CHAPTER 16

The Programmable Blackboard Model of Reading

J. L. McCLELLAND

In 1975, Rumelhart outlined a model of reading called the *interactive model*. That model, inspired by the HEARSAY model of speech understanding (Reddy, Erman, Fennell, & Neely, 1973), supposed that reading involved simultaneous processing at a large number of levels, including visual feature, letter, word, syntactic, and semantic levels. Hypotheses at each level were activated when active hypotheses on adjacent levels suggested them and competed with alternative hypotheses at the same level. This model, of course, was a precursor of the interactive activation model of word recognition and of the approach that underlies this whole book.

In the interactive model of reading, the activation of hypotheses was guided by a set of structures called "knowledge sources," each of which had expertise with respect to a particular aspect of reading. For example, a lexical knowledge source that contained knowledge of the letter sequences that made up each word was proposed, along with a syntactic knowledge source, a semantic knowledge source, and an orthographic knowledge source containing knowledge of the appearance of the letters.

An important aspect of the interactive model of reading was parallel processing. Processing was supposed to occur in parallel both within and between levels, so that hypotheses could influence and be influenced by other hypotheses spanning large stretches of the input. However, no specific implementation of the mechanism was proposed. In HEARSAY, although the *conception* was parallel, the *reality* was quite

sequential—each knowledge source could only be directed to a single small part of HEARSAY's BLACKBOARD at a time. The result was a model that was computationally very cumbersome and excruciatingly slow. Eventually, HEARSAY was abandoned in favor of a model in which the knowledge that guided processing was precompiled, by brute force, into a Markov chain.

PDP models such as the interactive activation model of word perception and the TRACE model of speech perception have tried to capture the parallelism inherent the conception of HEARSAY. (See Chapter 15.) However, these models differ from HEARSAY in a fundamental way. Instead of having a knowledge source that can be applied to an input, they build the knowledge into the connections between the units out of which the mechanisms are built. Thus, for example, in the word perception model, the knowledge that guides processing is built into the connections between the units. By making several copies of the same connection information, it is possible to allow parallel processing.

To make this point clear and to emphasize some of the properties of this aspect of the model, a sketch of the model is presented in Figure 1.

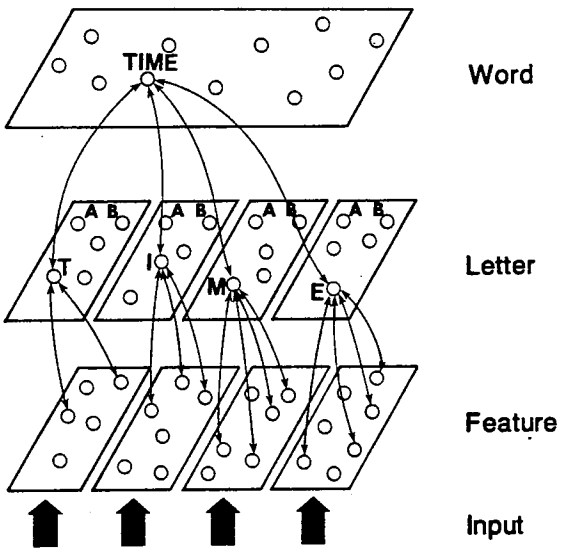

FIGURE 1. A sketch of the interactive activation model of word perception. Units within the same rectangle stand for incompatible alternative hypotheses about an input pattern, and are all mutually inhibitory. The bidirectional excitatory connections between levels are indicated for one word and its constituents. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, *9*, p. 115. Copyright 1985 by Ablex Publishing. Reprinted by permission.)

The figure brings out the fact that the model achieves parallel processing of all of the letters in a four-letter display by having four separate copies of the feature and letter units, and four separate copies of the connections between them. Parallel processing of more than one word at a time would require creating another copy of the entire network. In the TRACE model Elman and I did just this, creating a large number of copies of the entire connection network.

Now, the parallel processing permitted in models like the word perception model and the TRACE model are important to the functioning of these models, since parallel processing permits exploitation of mutual constraints, and to a large extent it is the exploitation of mutual constraints that parallel distributed processing is all about. But the massive reduplication of hardwired connection information leaves something to be desired as a way of providing a mechanism to exploit these mutual constraints. For one thing, it dedicates a lot of hardware to a single task. Also, even if we were willing to give up all the hardware, there would still be a serious problem with learning. In models of this sort, learning amounts to changing the strengths of the connections among units, based on their simultaneous activation. This kind of learning is, as was stressed in Chapter 1, *local* to the particular connections in which it occurred. Thus, any learning that occurred in one letter-processing channel of the word perception model would not be available for processing letters in other channels.

I was not pleased with this state of affairs. It seemed to me that if parallel distributed processing was going to prove viable, some way would have to be found to have a central knowledge representation, such as in HEARSAY, that could be made available for processing items occurring at different places in a visual display or at different points in time.

One obvious solution is just to "go sequential," and put the knowledge in a central location and map inputs into it one at a time. We have already reviewed a way that this can be done within the PDP framework in the introduction to this section of the book. The trouble with this solution is that it eliminates parallel processing, and thus the benefits thereof. Obviously, at some point we will have to go sequential—and I will start to do so at a later point in this chapter. But I was not happy with the possibility that the only way we could achieve parallel processing was through the reduplication of connection information. I sought, in short, a mechanism for achieving parallel processing without reduplication of connection information. This chapter reports on the results of my explorations in search of such a mechanism.

The organization of the chapter is as follows. The first section develops a model for processing two words at a time, using a single

central representation of the knowledge of the letter patterns that make up words. The model is applied to some recent data collected by Mozer (1983) on the processing of two-word displays. The second section extends the ideas developed in the first part to a more complex processing structure called the *programmable blackboard*, a structure analogous to the Blackboard in HEARSAY. That model is applied to a number of phenomena in word recognition not covered by the original interactive activation model. It also gives an account of several important aspects of reading a line of print, including the integration of information over successive fixations in reading. The final section discusses the proposed mechanisms in more general terms, and describes how they might be extended to the processing of the syntactic and semantic content of sentences.

CONNECTION INFORMATION DISTRIBUTION

This section describes a simple system for programming parallel processing structures in response to ongoing task demands and applies it to the processing of words in one- and two-word displays. The system consists of a set of programmable modules. Each module is a network of processing units very similar to those in other PDP models, such as the interactive activation model of word perception. The difference is that these units are not dedicated permanently to stand for particular hypotheses, and the knowledge that determines the pattern of excitatory and inhibitory interactions is not hardwired into the connections between them. Rather, the connections in the network are programmable by inputs from a central network in which the knowledge that guides processing is stored.

The first part of this section describes an individual programmable network. Later parts describe the structures needed to program such networks in response to ongoing processing demands.

A Programmable Network

Figure 2 presents a very simple hardwired network. The task of this section is to see how we could replace this hardwired network with one that could be programmed to do the same work. The network shown in the figure is a very simple interactive activation system, consisting only of a letter and a word level. The figure is laid out differently from the previous figure to highlight the excitatory connections between the

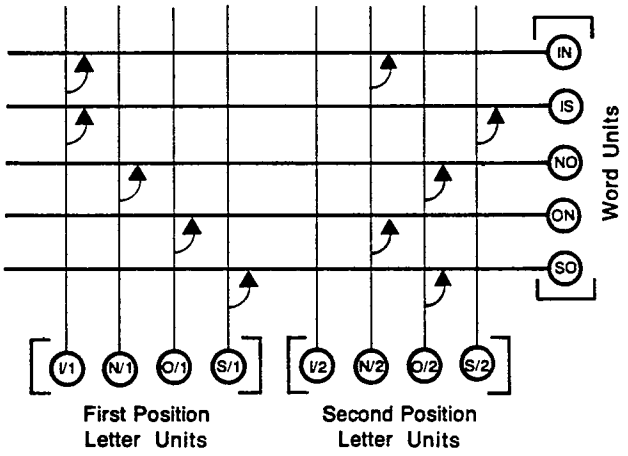

FIGURE 2. An extremely simple connectionist mechanism, capable of processing one two-letter string made up of the letters *I*, *N*, *O*, and *S*. The model knows only the five words that can be made of two of these letters, namely *IN*, *IS*, *NO*, *ON*, and *SO*. No top-down connections are included in this simple model. Units bracketed together are mutually inhibitory. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, *9*, p. 118. Copyright 1985 by Ablex Publishing. Reprinted by permission.)

units and lay them out in a way which will be convenient as we proceed.

In this simple network, there are detectors only for the letters *I*, *N*, *O*, and *S* in each of two letter positions. At the word level, there is a detector for each of the English words that can be made out of two of these letters. For simplicity, this model contains only letter-to-word connections; another matrix would be needed to capture word-to-letter feedback. Units that are in mutual competition are included in the same square brackets. This is just a shorthand for the bidirectional inhibitory connections, which could also be represented in another connection matrix.

In this diagram, letter units are shown having output lines that ascend from them. Word units are shown having input lines that run from left to right. Where the output line of each letter unit crosses the input line of each word unit, there is the possibility of a connection between them.

The knowledge built into the system, which lets it act as a processor for the words *IN*, *IS*, *NO*, *ON*, and *SO*, is contained in the excitatory connections between the letter and word units. These are represented by the filled triangles in the figure.

Now we are ready to see how we could build a programmable network, one that we could *instruct* to behave like the hardwired network shown in Figure 2. Suppose that instead of fixed connections from specific letter units to particular word units, there is a *potential* connection at the junction between the output line from each letter unit and the input line to each word unit. Then all we would need to do to "program" the network to process the words *IN*, *IS*, *NO*, *ON*, and *SO* correctly would be to send in signals from outside turning on the connections that are hardwired in Figure 2. This proposal is illustrated in Figure 3.

Multiplicative interactions yield programmable connections. At first glance, the notion of sending instructions to connections may seem to be adding a new kind of complexity to the basic processing elements out of which connectionist mechanisms are built. Actually, though, all we really need to do is to let each connection multiply two signals before passing along the result.

This point may be appreciated by considering the following equation. For the standard connections used in most connectionist models, the

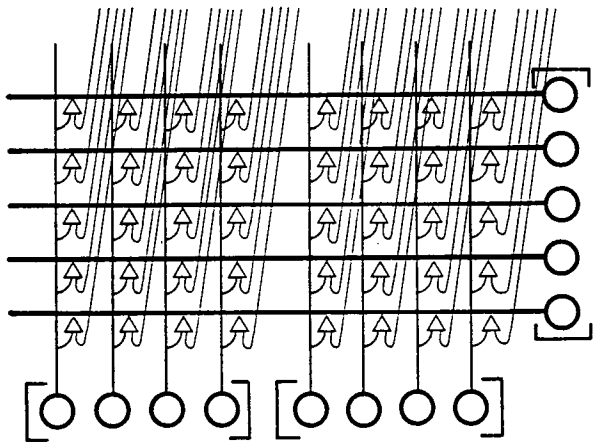

FIGURE 3. A programmable version of the simplified activation model shown in Figure 2. Each triangle represents a *programmable connection* that can be turned on by a signal coming from the central knowledge store, shown here as lying outside the figure to the upper right. If the triangular connections pass the product of the two signals arriving at their base along to the receiving unit, the lines coming into the matrix from above can be thought of as programming the network. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 119. Copyright 1985 by Ablex Publishing. Reprinted by permission.)

time-varying signal to some unit i from some unit j is multiplied by the fixed weight or connection strength w_{ij} to determine the value of the input to i from j:

$$input_{ij}(t) = signal_j(t) \times w_{ij}.$$

All we are assuming now is that the signal from unit j is multiplied by a second time-varying signal, for example, the signal arising from some other unit k instead of the fixed connection strength w_{ij}:

$$input_{ij}(t) = signal_j(t) \times signal_k(t).$$

We can think of the signal from unit k as *setting the strength* of the connection to i from j. When the value of the second signal is greater than 0, we will say that the connection to i from j is *active.*

Function and implementation of programmable connections. Multiplicative connections of the kind proposed here were introduced in Chapter 2, and the increase in computational capability that their introduction affords is considered in Chapter 10. The specific idea of using a second signal to modulate connections has been used in other connectionist models. Hinton (1981b) used such a scheme to map inputs from local (retinocentric) feature detectors onto central (object-centered) feature detectors in a viewpoint-dependent way. My use of multiplicative connections here was inspired by Hinton's. J. A. Feldman and Ballard (1982) have also suggested the idea of making connections contingent on the activation of particular units. The general notion of using one set of signals to structure the way a network processes another set of signals has previously been proposed by Sejnowski (1981) and Hinton (1981a).

Let us briefly consider the functional significance of programmable connections. In essence, what connections do in PDP models of perceptual processing is specify *contingencies* between *hypotheses.*[1] A positive weight on the connection to unit i from unit j is like the conditional rule, "if j is active, excite i." Fixed connections establish such contingencies in a fixed, permanent way. Programmable connections allow us to specify what contingencies should be in force, in a way which is itself contingent on other signals. By using multiplicative interactions between signals, in place of fixed connections, we now have a way of setting from outside a network the functional connections or contingencies between the units inside the network. This means that we can dynamically program processing modules in response

[1] I would like to thank Geoff Hinton for pointing out the relation between connections and contingencies.

to expectations, task demands, etc. The little module shown in Figure 3 could be used for a variety of different processing tasks, if different connection patterns were sent into it at different times. For example, if we sent in different signals from outside, we could reprogram the module so that the word units would now respond to the two-letter words in some other language. In conjunction with reprogramming the connections from feature level units to the letter units, we could even assign the network the task of processing words in a language with a different alphabet or of processing completely different kinds of patterns.

At a neurophysiological level, multiplicative or quasi-multiplicative interactions between signals can be implemented in various ways. Neurons can implement multiplication-like interactions by allowing one signal to bring the unit's activation near threshold, thereby strongly increasing the extent to which another signal can make the unit fire (Sejnowski, 1981). There are other possibilities as well. A number of authors (e.g., Poggio & Torre, 1978) have suggested ways in which multiplication-like interactions could take place in subneuronal structures. Such interactions could also take place at individual synapses, though there is little evidence of this kind of interaction in cortex. For a fuller discussion of these issues, see Chapters 20 and 21.

The Connection Information Distribution Mechanism

We are now ready to move up to a complete Connection Information Distribution (CID) mechanism containing a number of programmable modules along with the structures required to program them. The basic parts of the mechanism are shown and labeled in Figure 4; they are shown again, with some of the interconnections, in Figure 5.

The CID mechanism consists of a central knowledge store, a set of programmable modules, and connections between them. The structure is set up in such a way that all of the connection information that is specific to recognition of words is stored in the central knowledge store. Incoming lines from the programmable modules allow information in each module to access the central knowledge, and output lines from the central knowledge store to the programmable modules allow connection activation information to be distributed back to the programmable modules.

The two programmable modules are just copies of the module shown in Figure 3. It is assumed that lower-level mechanisms, outside of the model itself, are responsible for aligning inputs with the two modules, so that when two words are presented, the left word activates

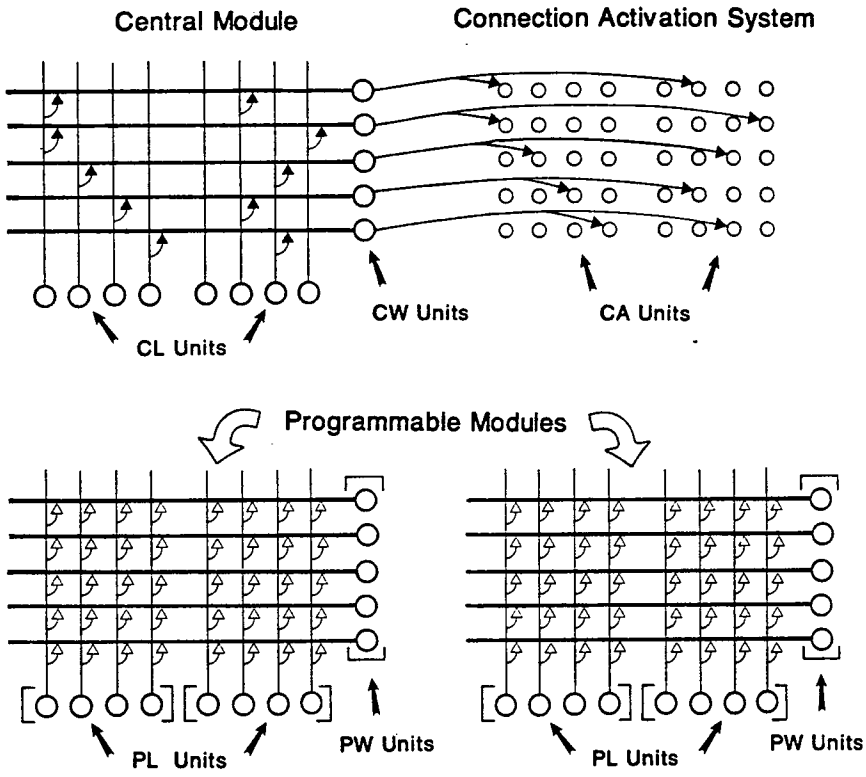

FIGURE 4. A simplified example of a connection information distribution (CID) mechanism, sufficient for simultaneous bottom-up processing of two two-letter words. The programmable modules consist of the programmable letter (PL) units, programmable word (PW) units, and programmable connections between them (open triangles). The central module consists of a set of central letter (CL) units, a set of central word (CW) units, and hardwired connections between them (filled triangles). The connection activation system includes the central word units, a set of connection activation (CA) units, and hardwired connections between them. Connections between the central knowledge system (central module plus connection activation system) and the programmable modules are shown in the next figure. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 122. Copyright 1985 by Ablex Publishing. Adapted by permission.)

appropriate programmable letter units in the left module, and the right one activates appropriate programmable letter units in the right module.

The central knowledge store. The knowledge store in the CID mechanism is shown at the top of Figure 4. This is the part of the mechanism that contains the word-level knowledge needed to program

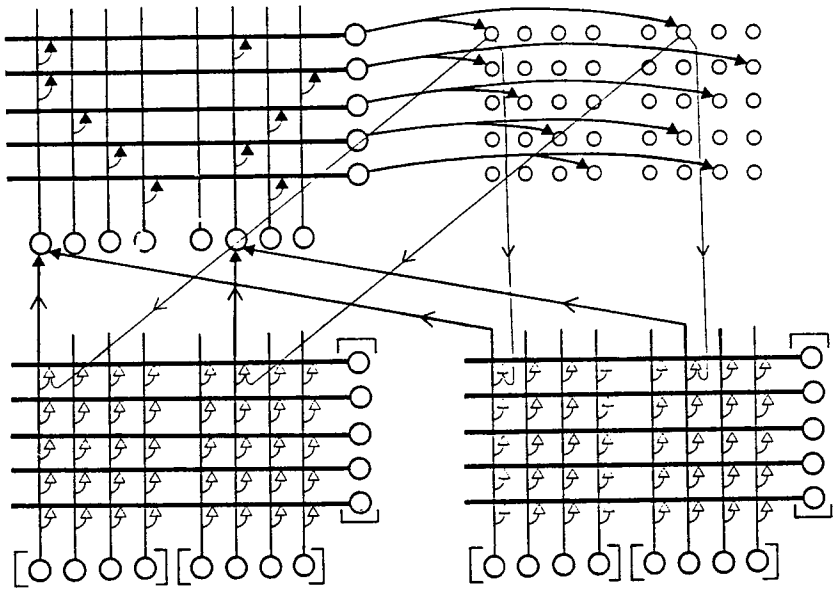

FIGURE 5. Each CA unit projects to the corresponding connection in both programmable modules, and each central letter unit receives projections from the corresponding programmable letter unit in both programmable modules. The inputs to two central letter units and the outputs from two CA units are shown. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 124. Copyright 1985 by Ablex Publishing. Reprinted by permission.)

the programmable modules. It consists of two parts. One part is called the *central module*, and the other part is called the *connection activation system*.

The central module consists of central letter units, central word units, and connections between the central letter units and the central word units. The letter units in the local modules project to the letter units in the central module, so that whenever a particular letter unit is active in either programmable module, the corresponding central letter unit is also (Figure 5). Note that the correspondence of local and central letter units is quite independent of what letters these units stand for.

The central letter units are connected to the central word units via connections of the standard, hardwired type. These connections allow patterns of activation at the letter level to produce corresponding activations at the word level, just as in the original interactive activation

model. However, it should be noted that the central word unit activations are based on a superposition of the inputs to each of the two programmable modules. Thus, the activations in the central letter units do not specify which module the letters came from, though relative position within each module is preserved. Thus, activations in the central module do not distinguish between the input *IN SO* and the input *SO IN* or even *SN IO*. In short, the central module cannot correctly determine which aspects of its inputs belong together.

The second part of the central knowledge system, the connection activation system, also consists of two sets of units and their interconnections. One of these sets of units is the set of central word units—they belong both to the central module and to the connection activation system. The other set is the set of connection activation (CA) units. The purpose of the connection activation system is to translate activations of central word units into activations of connections appropriate for processing the corresponding words in the local modules. The CA units serve as a central map of the connections in each of the programmable modules, and provide a way to distribute connection information to all of the programmable modules at the same time. (The CA units are not strictly necessary computationally, but they serve to maintain the conceptual distinction between that part of the mechanism that contains the knowledge about words and the parts that simply distribute that knowledge to the local modules). There is one CA unit corresponding to the connection between a particular programmable letter unit and a particular programmable word unit. I have arranged the CA units in Figure 4 to bring out this correspondence. *Each* CA unit projects to the corresponding connection in *both* programmable modules. I have illustrated the projections of two of the CA units in Figure 5. For example, the top-left CA unit corresponds to the connection between the left-most programmable letter unit and the top-most programmable word unit. This CA unit projects to its corresponding connection in each of the programmable modules and provides one of that connection's two inputs. So, when a particular CA unit is active, it activates the corresponding connection in *all* of the programmable modules. In this way it acts as a master switch.

At a functional level, we can see each CA unit as standing for a contingency between two activations. Thus, if we index the programmable letter units by subscript i, and the programmable word units by j, the ijth CA unit stands for the contingency, "if letter unit i is active, excite word unit j." Thus, we can think of the CA units as contingency activation units, as much as connection activation units. When we activate a CA unit (to a certain degree) we are implementing the contingency it represents (with a corresponding strength) in both of the programmable modules at once.

The central word units, of course, are responsible for activating the CA units. There are excitatory connections from each word unit to each of the CA units for the connections needed to process the word. For example, the central word unit for *IN* activates two CA units. One is the CA unit for the connection between the left-most programmable letter unit and the top-most programmable word unit. The other is the CA unit for the connection from the sixth programmable letter unit from the left to the same programmable word unit. These connections effectively assign the top programmable word unit to be the detector for *IN* (assuming, of course, that lower levels of processing have been arranged so that *I* in the first position and *N* in the second position activate the appropriate programmable letter units).

In summary, the CID mechanism consists of (a) the programmable modules; (b) the central knowledge store, including the central module and the connection activation system; (c) converging inputs to the central knowledge store from the programmable modules; and (d) diverging outputs from the central knowledge store back to the programmable modules.

We can now see how this mechanism allows the programmable modules to be programmed dynamically in response to current inputs. When an input causes activations in some of the programmable letter units in one of the programmable modules (say, the programmable letter units for *I* in the first position and *N* in the second position of the left programmable module), these activations are passed to the corresponding central letter units. From the central letter units they activate the central word unit for *IN*. Central word units for patterns that overlap partially with the input (such as *IS* and *ON*) also receive excitation, but only in proportion to their overlap with the input. The central word units pass activation to the CA units, and these in turn pass activation back to the connections in both programmable modules. Connections are only turned on to the extent that they are consistent with the input. When different patterns are presented to each programmable module, connections appropriate for both patterns are turned on, thereby programming both programmable modules to process either pattern. Central word units—and therefore connections—are also turned on for any words that appear in the superimposed input from the two programmable modules. However, the results of processing in each programmable module still depend on the activations of the programmable letter units. Thus the appropriate programmable word unit will tend to be the most active in each local module. Although the output of the central module does not specify which word was presented to which local module, this information is represented (though with some tendencies to error, as we shall see) in the outputs of the local modules.

The correspondence between programmable and central units in the CID mechanism illustrated in Figures 4 and 5 may lead some readers to feel that the local units are really dedicated to process the particular word I have said that the mechanism programs it to process. This is an artifact of the use of one unit to represent each word. If distributed representations are used instead, each local output unit unit can be programmed in different ways on different occasions, depending on which of a large number of different distributed patterns the local modules are programmed to produce. The discussion that follows is generally applicable to both local and distributed CID mechanisms; I chose to use the local case because I thought it was generally easier to grasp intuitively. The main difference between local and distributed CID mechanisms is that the latter make much more efficient use of the units in the programmable modules, as discussed in Chapter 12.

Computer Simulation of Word Recognition Using the CID Mechanism

To examine the behavior of CID mechanisms in more detail, I implemented a CID version of the interactive activation model of word perception. The model, which I just call CID ("Sid"), is scaled-up from the example we have been considering so that it can process two strings of four letters each. Only three or four different letter alternatives were allowed in each position within each string. These were *B*, *L*, *P*, and *S* in the first position; *A*, *E*, *I*, and *O* in the second position; *N*, *R*, and *V* in the third position; and *D*, *E*, and *T* in the fourth position. The lexicon used in the simulation consisted of the 32 words shown in Table 1.

Like the smaller-scale version shown in the figures, the model consisted of two programmable modules, one for each of the two-letter strings, and a central knowledge store consisting of the central module and the connection activation system. Each programmable module had 16 programmable letter units and 32 programmable word units. The programmable letter units were grouped into four groups of four, with each group to be used for letters in one display location. The members of each group had mutual, hardwired, inhibitory connections. Similarly, all of the programmable word units in each module were mutually inhibitory. Each programmable module contained $16 \times 32 = 512$ programmable connections, and there were 512 CA units, one for each each programmable connection. The central module contained 16 letter and 32 word units, like the programmable modules. There were no inhibitory connections either between the central word units or between the central letter units. The connections between the central letter

TABLE 1

THE 32 WORDS USED IN THE SIMULATIONS

BAND	BARE	BEND	BIND
BIRD	BOND	BONE	BORE
LAND	LANE	LARD	LEND
LINE	LINT	LIVE	LONE
LORD	LOVE	PANE	PANT
PART	PINE	PINT	POND
PORE	PORT	SAND	SANE
SAVE	SEND	SORE	SORT

Note: From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 126. Copyright 1985 by Ablex Publishing. Reprinted by permission.

units and the central word units and the connections from the central word units to the appropriate CA units were hardwired with the connection information needed to make the central letter units activate the right central word units and to make the central word units activate the right CA units.

Inputs to the simulation model were simply specifications of bottom-up activations to the programmable letter units in either or both programmable modules. Inputs were presented when all the units in the model were at their resting activation values and turned off after some fixed number of time cycles.

The only substantive difference between CID and the original interactive activation model is in the strengths of the excitatory connections between units. In CID, these strengths vary as a function of the current input, while in the original model they were fixed. Highly simplified activation rules are used to capture the essence of the connection activation process via the central letter units, central word units, and CA units. The activation of a particular central letter unit is simply the number of input units projecting to it that have activations greater than 0. Thus, the activation of a particular central letter unit just gives a count of the corresponding programmable letter units that are active. The activation of a central word unit is just the sum of the active central letter units that have hardwired connections to the central letter unit. The activation of a CA unit is just the activation of the central word unit that projects to it, and this value is transmitted unaltered to the corresponding programmable connection in each programmable module.

The net effect of these assumptions is to make the activation of the connections coming into a particular programmable word unit proportional to the number of active units for the letters of the word, summed over both modules. Active letter units count only if they stand for letters in appropriate positions within the programmable module of origin.

Output. So far we have said nothing about how the activations that arise in the programmable modules might give rise to overt responses. Following the original interactive activation model, I assume there is a readout mechanism of unspecified implementation which translates activations at either the letter or the word level into overt responses.[2] The readout mechanism can be directed to the word or the letter level of either module, and at the letter level it can be directed to a particular letter position within a module. In cases where more than one stimulus is to be identified on the same trial, the readout of each of the items is independent.

The probability of choosing a particular response depends on the strength of the unit corresponding to that response divided by the sum of the strengths of all the relevant alternatives (e.g., units for words in the same position) following the formulas introduced in Chapter 15.

The main import of these assumptions for present purposes is that the probability of a particular response is solely a function of the activations of units relevant to the response. All interactions between display items are thus attributed to the unit and connection activation mechanisms, and not to the readout mechanisms themselves.

RESULTS OF THE SIMULATIONS

Two principle findings emerged from working with the simulation model. First, when processing a single word, the CID mechanism causes the model to behave as though it were sharply tuned to its inputs, thereby mimicking the benefits of the bottom-up inhibition used in the original word perception model without actually incurring any of its deficiencies. Second, when processing two words at a time, the connection activation scheme causes the model to make errors similar to those made by human subjects viewing two-word displays. These

---

2 There must be coordination between the readout mechanism and the CID mechanism. For example, it would not do for the system to program the top-most letter unit to represent the word *BAND*, say, if the readout mechanism took this unit to correspond to some other word. The problem is a general one and is no different in programmable networks than it is in standard ones.

errors arise as a result of the essential characteristics of the CID mechanism.

One Word at a Time: The Poor Get Poorer

In the original word perception model, bottom-up inhibition from the letter level to the word level was used to sharpen the net bottom-up input to word units. For example, consider a display containing the word *SAND*. Due to bottom-up inhibition, units for words matching only three of the four letters shown (e.g., *LAND*) would receive less than 3/4 as much net bottom-up excitation as the unit for the word *SAND* itself.

The CID version of the model closely emulates this feature of the original, even though it lacks these bottom-up inhibitory connections. In CID, the activation of the *connections* coming into a word unit varies with the number of letters of the word that are present in the input. At the same time, the number of inputs to these same connections from the programmable letter units also varies with the number of letters in the input that match the word. The result is that in the CID version of the model, the amount of bottom-up activation a programmable word unit receives varies as the *square* of the number of letters in common with the input. Poorer matches get penalized twice.

In working with the original model, Rumelhart and I picked values for the bottom-up excitation and inhibition parameters by trial and error as we searched for a set of parameters that allowed the model to fit the results of a large number of experiments. The values we hit upon put the strength of bottom-up inhibition at 4/7 the strength of bottom-up excitation. For words that share two, three, or all four letters in common with the input, this ratio produces almost exactly the same relative amounts of net bottom-up activation as is produced by the CID mechanism (Table 2). Words with less than two letters in common received net bottom-up inhibition in the old version, whereas in the CID version they receive little or no excitation. In both cases their activation stays below zero due to competition, and thus they have no effect on the behavior of the model.

This analysis shows that the CID version of the model can mimic the original, and even provides an unexpected explanation for the particular value of bottom-up inhibition that turned out to work best in our earlier simulations. As long as the bottom-up input to the letter level was unambiguous, the correspondence of the CID version and a no-feedback version of the original model is extremely close.

When the bottom-up input to the letter level is ambiguous, however, there is a slight difference in the performance of the two versions of

TABLE 2

ONE WORD AT A TIME: BOTTOM-UP ACTIVATIONS OF SEVERAL WORD UNITS IN THE ORIGINAL AND CID VERSIONS OF THE INTERACTIVE ACTIVATION MODEL

Input: SAND

Unit	Letters Shared w/Input	Original		CID Version	
		Relative Activation	Ratio	Relative Activation	Ratio
SAND	4	4	-	4×4	-
LAND	3	3 - 4/7	.61	3×3	.56
LANE	2	2 - 8/7	.21	2×2	.25

Note: Ratio is the net bottom-up activation of the unit, divided by the net bottom-up activation of the unit for SAND. From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, *9*, p. 129. Copyright 1985 by Ablex Publishing. Reprinted by permission.

the model. This actually reveals an advantage of eliminating bottom-up inhibition similar to some of the advantages discovered in the discussion of the TRACE model (see the previous chapter). Consider the input to a word unit from the letter units in a particular letter position. In the original model, if three or more letter candidates were active, two of them would always produce enough bottom-up inhibition to more than outweigh the excitatory effect any one of them might have on the word. For example, if *E*, *F*, and *C* are equally active in the second letter position, *F* and *C* together would inhibit the detectors for words with *E* in second position more than *E* will excite them. Thus, if three letters are active in all four letter positions, no word would ever receive a net excitatory input. This problem does not arise in the CID version because there is no bottom-up inhibition. Thus, the CID version can pull a word out of a highly degraded display in which several letters are equally compatible with the feature information presented, while the original model cannot. It thus appears that CID gives us the benefits of bottom-up inhibition without the costs.

Two Words at a Time: Interference and Crosstalk

So far we have seen how CID retains and even improves on some of the important aspects of the behavior of the original model. Now, I

will show how CID captures important aspects of the data obtained in experiments in which subjects are shown two words at a time. Here the CID architecture becomes essential, since simultaneous processing of two patterns introduces considerations that do not arise in the processing of one pattern at a time.

When letters are presented to both modules, *all* of the letters are combined to turn on connections that are distributed to *both* of the programmable modules. The result is that the connections appropriate for the word presented in one module are turned on in the other module as well. This biases the resulting activations in each module. The programmable word unit for the word presented to a particular module will generally receive the most activation. However, the activation of programmable word units for words containing letters presented to the other module is enhanced. This increases the probability that incorrect responses to one of the words will contain letters presented in the other.

At first this aspect of the model disturbed me, for I had hoped to build a parallel processor that was less subject to crosstalk between simultaneously presented items. However, it turns out that human subjects make the same kinds of errors that CID makes. Thus, though CID may not be immune to crosstalk, its limitations in this regard seem to be shared by human subjects. I'll first consider some data on human performance, and then examine in detail why CID behaves the same way.

The data come from a recent experiment by Mozer (1983). In his paradigm, a pair of words (e.g, *SAND LANE*) is displayed, one to the left and one to the right of fixation. The display is followed by a patterned mask which occupies the same locations as the letters in the words that were presented. In addition, the mask display contains a row of underbars to indicate which of the two words the subject is to report. Subjects were told to say the word they thought they saw in the cued location or to say "blank" in case they had no idea.

In his first experiment, Mozer presented pairs of words that shared two letters in common. The pairs of words had the further property that either letter which differed between the two words could be transposed to the corresponding location in the other word and the result would still be a word. In our example *SAND-LANE*, *SAND* and *LANE* have two letters in common, and either the *L* or the *E* from *LANE* can be moved into the corresponding position in *SAND*, and the result would still be a word (*LAND* and *SANE*). Of course, it was also always true with these stimuli that the result would be a word if both letters "migrated." The duration of the two-word display was adjusted after each counterbalanced block of trials in an attempt to home in on a duration at which the subject would get approximately 70% of the

whole-word responses correct. Thus, the overall error rate was fixed by design, though the pattern of errors was not.

The principal results of Mozer's experiment are shown in Table 3. Of the trials when subjects made errors, nearly half involved what Mozer called "migration errors"—errors in which a letter in the context word showed up in the report of the target. To demonstrate that these errors were truly due to the presentation of these letters in the context, Mozer showed that these same error responses occurred much less frequently when the context stimulus did not contain these letters. Such "control" errors are referred to in the table as pseudo-migration errors.

As I already suggested, migration errors of the type Mozer reported are a natural consequence of the CID mechanism. Since the letters from both words are superimposed as they project onto the central module, the connections for words whose letters are present (in the correct letter position) in either of the two input strings are strongly activated in both programmable modules. The result is that programmable units for words containing letters from the context are more easily activated than they would be in the absence of the input presented to the other module.

Table 4 compares relative programmable word unit activations for various words, for two different cases: In one case, the word *SAND* is

TABLE 3

METHOD AND RESULTS OF MOZER (1983), EXPERIMENT 1

Method:

Example Display	SAND LANE
Target Cue	———

Results:

Response Type	Example	% of Total
Correct response	(SAND)	69.0
Single migration	(SANE or LAND)	13.3
Double migration	(LANE)	0.5
Other		17.2
Total		100.0
Pseudo-migration*		5.3

*Pseudo-migration rate is the percentage of reports of the given single migration responses (SANE,LAND) when a context word which does not contain these letters is presented. In this example, the context string might have been BANK. From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 131. Copyright 1985 by Ablex Publishing. Reprinted by permission.

presented alone; in the other, it is presented in the context of the word *LANE*. When *SAND* is presented alone, all words that share three letters with it receive $(3/4)^2$ or 9/16 as much bottom-up activation as the unit for *SAND* itself—we already explored this property of the CID model in the previous section. When *SAND* is presented with *LANE*, however, words fitting the pattern (*L* or *S*)-*A*-*N*-(*D* or *E*) all have their connections activated to an equal degree because of the pooling of the input to the connection activation apparatus from both modules. These words are, of course, *SAND* and *LANE* themselves and the single migration error words *LAND* and *SANE*. Indeed, over both letter strings, there are 6 occurrences of the letters of each of these words (the *A* and the *N* each occur twice). The result is that the excitatory input to the programmable word units in the left module for *LAND* and *SANE* is 3/4 of that for *SAND*, as opposed to 9/16. Other words having three letters in common with the target have their connections less activated. Their bottom-up activation is either 5/8 or 1/2 that of *SAND*, depending on whether two of the letters they have in common with the target are shared with the context (as in *BAND*) or not (as in *SEND*). Thus, we expect *LAND* and *SANE* to be reported more often than other words sharing three letters in common with *SAND*.

The reader might imagine that the effect would be rather weak. The difference between 3/4 and 5/8 or 1/2 does not seem strikingly large. However, a raw comparison of the relative bottom-up activation does not take into account the effects of within-level inhibition. Within-level inhibition greatly amplifies small differences in bottom-up activation.

TABLE 4

TWO WORDS AT A TIME: CROSSTALK.
RELATIVE BOTTOM-UP ACTIVATIONS PRODUCED BY SAND
PRESENTED EITHER ALONE OR WITH LANE AS CONTEXT

	alone		with LANE	
	activation	ratio	activation	ratio
SAND	4×4	-	4×6	-
LAND	3×3	.56	3×6	.75
BAND	3×3	.56	3×5	.62
SEND	3×3	.56	3×4	.50
LANE	2×2	.16	2×6	.50

Note: Ratio refers to the bottom-up activation of the unit, divided by bottom-up activation of SAND. From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 132. Copyright 1985 by Ablex Publishing. Reprinted by permission.

This is especially true when two or more units are working together at the same level of activation. In this case, the units for *LAND* and *SANE* act together. Neither can beat out the other, and both "gang up" on those receiving slightly less bottom-up activation, thereby pushing these other alternatives out. This "gang effect" was also observed in the original version of the word perception model. The results of this feature of the model are illustrated in Figure 6. Through the mutual inhibition mechanism, *SAND* and *LANE* come to dominate other words that share three letters in common with the target. Some of these, in turn, dominate words that have only two letters in common with the target, including, for example, *LANE*, even though the connections for *LANE* are strongly activated. This result of the simulation agrees with the experimental result that double or "whole-word" migrations are quite rare in Mozer's experiment, as shown in Table 3.

Mozer (1983) reported several additional findings in his study of the processing of two-word displays. A full discussion of these effects can be found in McClelland (1985). Suffice it to say here that the major findings of Mozer's experiments are consistent with what we would expect from CID.

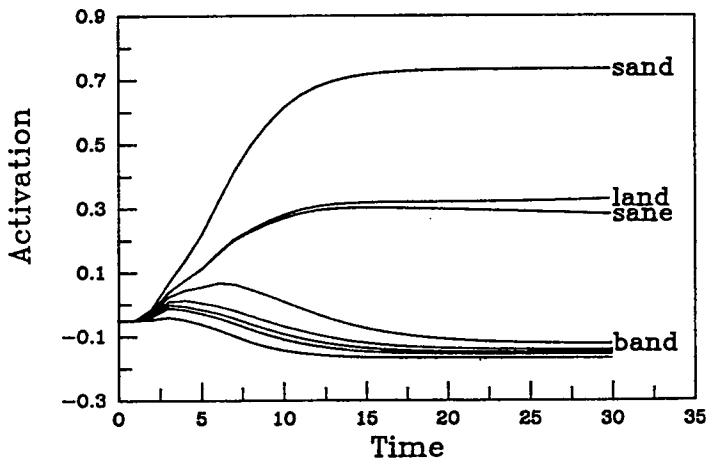

FIGURE 6. Activation curves for various programmable word units in the module to which *SAND* is shown when the input to the other module is *LANE*. The horizontal axis represents time cycles from the onset of the two-word display. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly" by J. L. McClelland, 1985, *Cognitive Science*, 9, p. 133. Copyright 1985 by Ablex Publishing. Reprinted by permission.)

THE PROGRAMMABLE BLACKBOARD MODEL

The model described in the previous section provides an illustrative example of the way in which knowledge in a central processing structure can be used to program local processing structures in response to ongoing processing demands, but it is a long way from a fully adequate model of the reading process, even excluding syntactic and semantic processing levels above the word. This section describes a more sophisticated model. The model is incomplete, but I believe it takes us several steps further toward a complete model of the reading process.

Overlapping the Programmable Processing Structures

In the preceding section, as in the original word perception model, we considered words of a fixed length, and we assumed that some mechanism not included in the model would solve the problem of aligning the inputs appropriately with the programmable letter units in each of the local modules.

Obviously, any plausible model of reading must be capable of accommodating arbitrary strings of text, including words of different lengths, without requiring prealignment of each word with a specific location in a module.

In the TRACE model described in the previous chapter, Elman and I dealt with this problem by allowing units in adjacent slots to have overlapping "receptive fields." The ideas from that model can be applied to reading as follows. Suppose we have several sets of letter units, one for each of a reasonably large number of letter positions in a string of text. Each of these sets of units will be the top end of a position-specific letter-processing channel, like the ones in the original word perception model. Let's suppose we can present letter strings so that the left letter projects to any one of the letter channels and adjacent letters in the string activate letter units in adjacent slots. Then the model would be able to process words starting in any slot if there were a number of overlapping word processing channels, one starting in every slot. If a string like *BINK*/4 was shown (*BINK* starting in the fourth letter position), units for *BLINK*/3, *INK*/5, *BIN*/4, and *SLINKY*/3 would all be activated, along with units for *BIND*/4, *SINK*/4 and others than would have been activated in the original model. These units would all produce feedback to the units for the letters they contain in the appropriate positions. This would allow conspiracies of partial activations of words of various lengths. The word units could also compete with each other