rate* (discussed below).[5]

The second stage begins after all the connections from hidden nodes to output nodes have been adjusted. At this point, using a process of the sort that is sometimes called *blame-assignment*, the algorithm computes the extent to which each hidden node has contributed to the overall error. The connection weights from a given input node *i* to a given hidden node *h* are adjusted by multiplying the activation value of *i* times the *blame score* for *h*, scaled by the value of the learning rate (a parameter that is discussed in the next section). The way in which back-propagation adjusts the connection weights that feed hidden nodes is thus very much analogous to the way in which the delta rule adjusts connection weights that feed output nodes, but with the blame-assignment score substituting for the difference between target and observed values.

The equations are as follows. Connections from hidden unit *h* to output node *o* are adjusted by Δw_{ho}, where

$$\Delta w_{ho} = \eta \delta_o a_h$$

and

$$\delta_o = (t_o - a_o) a_o (1 - a_o)$$

Connections from input unit *i* to hidden node *h* are adjusted by Δw_{ih}, where

$$\Delta w_{ih} = \eta \delta_h a_i$$

and

$$\delta_h = a_h (1 - a_h) \sum_k \delta_k w_{kh}.$$

Algorithms like back-propagation are known as *gradient-descent* algorithms. To understand this metaphor, imagine that after each trial we calculate the difference between the target and the observed output (that is, the output that the model actually produces). This difference, a measure of error, could be thought of as a point on a hilly terrain: the object is to find the lowest point (the solution with the smallest overall error).

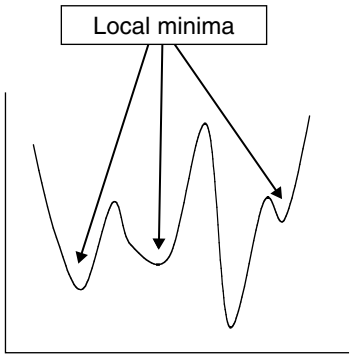

Figure 2.9
The hill-climbing metaphor. Arrows point to locations where error is low and small steps would lead only to greater error.

An inherent risk is that if we use an algorithm that is not omniscient we could get stuck in a *local minimum* (see figure 2.9). A local minimum is a place from which no small step that we can take will immediately lead to a better solution.

In simple tasks, networks trained with back-propagation typically reach an adequate solution, even if that solution is not perfect. It is more controversial whether these algorithms are adequate in more complex tasks (for further discussion of the issue of local minima, see Rumelhart, Hinton & Williams, 1986; Tesauro & Janssens, 1988).

2.1.7 Learning Rate

Learning algorithms such as back-propagation use a parameter known as *learning rate,* a constant that is multiplied by the error signal and node activations. In most models, the learning rate is relatively small, leading to learning that is necessarily gradual. The two principled reasons that learning rates tend to be small are both nicely explained by McClelland, McNaughton, and O'Reilly (1995, p. 437):

> Accuracy of measurement will increase with sample size, and smaller learning rates increase the effective sample size by basically using the network to take a running average over a larger number of recent examples.
>
> Gradient descent procedures . . . are guaranteed to lead to an improvement, but only if infinitesimally small adjustments are made to the connectionist weights at each step. . . . After each pass through the training set, the weights can be changed only a little; otherwise, changes to some weights will undermine the effects of changes to the

others, and the weights will tend to oscillate back and forth. With small changes, on the other hand, the network progresses a little after each pass through the training corpus.

2.1.8 Supervision

Because models that are trained by back-propagation require an external teacher, they are said to be *supervised*.[6] An obvious question that arises with respect to any supervised model is, Where does the teacher or supervisor come from? Some critics of the multilayer-perceptron approach would like to dismiss all supervised models on the basis of the implausibility of the supervisor, but such wholesale criticism is unfair. Some models do depend on a teaching signal that is not plausibly available in the environment, but in other cases the teaching signal may be a piece of information that is plausibly available in the environment. For example, in the sentence-prediction network that is described below, the input to the model is a word in a sentence, and the target is simply the next word in that sentence. It does not seem unreasonable to suppose that a learner has access to such readily available information. The question of whether the teacher is plausible must be raised separately for each supervised model.

2.1.9 Two Types of Multilayer Perceptrons

All the examples that I have discussed so far are called *feedforward networks* because activation flows forward from the input nodes through the hidden nodes to the output nodes. A variation on the feedforward network is another type of model known as the *simple recurrent network* (SRN) (Elman, 1990), itself a variation on an architecture introduced earlier by Jordan (1986). Simple recurrent networks differ from feedforward networks in that they have one or more additional layers of nodes, known as *context units*, which consist of units that are fed by the hidden layer but that also feed back into the (main) hidden layer (see figure 2.10). The advantage of these more complex models, as is made clear later in this chapter, is that, unlike feedforward networks, simple recurrent networks can learn something about sequences of elements presented over time.

2.2 Examples

The vast majority of the connectionist models that have been used in discussions of cognitive science are multilayer perceptrons, either feedforward networks or simple recurrent networks. Among the many domains in which such models have been used are the acquisition of linguistic inflection (e.g., Rumelhart & McClelland, 1986a), the acquisi-

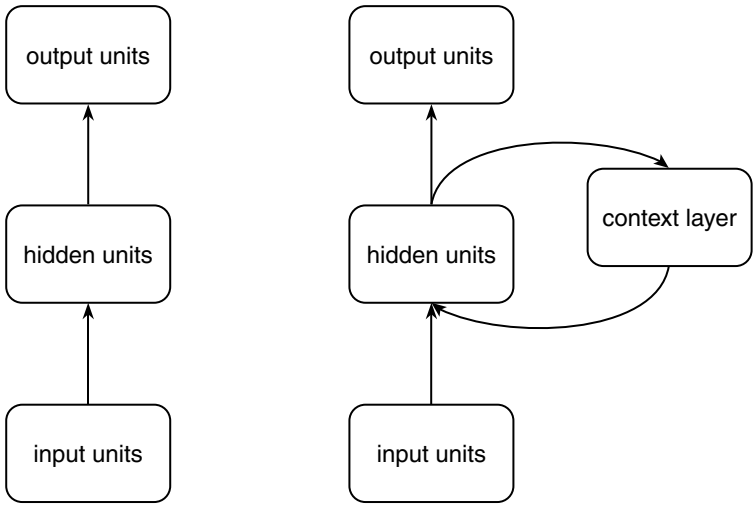

Figure 2.10
A feedforward network (left) and a simple recurrent network (right).

tion of grammatical knowledge (Elman, 1990), the development of object permanence (Mareschal, Plunkett & Harris, 1995; Munakata, McClelland, Johnson & Siegler, 1997), categorization (Gluck & Bower, 1988; Plunkett, Sinha, Møller & Strandsby, 1992; Quinn & Johnson, 1996), reading (Seidenberg & McClelland, 1989), logical deduction (Bechtel, 1994), the "balance beam problem" (McClelland, 1989; Shultz, Mareschal & Schmidt, 1994), and the Piagetian stick-sorting task known as *seriation* (Mareschal & Shultz, 1993). This list is by no means comprehensive; many more examples can be found in books, journals, and conference proceedings. In this section I focus on two particular examples that are well known and that exemplify the two major classes of multilayer perceptrons—feedforward networks and simple recurrent networks. Each of these examples has played a pivotal role in discussions about the implications of connectionism for symbol-manipulation.

2.2.1 The Family-Tree Model: A Feedforward Network

The family-tree model, described by Hinton (1986), was designed to learn about the kinship relations in the two family trees depicted in figure 2.11. These two family trees are *isomorphic,* which is to say that they map onto one another perfectly; each family member in one family tree corresponds to a family member in the other family tree.

The model itself, depicted in figure 2.12, is a multilayer perceptron, with activation flowing strictly from input nodes through the output

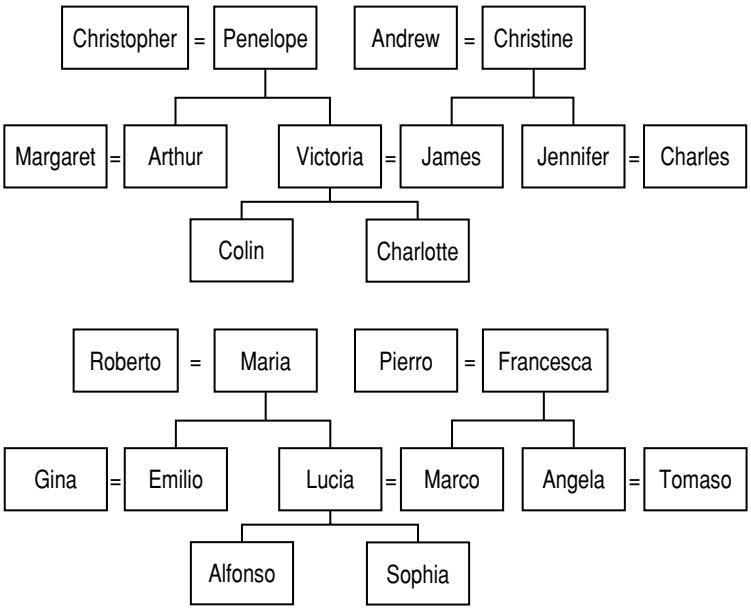

Figure 2.11
The two isomorphic family trees used in Hinton (1986). The symbol = indicates the meaning *married to.* For example, Penelope is married to Christopher and is the mother of Arthur and Victoria.

nodes. Particular facts are encoded as input pairs. Each input node in the model encodes either one of the 24 individuals depicted in the two family trees or one of 12 familial relationships (*father, mother, husband, wife, son, daughter, uncle, aunt, brother, sister, nephew,* and *niece*). Output nodes represent particular individuals. Given the 12 possible familial relationships that are encoded by the relationship input units and given the two family trees that Hinton used, there are a total of 104 possible facts of the form *X is the Y of Z*, such as *Penny is the mother of Victoria and Arthur.*

Initially, the model's weights were randomized. At this point, the model responded randomly to terms such as *father, daughter,* and *sister* and did not know any specific facts, such as which people were the children of Penelope. But through the application of back-propagation,[7] the model gradually learned specific facts. Hinton argued that the model learned something about the kinship terms (*father, daughter,* and so on) on which it is trained. (I challenge Hinton's argument in chapter 3.)

Rather than training the model on all 104 of these facts, Hinton left four facts in reserve for testing. In particular, he conducted two test runs

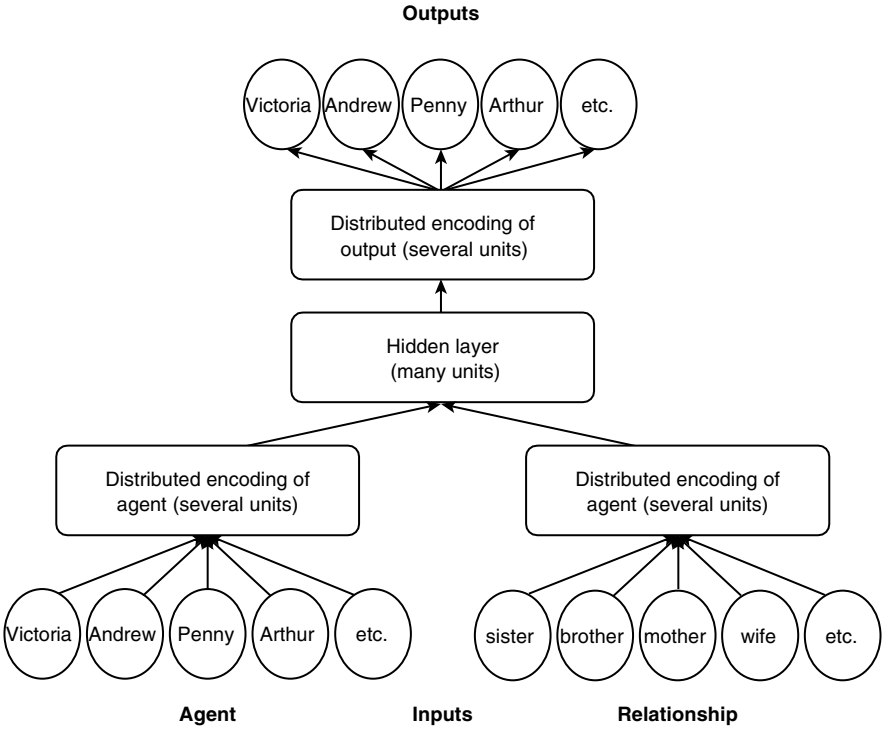

Figure 2.12
Hinton's (1986) family-tree model. Circles indicate units; squares indicate sets of units. Not all units or all connections are shown. Inputs to the model are indicated by activating one agent unit and one relationship unit. The set of patients corresponding to that agent and relationship are activated within the output bank. All activation flows forward, from the input to the output.

of this model, each time training the model on exactly 100 of the 104 possible facts. The test runs differed from each other in the set of initial random weights that were used; the two test runs might be thought of as roughly analogous to two different experimental subjects. On one test run, the model got all four test cases correct, and on the second test run it got three of four correct, both times showing at least some ability to generalize to novel cases.

Part of what makes the model interesting is that the hidden units appear to capture notions such as "which generation a person belongs to . . . [and] which branch of the family a person belongs to" that are not explicitly encoded in the input. McClelland (1995, p. 137) took Hinton's model to show "that it was possible to learn relations that cannot be expressed in terms of correlations between given variables. What . . .

[the] network did was discover new variables into which the given variables must be translated." Similarly, Randall O'Reilly (personal communication, February 6, 1997) argued that Hinton's "network developed (through learning with backprop) abstract internal representations in the 'encoding' hidden layers and then, in a subsequent layer, encoded relationship information in terms of these abstracted internal representations."

2.2.2 The Sentence-Prediction Model: A Simple Recurrent Network

Another important and influential multilayer perceptron, in this case a simple recurrent network rather than a feedforward network, is the sentence-prediction model, as described by Elman (1990, 1991, 1993). A simplified version of the sentence-prediction model is given here, in figure 2.13. The model is much like a standard feedforward network, but as I indicated earlier, it is supplemented with a *context layer* that records a copy of the state of the hidden layer. This context layer feeds back into

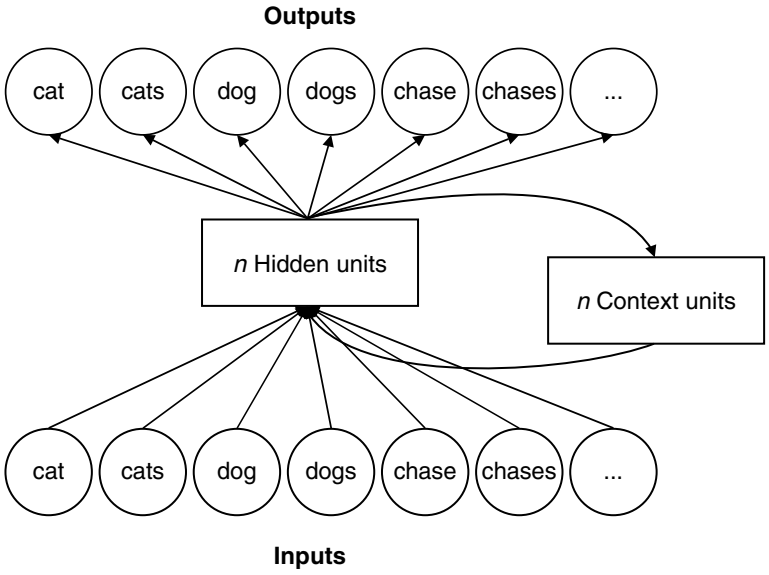

Figure 2.13
A simplified version of Elman's (1990, 1990, 1993) sentence-prediction model. Circles (input nodes and output nodes) represent particular words; The input to the model is presented with a single word at each time step; the target is the next word in that sequence. Rectangles contain sets of units. Each hidden unit projects to one context unit with a fixed weight of 1.0. Each context unit feeds into every hidden unit, with modifiable connection weights. Elman's model has 26 input nodes and 26 output nodes.

the hidden layer at the next time step. At any given point, the activation levels of the hidden units depend not only on the activation of the input units but also on the state of these context units. In this way, the units in the context layer serve as a sort of memory of the model's history.

The sentence-prediction model was trained on a series of sentences taken from a semi-realistic artificial grammar that included 23 words and a variety of grammatical dependencies such as subject-verb agreement (*cats love* and *cat loves*) and multiple embeddings. At each time step, the input to the model is the current word (indicated by the activation of some node), and the target output is the next word in the current sentence.

The weights of the model (except the weights from the hidden unit to the context layer, which are fixed) were adjusted by the back-propagation algorithm. Once trained, the model was often able to predict plausible continuations for strings such as *cats chase dogs* and even more complicated strings such as *boys who chase dogs see girls*—without any explicit grammatical rules. For this reason, the simple recurrent network has been taken as strong evidence that connectionist models might obviate the need for grammatical rules. For example, P. M. Churchland (1995, p. 143) writes that

> The productivity of this network is of course a feeble subset of the vast capacity that any normal English speaker commands. But productivity is productivity, and evidently a recurrent network can possess it. Elman's striking demonstration hardly settles the issue between the rule-centered approach to grammar and the network approach. That will be some time in working itself out. But the conflict is now an even one. I've made no secret where my own bets will be placed.

Churchland is not alone in his enthusiasm. According to a survey of citations in the span 1990 to 1994 (Pendlebury, 1996), Elman's (1990) discussion of the simple recurrent network was the most widely cited paper in psycholinguistics and the eleventh most cited paper in psychology.

2.3 How Multilayer Perceptrons Have Figured in Discussions of Cognitive Architecture

The idea that connectionist networks might offer an alternative to symbol-manipulation started to become prominent when J. A. Anderson and Hinton (1981, pp. 30–31) wrote that "[w]hat we are asserting is that the symbol-processing metaphor may be an inappropriate way

of thinking about computational processes that underlie abilities like learning, perception, and motor skills. . . . There are alternative models that have different computational flavor and that appear to be more appropriate for machines like the brain, which are composed of multiple simple units that compute in parallel." The idea became even more prominent in 1986 with the publication of an influential paper by Rumelhart and McClelland (1986a). Rumelhart and McClelland presented a two-layer perceptron that captures certain aspects of children's acquisition of the English past tense. They suggest that their model can "provide a distinct alternative to . . . [rules] in any explicit sense" (for discussion, see section 3.5). Elsewhere in the same book, Rumelhart and McClelland (1986b, p. 119) clearly distance themselves from those who would explore connectionist implementations of symbol-manipulation when they write, "We have not dwelt on PDP implementations of Turing machines and recursive processing engines [canonical machines for symbol-manipulation] because we do not agree with those who would argue that such capabilities are of the essence of human computation."

Similarly, Bates and Elman (1993, p. 637) suggest that their particular connectionist approach "runs directly counter to the tendency in traditional cognitive and linguistic research to seek 'the rule' or 'the grammar' that underlies a set of behavioral regularities. . . . [These systems] do not look like anything we have ever seen before." And Seidenberg (1997, p. 1600) writes that the kind of network he advocates "incorporates a novel form of knowledge representation that provides an alternative to equating knowledge of a language with a grammar. . . . Such networks do not directly incorporate or implement traditional grammars."

Still, although such claims have received a great deal of attention, not everyone who advocates multilayer perceptrons denies that symbol-manipulation plays a role in cognition. A somewhat weaker but commonly adopted view holds that symbol-manipulation exists but plays a relatively small role in cognition. For example, Touretzky and Hinton (1988, pp. 423–424) suggest that there is an important role for connectionist alternatives to symbol-manipulation: "many phenomena which appear to require explicit rules can be handled by using connection strengths." But at the same time they allow for connectionist models that implement rules, when they write that "we do not believe that [the fact the some phenomena can be handled without rules] . . . removes the need for a more explicit representation of rules in tasks that more closely resemble serial, deliberate reasoning. A person can be told an explicit rule such as '*i* before *e* except after *c*' and can then apply this rule to the relevant cases."

2.4 The Appeal of Multilayer Perceptrons

Whether multilayer perceptrons turn out to be the best account of all of cognition, of some of it, or of none of it, it is clear that they have attracted a great deal of attention. As Paul Smolensky wrote in 1988 (p. 1), "The connectionist approach to cognitive modeling has grown from an obscure cult claiming a few true believers to a movement so vigorous that recent meetings of the Cognitive Science Society have begun to look like connectionist pep rallies."

Why have so many people focused on these models? It is not because the models have been shown to be demonstrably better at capturing language and cognition than alternative models. Most discussions of particular models present those models as being *plausible* alternatives, but with the possible exception of models of certain aspects of reading, few models have been presented as being *uniquely* able to account for a given domain of data. As Seidenberg (1997, p. 1602) puts it, "The approach is new and there are as yet few solid results in hand."

2.4.1 Preliminary Theoretical Considerations

The argument for eliminating symbol-manipulation thus rests not so much on empirical arguments against symbol-manipulation in particular domains but instead primarily on what one might think of as preliminary theoretical considerations. One reason that multilayer perceptrons seem especially attractive is that they strike some scholars as being more "more compatible than symbolic models with what we know of the nervous system" (Bechtel & Abrahamsen, 1991, p. 56). Nodes, after all, are loosely modeled on neurons, and the connections between nodes are loosely modeled on synapses. Conversely, symbol-manipulation models do not, on their surface, look much like brains, and so it is natural to think of the multilayer perceptrons as perhaps being more fruitful ways of understanding the connection between brain and cognition.

A different reason for favoring multilayer perceptrons is that they have been shown to be able to represent a very broad range of functions. Early work on connectionism virtually died out with Minsky and Papert's (1969) proof of limitations on networks that lacked hidden layers; advocates of the newer generation of models take heart in the broader representational abilities of the newer models. For example, P. M. Churchland (1990) has called multilayer perceptrons "universal function approximators" (see also Mareschal and Shultz, 1996). A *function approximator* is a device that takes a set of known points and interpolates or extrapolates to unknown points. For instance, a device that maps between motor space (a space defined in terms of forces and joint

angles) and visual space can be thought of as learning a function; likewise, the mapping between the stem of a verb and its past tense can be thought of as a function. For virtually any given function one might want to represent, there exists some multilayer perceptron with some configuration of nodes and weights that can approximate it (see Hadley, 2000).

Still others favor multilayer perceptrons because they appear to require relatively little in the way of innate structure. For researchers drawn to views in which a child enters the world with relatively little initial structure, multilayer perceptrons offer a way of making their view computationally explicit. Elman et al. (1996, p. 115), for instance, see multilayer perceptron models as providing a way to "simulate developmental phenomena in new and . . . exciting ways . . . [that] show how domain-specific representations can emerge from domain-general architectures and learning algorithms and how these can ultimately result in a process of modularization as the end product of development rather than its starting point."

Multilayer perceptrons are also appealing because of their intrinsic ability to learn (Bates & Elman, 1993) and because of their ability to *gracefully degrade:* they can tolerate limited amounts of noise or damage without dramatic breakdowns (Rumelhart & McClelland, 1986b, p. 134). Still others find multilayer perceptrons to be more parsimonious than their symbolic counterparts. For example, multilayer perceptron accounts of how children inflect the English past tense hold that children use the same mechanism for inflecting both irregular (*sing-sang*) and regular (*walk-walked*) inflection, whereas rule-based accounts must include at least two mechanisms, one for regular inflection and another for exceptions to the rule. (For further discussion of models of inflection, see section 3.5.)

2.4.2 Evaluation of Preliminary Considerations

None of the preliminary considerations that apparently favor multilayer perceptrons—biological plausibility, universal function approximation, and the like—is actually decisive. Instead, as is often the case in science, preliminary considerations do not suffice to settle scientific questions. For example, although multilayer perceptrons can approximate a broad range of functions (e.g., Hornik, Stinchcombe & White, 1989), it is not clear that the range is broad enough. Hadley (2000) argues that these models cannot capture a class of functions (known as the partial recursive functions) that some have argued capture the computational properties of human languages.

Whether or not these models can in principle capture a broad enough range of functions, the proofs of Hornik, Stinchcombe, and White apply

only to networks that have an arbitrary number of hidden nodes. Such proofs do not show that a *particular* network with fixed resources (say, a three-layer network with 50 input nodes, 30 hidden nodes, and 50 output nodes) can approximate any given function. Rather, these kinds of proofs show that for every function within some very broad class there exists *some* connectionist model that can model that function—perhaps a different model for each function. Furthermore, the proofs do not guarantee that any particular network can *learn* that particular function given realistic numbers of training examples or with realistic numbers of hidden units. They in no way guarantee that multilayer perceptrons can generalize from limited data in the way that humans do. (For example, we will see in chapter 3 that even though all multilayer perceptrons can represent the "identity" function, in some cases they cannot learn it.) In any case, all this talk of universal function approximators may be moot. Neither the brain nor any actually instantiated network can literally be a universal function approximator, since the ability to approximate any function depends (unrealistically) on having infinite resources available.[8] Finally, just as one can build some multilayer network to approximate any function, one can build some symbol-manipulating device to approximate any function.[9] Talk about universal function approximation is thus a red herring that does not actually distinguish between multilayer perceptrons and symbol-manipulation.

Similarly, at least for now, considerations of biological plausibility cannot choose between connectionist models that implement symbol-manipulation and connectionist models that eliminate symbol-manipulation. First, the argument that multilayer perceptrons are biologically plausible turns out to be weak. Back-propagating multilayer perceptrons lack brainlike structure and differentiation (Hubel, 1988) and require synapses that can vary between being excitatory and inhibitory, whereas actual synapses cannot so vary (Crick & Asunama, 1986; Smolensky, 1988). Second, the ways in which multilayer perceptrons are brainlike (such as the fact that they consist of multiple units that operate in parallel) hold equally for many connectionist models that are consistent with symbol-manipulation, such as the temporal-synchrony framework (discussed in chapters 4 and 5) or arrays of McCulloch-Pitts neurons arranged into logic gates.

The flip side of biological plausibility is biological implausibility. Some people have argued against symbol-manipulation on the grounds that we do not know how to implement it in the brain (e.g., Harpaz, 1996). But one could equally argue that we do not know how to implement back-propagation in the brain. Claims of biological implausibility are most often merely appeals to ignorance that can easily mislead. For example, we do not yet know exactly how the brain encodes short-term

memory, but it would be a mistake to conclude that the psychological process of short-term memory is "biologically implausible" (Gallistel, 1994). Connectionism should not be in the business of sticking slavishly to what is know about biology, since so little is known. As Elman et al. (1996, p. 105) put it, "There is obviously a great deal which remains unknown about the nervous system and one would not want modeling to always remain several paces behind the current state of the science." For now, then, considerations about biological plausibility and biological implausibility are simply too weak to choose between models.[10] In short, there is no guarantee that the right answer to the question of how cognition is implemented in the neural substrate will be one that appears to our contemporary eyes to be "biologically plausible." We must not confuse what currently seems biologically plausible with what actually turns out to be biologically real.

The other preliminary considerations are likewise not adequate for choosing between architectures. For example, neither the ability to learn nor the ability to degrade gracefully is unique to multilayer perceptrons. Modeling learning is a core focus of canonical symbolic models of cognition such as SOAR (Newell, 1990) and models of grammar learning such as those described by Pinker (1984). And while some symbolic systems are not robust with respect to degraded input, others are (Fodor & Pylyshyn, 1988). For example, Barnden (1992b) describes a symbolic analogy-based reasoning system that is robust to partial input. A variety of symbol-manipulating mechanisms can recover from degraded input, ranging from error-correction algorithms that check the accuracy of transmitted information to systems that seek items that share a subset of attributes with some target. Whether these mechanisms are adequate to account for the ability of humans to recover from degraded input remains to be seen; for now, there is little in the way of relevant empirical data.

Another question that is logically independent of the distinction between connectionist models that would and would not implement symbol-manipulation is the question of whether the mind contains a great deal of innate structure. Although multilayer perceptrons typically have relatively little innate structure, it is possible in principle to prespecify their connection weights (for an example of a system in which connection weights are in fact to some extent prespecified, see Nolfi, Elman & Parisi, 1994). Similarly, although many symbol-manipulating models have a great deal of innate structure, not all do (e.g., Newell, 1990).

Finally, although it is true that one could argue that multilayer perceptrons are more parsimonious than symbolic models, one could equally argue that they are less parsimonious. As McCloskey (1991)

notes, one could argue that networks with thousands of connection weights have thousands of free parameters. Because biological systems are clearly complex, constraining ourselves a priori to just a few mechanisms may not be wise. As Francis Crick (1988, p. 138) puts it, "While Occam's razor is a useful tool in physics, it can be very a dangerous implement in biology." In any case, parsimony chooses only between models that adequately cover the data. Since we currently lack such models, applying parsimony is for now premature.

In short, none of these preliminary considerations forces us to accept—or reject—multilayer perceptrons. Since they can be neither accepted or rejected at this point, it is now time that we begin to evaluate them on other grounds. We must also begin to confront the thorny question of whether multilayer perceptrons serve as implementations of or alternatives to symbol-manipulation, a question that turns out to be more difficult than it first appears.

2.5 Symbols, Symbol-Manipulators, and Multilayer Perceptrons

First, though, before we examine what I think truly distinguishes multilayer perceptrons from symbol-manipulation, it is important to clear up a red herring. A number of people seem to think that a key difference between multilayer perceptrons and symbol-manipulators is that the latter make use of symbols but the former do not. For example, Paul Churchland (1990, p. 227) seems to suggest this when he writes

> An individual's overall-theory-of-the-world, we might venture, is not a large collection or a long list of stored symbolic items. Rather, it is a specific point in that individual's synaptic weight space. It is a configuration of the connection weights, a configuration that partitions the system's activation-vector space(s) into useful divisions and subdivisions relative to the inputs typically fed to the system.

Book titles like *Connections and Symbols* (Pinker & Mehler, 1988) seem to further this impression. But what I want to do in this brief section is to persuade you that it is not terribly valuable to think of the difference between competing accounts of cognitive architecture as hinging on whether the mind represents symbols.

The trouble is that there are too many different ways of defining what is meant by a symbol. It is certainly possible to define the term *symbol* in a way that means that symbol-manipulators have them and multilayer perceptrons do not, but it is just as easy to define the term in a way that entails that both symbol-manipulators and multilayer perceptrons have them. It might even be possible to define the term in such a way that

neither classical artificial intelligence (AI) programs (which are usually taken to be symbol-manipulators) nor multilayer perceptrons have them (for further discussion of this latter possibility, see Searle, 1992).

On pretty much anyone's view, to be a symbol is, in part, to be a representation. For example, the word *cats* as it appears on this page is a symbol in the external world that stands for cats. (More precisely, either for cats in the world or the idea of cats; this is not the place to worry about that sort of concern). Advocates of symbol-manipulation assume that there is something analogous to external symbols (words, stop signs, and the like) inside the head. In other words, they assume that there are mental entities—patterns of matter or energy inside the head—that represent either things in the world or mental states, concepts or categories.

If all it took to be a symbol was to be a mental representation, probably all modern researchers would agree that there were symbols. Hardly anyone since Skinner has doubted that there are mental representations of one sort or another. What might be less obvious is that advocates of multilayer perceptrons are committed to at least one of the sorts of mental representations that is often taken to be symbolic: the representation of categories or *equivalence classes.*

A programmer building a classical AI model might assign a particular pattern of binary bits to represent the idea of a cat; a programmer building a multilayer perceptron might assign a particular node to represent the idea of a cat. In both approaches, the representation of CAT is context-independent: every time the computer simulation—whether a classical AI model or a multilayer perceptron model—is representing cat, it does the same thing. With respect to such an encoding, all cats are represented equivalently.

There has been some confusion in the literature on this point. For example, people have talked about Elman's sentence-prediction model as if it had context-dependent representations of its input words. But in fact, the input nodes are context-independent (the word *cat* always turns on the same node regardless of where in a sentence it appears), and the hidden nodes do not truly represent individual words; instead, the hidden units represent sentence fragments. So it's not that *cat* is represented differently by the hidden units in the sentence *cats chase mice* as opposed to the sentence *I love cats.* It's that those two particular *sentence fragments* happen to elicit different patterns of hidden unity activity. The only representation of *cat* per se is the activation of the input unit **cat,** and that activation is context-independent. A more general version of the suggestion about Elman's model is Smolensky's (1988, 1991) claim that connectionist "subsymbols" are context-dependent, but Smolensky never spells out exactly how this works. The actual examples he gives of

representations are invariably grounded in lower-level features that are themselves context-independent. For example, *cup of coffee* is grounded in context-independent features like **+porcelain-curved-surface.** Hence the subsymbol-symbol distinction seems to be a distinction without a difference.

Also often mentioned in these sorts of discussions is the distributed-versus-localist distinction. A great many people have made it seem that multilayer perceptrons are special because they make use of distributed representations. For example, instead of representing CAT with a single node, CAT might be represented by a set of nodes like **+furry, +four-legged, +whiskered,** and the like. But not all multilayer perceptrons use distributed representations. Elman's sentence-prediction model for example, uses a single node for each distinct word that it represents. Moreover, not all symbol-manipulators use localist representations. For example, digital computers are canonical symbol-manipulators, and some of their most canonical symbols are distributed encodings. In the widely adopted ASCII code, every instance of the capital letter A is represented by one set of 1s and 0s (that is, 01000001), and every instance of the capital letter B by a different set of 1s and 0s (that is, 01000010).[11] As Pinker and Prince (1988) point out, distributed phonological representations are the hallmark of generative phonology (e.g., Chomsky & Halle, 1968).

My point is that attempts to differentiate multilayer perceptrons from symbol-manipulators cannot rest on questions such as whether there are context-independent mental representations of categories or whether mental representations are distributed. Indeed, one might argue that we ought to look elsewhere in trying to differentiate multilayer perceptrons and symbol-manipulators. For example, for Vera and Simon (1994, p. 360), multilayer "connectionist systems certainly differ in important respects from 'classical' [symbol-manipulating] simulations of human cognition . . . [but] symbolic-nonsymbolic is not one of the dimensions of this difference."

But Vera and Simon's view is not the only possible view. Others argue that symbolhood rests on far more than the ability to represent context-independent categories. For example, one view is that something can be a symbol only if it can appear in a rule (e.g., Kosslyn & Hatfield, 1984). Another view is that a symbol must be able to participate in certain kinds of structured representations (e.g., Fodor & Pylyshyn, 1988). And it seems pretty clear that one might want symbols that stand for particular individuals (Felix) rather than categories (CATS).

My own view is that these cases simply point to a taxonomy of different kinds of things that symbols can stand for—namely categories (CATS), variables (**x,** as in for all **x,** such that **x** is category **y**), computational

operations (+, –, concatenate, compare, etc.), and individuals (Felix). To my mind, a system that can use representations for even one of those four kinds of things counts as having symbols. After all, any given classical AI program may use only a subset of those four kinds of representations. For example, a tic-tac-toe–playing program might not have any need for structured representations or a difference between kinds and individuals but might need variables and operations. Since multilayer perceptrons have context-independent representations of categories, I count them as having symbols.

Whether or not you agree with my permissive view, it is clear that we are simply delaying the inevitable. The interesting question is not whether we want to call a system that has context-independent representations of categories *symbolic* but rather whether the mind is a system that represents variables, operations over variables, structured representations, and a distinction between kinds and individuals.