Chapter 3

Relations between Variables

In a simple perceptron, patterns are recognized before "relations"; indeed, abstract relations, such as "A above B" or "the triangle is inside the circle" are never abstracted as such, but can only be acquired by means of a sort of exhaustive rote-learning procedure, in which every case in which the relation holds is taught to the perceptron individually.
—Rosenblatt (1962, p. 73)

3.1 The Relation between Multilayer Perceptron Models and Rules: Refining the Question

Computer programs are in large part specified as sets of operations over variables. For example, the cost of a set of widgets that a customer has ordered might be calculated by multiplying the contents of a variable that represents the cost per widget times the contents of a variable that represents the number of widgets: **total_cost = item_cost * number_ordered.**

Does the mind make use of something analogous? Does it have a way of representing variables and a way of representing relations between variables? Proponents of symbol-manipulation assume that the answer is yes—that we make use of open-ended schemas such as "form a progressive of any verb by adding *-ing* to its stem" (such as *walk-walking*). Because such schemas are much like algebraic equations (**prog = stem** + *ing*), I refer to them as *relations between variables* or *algebraic rules.*

Although it seems clear enough that we can manipulate algebraic rules in "serial, deliberate reasoning", not everybody agrees that such abstract relationships between variables play an important role in other aspects of language and cognition. For example, as mentioned earlier, Rumelhart and McClelland's (1986a) two-layer perceptron was an attempt to explain how children might acquire the past tense of English without using anything like an explicit rule.[1]

What I want to do here is to clarify the relationship between multilayer perceptrons and devices that perform operations over variables. As far as I can tell this relationship has never been clearly specified (definitely not in my own earlier writings). The relationship between multilayer perceptrons and devices that compute *operations* over variables is much more subtle than has been realized. A better understanding of that relationship will help clarify whether the mind does in fact make use of operations over variables and also clarify how such operations can be implemented in a neural substrate.

To make the strongest possible case that the mind does in fact implement operations over variables, I focus on what I call *universally quantified one-to-one* mappings (UQOTOM). The terms *universally quantified* and *one-to-one* come from logic and mathematics. A function is *universally quantified* when it applies to all instances in its domain. Such a function might be specified as, say, "For all x such that x is an integer" or "For all x such that x is a verb stem." A function is *one-to-one* if each output maps onto a single input in its domain. For example, in the function $f(\mathbf{x}) = \mathbf{x}$, the output 6 corresponds to the input 6 (and no other); the output 3,252 corresponds to the input 3,252 (and no other); and so forth. In the function $f(\mathbf{x}) = 2\mathbf{x}$, the output 6 corresponds to the input 3 (and no other), and so on. (One example of a function that is not one-to-one is the many-to-one function that equals 1 if $\mathbf{x}$ is odd, 0 if $\mathbf{x}$ is even.)

Two particularly important functions that are both universally quantified and one-to-one are *identity* ($f(\mathbf{x}) = \mathbf{x}$, comparable to the "copy" operation in a computer's "machine language") and *concatention* ($f(\mathbf{x}, \mathbf{y}) = \mathbf{xy}$, such as **past = stem** concatenated with *-ed*).[2] In what follows, I frequently use the example of identity, but identity is just one among many possible UQOTOM.

I do not mean to suggest that UQOTOM are the only mappings people compute. But UQOTOM are especially important to the arguments that follow because they are functions in which every new input has a new output. Because free generalization of UQOTOM would preclude memorization, evidence that people (or other organisms) can freely generalize UQOTOM would be particularly strong evidence in support of the thesis that people (or other organisms) can perform operations over variables. (A UQOTOM is not the only kind of mental operation that might reasonably be called an operation over variables. There may be other kinds of operations over variables as well, such as one that determines whether a given number is odd or even. But because it is harder to be certain about the mechanisms involved in those cases, I leave them open.)

3.1.1 Can People Generalize UQOTOM?

There is ample evidence, I think, that people can generalize universally quantified one-to-one mappings. To illustrate this, I start with a very

Table 3.1
Input and output data.

Training Item	
Input	Output
1010	1010
0100	0100
1110	1110
0000	0000
Test Item	
1111	?

artificial example. Imagine that you are trained on the input and output data given in table 3.1. If you are like other people whom I have asked, you would guess that in the test item the output that corresponds to input item [1111] is [1111]. But that is not the only inference that you could draw. For example, in the training data, the rightmost column is always 0: there is no direct evidence that the rightmost column could ever be a 1. So you might decide that the output that corresponds to test item [1111] is [1110]. That inference, too, would be perfectly consistent with the data, yet few if any people would make it. (We see later that some networks do.) One way of describing the inference that people tend to draw is to say that they are generalizing a one-to-one function, such as identity or sameness, universally.

More natural examples can readily be found. For instance, we can form the progressive of any English verb stem—even an unusual-sounding one—by concatenating it with the suffix *-ing,* hence *walk-walking, jump-jumping,* and, in describing what Yeltsin might have done to Gorbachev, *outgorbachev-outgorbacheving.* Similarly, (modulo a set of exceptions) we can apply the *-ed* past-tense formation process equally freely, with *wug-wugged* (Berko, 1958) and *outgorbachev-outgorbacheved* (Marcus, Brinkmann, Clahsen, Wiese & Pinker, 1995; Prasada & Pinker, 1993).

Our processes of sentence formation seem equally flexible and freely generalizable to new cases. For example, we can form a sentence combining any *noun phrase* (say, *the man who climbed up a hill*) with any *verb phrase* (say, *came down the boulevard in chains*).[3] Likewise, our intuitive theories (Carey, 1985; Gopnik & Wellman, 1994; Keil, 1989) seem to consist at least in part of bits of knowledge about the world that can be freely generalized. For example, part of our knowledge about biology is that (other things being equal) when animals bear offspring, the babies are of the same species as their parents (Asplin & Marcus, 1999; Marcus, 1998b). This bit of knowledge can be freely generalized, allowing us, for instance, to infer that the gerenuk (a bovid found in Eastern Africa) gives birth to gerenuks.

Another straightforward instance of a UQOTOM is reduplication. Reduplication, or immediate repetition, is found in pluralization (for example, in Indonesian the plural of *buku* ("book") is *buku-buku*) and even in syntax, as Ghomeshi, Jackendoff, Rosen, and Russell (1999) have recently pointed out, with examples such as, "Are you just shopping, or are you shopping-shopping?" meaning, roughly, "Are you shopping casually or seriously?" (Dear reader, are you just reading this, or are you reading-reading it?) The "opposite" of reduplication (also a UQOTOM), so to speak, is a process that allows anything but reduplication. For example, a constraint of Hebrew word formation is that adjacent consonants in a root must not be identical; Berent and her colleagues (Berent, Everett & Shimron, 2000; Berent & Shimron, 1997) have shown that speakers freely generalize this constraint to novel items.

My own recent research suggests that the ability to freely generalize patterns like reduplication has roots quite early in development. My colleagues and I have found that seven-month-old infants can freely generalize (Marcus, Vijayan, Bandi Rao & Vishton, 1999). In our experiments, infants listened for two minutes to "sentences" from one of two artificial grammars. For instance, some subjects heard sentences constructed from an ABA grammar, such as *ga na ga* and *li ti li,* while others heard sentences constructed from an ABB grammar. After this two-minute *habituation,* infants were exposed to test sentences that were made up entirely of novel words. Half of the test sentences were consistent with the sentences that the infant had heard in the two-minute habituation; half were not. The point was to test whether infants were able to extract some sort of abstract structure from the habituation and to test whether they could freely generalize. To assess this, we measured how long infants looked at flashing lights that were associated with speakers playing test sentences. Based on prior work by Saffran, Aslin, & Newport (1996), we predicted that infants who could distinguish the two grammars and generalize them to new words would attend longer during the inconsistent items. For example, infants that were trained on the ABA grammar should look longer during, say, *wo fe fe* than *wo fe wo.* As predicted, infants looked longer at the inconsistent items, suggesting that infants were indeed sensitive to the abstract structure of the artificial grammar on which they were trained. Because the words in the test sentences and the words in the training sentences were different, our experiments suggest that the infants were able to freely generalize (and that they could do so without explicit instruction).

Additional experiments showed that infants were not relying simply on the presence or absence of immediately reduplicated items: infants could also distinguish between an AAB grammar and an ABB grammar. In principle, infants could have made such a distinction based purely on the last two words, but pilot data that I reported in Marcus (1999) shows that infants are capable of distinguishing grammars such as AAB versus BAB that do not differ in the final two words. Still other experiments, by Gomez and Gerken (1999), point to similar abilities in twelve-month-old infants.

Although I take the evidence for free generalization to be strong, I am not claiming that *every* generalization that we draw is freely generalized across all potential instances in its domain. For example, some of the generalizations that we draw in the area of motor control may be far more restricted. Ghahramani, Wolpert, and Jordan (1996) conducted an adaptation experiment in which subjects used a computer mouse to point to computer-generated visual targets. Subjects received feedback, but only for one or two specific locations; when they pointed outside these locations, they received no feedback. Unbeknownst to the subjects, the feedback (in the one or two designated locations in which it was supplied) was secretly altered. This altered feedback caused subjects to alter their pointing behavior, but rather than compensating equally across the motor space, subjects compensated for the altered visual feedback most strongly in the locations at which they have received feedback. In other words, rather than transferring across the board, the degree to which subjects transferred declined rapidly as a function of the distance from the locations on which they were trained. Rather than learning something that held universally, in this case subjects learned something that seemed to pertain to only a few of its possible inputs. More broadly speaking, in each domain in which there is generalization, it is an empirical question whether the generalization is restricted to items that closely resemble training items or whether the generalization can be freely extended to all novel items within some class.

3.1.2 Free Generalization of UQOTOM in Systems That Can Perform Operations over Variables

To a system that can make use of algebralike operations over variables, free generalization comes naturally. For example, the information that we extracted from table 3.1 could be represented as an expression of the universally quantified, one-to-one identity mapping, $f(\mathbf{x}) = \mathbf{x}$. We could then calculate the output that corresponds to the test item, f(1111) by *substituting* the instance *1111* into the variable **x** on the right-hand side of the equation.

Defined by such a substitution process, an operation over a variable is indifferent as to whether the instantiation of that variable is familiar or unfamiliar.[4] We do not care which examples of that variable we have seen before; the operation over variables can be *freely generalized* to any instantiation.

Looking something up in a table does not count as applying an abstract relationship between variables. For example, if we have a table that tells us that entry 1 corresponds to Adam, 2 to Eve, 3 to Cain, and 4 to Abel, there is no interesting sense in which the computation being performed is a systematic, unbounded operation over variables. Algebraic rules are not finite tables of memorized facts or relationships between specific instances but open-ended relationships that can be freely generalized to all elements within some class.

3.1.3 Implementing Operations over Variables in a Physical System

How might a system that can perform operations over variables be implemented in a physical system? One simple way to do this is to use a set of buckets. One bucket represents the variable **x,** and another bucket represents the variable **y.** The *instantiation* of a given bucket is indicated by the bucket's contents. To set **x** equal to the value of 0.5, we *fill* the bucket representing the variable **x** half way. To copy the contents of variable **x** into variable **y,** we literally pour the contents of **x** into **y.**

A given variable could also be represented by using more than one bucket. For example, if we want variable **x** to represent varying amounts of pocket change, we could use one bucket to represent the number of quarters, another to represent the number of dimes, another to represent the number of nickels, and another to represent the number of pennies. The total amount of currency thus is represented by the four-bucket ensemble. Just as we can define simple universally quantified one-to-one operations such as *copy* in the single-bucket case, we can define simple universally quantified one-to-one operations in the multiple bucket case. The key to doing this is that we must do the same thing *in parallel for each individual bucket.* To copy the contents of variable **x** (represented by four buckets) into the contents of variable **y** (represented by four buckets), we must copy the contents of the **x** bucket that represents the number of quarters into the **y** bucket that represents the number of quarters, and so forth, for the dimes, nickels, and pennies—a strategy that might be described by the Latin phrase *mutatis mutandis,* which is loosely translated as "repeat as necessary, changing what needs to be changed."

This basic insight of *mutatis mutandis* is at the core of how modern digital computers implement operations over variables. Much as in our multiple bucket example, computers represent numerical quantities

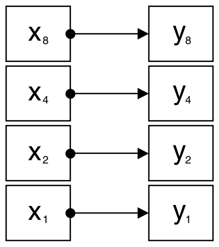

Figure 3.1
A simple circuit for implementing a "copy" operation in a computer that represents variables (here, **x** and **y**) using sets of bits.

and other kinds of information using sets of *binary registers* (sometimes known as *bits*). These binary registers can be thought of as analogous to buckets that are always either full or empty. Operations are defined in parallel over these sets of binary bits. When a programmer issues a command to copy the contents of variable **x** into variable **y,** the computer copies in parallel each of the bits that represents variable **x** into the corresponding bits that represent variable **y,** as depicted in figure 3.1.

3.2 Multilayer Perceptrons and Operations over Variables

The distinction between encoding a variable with a single bucket and encoding a variable with a set of buckets is helpful because the relationship between multilayer perceptrons and operations over variables can be understood in similar terms. In essence, the key question is whether a given input variable in a particular network is encoded using one node or a set of nodes.

For example, consider the encoding schemes used by various models of children's understanding of so-called balance-beam problems. In these problems, a child must predict which side of a balance beam will go down. In these simulations, the input to a model consists of four variables, **number-of-weights-on-the-left-side, distance-of-left-weights-from-fulcrum, number-of-weights-on-the-right-side,** and **distance-of-right-weights-from-fulcrum.** As figure 3.2 illustrates, one option is to allocate one node to each of these variables, with any given variable taking values such as 1.0, 2.0, or 3.0 (Shultz, Mareschal & Schmidt, 1994). Another option is to use a set of nodes for each variable, with each particular node representing some particular number of weights (McClelland, 1989).

This difference—in whether a particular variable is encoded by one node or by many nodes—is *not* the same as the difference between

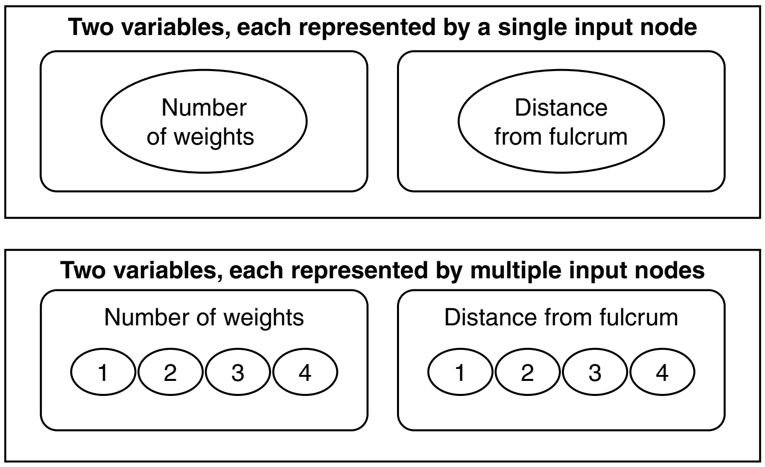
Two variables, each represented by a single input node
Number of weights
Distance from fulcrum
Two variables, each represented by multiple input nodes
Number of weights
1
2
3
4
Distance from fulcrum
1
2
3
4

Figure 3.2
The balance-beam task: two different ways of encoding how many weights are on the left and how far those weights are from the fulcrum on the right. Top panel: An input encoding scheme in which a single node is devoted to the encoding of each variable. Bottom panel: An input encoding scheme in which a set of nodes is devoted to the encoding of each variable. In the top panel, both the number of weights and the distance from the fulcrum are encoded locally. In the bottom panel, both the number of weights and the distance from the fulcrum are encoded in distributed fashion, using banks of nodes. If there were three weights on the left side, the coding scheme depicted in the top panel would activate the number-of-weights scheme to a level of 3.0, while the coding scheme depicted in the bottom panel would activate to a level of 1.0 the 3 node in the bank of weights representing the number of weights. Hidden units and output units are not shown.

localist and distributed representations. While all models that use distributed representations allocate more than one variable per node, it is not the case that all localist models allocate a single node per variable. In fact, most localist models allocate more than one node per variable. Consider Elman's sentence-prediction model. Here, the input to the model is a single variable that we might think of as **current word.** Although any given instantiation of that variable (say, *cat*) will activate only a single node, every input node can potentially indicate an instantiation of the variable current word. For example, the node for *dog* might not be active at this moment, but it might be active during the presentation of another sentence. The sentence-prediction model is thus an example of a localist model that allocates multiple nodes to a single input variable. Again, what is relevant here is not the sheer number of input units but rather the *number of input units allocated to representing each input variable.*

It is also important to draw a distinction between the contrast I am drawing and another often overlapping contrast between analog and binary encoding schemes. As it happens, many models that allocate just one node per variable rely on continuously varying input nodes rather than binary input nodes (*analog encoding*), whereas models that use multiple nodes typically use binary encoding schemes. But it is possible to have an input variable that is represented by a single node that takes on discrete values or by a set of nodes that take on continuous activation values. What is important for present purposes is not whether a node is analog or binary but rather whether a given variable is represented by a single node or many.

3.2.1 Models That Allocate One Node to Each Variable

With this distinction—between representational schemes that allocate one node per variable and representational schemes that allocate more than one node per variable—firmly in mind (and clearly distinguished from the separate question of localist versus distributed encoding), we are now ready to consider the relation between multilayer perceptrons and systems that represent and generalize operations over variables.

As I warned in chapter 1, my conclusions may not be what you expect. I argue neither that multilayer perceptrons cannot represent abstract relationships between variables nor that they must represent abstract relationships between variables. Simple claims like "Multilayer perceptrons cannot represent rules" or "Multilayer perceptrons always represent 'concealed' rules" simply are not correct. The real situation is more complex—in part because it depends on the nature of a given model's input representations.

Models that allocate a single node to each input variable behave very differently from models that allocate more than one node to each input variable. Models that allocate a single node to each input variable are (with some caveats) simpler than models that allocate multiple nodes to each variable. One-node-per-variable models, it turns out, can and indeed (the caveats in note 5 notwithstanding) must represent universally quantified one-to-one mappings.[5] For example, the model illustrated in figure 3.3 can represent—and freely generalize—the identity function if it uses a connection weight of 1.0 (and linear activation function with a slope of one and intercept of zero). With the same activation function but a connection weight of 2.0, the model can represent and freely generalize the function $f(\mathbf{x}) = 2\mathbf{x}$, or any other function of the form $f(\mathbf{x}) = m\mathbf{x} + b$—each of which is a UQOTOM.

From the fact that (caveats aside) such models can represent only UQOTOM and no other functions, it follows directly that all that a learning algorithm can do is choose between one universally-quantified one-

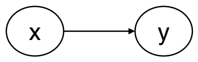

Figure 3.3
A network that uses one node to represent each variable.

to-one mapping and another, such as f(**x**) = **x** versus f(**x**) = 1.5**x,** f(**x**) = 2**x,** and so on. Such models cannot learn arbitrary mappings. (For example, they cannot learn to map an input number that specifies the alphabetical order of a person in a phonebook to an output that specifies that person's telephone number.) As such they provide a candidate hypothesis for how operations over variables can be implemented in a neural substrate and not for a mental architecture that eliminates the representation of abstract relationships between variables.

3.2.2 Models That Allocate More Than One Node per Variable

Models that allocate more than one node per variable too, can represent universally quantified one-to-one mappings (see, for example, the left panel of figure 3.4), but they do not have to (see the right panel of figure 3.4). When such a network represents identity or some other UQOTOM, it represents an abstract relationship between variables—which is to say that such a network implements an *algebraic rule.*

Advocates of multilayer perceptrons might resist the claim that I am making here, for I am claiming that some multilayer perceptrons (such as the one in the left panel) implement—rather than eliminate—algebraic rules. In hindsight, though, my claim should seem obvious, perhaps even banal. After all, a network that implements the identity (that is, "copy") function using a set of connections such as in the left panel has essentially the same wiring diagram as a digital logic chip that implements a copy function.

My remarks so far have been purely about representation, not about generalization. To sum them up, models that allocate a single node to each variable have (putting aside the worries about nonlinear activation functions and arbitrary representational schemes) no choice but to represent abstract relationships between variables, whereas models that allocate multiple nodes to each variable sometimes represent abstract relationships between variables and sometimes do not: what they represent is a function of what their connection weights are. In multiple-nodes-per-variable multilayer perceptrons, some connection weights represent UQOTOM, others represent many-to-one mappings, and still others can represent purely arbitrary mappings.

As such, multilayer perceptrons that allocate more than one node to each variable are quite flexible. One might ask whether this flexibil-

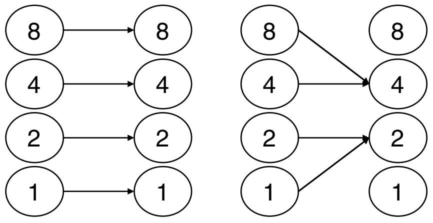

Figure 3.4
UQOTOM and many-to-one mappings in models that represent a single variable with a set of nodes. Left panel: A one-to-one mapping from a single input variable to a single output variable. Right panel: A many-to-one mapping from a single input variable to a single output variable. Only connections with non-zero weights are shown.

ity suggests that multiple-nodes-per-variable multilayer perceptrons are the best way of implementing abstract relationships between variables in a neural-like substrate. What I suggest in the next section is that their flexibility is both an asset and a liability and that the liability is serious enough to motivate a search for alternative ways in which abstract relationships between variables can be implemented in a neural (or neural-like) substrate.

Learning The flexibility in what multiple-nodes-per-variable models can represent leads to a flexibility in what they can learn. Multiple-nodes-per-variable models can learn UQOTOMs, and they can learn arbitrary mappings. But what they learn depends on the nature of the learning algorithm. Back-propagation—the learning algorithm most commonly used—does not allocate special status to UQOTOMs. Instead, a many-nodes-per-variable multilayer perceptron that is trained by back-propagation can learn a UQOTOM—such as identity, multiplication, or concatenation—*only if it sees that UQOTOM illustrated with respect to each possible input and output node.*

For example, the data in table 3.1, as mentioned earlier, might be thought of as illustrating the identity function. But the data do not exemplify *all* possible instances of the identity function. Instead, they illustrate only a systematically restricted *subset* of the instances of the identity function: in every training case the rightmost column in the target output is 1 and is never 0.

If we thought about this in geometric terms, we might call the set of possible inputs the *input space,* the set of inputs on which the model is trained the *training set,* and the area of the input space in which the training set is clustered the *training space.* Inputs with the rightmost column of 0 (whether or not they are in the training set) are in the training space, but inputs with the rightmost column of 1 are *outside the training*

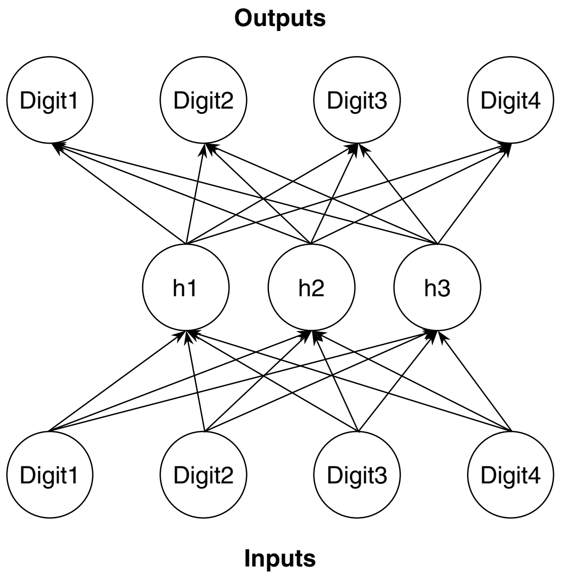

Figure 3.5
A multilayer network with distributed input and output representations.

space. (If we construe the inputs in table 3.1 as binary numbers, the even numbers lie inside the training space, and the odd numbers lie outside the training space.)

Many-nodes-per-variable multilayer perceptrons that are trained by back-propagation can generalize one-to-one mappings within the training space, but assuming that the inputs are binary (such as 0 or 1, –1 or +1, +voiced or –voiced, +cat or –cat, and so on), they cannot generalize one-to-one mappings outside the training space.[6]

For example, in a recent series of simulations, I found that if the simple network illustrated in figure 3.5 is trained only on inputs with a rightmost digit of 0, it will not generalize identity to inputs with a rightmost digit of 1 (Marcus, 1998c). Instead, whether the rightmost digit is a 1 or a 0, the model always returns an output in which the rightmost digit is 0. For example, given the input 1111, the model generally returns 1110, an inference that is mathematically justifiable but plainly different from what humans typically do.

Put informally, the network has no way to tell that all four columns should be treated uniformly. People may not always treat the columns uniformly, but certainly under some conditions they can, and these conditions pose difficulties for the many-nodes-per-variable models that are trained by the back-propagation learning algorithm.

Training independence A bit more formally, we can say that many-nodes-per-variable multilayer perceptrons that are trained by back-propagation cannot generalize one-to-one mappings between nodes. This is because the learning that results from back-propagation is, in an important sense, *local.* As McClelland and Rumelhart (1986, p. 214) put it, these models "change the connection between one unit and another based on information that is locally available to [a given] connection." This localism has the consequence that if a model is exposed to a simple UQOTOM relationship (such as identity) for some subset of the inputs that leaves some nodes untrained, it will not generalize that UQOTOM function to the remaining nodes.

The fact that a multiple-nodes-per-variable multilayer perceptron cannot generalize a UQOTOM function to a node that lies outside the training space follows from the equations that define back-propagation. The equations lead to two properties that I call *input independence* and *output independence* or, collectively, *training independence* (Marcus, 1998c). Input independence is about how the connections that emanate from input nodes are trained. First, when an input node is always off (that is, set to 0), the connections that emanate from it will never change. This is because the term in the equation that determines the size of the weight change for a given connection from input node **x** into the rest of the network is always multiplied by the activation of input node **x**; if the activation of input node **x** is 0, the connection weight does not change. In this way, what happens to the connections that emanate from an input node that is never turned on is *independent* of what happens to connections that emanate from other input nodes. (If the input node never varies but is always set to some value *v* other than 0, the mathematics becomes more complex, but it appears to be true empirically that in such cases the model does not learn anything about the relation between that input node and the output, other than to always set the output node to value *v.*)

Output independence is about the connections that feed into the output units. The equations that adjust the weights feeding an output unit *j* depend on the difference between the observed output for unit *j* and the target output for unit *j* but *not on the observed values or target values of any other unit.* Thus the way the network adjusts the weights feeding output node *j* must be independent of the way the network adjusts the weights feeding output node *k* (assuming that nodes *j* and *k* are distinct). This means not that there is never any dependence between output nodes but that the only source of dependence between them is their common influence on the hidden nodes, which turns out not to be enough. At best, the mutual influence of output nodes on input-to-hidden-layer connections may under some circumstances lead to felicitous encodings of the hidden nodes. We might think of such hidden units as *quasi-input nodes.* The crucial point is that no matter how felicitous the choice of quasi-input units may be, the network must always *learn* the mapping between these quasi-input nodes and the output nodes. Since this latter step is done independently, the mutual influence of the output nodes on input-to-hidden-layer connections is not sufficient to allow the network to generalize a UQOTOM between nodes.

Training independence leads other standard connectionist learning algorithms to behave in similar ways. For example, the Hebbian rule ensures that any weight that comes from an input unit that is set to 0 will not change, since the weight change is calculated by multiplying the input unit's activation by the output unit's activations times some constant. Again, multiplying by 0 guarantees that no learning will take place. Likewise, when the Hebbian algorithm adjusts the weights feeding into some output node j, the activations for all nodes $k \neq j$ are irrelevant, and hence multiple-nodes-per-variable perceptrons that are trained by the Hebbian algorithm do not generalize UQOTOM between nodes.

Extending a new function to a node that has already been trained Training independence does not limit just the ability of networks to generalize to nodes that were never used, but also the ability of networks to generalize between what we might call *known nodes*—nodes in which both feature values have appeared in the input. For example, consider the model shown in figure 3.6. I trained this network to do two different things. If the rightmost node was activated, the model was to copy the remainder of the input; if the rightmost node was not activated, the model was to invert the remainder of the input (that is, turn each 1 into a 0 and each 0 into a 1, such as 1110 into 0001).

I trained this network on inversion for all 16 possible inputs and then trained it on identity just for the numbers in which digit 4 equaled 0. As before, the network was unable to generalize to 1111, despite having had ample experience with the digit 4 input node in the inversion function. *The problem of transferring from node to node is not restricted to untrained nodes:* networks trained with localist algorithms such as backpropagation never transfer UQOTOM between nodes.

Training independence, mathematics, and modeling Let me stress that there is no flaw in the training algorithm itself. What these localist learning algorithms do is not a mathematical aberration, but rather an induction that is perfectly well licensed by the training data. For example, given the training data, the conditional probability that the rightmost digit would be a 1 is exactly 0. The model thus extends a conditional probability in a way that is mathematically sound.

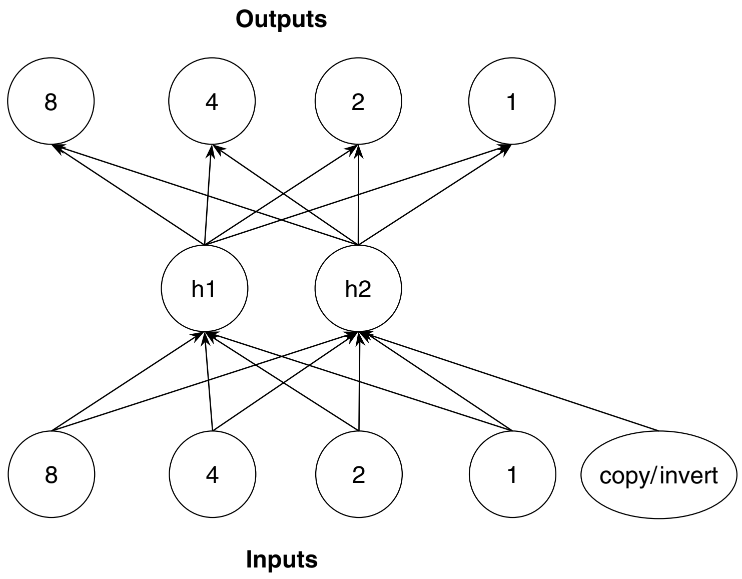

Figure 3.6
A model that performs either the "copy" function or the "invert" function, depending on the activity of the rightmost input node.

If there were no cases in which organisms could freely generalize on the basis of limited input, training independence might not be a problem. In tasks in which subjects cannot freely generalize, a model that does its training independently may actually be preferred over a model that can learn only relationships that apply to all instances of a class. A localist algorithm in which there is training independence is a liability only *if it is used to capture phenomena in which an organism can freely generalize.* In cases where organisms cannot freely generalize, it is possible that localist algorithms may be appropriate.

But in some cases it appears that humans can freely generalize from restricted data, and in these cases many-nodes-per-variable multilayer perceptrons that are trained by back-propagation are inappropriate. This fact is worth pointing out because the literature on connectionist models of cognitive science is filled with multiple-nodes-per-variable multilayer perceptron models that are trained by back-propagation, and many of those models are aimed at accounting for aspects of mental life in which humans *do* appear to be able to freely generalize from incomplete input data. For example, Hinton's family-tree model (described in chapter 2) tried to learn abstract relations like *sibling of.* It seems quite clear that humans can freely generalize such relations. A human knows that if Dweezil is the sibling of Moon, Moon must be the sibling of Dweezil. But the symmetry of such relationship is lost on Hinton's family-tree model: each new person must be represented by a new node, each new node is treated independently, and hence the network does not infer that Moon must be the sibling of Dweezil. (In Hinton's discussion of the family-tree model, the problem of generalizing outside the training space is not addressed. Hinton's tests of the model were always within the training space—tests of whether the model could infer some fact about a family member about which many facts were already known. Cases such as the Dweezil-Moon example were never tested.)

Similarly, Elman's sentence-prediction model seems to be aimed squarely at cases in which humans can freely generalize—at accounting for how we acquire syntactic relationships between categories. To illustrate one way in which training independence would undermine the sentence-prediction model, in Marcus (1998c) I reported a series of simulations in which I trained the sentence-prediction model on sentences such as *a rose is a rose, a lily is a lily,* and *a tulip is a tulip.* Humans would predict that the continuation to the sentence fragment *a blicket is a* _____ is *blicket,* but my simulations showed that Elman's network (assuming that each word is represented by a separate node) does not. (Once again, the issue is not about new nodes per se but about generalizing a UQOTOM between nodes. In a follow-up to that experiment I showed that pretraining the simple recurrent network on sentences such as *the bee sniffs the blicket* and *the bee sniffs the rose* did not help the network infer that the continuation to a *blicket is a* _____ is *blicket.*)

In a reply, Elman (1998) obscured the issues, by showing one way in which the sentence-prediction network could generalize a function that was not one-to-one within the training space. But showing that the sentence-prediction network could generalize a function that was *not* one-to-one does not bear on my point that such a network cannot generalize (outside the training space) functions that are one-to-one. The bottom line is that humans can freely generalize one-to-one mappings but that multilayer perceptron models that allocate multiple nodes per variable and are trained with localist learning algorithms cannot. For these cases, we must seek alternative models.

3.3 Alternative Ways of Representing Bindings between Variables and Instances

Cases in which humans can freely generalize UQOTOM on the basis of restricted data are problematic for multiple-nodes-per-variable multilayer perceptrons trained by back-propagation. But this does not mean that no model of any sort could capture free generalization from restricted data.

In general, what is required is a system that has five properties. First, the system must have a way to distinguish variables from instances, analogous to the way mathematics textbooks set variables in italic type (x) and constants in bold type (**AB**). Second, the system must have a way to represent abstract relationships between variables, analogous to an equation like $\mathbf{y} = \mathbf{x} + 2$. Third, the system must have a way to bind a particular instance to a given variable, just as the variable x may be assigned the value 7. Fourth, the system must have a way to apply operations to arbitrary instances of variables—for example, an addition operation must be able to take any two numbers as input, a copying operation must be able to copy any input, or a concatenation operation must be able to combine any two inputs. Finally, the system must have a way to extract relationships between variables on the basis of training examples.

3.3.1 Variable Binding Using Nodes and Activation Values in a Multilayer Perceptron

We have already seen one simple model that meets these five criteria: a model in which a single input node connects to a single output node (with a linear activation function). In this model, variables are represented distinctly from instances: the nodes represent variables, and the activation values indicate instances. The connection weight indicates the relation between the variables (for example, it is 1.0 if the output variable always equals the input variable). Bindings are indicated by the activation values. The structure of the network guarantees that all instances of a variable will be treated in the same way. The learning algorithm (either back-propagation or the Hebbian algorithm will work) is constrained such that all it can do is change that single connection weight; each possible (changed) value of the connection weight simply indicates a different relationship between variables. Consequently, such a model can freely generalize the identity relationship on the basis of a very small number of training examples.

Still, although a one-node-per-variable system can readily represent functions such as identity or multiplication, such a system cannot so easily represent many other important one-to-one mappings. For example, it is not obvious how one would implement an operation that combines a verb with its suffix or an operation that adjoins one part of a syntactic tree with another. Because the range of operations that one might represent seems fairly limited, it is worth considering alternatives.

What about the more complex model in which variables are represented by sets of nodes? Instances are again represented by activation values; the difference is that only some sets of connection weights implement *operations that apply uniformly to all possible instances.* Taken in conjunction with a learning algorithm such as back-propagation, this is not a good thing, for as we saw, UQOTOM are not generalized outside the training space. But this does not mean that one could not use a different kind of learning algorithm. Goldrick, Hale, Mathis, and Smolensky (1999) are working on developing learning algorithms that relax the assumption of localism that leads to training dependence. It is too early to fully evaluate their approach, but it clearly merits further study. Should they succeed, an important open question will be whether the resulting learning algorithm is one that provides an alternative to operations over variables or an implementation thereof.

3.3.2 Conjunctive Coding

In multilayer perceptrons, the current instantiation of a given variable is indicated by a pattern of activity. There are a number of other possible ways to indicate the binding between a variable and its current instance. One possibility is to devote specific nodes to particular combinations of a variable and an instance. For example, node A might be activated if and only if the subject of some sentence is *John,* node B might be activated if and only if the subject of that sentence is *Mary,* and node C might be activated if and only if the object of some sentence is *John.* This sort of system provides a way of temporarily binding variables and instances but is not by itself a way of implementing operations over variables. For that, some additional mechanisms are required.

It seems likely that conjunctive coding plays some role in our mental life. For example, experiments with single-cell recordings by Goldman-Rakic and others (Funashi, Chafee & Goldman-Rakic, 1993) have indicated that certain neurons are most strongly activated when a particular object appears in a particular position. It does not seem unreasonable to assume that these neurons conjunctively encode combinations of objects in particular positions.

But the brain must rely on other techniques for variable binding as well. Conjunctive codes do not naturally allow for the representation of binding between a variable and a novel instance. The fact that *Dweezil is the agent of loving* can be represented only if there is a node that stands for **agent-of-loving-is-Dweezil.** It seems implausible to suppose that all necessary nodes are prespecified, yet it also seems problematic to think that there would be a mechanism that could manufacture arbitrary conjunctive nodes on the fly. Moreover, conjunctive encoding schemes may require an unrealistically large number of nodes, proportional to the number of variables times the number of possible instances. (As I show

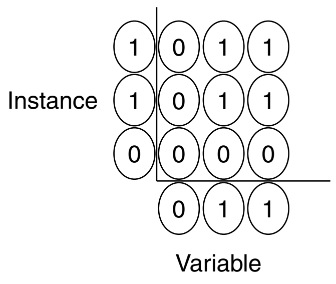

Figure 3.7
A tensor product representation of a binding between a variable (**agent**) and an instance (*John*).

in chapter 4, this becomes especially worrisome if the instances can be complex elements such as *the boy on the corner.*)

3.3.3 Tensor Products

A more general, more powerful way of doing conjunctive binding is the *tensor product* proposed by Smolensky (1990). A tensor product is a way of representing a binding between a variable and an instance. A tensor product is not (by itself) a way of representing a relationship between variables or a way of applying operations to variables. Further machinery would be required to represent or extend relationships between variables. I do not discuss such machinery here but instead focus only on how tensor products represent bindings between variables and instances.

In the tensor product approach, each possible instance and each possible variable is represented by a vector. A particular binding between a particular variable and a particular instance is represented by applying a process analogous to multiplication. The resulting combination, a *tensor product,* is a vector of greater dimensionality.

To illustrate how the model might encode the binding between the variable **agent** and the instance *John,* let us suppose that John is represented by the vector 110 and **agent** by the vector 011. Figure 3.7 illustrates the encoding of John on the y-axis and the encoding of **agent** on the *x*-axis. The resulting tensor product that represents their binding (corresponding to the 3×3 set of nodes in the top right corner of the figure) would be the two-dimensional vector

$$\begin{array}{ccc} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{array}$$

One way in which tensor products differ from the simple conjunctive scheme (described in the previous section) is in the role of a given node. In the simple conjunctive scheme, each node is dedicated to the representation of a single particular binding (for example, one node is turned on only if John is the agent, another if Peter is the agent, and so forth). In contrast, in the tensor product scheme, every node participates in every binding.

The tensor product scheme has at least two important advantages over the simple conjunctive scheme. First, it is potentially more efficient. The simple conjunctive scheme requires $i*v$ nodes, where i is the number of instances and v is the number of variables. The tensor product scheme requires $a*b$ nodes, where a is the length of the vector encoding the instance and b the length of the vector encoding the variable. If there are, say, 128 possible instances and 4 possible variables, the tensor product scheme is considerably more efficient, requiring $7 + 2 + 14 = 23$ nodes, 7 nodes to represent the instance, 2 to represent the variable, and 14 to represent any possible combination of the two. The simple conjunctive scheme requires $128 * 4 = 512$ nodes. Second, the tensor product scheme can more readily cope with the addition of new possible instances. Assuming that the new instance can simply be assigned a new vector, representing a binding containing that instance is simply a matter of plugging a new vector into the preexisting tensor product machinery. Nonetheless, despite these advantages, I suggest in chapter 4 that tensor products are not plausible as an account of how we represent recursively structured representations.

3.3.4 Registers

A limitation of the binding schemes discussed so far is that none provides a way of *storing* a binding. The bindings that are created are all entirely transitory, commonly taken to be constructed as the consequence of some current input to the system. One also needs a way to encode more permanently bindings between variables and instances.

One way to do this is to use devices that have two or more stable states. For example, digital computers often make use of flip-flops—binary or *bistable* devices that can be set to either on or off and then maintained in that state without further input. (Registers need not be bistable; they need only have more than one stable state. For example, mechanical cash registers use memory elements with 10 stable states each (0, 1, 2, . . . 9)—each memory element for the number of pennies, one for the number of tens of pennies, one for the number of dollars, one for the number of tens of dollars, and so on. If registers are used in the human brain, they might be bistable, like those in a digital computer, but might not be; I am not aware of any evidence that directly bears on this question.)

Registers are central to digital computers; my suggestion is that registers are central to human cognition as well. There are several ways in which stable but rapidly updatable registers could be constructed in a neural substrate. For example, Trehub (1991) proposed that autaptic cells—cells that feed back into themselves—could serve effectively as rapidly updatable bistable devices. This idea has its origins in Hebb's (1949) notion of a "cell-assembly." A related proposal comes from Calvin (1996), who proposed a set of hexagonal self-excitatory cell assemblies that could serve as registers.

Along these lines, it should be clear that although multilayer perceptrons do not directly provide for registers, it is an easy matter to construct bistable registers out of nodes and connections. All that is really required is a single node that feeds back into itself. As Elman et al. (1996, p. 235) showed, with the right connection weight, a single node that feeds back into itself becomes a bistable device. If the input is 0, the output tends to go to 0; if the input is 1.0, the output tends to go to 1.0. If the input is 0.5, which we can think of as the absence of a write-to-memory operation, the output tends to remain unchanged. Once the input is taken away, the model tends to remain stable at one or another *attractor point* (0.0 or 1.0). The model then holds stable at the attractor point, just like a flip-flop. The key here is to use the self-feeding node as a memory component within a more structured network. Although one can use a simple node connected to itself as part of a more complex system that performs operations over variables, standard multilayer perceptrons do not make a distinction between components for processing and components for memory.

Although it is often assumed that knowledge is stored in terms of changes in between-cell (synaptic) connection weights, it is logically possible that knowledge is stored within cells. A given neuron could, for example, store values internally by modulating cell-internal gene expression. We know, for example, that cells have the sort of memory that indicates their type (Rensberger, 1996); when a cell divides, its type of memory is generally inherited by its offspring. These mechanisms, or other mechanisms such as the reciprocal modulation of ion channels (Holmes, 1998), could provide an intracellular basis for registers.

Registers, however they are implemented, can provide a basis not only for variable binding but also, more generally, for the kinds of memory in which we learn things on a single trial. Such rapidly updatable memory clearly plays an important role throughout our mental life. A typical example comes from Jackendoff and Mataric (1997, p. 12):

> Coming to work in the morning, I park my car in parking lot E instead of parking lot L, where I usually park. At the end of the day, if I am inattentive, I may head for lot L. But if I quickly think "Where did I park my car?" I remember, and head correctly for lot E. . . . Here, despite the fact that an association of my car with lot L is well trained into me, I remember to go to lot E on the basis of one occurrence.

Whatever rapidly updatable neural circuitry supports these kinds of everyday experiences could also be used to support registers that store instances of variables.[7]

3.3.5 Temporal Synchrony

Although I personally suspect that (at least some) registers will be defined in terms of physically isolable parts of the brain (cells, circuits, or subcell assemblies), several other possibilities have been proposed in the literature. Most prominent among these alternative possibilities is *temporal synchrony* (also known as *dynamic binding*) (Hummel & Biederman, 1992; Hummel & Holyoak, 1993; Konen & von der Malsburg, 1993; Shastri & Ajjanagadde, 1993), which we can think of as a framework for representing registers in time rather than in space.

In the temporal synchrony framework, both instances and variables are represented by nodes. Each of these nodes oscillates on and off over time. A variable is considered to be bound to its instance if both fire in the same rhythmic phase. For example, suppose we want to bind (the instance) **Sam** to the role (variable) **action-of-selling.** As sketched in figure 3.8, nodes for the variable **agent-of-selling** and the instance **Sam** oscillate simultaneously in a rhythmic cycle. (Meanwhile, **book** and **object-of-selling** also resonate together but in a different phase than **Sam** and **agent-of-selling.**)

Temporal synchrony is, by itself, simply a way of representing bindings between variables and instances and is not a way of performing operations over those instances. Fortunately, it is possible to build mechanisms that operate over those bindings. For example, Holyoak and Hummel (2000) have shown that an analogical reasoning system that uses temporal synchrony to represent variable bindings can generalize the identity task described earlier in this chapter. Similarly, Shastri and his colleagues (Mani & Shastri, 1993; Shastri & Ajjanagadde, 1993) have shown how temporal synchrony can play a role in rapid inference (and, as we see later in this chapter, Shastri and Chang, 1999, have shown how temporal synchrony can play a role in a simulation of the Marcus et al., 1999, infant results).

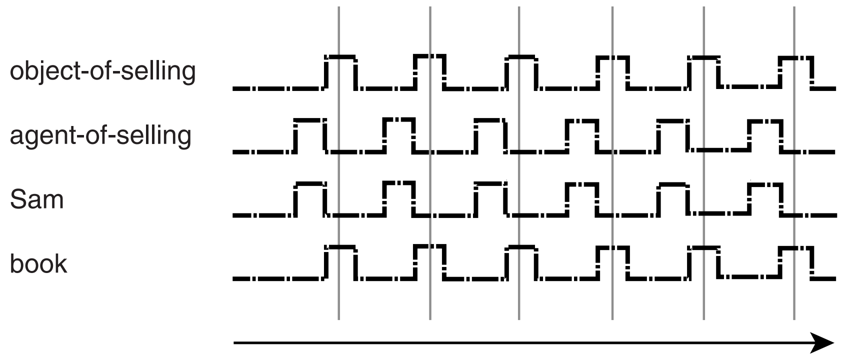

Figure 3.8
Illustration of the representation of *Sam sold a book* in the temporal synchrony framework. The x-axis indicates time. A variable (**agent-of-selling**) and an instance (**Sam**) are considered to be bound if and only if they oscillate in synchrony. In this illustration, **book** and **object-of-selling** are in synchrony, as are **Sam** and **agent-of-selling.** Thin gray vertical bars indicate the synchrony of peaks in the oscillations that corresponding to **object-of-selling** and **book.**

These proposals are motivated by the suggestions of neuroscientists such as von der Malsburg (1981) and Singer et al. (1997) that the synchronization of the activity of neurons may play an important role in neural computation. (Not everybody agrees that synchronization plays an important role; for some skeptical views, see some of the *Behavioral and Brain Science* commentaries that appear with Shastri and Ajjanagadde, 1993.)

My own view is that temporal synchrony might well play a role in some aspects of vision, such as grouping of parts of objects, but I have doubts about whether it plays as important a role in cognition and language. One potential limitation to the temporal synchrony framework is that such a system is likely to be able to keep distinct only a small finite set of phases, typically estimated as less than 10. Hence such a system can simultaneously represent only a small set of bindings. Of course, with respect to *short-term memory,* the number-of-distinct-phases limitation could turn out to be a virtue. Shastri and Ajjanagadde (1993) have suggested that the limitation on the number of phases can capture limits in rapid reasoning (but see Lange and Dyer, 1996), while Hummel and Holyoak (1997) have suggested that the limitation on phases can help to account for some phenomena in our computation of analogy. But it is plain that as a means for representing *long-term* bindings between variables and their instantiations, temporal synchrony is inadequate. We probably can represent millions of bindings (such as facts about who did what to whom) in long-term memory, yet on nobody's account can the brain keep distinct millions of phases. Another limitation, which I take up in chapter 4, concerns the representation of complex structure.

3.3.6 Discussion

In this section, I have suggested that a system of registers, implemented either intracellularly or intercellularly, can serve as a substrate for representing variable bindings. But even if I am right, and even if we knew what kind of neural substrate supported registers, we would be far from understanding how relationships between abstract variables are represented and generalized.

Variables are one part of the story, operations over those variables another. To clarify the difference, consider the distinction in digital computers between registers and *instructions.* Registers store values; instructions, such as "copy" and "compare," manipulate those values. My hunch is that the brain contains a similar stock of basic instructions, each defined to operate over all possible values of registers.

Even if my hunch is right, and even if we could identify exactly what is in the brain's basic set of mental instructions, an important open question would be about how those instructions (in other words, operations over variables) are combined. Digital computers depend on programmers who specify which instructions to use to complete a task. Their job is often made easier by using a *compiler* or *interpreter* that translates a high-level description in a programming language such as C++ or Java into the *machine language* description of what to do in terms of the instructions that are built into the microprocessor.

In a few cases, the mind may depend on something vaguely analogous, inasmuch as we can (unconsciously) translate high-level descriptions such as *repeat each word that I say* or *clap your hands when I say a word that has the letter* e *in it* into some sort of brain-usable format (Hadley, 1998). But in some cases we manage to extract a relationship between variables on the basis of training examples, without being given an explicit high-level description. Either way, when we learn a new function, we are presumably choosing among ways of combining some set of elementary instructions.

In any case, even though we are a long way from being able to say what the basic instructions might be and further from being able to say how they are combined, we are in a position to begin. What I have shown thus far in this chapter is that the fusion of vector coding and localist training algorithms is not enough to account for free generalizations. Whether or not you are satisfied with the register-based alternative that I have advocated, I hope to have persuaded you that the question is not *whether* the mind performs operations over variables but *how.*

3.4 *Case Study 1: Artificial Grammars in Infancy*

To further illustrate the importance of systems that can represent abstract relations between variables, in the remainder of this chapter I consider two domains in which a large number of connectionist models have been proposed. The proliferation of models in these domains allows us to consider what architectural properties of particular models are and are not crucial to their operation.

The first case study comes from the *ga ti ga* infant experiments that I described in section 3.1. In less than a year since these results were published, at least nine distinct models have been proposed. Before I compare these models, I want to make clear that my colleagues and I were not arguing against *all possible* neural network models. Although researchers such as Shultz (1999) have wrongly attributed to us the claim that "neural network models are unable to account for data on infant habituation," we meant no such thing. Instead, as we said in our original report, our goal was "not to deny the importance of neural networks" but rather "to try to characterize what properties the right sort of neural network architecture must have" (Marcus, Vijayan, Bandi Rao & Vishton, 1999, p. 80).

In fact, there are many ways of trying to capture our results in a neural network substrate. The issue is whether the right kind of neural network is one that implements variables, instances, and operations over variables. This issue turns out to be complex because not every author that has described a model that incorporates variables, instances, and operations over variables has done so explicitly. Let us turn now to the models and try to understand how they work.[8]

3.4.1 *Models That Do Not Incorporate Operations over Variables*

A simple recurrent network The first model is a nonmodel, a demonstration that I myself conducted. I simply took Elman's sentence-prediction network and showed that it could not (unchanged) capture our infant results. In keeping with the general strategy adopted by Elman, I set up the infant task as a *prediction task.* That is, during training, the model was given "sentences" one word at a time, with the target at a given point being the next word in that sentence. For example, given the sentence fragment *ga ta,* in the ABA condition the model's target would be *ga,* whereas in the ABB condition the model's target would be *ta.* The test of the model's success was to see whether it could predict proper continuations to novel sentence fragments such as *wo fe* (for example, the target was *wo* in the ABA condition).

What I found—that the model was not able to predict the proper continuations—should not be surprising, given the discussion of training independence earlier in this chapter. Following Elman's standard practice, each novel word was represented by a new node. Since the sentence-prediction network does not generalize between nodes (putting aside the caveat about hidden units described earlier), the model could not predict how a sentence fragment should be continued. Because the model's inability to capture the infant data is due to the underlying training independence, it follows that the simple recurrent network would not be able to capture the infant results regardless of what the learning rate was, regardless of how many hidden nodes there were, and regardless of how many hidden layers were present.

One might ask, though, whether distributed representations (those in which each node represents not a word but a part of a word) could solve this problem. Indeed, when I first described the problems of training independence and how they undermined certain kinds of connectionist models, a common response was to suggest that the problems could be remedied by using distributed representations. For example, in a response to an earlier discussion of mine, Elman (1998, p. 7) wrote that "localist representations are useful but not necessary to the connectionist models Marcus is concerned about," implying that distributed representations might allow networks to overcome the problems of training independence.

But distributed representations are not without costs. Models that make use of distributed representations can be subject to a problem known as the *superposition catastrophe* (Hummel & Holyoak, 1993; von der Malsburg, 1981). This term refers to what happens when one tries to represent multiple entities simultaneously with the same set of resources. To take a simple example, suppose that we represented *a* as [1010], *b* as [1100], *c* as [0011], and *d* as [0101]. Given such a representational scheme, a single set of units would be unable to represent unambiguously the simultaneous activation of *a* and *d* because the combination of the two [1111] would also be the combination of *b* and *c.* As Gaskell (1996, p. 286) observes, the consequence is that "distributed systems cannot implement localist activation models literally."

The superposition catastrophe matters with respect to the sentence-prediction network because the goal of the network is to represent a *set* of possible continuations, and the network needs to be able to do so unambiguously. For example, if the model is trained on the sentences *cats chase mice, cats chase dogs,* and *cats chase cats,* the optimal response to the sentence fragment *cats chase* is to activate simultaneously *mice, dogs,* and *cats.* If the output representations are localist, a network needs only to activate simultaneously the relevant nodes. But if the output representations are genuinely distributed (with nouns and verbs truly overlapping), it becomes much more difficult to activate all and only the nouns.

After all, by hypothesis, the resources that represent nouns would overlap with the resources that represent verbs. For example, if the distributed representations encoded phonology, activating all the sounds that occur in nouns would be tantamount to activating all the sounds that occur in verbs. A model that represented words by phonological distributed representations would therefore be unable in general to keep nouns and verbs distinct.

The same holds even for far more arbitrary distributed representation. For example, in an unpublished (but widely circulated) technical report, Elman (1988) tested a version of the simple recurrent network that—in contrast to the later published versions—did use distributed output representations. Each word was assigned a random 10-bit binary vector. For example, each instance of the word *woman* was assigned the code [0011100101], each instance of the word *cat* was assigned the code [0101110111], each instance of the word *break* was assigned the code [0111001010], and each instance of the word *smell* was assigned the code [1111001100]. Because the representations of different words overlap, it was not possible for the model to unambiguously represent all and only the possible continuations to a given string—regardless of what computations the model performed. The best that the model could do was to guess that the continuation would be the *average* of all the nouns, but if patterns are truly assigned randomly, that average is just as likely to correspond to some particular noun as it is to correspond to some verb. (Indeed, since the codes for words are randomly assigned, it is a consequence of the laws of probability that as the size of the vocabulary increases, the average of the nouns and the average of the verbs would tend to become indistinguishable.) The practical consequence is that the output nodes of the sentence-prediction network could not distinguish between nouns and verbs if it used random output representations. Elman (1988, p. 17) reported that the distributed-output network's "performance at the end of training, measured in terms of performance, was not very good." At the end of five passes through 10,000 sentences, "the network was still making many mistakes."

The superposition catastrophe also renders Hinton's family-tree model incompatible with distributed output representations. Consider a statement such as *Penny is the mother of X.* The response for X is *Arthur AND Victoria.* In the localist output version of the family tree, the model simply needs to activate simultaneously both the **Arthur** node and the **Victoria** node. In a distributed output model, there is no suitable target. Imagine, for instance, that Arthur is encoded by activating only input nodes 1 and 2, Victoria by nodes 3 and 4, Penny by nodes 1 and 3, and Mike by nodes 2 and 4. To indicate that Arthur and Victoria are both sons of Penny, this distributed output version of the family-tree model needs to activate nodes 1, 2, 3, and 4: exactly the same set of nodes as it uses to indicate Penny and Mike.

In any case, distributed representations can be of help only if the items to which one must generalize have only those contrasts which the model learned. We designed the second and third experiments of our infant learning study so that a model that encodes inputs by means of binary phonetic features (+/– voiced, +/– nasal, and so forth) is unable to capture our results. For example, the test words vary in the feature of voicing (e.g., if the A word is voiced, the B word is unvoiced), but the habituation words are all voiced and thus provide no direct information about the relation between voiced and unvoiced consonants. As I confirmed in further simulations with the sentence-prediction network, changing from locally encoded inputs to phonetically encoded inputs has no effect. (Further details about the simulations I conducted using the sentence-prediction network are provided on my web site at http://psych.nyu.edu/gary/science/es.html.)

Although there is no way for the sentence-prediction network to correctly predict the right continuations for the test items, Christiansen and Curtin (1999) have claimed that a slight variant on the sentence-prediction network can capture our data. Their model is essentially the same as the phonetically encoded sentence-prediction network, but it has an additional word boundary unit. The basis for their claim that they can model our data is that in the test phase their model predicts word boundaries better during presentations of inconsistent items than during presentations of consistent items—a pattern that could (in tandem with a further assumption that infants look longer when it is easier to find word boundaries) explain our results. But Christiansen and Curtin are forced to assume (implicitly) that infants can discern word boundaries in the test items but not in the habituation items. This entirely unmotivated assumption makes little sense, since the gaps between words were identical (250 ms) in both habituation and test. Furthermore, Christiansen and Curtin offer no account of why the model should show the particular preference that it does: Why should grammaticality correlate *negatively* with segmentability? The result may in fact be nothing more than noise. Christensen and Curtin provided no statistical tests of their main result.[9]

A simple recurrent network with "weight-freezing" A slight variation of the simple recurrent network, proposed by Altmann and Dienes (1999), comes closer to robustly capturing our results. The model is illustrated in figure 3.9. In many ways, this model is like a standard sentence-prediction network. It shares roughly the same architecture and shares the assumption that sentences are input in a word-by-word fashion, with

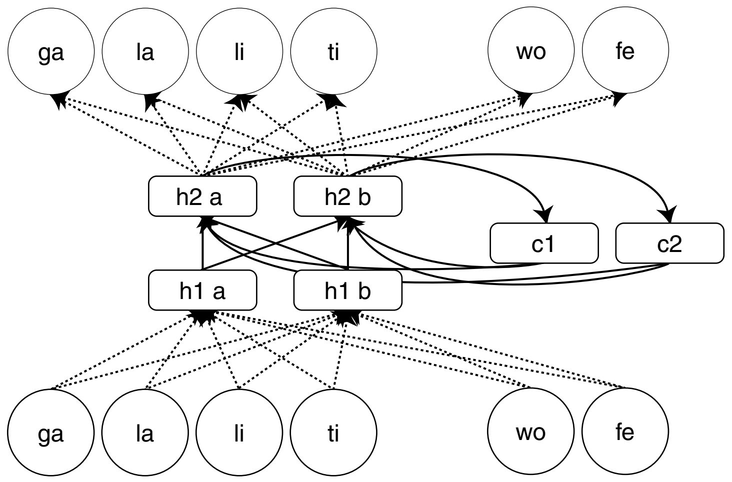

Figure 3.9
Altmann and Dienes's (1999) variant on the simple recurrent network. This model includes an additional hidden layer and a mechanism that during training selectively freezes the weights of the connections that run between hidden layers and from the hidden layer to the context units. In the habituation period all connections can vary freely except those from h2 to c1 (which are fixed at 1.0). In the test phase the only connections that can vary are those drawn in dotted lines.

the target always being the next word in a sentence. What might not be obvious from Altmann and Dienes's discussion of the model is that—like the standard sentence-prediction network—the Altmann-Dienes network is not actually able to predict on the first trial how a given test fragment should be continued. Instead, Altmann and Dienes base their claim that the model can capture our infant results on a kind of savings effect. *Savings* is a term psychologists use to describe an advantage to learning a second set of items given training on a first set. Altmann and Dienes showed that there is greater savings in learning *consistent* test items than in learning *inconsistent* test items. For example, an Altmann-Dienes network that is trained on sentences like *ga ta ga* learns the new sentence *wo fe wo* faster than it learns the new sentence *wo fe fe.*

It is not entirely clear why the model should show such a savings effect, but based on some pilot testing I believe that the savings effect is robust and that it probably stems from the two key differences between this model and the original simple recurrent network: an additional layer of hidden units and an external device that "freezes" the weights between the two hidden layers during testing.

The additional hidden layer means that rather than learning about relationships between input units, the Altmann-Dienes model learns about relationships between *hidden unit encodings of the inputs.* In other words, the first hidden layer in the Altmann-Dienes model (that is, the one closer to the input nodes) bears the same relationship to the second hidden layer as the input nodes in a standard simple recurrent network bear to the (only) hidden layer in that type of network. The consequence is that the layer that feeds into the output units learns not about relationships between input units but about relationships between hidden unit *recodings* of the input units.

By itself, the additional hidden layer makes no difference: training independence still applies. But the additional layer is combined with the novel weight-freezing mechanism, and the combination of the two seems to make it easier to learn consistent items than inconsistent items. If the test items are consistent with the habituation items, the model can acquire a new test sentence by forcing the set of test words to elicit patterns of hidden unit activity that correspond to the patterns elicited by the original set of input words. Since the model already "knows" how to deal with those encodings, learning is relatively efficient. In contrast, if a given test item is inconsistent with the habituation sentences, the model must learn both new encodings and new relationships between those encodings. This process is impaired by the freezing of the connections from hidden layer 1 to hidden layer 2, and so there is an advantage to learning consistent items.

But while the Altmann-Dienes (1999) model (arguably)[10] captures the empirical results reported in Marcus, Vijayan, Bandi Rao, and Vishton (1999), the model does not fully capture the spirit of the Marcus et al. results: it does not truly derive a UQOTOM. For example, in simulations I found that once the Altmann-Dienes model was trained on *la ta la,* it predicted that a child would look longer at a consistent item such as *ta la ta* than at an inconsistent item *ta la la,* apparently because the model has learned to predict that *la* is a likely third word. Children would, I suspect, do the opposite.

Although I have not tested that particular prediction, Shoba Bandi Rao and I tested a similar one, also derived from the model, comparing new infant data with new simulation data. In the simulations, all of the habituation items were the same as in our original experiments. I gave the model a chance to map *wo fe wo* onto *ga ti ga* and then tested it on *fe wo wo* versus *fe wo fe.* The model, presumably driven by information about the final word rather than the abstract ABA structure, favored *fe wo wo* (or, cashed out as looking time, the model predicted that the infants would look longer at *fe wo fe*).

In contrast, we found that infants look longer at *fe wo wo* than at *fe wo fe* (Marcus & Bandi Rao, 1999). Thus while the Altmann and Dienes architecture does offer a bona fide alternative account of the original Marcus et al. results, it does not truly extract a UQOTOM, and our additional data suggest that it does not appear to yield an account of what infants actually do. Children seem to freely generalize the ABA sequence, ignoring facts like whether *wo* appears as the third word, whereas the Altmann-Dienes model is driven only be more particular, less general kinds of information.

3.4.2 Models That Incorporate Operations over Variables

A simple recurrent network trained by an external supervisor What other alternatives are there? Seidenberg and Elman (1999a) proposed one possible solution. Their model has two parts—a simple recurrent network and an external supervisory device. The network part of the model is much like the simple recurrent networks described earlier in this chapter, but the system as a whole differs. Whereas standard versions of the SRN are trained by a signal that is readily available in the environment (the next word in a sentence), Seidenberg and Elman's model is trained by an external supervisor that itself applies a rule of the form "For all syllables **x, y,** if **x** = **y,** then output 1 else output 0."

Since the existence of an external supervisor that incorporates a rule makes the difference between the system working and a nearly identical system not working, it seems that the rule is a crucial component of the overall system.[11] Unfortunately, Seidenberg and Elman (1999a) do not give an account of how the supervisor's rule could itself be implemented in the neural substrate, so their model tells little about how rules might be implemented in a neural substrate.

A feedforward network that uses nodes as variables Shultz (1999) showed how an autoassociator network (one in which the target is always the same as the input) could capture our results. Crucial to the success of the model is the encoding scheme. Rather than using nodes to represent particular words (as in the sentence-prediction network) or the presence or absence of particular phonetic features (à la Seidenberg and Elman, 1999a) Shultz uses each node as a variable that represents a particular position in the sentence. In other words, rather than using a many-nodes-per-variable encoding scheme, Shultz uses a one-node-per-variable encoding scheme.

In all, Shultz uses three input nodes and three output nodes, each of which represents a word position. One input node represents the variable **first word** in the input sentence, another represents the variable **second word** in the sentence, and the remaining one represents the variable **third word** in the sentence.[12] Likewise, each output node represents a particular word.[13] The nodes serve as variables, and their activation values represent specific instances. For example, if the first word is *ga,* Shultz turns on the first input node with a value of 1; if the first word is *li,* Shultz turns on the first input node with a value of 3; if it is *ni,* he turns on the first input node with a value of 7.

As was shown earlier, a connection that runs from an input node that represents a variable to a hidden node with a connection weight of 1 is simple implementing an operation that copies the contents of one variable to the contents of another. Since the connection treats all possible instances equally, the copy operation applies equally to all possible instantiations of the input variable.

The task of Shultz's model is auto-association. The measure of the model that Shultz adopted is how closely the output units reflect the input units. The idea is that the model will better auto-associate (copy) inputs that are consistent with habituation than inputs that are inconsistent with training.

While Shultz does not provide information about what connection weights the network actually uses, it is easy to see how this network could capture the results in a way that implements operations that are defined for all instances of some variable (or what I have called *algebraic rules*). For example, figure 3.10 shows how a simplified version of Shultz's model can (transparently) implement operations over variables.

In a similar but slightly more complex work (published prior to Shultz's 1999 article), Negishi (1999) shows how a modified simple recurrent network can use nodes that represent variables to capture our results. Negishi's model is slightly more complex, in part because each word is encoded by means of two variables, but the general point remains the same: it relies on using nodes as variables, with connections indicating operations that must apply to all instances of a class, rather than indicating operations that pertain only to those elements that contain some particular feature.

A still more complex variation on this theme was presented by Gasser and Colunga (1999). The authors use a set of *micro-relational units,* each of which recodes the sameness or difference between two particular items. For example, one micro-relational unit responds in accordance with the degree of similarity between the first word and the last word. In effect, these microrelational units work like an instruction in a microprocessor that calculates the cosine between any two numbers **x** and **y.** What is crucial is that the behavior of these microunits is not conditioned on experience but rather, as in a microprocessor, defined in advance for all possible instances of **x** and **y.** As Gasser and Colunga note, their model would be able to capture our results with such units.

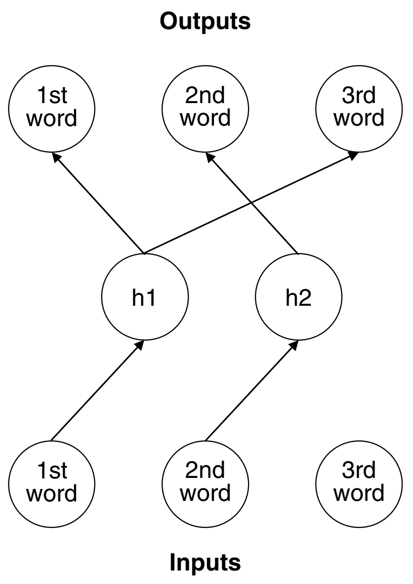

Figure 3.10
A simplified version of Shultz's (1999) model. This set of weights encodes the ABA grammar in a feedforward network that uses nodes to represent variables rather than particular words or properties of sets of words. This model auto-associates ABA sentences better than ABB sentences.

A temporal synchrony model Shastri and Chang (Shastri, 1999; Shastri & Chang, 1999) have implemented a different sort of model. Unlike the models of Shultz and of Altmann and Dienes, this model was not implemented as an apparent argument against the idea that children represented algebraic rules, but as an explicit suggestion about how such rules can be implemented in a neural substrate. Following the temporal synchrony framework, Shastri and Chang use one set of nodes to represent temporal variables (**1st word, 2nd word, 3rd word**) and another set of nodes to represent phonetic features (**+voiced,** and so on). Rules are represented by links between the variables. In essence, the ABA rule is represented by having a single "hidden" node that is linked to both **1st word** and **3rd word.** This assembly forces the nodes representing first and third words to resonate in synchrony, thereby binding them to the same instantiations.

An abstract recurrent network Another approach is to use registers. In effect, the abstract recurrent network model of Dominey and Ramus (2000), depicted in figure 3.11 does just that. Dominey and Ramus tested what would happen if the model did not have the register-like

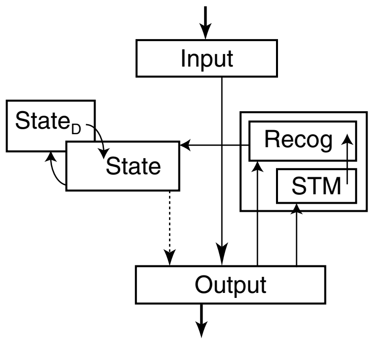

Figure 3.11
The abstract recurrent network of Dominey and Ramus (2000). Reprinted by permission of the publisher.

component. They found that the registerless version of the model could not capture our results but that the version that incorporates registers and an operation that compares the values of those registers with the current input was able to capture our results. Supporting our view (albeit perhaps reluctantly) Dominey and Ramus (2000, p. 121) conclude that "Even though, like Seidenberg (1997), we feel that the statistical properties of the input have too often been overlooked, both Marcus et al.'s experiments and our simulations show that learning cannot be reduced to the discovery of statistical regularities on the surface." Instead, they note that the version of the model that could capture our results differs from the version that could not in that it "includes the recognition function, which is a comparator, a typically nonassociationist mechanism"—one that applies the same operation to all instances of a variable.

3.4.3 Summary

The bottom line should be clear: what makes the successful connectionist models work is a facility for representing operations that apply to all instances in a class. As the summary given in table 3.2 makes clear, the few connectionist models that do not incorporate any sort of genuine operation over variables cannot capture our results, whereas all of the models that do implement operations over variables can capture our results.

3.5 Case Study 2: Linguistic Inflection

Perhaps the only test case in which there is a wider array of connectionist models is the domain of linguistic inflection. Elman et al.'s, 1996, review of connectionist models of development devotes more pages to

Table 3.2
Models of rule learning in infants.

Model	Architecture	Encoding	Relies on an external device that incorporates a rule	Works with sets of distributed phonetic features	Incorporates an operation that applies to all instances	Captures free generalization to novel items
Marcus (see text)	Simple recurrent network	Localist or distributed	No	Yes	No	No
Altmann and Dienes (1999)	Modified simple recurrent network	Localist	No	N/A	No	No (see text)
Christiansen and Curtin (1999)	Simple recurrent network	Distributed	No	Yes	No	No (small differences that have not been shown to be reliable)
Dominey & Ramus (2000), Model A	Abstract recurrent network	Localist	No	N/A	No	No
Dominey & Ramus (2000), Model B	Temporal recurrent network	Localist	No	N/A	Yes	Yes
Negishi (1999)	Modified simple recurrent network	Analog	No	N/A	Yes	Probably yes
Seidenberg & Elman (1999a)	Simple recurrent network	Distributed	Yes	Yes	Yes (in teacher)	Probably yes
Shastri & Chang (1999)	Temporal synchrony	Distributed	No	Yes	Yes	Probably yes
Shultz (1999).	Feed forward network	Analog	No	N/A	Yes	Probably yes

inflection than any other empirical topic; at least 21 different models have been proposed. Most focus on the English past tense.

3.5.1 Empirical Data

What sort of empirical data can be used to choose among these models? Most of the empirical data in this literature has been collected in the context of a model originally proposed by Pinker and Prince, and defended by several others, including myself. That model includes a rule-based component for inflecting regular verbs (*walk-walked*) and an associative memory, perhaps perceptron-like, for inflecting irregular verbs (*sing-sang, go-went,* and so forth). On this view, the irregular component takes precedence over the operation of the rule-based component. Consistent with this model, a great deal of evidence suggests that regulars and irregulars behave in qualitatively different ways (Berent, Pinker & Shimron, 1999; Clahsen, 1999; Kim, Marcus, Pinker, Hollander & Coppola, 1994; Kim, Pinker, Prince, & Prasada, 1991; Marcus, 1996b; Marcus, Brinkmann, Clahsen, Wiese & Pinker, 1995; Marcus et al., 1992; Pinker, 1991, 1995, 1999; Pinker & Prince, 1988; Prasada & Pinker, 1993; Ullman, 1993). For example, Prasada and Pinker (1993) showed that generalizations of irregular patterns are sensitive to similarity to stored forms but that generalizations of the regular pattern are not sensitive to similarity. It seems more natural to inflect the novel verb *spling* (which resembles other irregulars such as *sing* and *ring*) as *splang* than to inflect the novel verb *nist* (which does not closely resembles any irregular) as *nast*, even though both verb stems undergo the same vowel change. In contrast, it seems no more natural to inflect *plip* (which resembles regulars such as *rip, flip,* and *slip*) as *plipped* than to inflect *ploamph* (which does not closely resemble any regular) as *ploamphed.* Further evidence also suggests regulars and irregulars are processed in different brain areas (Jaeger et al., 1996; Pinker, 1999; Ullman, Bergida & O'Craven, 1997) and may be (doubly) dissociable in patient populations (Marslen-Wilson & Tyler, 1997; Ullman et al., 1997).

In what follows, I use three criteria for evaluating competing models. The first is that a model should be able to add *-ed* freely to novel words, even those with unfamiliar sounds. For example, Berko (1958) showed that children tend to generalize *-ed* inflection to novel words like *wug: This is a man who knows how to wug. What did he do yesterday? He* ________. Similarly, adults seems to be able to freely generalize *-ed* to words of virtually any sound. We might say that *Yeltsin outgorbacheved Gorbachev,* even if we do not know any other verb that sounds like *outgobachev.* A further bit of evidence that the operation of *-ed* is rule-like is that children seem to be able to apply it even to verb stems that are homophonous with irregular verbs. When they are told *This is a ring. Now I'm ringing your finger. What did I just do?,* adults (Kim, Pinker, Prince &

Prasada, 1991) and even three-year-old children (Kim, Marcus, Pinker, Hollander & Coppola, 1994) respond *You just ringed my finger* not *You just rang my finger.*

The second criterion is about frequency. Although adding *-ed* is the most common way of inflecting English verbs, *-ed*'s qualitative status as a default (it can be added to unusual sounding words, to verbs derived from nouns, and so forth) does not appear to depend on its high frequency, whether measured in terms of the number of distinct verbs (types) or the number of distinct occurrences of those verbs (tokens). Instead, we find cases like the German *-s* plural—a suffix that applies to fewer than 10 percent of the nouns (measured by types or tokens) and yet behaves in ways that are qualitative virtually identical to English default inflection (Marcus, Brinkmann, Clahsen, Wiese & Pinker, 1995). For example, just as we would say that *Last night we had the Julia Childs over for dinner,* a German speaker would say that *I read two Thomas Manns* rather than *two Thomas Männer.* An adequate model should therefore produce defaultlike effects even when the regular pattern is no more common or even less common than the irregular patterns.

The third important criterion for evaluating competing models is that when people do apply a default suffix, they almost always apply it to a verb's stem rather than to an inflected version of the stem. Children, for example, produce errors like *breaked* about 10 times as often as errors like *broked* (Marcus et al., 1992). Similarly, given a novel verb like *spling,* adults may produce *splang* or *splinged,* but they hardly ever produce *splanged* (Prasada & Pinker, 1993). An adequate model should thus avoid *splanged*-like blends in which *-ed* is added to something other than a verb's stem.[14]

3.5.2 Three Criteria Applied

Which models best capture these empirical data? Paralleling my discussion of models of artificial grammar learning, I suggest again that adequate models must incorporate some sort of machinery for representing abstract relationships between variables. In the remainder of this chapter, I review the connectionist models of inflection, dividing them into models that explicitly implement abstract relationships between variables, models that do not implement abstract relationships between variables, and models that are billed as alternatives to symbol-manipulation but that nonetheless turn out to implement abstract relationships between variables.

Table 3.3 lists 21 connectionist models of inflectional systems, the vast majority of which are multilayer perceptrons, giving details about their architectures, encoding schemes, and training regimes; I taxonomize and evaluate them in the next several pages.

Table 3.3
Past-tense models.

Type of model	Reference	Input	Output
Feedforward network	Rumelhart & McClelland (1986a)	Phonology	Phonology
Feedforward network	Egedi & Sproat (1991)	Phonology	Phonology
Feedforward network	MacWhinney & Leinbach (1991)	Phonology	Phonology
Feedforward network	Plunkett & Marchman (1991)	Phonology	Phonology
Attractor network	Hoeffner (1992)	Semantics and syntax	Phonology
Feedforward network	Daugherty & Seidenberg (1992)	Phonology	Phonology
Feedforward network	Daugherty & Hare (1993)	Phonology	Phonology
Feedforward network	Plunkett & Marchman (1993)	Phonology	Phonology
Feedforward network	Prasada & Pinker (1993)	Phonology	Phonology
Feedforward network	Bullinaria (1994)	Phonology	Phonology
Simple recurrent network	Cottrell & Plunkett (1994)	semantics	Phonology
Feedforward network	Forrester & Plunkett (1994)	Verb ID	Class ID
Feedforward network	Hare & Elman (1995)	Phonology	Class ID
Hybrid (see text)	Hare & Elman (1995)	Phonology	Phonology
Hybrid (see text)	Westermann & Goebel (1995)	Phonology	Phonology
Feedforward network	Nakisa & Hahn (1996)	Phonology	Class ID
Feedforward network	O'Reilly (1996)	Semantics	Phonology
Feedforward network	Plunkett & Nakisa (1997)	Phonology	Class ID
Feedforward network	Plunkett & Juola (1999)	Phonology	Phonology
Hybrid (see text)	Westermann (1999)	Phonology	Phonology
Feedforward network	Nakisa, Plunkett, & Hahn (2000)	Phonology	Phonology

Models that explicitly implement abstract relationships between variables A small number of models explicitly implement a rule-and-memory system, very much along the lines of what Pinker and I have advocated. The model that comes closest to ours was proposed by Westermann and Goebel (1995, p. 236) "in accordance with the rule-associative memory hypothesis proposed by Pinker (1991)" and incorporating a module that serves as a short-term memory to represent "the rule path of the dualistic framework" and a phonological lexicon to implemented the irregulars (see figure 3.12).

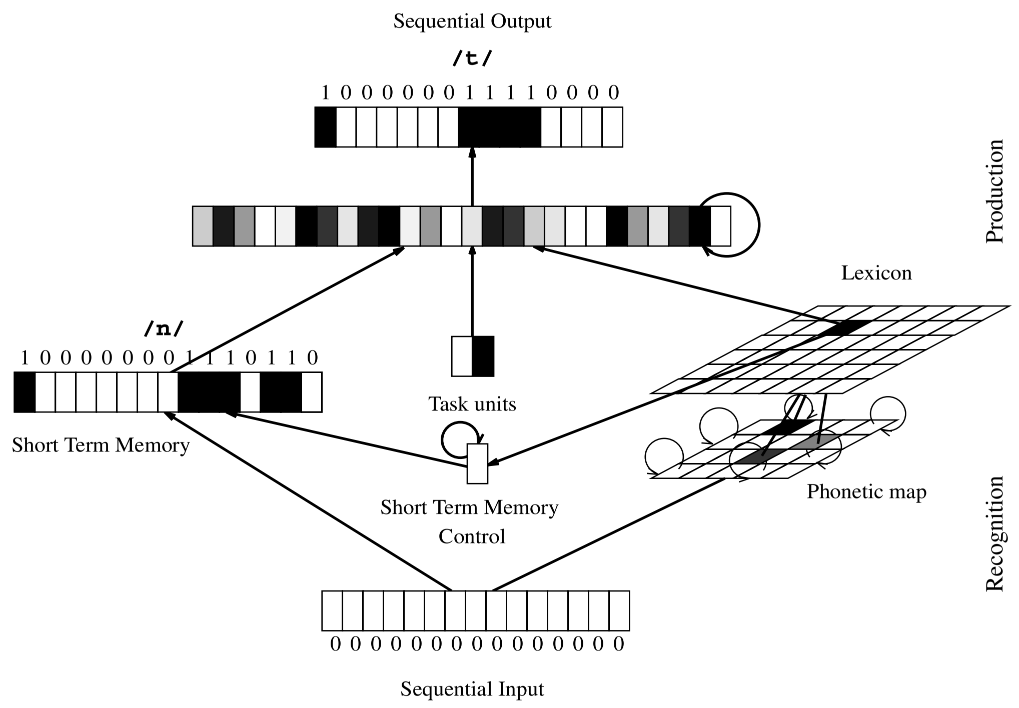

Figure 3.12
Westermann and Goebel's (1995) modular model of linguistic inflection. The phonetic map (right) serves as a way to associate irregulars. The short-term memory (left) serves as a way to copy the stem, implementing part of the process that adds a suffix to the stem. © Cognitive Science Society, Inc. Reprinted by permission.

Their model is, at least to some extent, able to capture a default in which the type frequency of regular verbs is not overwhelmingly greater than that of irregular verbs. Their model, which is trained on a corpus of only about 52 percent regular verbs, measured by types (45 percent measured by tokens), is able to generalize the regular pattern to three of four novel test words that do not sound similar to any of the training items.

A later model by Westermann (1999) is also able to capture the fact that people can freely generalize the regular inflection pattern to novel stems. Like Westermann's earlier model, Westermann's more recent model also builds in two routes. In this later model, one route depends on an abstract relation between variables that is implemented as a set of copy weights. These copy weights—which effectively build in the identity function prior to input—guarantee that the model can copy the stem of any verb, even prior to any training. Like the earlier model by Goebel and Westermann, Westermann's more recent model is able to capture free generalization of default inflection to novel, unusual-sounding verb stems, and it appears to be able to do so even in the absence of high type frequency for regular inflection.

Models that offer a genuine alternative to rules Although these rule-implementing models of Westermann do at least a reasonable job of capturing empirical data, most of the connectionist models on inflection have been presented with the aim of providing alternatives to rules. For example, the first and most famous connectionist model of inflection was proposed by Rumelhart and McClelland (1986a). As we saw earlier, these authors proposed their model as a "distinct alternative to the view that children learn the rule of English past tense formation in any explicit sense" (p. 267). Throwing down the gauntlet, Rumelhart and McClelland (1986a, p. 267) aimed to show that "a reasonable account of the acquisition of past tense can be provided without recourse to the notion of a 'rule' as anything more than a *description* of the language."

Their model, sketched in figure 3.13, works by taking a phonetically encoded input and transforming it into a phonetically encoded output. For example, the input to the model on a given trial is a phonetic description of the word *ring,* and the target output is *rang.* Words consist of sets of triples, known as Wickelfeatures. Simplifying slightly, the word *sing* is represented by the simultaneous activation of the triples #*si, sin, ing,* and *ng*#, where # is a special marker for the beginning or end of a word.

Unlike many of its successors, the Rumelhart-McClelland model lacked hidden units. Yet the model did surprisingly well, capturing some interesting qualitative phenomena. For example, although the model did not have any explicitly represented rules, it added *-ed* to some

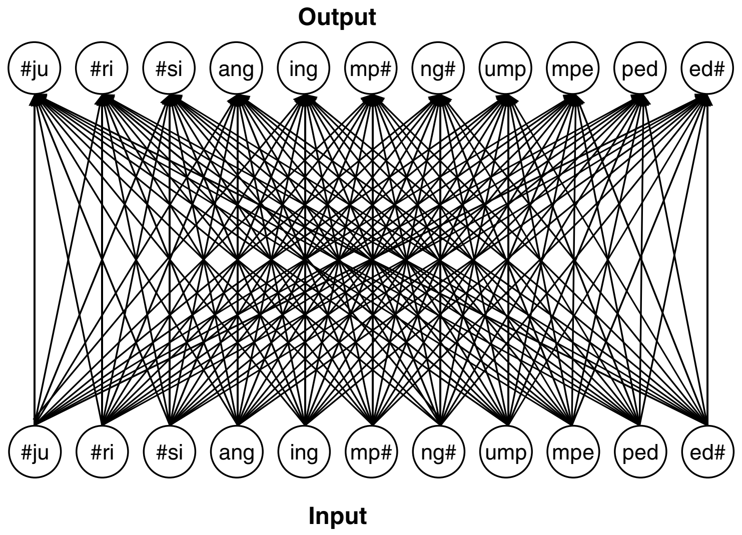

Figure 3.13
Rumelhart and McClelland's (1986a) two-layer pattern associator that represents words as sequences of three letters. Input and output nodes encode Wickelfeatures (sequences of three phonetic features) rather than sequences of three letters. The actual model has 460 input nodes, each of which is connected to each of 460 output nodes; all words are represented as subsets of those nodes.

novel verbs, which yielded "overregularizations" like *breaked* and *taked.* Likewise the model produced some correctly inflected irregular verbs before it first began to overregularize.

Nonetheless, it is now widely acknowledged that the model is seriously flawed. For example, the model's ability to capture a period of correct irregular use prior to overregularization[15] depends on an unrealistic, abrupt change from an almost entirely irregular input vocabulary to an almost entirely regular input vocabulary (Marcus et al., 1992; Pinker & Prince, 1988). Another problem is that the model is unable to generalize well to novel words, producing bizarre blends such as the past tense *membled* for *mail* and the past tense *imin* for the novel verb stem *smeeb* (Prasada & Pinker, 1993). In addition, the Wickelfeature system that the model uses to represent a word cannot keep certain pairs of words distinct, such the Australian language Oynkangand's words *algal* ("straight") and *algalgal* ("ramrod straight") (Pinker and Prince, 1988, p. 97). The model would also likely have trouble generalizing to a default that is low in frequency (Marcus, et al., 1995).

But if the model's limitations are by now widely acknowledged, there is far less consensus on what to do about these problems or on what aspects of the model's architecture are responsible for its limitations. Although Pinker and I attribute the limitations of the Rumelhart and McClelland model to its lack of rules, others attribute the limitations to the model's lack of a hidden layer. For example, McClelland (1988, p. 118) argues that "a problem with the [Rumelhart & McClelland, 1986] past-tense model is that it has no intervening layers of units between the input and the output. This limitation has been overcome by the development of the back-propagation learning algorithm (Rumelhart, Hinton & Williams, 1986)."

Echoing McClelland's remarks, Plunkett and Marchman (1991, p. 199) argue that "the use of a back-propagation algorithm in a network with hidden units represents a step forward in the application of PDP systems to problems of language processing and acquisition." Similarly, Hare, Elman, and Daugherty (1995, p. 607) acknowledge some of the criticisms raised by Prasada and Pinker (1993) but attribute the limitations to two-layer networks, suggesting that while the Rumelhart-McClelland model "was a remarkable contribution to the field, advances in learning theory have made it obsolete in certain respects and its shortcomings do not carry over to the more sophisticated architectures that have since been developed."

In keeping with these suggestions, many researchers have pursued more sophisticated multilayer perceptron models, models that are similar in spirit to Rumelhart and McClelland's model but are enhanced with a hidden layer and more plausible training regimes and phonetic representation schemes. Like the Rumelhart-McClelland model, many subsequent models have continued to treat the task of past tense acquisition as one of using a single network to learn a mapping between a phonologically represented stem and phonologically represented inflected form. In the words of Elman et al. (1996, p. 139), the goal of these models is to support a position in which "regular and irregular verbs . . . [are] represented and processed similarly in the same device."

Yet these models continued to face many of the same limitations that the earlier Rumelhart and McClelland model faced. It is striking that, contrary to the stated goals of Elman et al., no one has yet proposed a comprehensive single-mechanism model. Instead, what has been proposed is a series of models, each devoted to a different aspect of the past tense: one model for why denominal verbs (such as *ring* as in *ring a city with soldiers*) receive regular inflection (Daugherty, MacDonald, Petersen & Seidenberg, 1993), another for handling defaults for low-frequency verbs (Hare, Elman & Daugherty, 1995), another for distinguishing homonyms that have different past-tense forms (MacWhinney & Leinbach, 1991), and still another for handling overregularization phenomena (Plunkett & Marchman, 1993). These models differ from one another in their input representations, their output representations, and their training regimes. Far from showing how inflection can be implemented in a single device, these models—taken collectively—could just as easily be taken as evidence that *more than* one mechanism is necessary.

More to the point, the models that map from phonetic representations to phonetic representations still have trouble capturing the generalization of default inflection to unfamiliar words and still have trouble explaining how default inflection can be generalized in languages in which it is infrequent. For example, Plunkett and Marchman (1993) conducted a series of simulations in which they systematically varied the proportion of the input vocabulary that was regular, testing how likely each network was to generalize regular inflection to novel words that did not closely resemble any of the words on which the model had been trained. They found that "the level of generalizations . . . [is] closely related to the total number of regular verbs in the vocabulary" (p. 55). They also reported that "generalization is virtually absent when regulars contribute less than 50% of the items overall" (p. 55). Such models would thus have difficulty capturing default inflection where the default is not the most frequent pattern.

Such models also frequently produce blends, adding regular inflection to the past tense of the verb (such as *ated*) rather than to the verb stem (such as *eated*). For example, Plunkett and Marchman's (1993) model produced far more *ated*-type blends (6.8 percent) than *eated* type overregularizations (less than 1 percent), whereas children produce far more *eated* type overregularizations (4 percent in a sample of preschoolers) than *ated*-type blends (less than 1 percent) (Marcus et al., 1992).[16] Similarly, Daugherty and Hare's (1993) model produced such blends in half (six out of 12) of its responses to words containing novel vowels.

Why do networks that map phonetically encoded stems onto phonetically encoded past tense forms have such difficulties? It is instructive to think about how these networks inflect regular verbs. In the rule-and-memory model, novel regulars are inflected by a process that concatenates a variable (**verb stem**) with the *-ed* morpheme. As such it is automatically defined to apply equally to all verb stems, regardless of their sound. The operation of the rule may be suppressed by the associative system, so sometimes the rule may not be invoked at all (or alternatively, the rule might be invoked but its actual output suppressed.) But its output is uniform, unaffected by similarity: *walk* in, *walked* out; *outgorbachev* in, *outgorbacheved* out.

Implicit in this is a sort of identity operation. Putting aside the irregulars, part of the past tense of an English verb **x** is **x.** For example, one part of the past tense of *outgorbachev* is *outgorbachev* itself. The rule-memory model explains this by saying, effectively, that the past tense of the verb is a copy of the verb stem, suffixed by the *-ed* morpheme.

Pattern associators like Rumelhart and McClelland's (1986) model offer a different account of how novel regular verbs are inflected with regular inflection. In the place of an operation that is defined over the variable **verb stem,** they offer a set of lower-level associations in which phonologically defined parts of verb stems are associated with phonologically defined parts of past tense forms. Fundamentally, such models are many-nodes-per-variable models. This means that they must learn the "identity map" part of regular inflection piecemeal. If input nodes stand for phonemes, the models must learn identity for each phoneme separately; if input nodes stand for phonetic features, the models must learn identity for each phonetic feature separately.

Depending on the nature of the input representation, this piecemeal learning may or may not make learning the identity map part of inflection problematic. If the input nodes represent phonemes, a model would not be able to properly produce the past tense form that corresponds to an input verb that contains a novel phoneme. For example, if the sound /z/ as in the word *rouge* never appeared in training on verbs, a model that allocates a separate node to /z/ will not generalize to that node. Such a model thus will be incapable of explaining how a native speaker inflects *rouge* as *rouged* (as in what the aging film star played by Diane Wiest does to her cheeks in *Bullets over Broadway,* just prior to having a couple of drinks with John Cusack).

If the /z/ sound is represented by a set of phonetic features, all of which appear in training, inflecting *rouge* as *rouged* is not problematic. But going to phonetic representations is probably not a panacea. I suspect, for example, that English speakers could, at least in comprehension, distinguish between, say, inflected words that copy a stem that contains a novel feature and inflected words that omit that novel feature. For example, I suspect that English speakers would prefer *Ngame out!ngaioed !Ngaio* to *Ngame outngaioed !Ngaio,* so even a model that represents inputs in terms of phonetic features (rather than phonemes) would have trouble. (For that matter, further problems arise if it turns out that we can inflect unpronounceable glyphs—for example, if we recognize the well-formedness of the [written] sentence, *In sheer inscrutability, the heir apparent of the artist formerly known as Prince has out♀ed ♀.*)

In any case, swapping a process that operates over variables for a process that relates a regular verb and its past tense only in a piecemeal way results in a problem with blends. If there is no stem-copying process as such, nothing constrains the system to use the *-ed* morpheme only when the stem has been copied. Instead, whether the stem is transformed (as with an irregular) or copied (as with a regular) is an emergent property that depends on a set of largely independent, piecemeal processes. If the system learns that *i* sometimes changes to *a,* there is little to stop it from applying the *i-a* process at the same time as the process that causes *-ed* to appear at the end. The consequence is lots of blends, such as *nick-nucked,* that humans rarely if ever produce. Humans tend to make a discrete choice between *nack* and *nicked* because the pathway that adds *-ed* adds *-ed* to the **stem**; networks that lack a separate pathway from regulars have no such constraint. If *i* activates *a* but *ck* activates *cked,* a blend is produced. The bottom line—for models that lack a process defined over the variable **verb stem**—is this. Even if one uses low-level phonetic features to represent words, where there are irregulars it is difficult to correctly generalize the *-ed* pattern to novel unusual-sounding words without producing spurious blends.

Classifier models: Alternatives to rules? If these phonetics-to-phonetics models were the only alternatives to Westermann's (1999) models that explicitly implement rules, perhaps the controversy would already be over. What has kept the controversy going, I think, is that there is a wide variety of other connectionist models of inflection that operate on different principles and these models do not map phonetically encoded inputs into phonetically encoded outputs. These models—which are billed as alternatives to algebraic rules (that is, operations over variables)—do a better job of capturing the human data and their phonetics-to-phonetics cousins. But it turns out that each of these models either implements algebraic rules or depends on an external device that does.

One class of models, which I refer to as *classifiers,* produces as its output not a phonological description (such as /rang/ or /jumpd/) but simply a label. This label indicates whether a given input word belongs to, say, the *ing-ang* class or the *add -ed* class. The process of inflecting the input word is not complete until some external device concatenates the verb stem as input with the *-ed* subjects. There is of course, nothing wrong with relying on such an external device. But assuming the existence of such a device (unnecessary in the case of phonetics-to-phonetics models) is tantamount to including two algebraic rules: one that copies the stem and another that concatenates it with *-ed.*

By building in (offstage, as it were) operations such as "copy" and "concatenate," these models start with the relevant abstract relationships between variables and thus avoid the problems that would otherwise arise as a consequence of training independence. They do not, however, obviate the need for rules.[17]

The clean-up network: Implementation or alternative? As a final illustration, consider the two-part network proposed by Hare, Elman, and

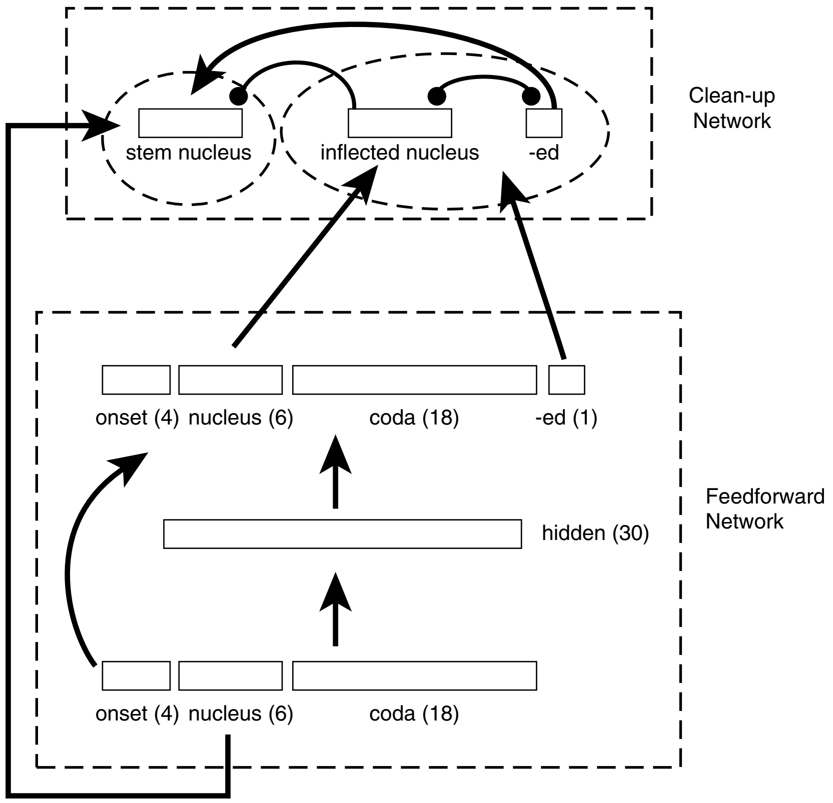

Figure 3.14
Hare, Elman, and Daugherty's (1995) hybrid model: A clean-up network and a feedforward network. Reprinted by permission.

Daugherty (1995) and illustrated in figure 3.14. Billed as an alternative to the rule-and-memory approach, this model effectively *implements* a rule-and-memory model. The model consists of two components—a feedforward network depicted at the bottom of the diagram and a clean-up network at the top. The feedforward network works much like any other phonetics-to-phonetics model and by itself does not implement a rule. But the single solid line that appears on the left side of the diagram, running from the input nodes to the clean-up network, does. This line actually represents a set of six connections that serve as a prewired copy operation—thereby finessing the training independence issues by guaranteeing in advance that all possible verb stems will be copied, even if the model received no training at all.[18]

In addition to a prewired copy operation that passes the stem along to the clean-up network, the Hare, Elman, and Daugherty (1995) model includes a mechanism that comes close to recapitulating the "blocking" mechanism that Pinker and I suggested modulates the relation between irregulars and regulars. The mechanism that we had advocated was, "Search for an irregular, use it if you find it, and otherwise fall back on the default." The Hare et al. model operates on essentially an identical principle. The feedforward network supplies a guess about how the input verb might be inflected if it were an irregular; this guess, along with the verb stem, is passed along to the clean-up network. The clean-up network, which learns nothing, is innately wired such that if the model's guess about the irregular is strongly activated, output nodes that represent the stem and the *-ed* suffix are suppressed. In contrast, if the irregular is weakly activated, both the stem and *-ed* are strongly activated. As in the classifier models, the suffixation process is actually handled by an external device. Rather than being an alternative to rules, the Hare et al. model relies on rules extensively.[19]

This case is particularly instructive because Hare, Elman, and Daugherty (1995) attribute the success of their model to its hidden layer and to its assumptions about the phonological distribution of the input words (that is, the similarity between different verbs of different classes). Because earlier work by Egedi and Sproat (1991) led us to be skeptical about the importance of hidden layers, Justin Halberda and I (Marcus & Halberda, in preparation) tested to see whether an implementation of the Hare et al. model that lacked a hidden layer would perform notably worse. We found that it did not, doing just as good a job of generalizing to novel, unfamiliar words as did a version of the model that included hidden layers. In contrast, we found that the clean-up network was crucial to the success of the Hare et al. model. A version of the model in which the clean-up network was removed did far worse than a version of the model that contained the clean-up network, producing far more blends than humans produce.

3.5.3 Discussion

The past tense question originally became popular in 1986 when Rumelhart and McClelland (1986a) asked whether we really have mental rules. Unfortunately, as the proper account of the past tense has become increasingly discussed, Rumelhart and McClelland's straightforward question has become twice corrupted. Their original question was "Does the mind have rules in anything more than a descriptive sense?" From there, the question shifted to the less insightful "Are there two processes or one?" and finally to the very uninformative "Can we build

Table 3.4
Models of inflection: Performance summary.

Type of model	Reference	Includes a separate pathway for regulars	Add *–ed* to unusual sounding unfamiliar words	Low-frequency default	Avoids blends
Feedforward network	Rumelhart & McClelland (1986a)	No	No	Not tested	No
Feedforward network	Egedi & Sproat (1991)	No	No	Not tested	No
Feedforward network	MacWhinney & Leinbach (1991)	No	No	Not tested	No
Feedforward network	Plunkett & Marchman (1991)	No	No	Not tested	No
Attractor network	Hoeffner (1992)	No	Not tested	Not tested	No
Feedforward network	Daugherty & Seidenberg (1992)	No	Yes (?)	Not tested	No (?)
Feedforward network	Daugherty & Hare (1993)	No	Yes (?)	Yes	No
Feedforward network	Plunkett & Marchman (1993)	No	No	No	No
Feedforward network	Prasada & Pinker (1993)	No	No	Not tested	No
Feedforward network	Bullinaria (1994)	No	Not tested	Not tested	No
Simple recurrent network	Cottrell & Plunkett (1994)	No	Not tested	Not tested	No
Feedforward network	Forrester & Plunkett (1994)	Classifier	N/A	Yes	N/A
Feedforward network	Hare & Elman (1995)	Classifier	*Yes*	*Yes*	N/A
Hybrid (see text)	Hare & Elman (1995)	*Yes*	*Yes*	*Yes*	*Yes*
Hybrid (see text)	Westermann & Goebel (1995)	*Yes*	*Yes*	*Yes*	*Yes*
Feedforward network	Nakisa & Hahn (1996)	Classifier	Not tested	Not tested	Yes
Feedforward network	O'Reilly (1996)	No	Not tested	Not tested	No
Feedforward network	Plunkett & Nakisa (1997)	Classifier	Not tested	Yes	N/A
Feedforward network	Plunkett & Juola (1999)	No	See note	Not tested	No
Hybrid (see text)	Westermann (1999)	Not tested	*Yes*	*Yes*	*Yes*
Feedforward network	Nakisa, Plunkett & Hahn (2000)	No	Not tested	Not tested	Yes

Note: Plunkett and Juola's (1999) model regularized some unfamiliar words, but it is not clear from their report whether it can generalize *–ed* to unusual-sounding unfamiliar words.

a connectionist model of the past tense?" The "two processes or one?" question is less insightful because the *nature* of processes—not the sheer number of processes—is important. A bipartite model can be built as a hybrid (as Pinker and I suggest), a bipartite symbolic model, or even a bipartite multilayer perceptron, with one "expert" devoted to regulars and another to irregulars (e.g., Jacobs, Jordan & Barto, 1991). Likewise one can build monolithic models from either architecture. The sheer number tells us little, and it distracts attention from Rumelhart and McClelland's original question of whether (algebraic) rules are implicated in cognition.

The "Can we build a connectionist model of the past tense?" question is even worse, for it entirely ignores the underlying question about the status of mental rules. The implicit premise is something like "If we can build an empirically adequate connectionist model of the past tense, we won't need rules." But as we have seen, this premise is false: many connectionist models implement rules, sometimes inadvertently.

Opponents of symbol-manipulation rarely consider this issue and instead take for granted that their models, by virtue of being connectionist, serve as refutations of variable-manipulating models. For instance Hare, Elman, and Daugherty's (1995) clean-up network did indeed overcome one of the key limitations of early connectionist models of the past tense. Because it is a connectionist model, Hare et al. took this model as a refutation of the rule-and-memory model. But as we have seen, the Hare et al. model reveals that it is not a genuine counter to the rule-and-memory model but virtually an implementation of it.

The right question is not "Can any connectionist model capture the facts of inflection?" but rather "What design features must a connectionist model that captures the facts of inflection incorporate?" If we take what the models are telling us seriously, what we see is that those connectionist models that come close to implementing the rule-and-memory model far outperform their more radical cousins. For now, as summarized in table 3.4, it appears that the closer the past tense models come to recapitulating the architecture of the symbolic models—by incorporating the capacity to instantiate variables with instances and to manipulate (here, "copy" and "suffix") the instances of those variables—the better they perform.

Connectionist models can tell us a great deal about cognitive architecture but only if we carefully examine the differences between models. It is not enough to say that some connectionist model will be able to handle the task. Instead, we must ask what architectural properties are required. What we have seen is that models that include machinery for operations over variables succeed and that models that attempt to make do without such machinery do not.