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.

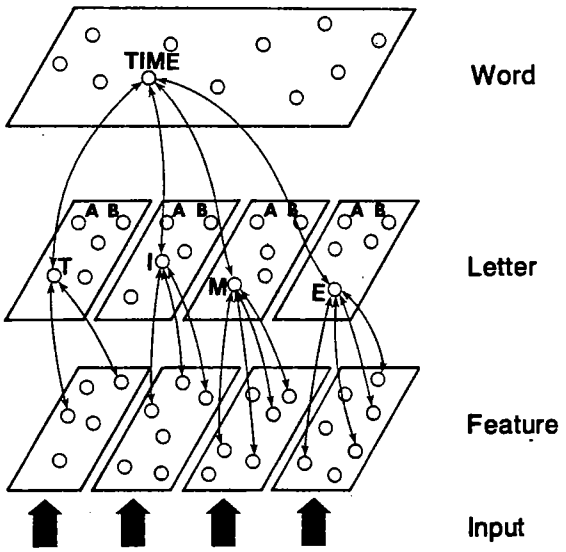

FIGURE 1. A sketch of the interactive activation model of word perception. Units within the same rectangle stand for incompatible alternative hypotheses about an input pattern, and are all mutually inhibitory. The bidirectional excitatory connections between levels are indicated for one word and its 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

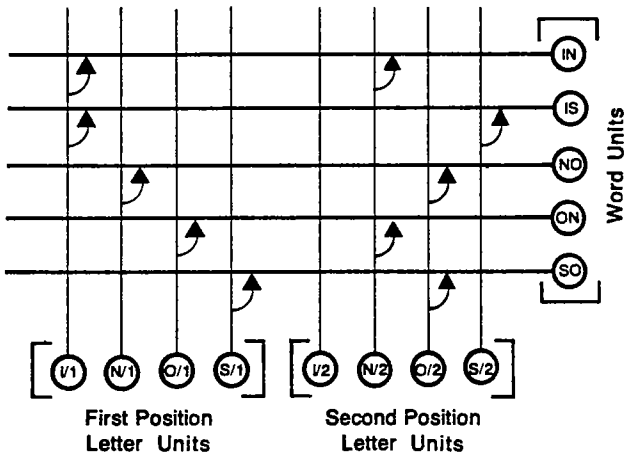

FIGURE 2. An extremely simple connectionist mechanism, capable of processing one two-letter string made up of the letters *I*, *N*, *O*, and *S*. The model knows only the five words that can be made of two of these letters, namely *IN*, *IS*, *NO*, *ON*, and *SO*. No top-down connections are included 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

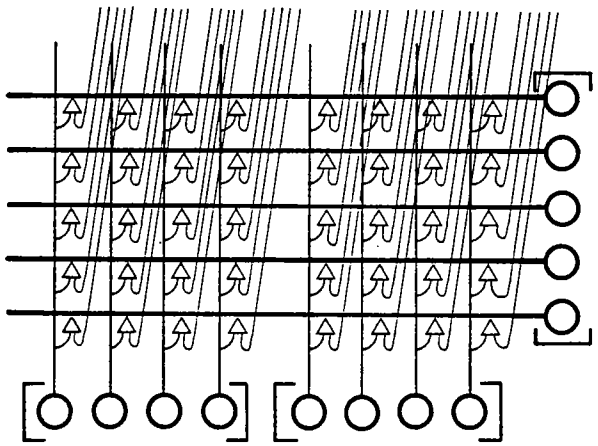

FIGURE 3. A programmable version of the simplified activation model shown in Figure 2. Each triangle represents a *programmable connection* that can be turned on by a signal coming from the central knowledge store, shown here as lying outside the figure to the upper right. If the triangular connections 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

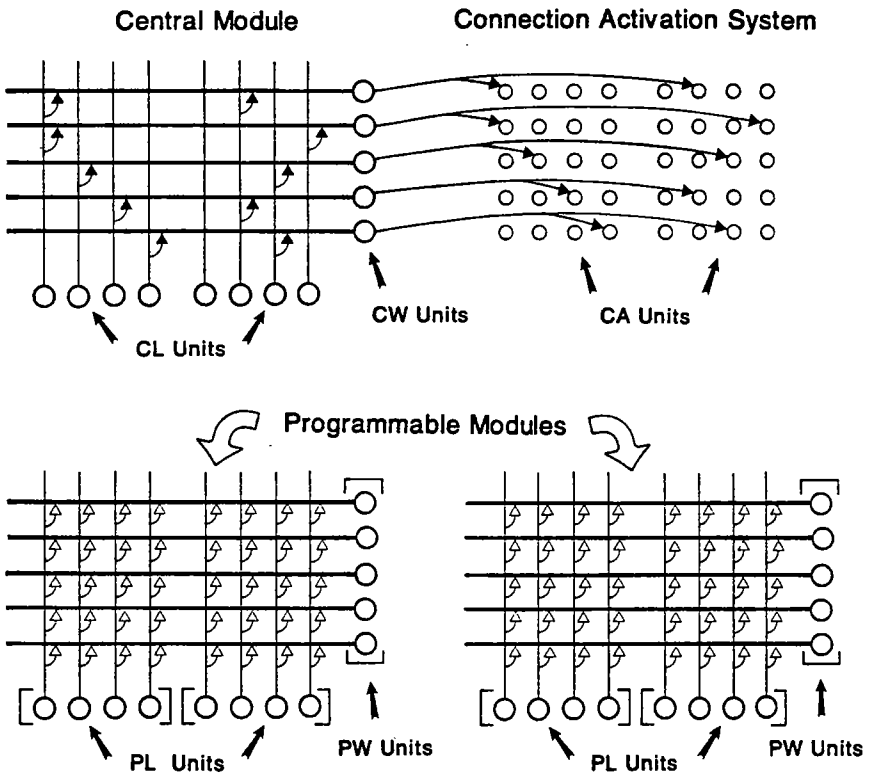

FIGURE 4. A simplified example of a connection information distribution (CID) mechanism, sufficient for simultaneous bottom-up processing of two two-letter words. The programmable modules consist of the programmable letter (PL) units, programmable word (PW) units, and programmable connections between 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

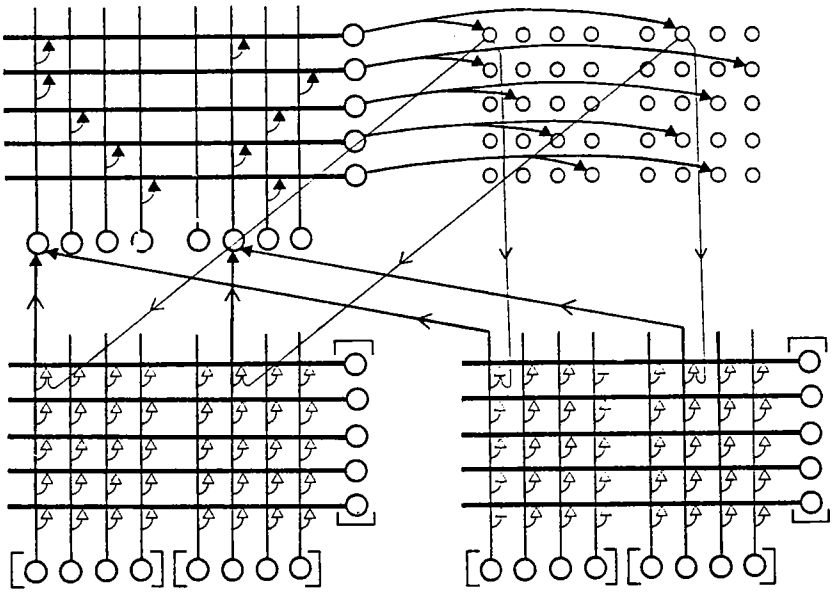

FIGURE 5. Each CA unit projects to the corresponding connection in both programmable modules, and each central letter unit receives projections from the corresponding programmable letter unit in both programmable modules. The inputs to two central letter units and the outputs from two CA units are shown. (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 Target Cue	SAND	LANE

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.

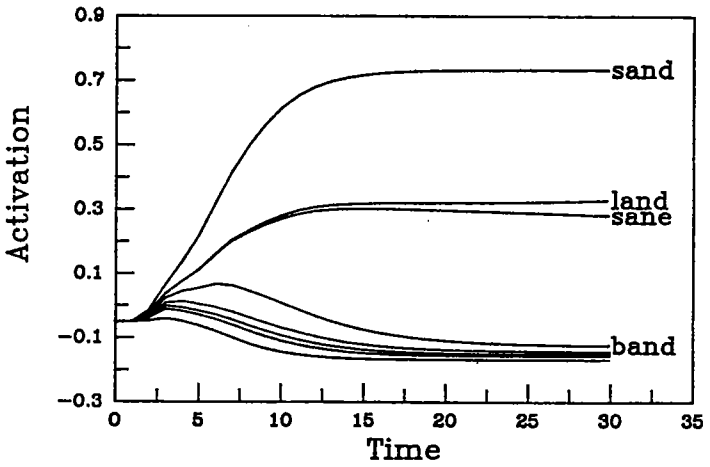

FIGURE 6. Activation curves for various programmable word units in the module to which *SAND* is shown when the input to the other module is *LANE*. The horizontal axis represents time cycles from the onset of the two-word display. (From "Putting Knowledge in its Place: A Scheme for Programming Parallel 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

to the extent that they spanned overlapping letter sequences, thereby allowing the model to settle on the best interpretation of each substring of letters separately. A hardwired version of this idea was implemented in the TRACE model of speech perception; the task now is to apply the same idea to a programmable processing structure.

The extension of the connection information distribution scheme to overlapping modules introduces a new problem. To see the problem, let's first look at a simple example of a hardwired word perception model with overlapping slots. Figure 7 illustrates such a case—a hardwired "blackboard" set up for processing the two-letter *I-N-O-S* words *IN*, *IS*, *NO*, *ON*, and *SO*, starting in any one of several letter positions. Each set of word units receives inputs from two adjacent sets of letter units. Members of each letter slot are mutually inhibitory. Members of word slots are mutually inhibitory to the extent that the words they stand for overlap at the letter level. As in the TRACE model, the idea is that word units should only compete in so far as they represent alternative interpretations of the contents of the same location.

A two-letter word can be presented to letter units in any two adjacent letter slots. The appropriate unit in the appropriate word slot will then become active. Thus, the second letter slot can represent the second letter of a two-letter word starting in the first position or the first letter of a word starting in the second position.

Let us consider what is involved in making a programmable version of this model. First, we would need to replace the hardwired connections in the figure with programmable connections. We would need a

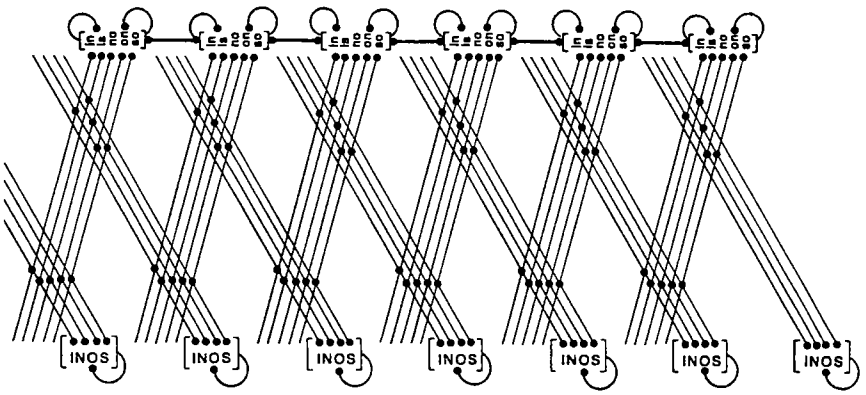

FIGURE 7. A hardwired processing structure for bottom-up processing of the words *IN*, *IS*, *NO*, *ON*, and *SO* presented to any two adjacent letter slots. Note that each letter slot participates in two different word slots, except at the edges.

lexical knowledge source, or central knowledge system, consisting of two sets of letter units, one for the first letter of the word and one for the second letter, and a set of word units. In addition, we would need a set of connection activation units, one for each programmable connection in each overlapping word-processing channel. Finally, we would need converging input lines coming into the central knowledge structure and diverging output lines from the CA units to the programmable connections.

A problem arises with the converging input lines. Since each letter slot can contain letters in either role within the word, the letter units in each letter slot would have to send outputs to the central letter units in each central letter channel. The problem with this would be that every occurrence of a particular letter, regardless of what role it turned out to play within a word, would turn on the connections for words which have that letter in them anywhere. Consider, for example, the display *–ON–* (dashes indicate blanks in the display). This will turn on the connections for all words having *O* in them anywhere, since the *O* will be treated as a potential first letter of a word beginning in the second position and as a potential second letter of a word beginning in the first position. Similarly, the *N* will turn on the weights for all words having *N* in them anywhere, since it will be treated as a potential first letter of a word beginning in the third position and a potential second letter of a word beginning in the second position. For this example, then, the connections for *NO* will be turned on just as much as the connections for *ON*. When multiple words are presented, this would tend to cause the model to make migration errors for nonhomologous positions. For example, in a version of the model with a forth overlapping module, the display *IS-NO* would activate the connections for the word *IN* just as much as the display *IS-ON*. We would therefore expect subjects to report *IN* as an incorrect report of the contents of the first two letter positions equally often in these two cases. More generally, migration errors would not be expected to honor relative position within the word. Analyses reported in Shallice and McGill (1978) reveal that nonhomologous migration errors are quite rare indeed.

Role-specific letter units. It appears that we need to have some way of treating letters differently depending on their role within the word. One poor way to do this would be to imagine that there is a separate detector in each letter channel for each letter in every role it might play. For our two-letter word case, letters can occur in either of two roles—as the first letter in a two-letter string, or as the last. So, if we went with role-specific letter detectors, each letter slot would contain eight letter units, one for each letter as the first letter in a word, and one for each letter as the second letter in a word.

Obviously this solution has serious weaknesses. For one thing, it duplicates units ad nauseam. If we wished to process longer words, we would need very large numbers of role-specific units in each letter slot.

Coarse coding. There is a better way. Suppose that each letter is represented, not as an activation of a single unit, but as a pattern of activation involving several active units. With such an encoding, we would imagine that a given letter—say, the letter *I*—produces a slightly different pattern when it occurs in different roles. All these different patterns will represent different versions of the letter *I*, and to the degree that they are similar they will have similar effects, but their differences can be used to produce a sufficient degree of differentiation of letters in different contexts to allow our model to work and to reduce the incidence of nonhomologous migration errors. This idea is an instance of the idea of coarse coding, which is described more fully in Chapter 3.

In the present application of the coarse coding idea, there are four detectors for each letter. Each detector serves as a partial specification of a letter/role combination. One of the detectors is activated when the letter is preceded by a blank; one when it is preceded by any other letter; one when it is followed by a blank; and one when it is followed by any other letter. With this scheme, the pattern of active letter/role units can indicate whether a particular letter is a separate string unto itself (blanks on both sides); the first letter in a string (blank on left, letter on right); the last letter in a string (letter on left, blank on right); or an internal letter in a string (letters on both sides).

Notationally, I will designate particular units by the letter they stand for in upper case, preceded or followed by an underbar to indicate a blank or by an *x* to indicate any letter. The code for a letter in a particular role would be given by two such specifications in the same parentheses, with a space between them. The code for an entire word is given by a series of letter codes in square brackets. For example, the word *EVE* would be coded [(_*E Ex*) (*xV Vx*) (*xE E*_)].

One property of this coding scheme is that similar letter roles produce overlapping patterns of activation. For example, the code for the letter *I* at the beginning of a word, (_*I Ix*), overlaps with the code for the same letter in medial position, (*xI Ix*). It also overlaps with the code for the same letter in isolation, (_*I I*_). Similarly, the code for a letter in final position, (*xI I*_), overlaps with the code for the same letter in medial position, (*xI Ix*), and with the code for the letter in isolation. However, there is no overlap of the codes for initial and final letters—disjoint subsets of units are activated in the two cases. Similarly, there is no overlap of codes for isolated letters and medial letters in strings of three letters or more.

Obviously this particular code could be improved upon to make all occurrences of a particular letter have something in common and to make nonterminal letters in different positions differ from each other. However, this scheme captures the essence of the coarse coding idea and works surprisingly well for words up to four letters long, so we will stick with it for now.

For our two-letter word example, this coding scheme does not buy us much. The advantages of the scheme only begin to emerge with longer words. However, for visualization, I represent a version sufficient for the two letter *I-N-O-S* words in Figure 8; some of the connections from the CA units to two of the programmable modules are shown in Figure 9. For two-letter words, we don't need all four different role specifications, so I've only included the *_x* and *x_* units where *x* stands for *I*, *N*, *O*, or *S*.

With these coarse-coded letter/role units in each letter channel, the units in each programmable letter channel project to a single central letter channel. Different role specifications are still preserved, so that the word *ON* will activate no central units in common with the word *NO*.

Other Enhancements in the Programmable Blackboard Model

Feedback. In describing CID, I did not discuss word-to-letter level feedback, although a version of that model has been implemented that incorporates this. For the programmable blackboard model, I wished to incorporate feedback to illustrate how the overlapping modules in CID allow feedback from activations of words of different lengths appropriately aligned with the input. I assume feedback is implemented by another set of programmable connections running from the programmable word units back down to the programmable letter units. (It is important to distinguish the connection activation process from word-to-letter level feedback. They are different things serving very different roles in the model). Because of the symmetry of the bottom-up and top-down activations, the very same set of CA units can be used to program both the bottom-up and the top-down connections. That is, the same CA unit that turns on the bottom-up connection from a particular letter unit to a particular word unit can also turn on the connection from that same word unit to the same letter unit.

Shifting the focus of attention. While there is some reason to believe we process more than one word at a time, it is clear that we do not process whole pages or even whole lines of text at a time. In reading, the eyes make saccadic jumps from word to word, stopping for

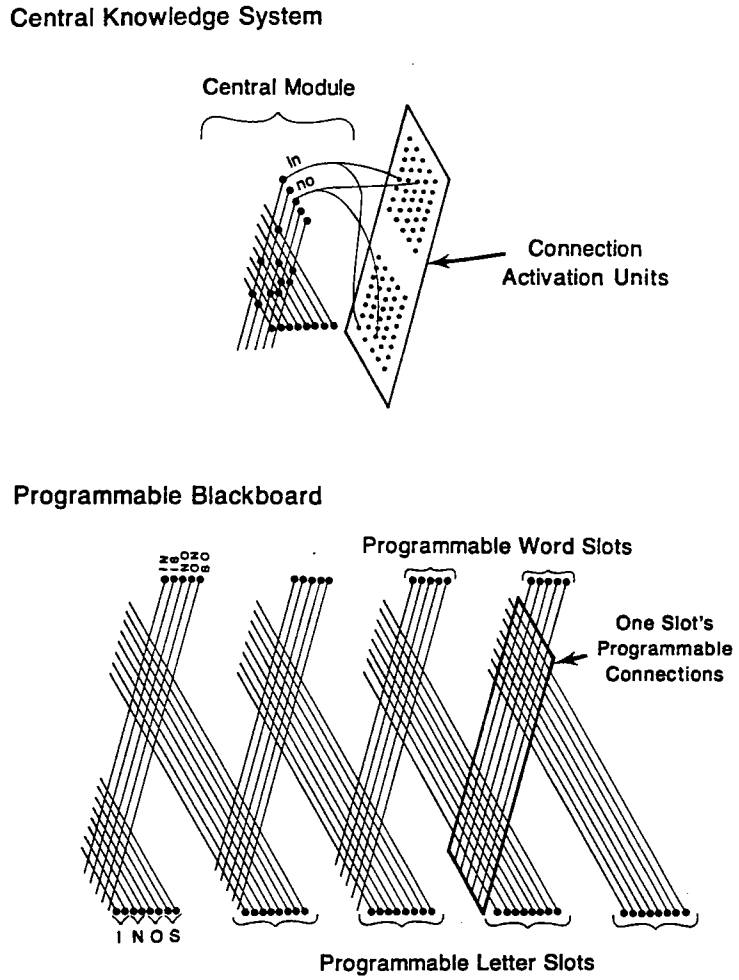

FIGURE 8. A simplified programmable blackboard sufficient for processing the two-letter *I-N-O-S* words, using coarse coded representations at the letter level. The hardwired connections between the central letter and word units are illustrated, as are the connections from the central word units to the CA units for the connections needed to process two of these words. The programmable letter units in each letter channel project to the central letter units, and the CA units project back to the programmable connections between the letter units and the word units. Some of these projections are shown in the next figure.

200-500 msec about once a word. Shorter words may be skipped and longer words may be fixated twice.

From these data, we might infer that the reading process is proceeding in a strictly sequential fashion, one word at a time. For certain

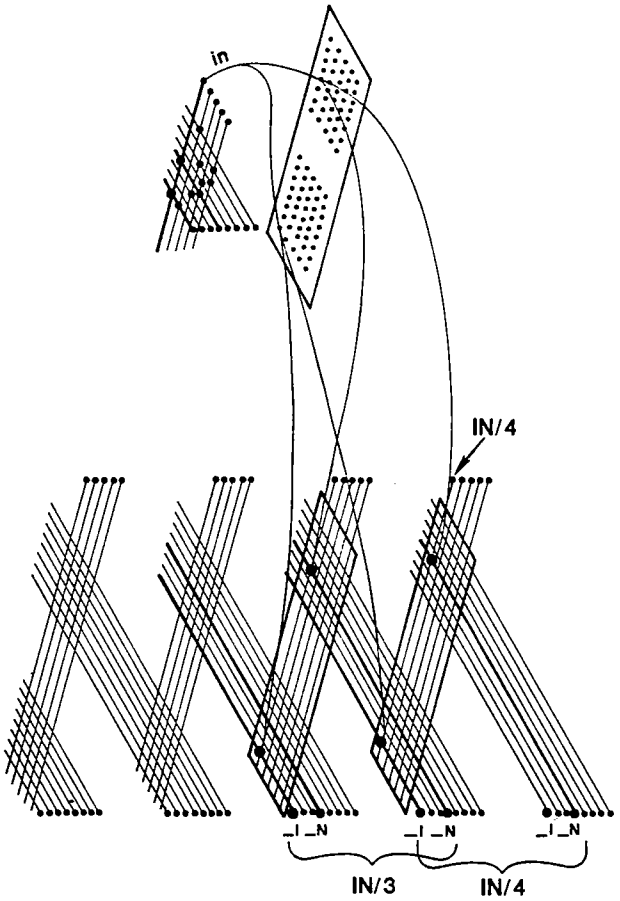

FIGURE 9. A replica of Figure 8, with the outgoing connections from the central word unit for *IN* to two of the programmable modules via the CA units. Also shown are the units for *_I* and *N_* in the third, fourth, and fifth letter slots, and the unit for *IN* in the fourth programmable word slot. The programmable letter units in each letter channel project to the central letter units, and the CA units project back to the programmable connections between the letter and the word units, but these connections are not actually shown.

purposes, as an approximation, it may not be unreasonable to treat it so. But there is evidence that we can process more than one word per fixation, though there is some degradation of performance with an increase in the contents of the display. Of course, this finding is compatible with sequential processing within fixations, but I will assume that within a fixation, all processing activity is taking place in parallel. However, I will assume that the higher resolution of the fovea and the

focusing of attention to the right of fixation causes some parts of the contents of the fixation to be processed more successfully and more completely than others.

A sketch that may help visualize these assumptions is shown in Figure 10. Several sets of programmable word and letter units are illustrated, with schematic illustrations of the connections between and within slots. Along the bottom, a fragment of a possible line of text is displayed with the letters lined up below the letter slots. The assumption is that the letters lying within the foveal region and the right penumbra are mapped into successive slots in the programmable blackboard. Converging inputs from these slots, and only these slots, are sent to the central letter units, and connection activations from the connection activation structures are projected back to the programmable connections in these slots only.

Connection activations are assumed to be "sticky": That is, it is assumed that once a connection has been programmed by the central knowledge structures, it stays that way. This allows interactive activation processes to continue in older parts of the programmable

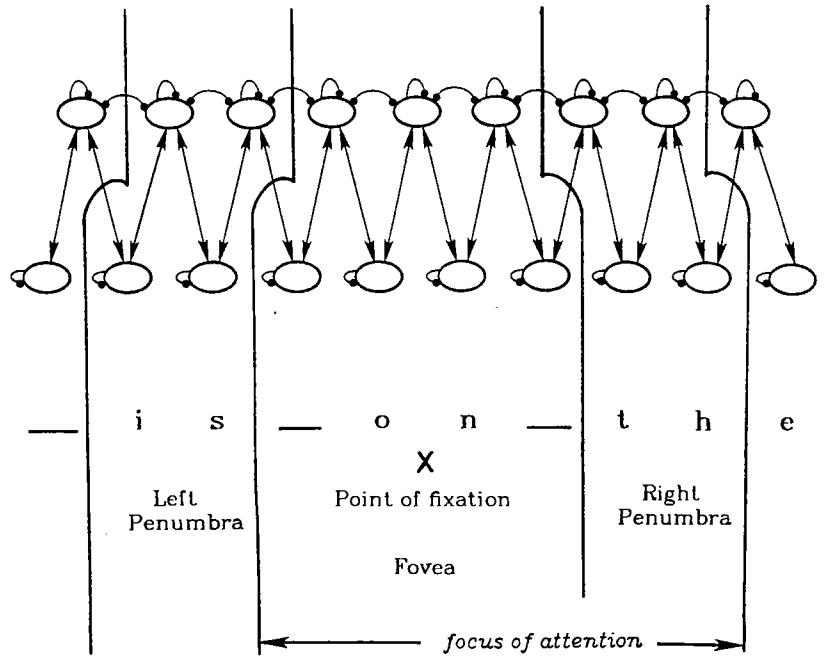

FIGURE 10. Schematic illustrations of the programmable blackboard, with illustrations of the fovea, the left and right penumbras, and the span of the focus of attention during a fixation in reading.

blackboard while they are being set up in newer parts as the eye moves along from left to right. So, what has already been processed in earlier fixations can continue to influence the processing of the contents of the current fixation, and, in case of unresolved ambiguities, can be influenced by the new information coming in.

To make this model work, we would need to imagine that when the eye moves, we change the mapping from a low-level, retinocentric representation of the contents of the current fixation into the programmable blackboard. The mapping mechanism proposed by Hinton (1981b) would be appropriate for doing this, assuming the activations of the "mapping units" could be updated in tandem with the commands to move the eyes. Hinton's scheme can also be used to map from the programmable letter units into the central letter units and from the connection activation units to the appropriate programmable connections.

Details of the Simulation Model

The simulation model developed to embody these assumptions is called PABLO, an approximate acronym for Programmable Blackboard. PABLO uses an alphabet of 9 letters, including blank (written as "_") and the letters *A*, *C*, *D*, *E*, *H*, *I*, *N*, and *T*. Its lexicon consists of all the words of one to four letters made up only of those letters that were found in a large computerized dictionary. Archaic forms, past tenses, plurals, proper nouns and abbreviations were deleted, leaving a lexicon of 92 words, as shown in Table 5. Also, blank was treated as a (special) one-letter word. As in TRACE (Chapter 15), blanks served to provide competition against words that would otherwise tend to invade the spaces between words.

PABLO contained a sufficient programmable blackboard to read lines of text 20 characters long. To accomplish this, it contained 20 overlapping programmable modules, each capable of being programmed for processing all of the words in the lexicon. Each module consisted of a set of programmable word units, four sets of programmable letter units shared with adjacent modules, and two sets of programmable connections between the letter units and the word units. One set of connections was used for bottom-up connections from the letter to the word level, the other for top-down connections from the word to the letter level.

There were 93 word units in each module, one for each of the 93 words (including blank). Each letter slot consisted of a complete set of letter/role specification units. Each set contained four units for each of

TABLE 5

THE WORDS USED IN PABLO

A	ACE	ACHE	ACID	ACNE	ACT
AD	ADD	AID	AIDE	AN	AND
ANT	ANTE	AT	CAD	CAN	CANE
CANT	CAT	CEDE	CENT	CHAT	CHIC
CHIN	CHIT	CITE	DAD	DATA	DATE
DEAD	DEAN	DEED	DEN	DENT	DICE
DIE	DIET	DIN	DINE	DINT	EACH
EAT	EDIT	END	ETCH	HAND	HAT
HATE	HE	HEAD	HEAT	HEED	HEN
HI	HIDE	HIND	HINT	HIT	I
ICE	IDEA	IN	INCH	INN	IT
ITCH	NEAT	NEED	NET	NICE	NINE
NIT	TACT	TAD	TAN	TEA	TEE
TEEN	TEN	TEND	TENT	THAN	THAT
THE	THEN	THIN	TIDE	TIE	TIN
TINE	TINT				

the nine letters, one for each partial role specification described above. Thus, there were 36 letter/role units in each letter slot.

Each complete programmable module included four letter slots. Thus, module i included letter slots i, $i + 1$, $i + 2$, and $i + 3$.

As in TRACE, competition between word units depends upon the extent of the overlap of the words the units represent. That is, each word unit inhibits every other word unit once for every letter position in which they overlap. Thus *CAT*/1 and *HAT*/1, which overlap in all three letter positions, inhibit each other three times. *CAT*/1 and *ATE*/2, which overlap in two letter positions, inhibit each other twice. Again, for simplicity, these inhibitory connections are hardwired.[3]

In addition to these programmable processing structures, there is a central knowledge structure consisting of a central module and a connection activation system. The central module consists of one set of central letter units, one set of 93 central word units, and one hardwired

[3] For these connections to be hardwired, knowledge of the length of the word the local word units will be allocated to must be built into the local modules. This state of affairs would obviously be unacceptable in the ultimate version of the model. These inhibitory interactions could, of course, be programmed, as suggested earlier. An alternative scheme, in which patterns of activation are coarse-coded at the word as well as the letter level, would get around the problem by using a kind of "natural competition" that occurs between patterns competing for activation over the same set of units.

connection matrix linking the central letter units to the central word units. The central unit for each word also has hardwired connections to the CA units appropriate for activating both bottom-up and top-down connections between the programmable letter units and the programmable word unit for this word.

An input to be presented to the model for processing could be a sequence of letters up to twenty letters long. The sequence could include blanks (represented by "_"). The blanks were treated as characters just as the letters were. The sequence of blanks and letters were used to specify which letter/role units should receive external excitatory input in each letter slot. The "_" character always activated the same two letter/role units no matter what preceded or followed it. So, for example, the input-specification *_CAT_* caused external input to the letter/role units for *_C* and *Cx* in the second slot, *xA* and *Ax* in the third letter slot, *xT* and *T_* in the fourth letter slot, and to the two letter/role units for "_" in the first and fifth slots.

Single and Multiple Fixations

The simulation model was studied in two modes: One involved short displays—one or at most two short words—which were presented for processing within a single fixation. The second mode involved whole lines of text. For the latter, I assumed that PABLO "read" them from left to right with one fixation per word. Fixation durations were fixed, for simplicity, though we know that they can be influenced by visual and linguistic factors (Just & Carpenter, 1980).

First I will describe the assumptions made for the multiple fixation case. It was assumed that, on each fixation, the spotlight of attention consisted of two regions: a *center* and a *penumbra*. The center of the spotlight consisted of several letter positions in the input centered around the point of fixation. The penumbra of the spotlight consisted of several more letter positions further to the right. Note that the parafoveal region to the left of the point of fixation is not included in the spotlight of attention. Evidence from a number of sources (e.g., Rayner, 1975) suggests that information in this region is not utilized.

Letters in both the center and the penumbra of the spotlight of attention are projected to the corresponding letter units. To keep things simple, I assume for present purposes that the letters in the penumbra simply produce weaker bottom-up activations than those in the center of the spotlight. A more realistic assumption would be that letters in the penumbra produce less precise activations (i.e., activations that encompass all the letters in the same shape-class) but I have foregone this assumption for now to keep things simple and as clear as possible.

The spotlight of attention falls on a particular place in the input text, and its contents are projected to the corresponding region in the programmable blackboard. This region is called the *illuminated region*. The programmable letter units in this part of the blackboard project to the central module, and the connection activation units project back to the programmable connections in the illuminated region.[4]

In single fixation mode, I assumed that both the display and the spotlight of attention were centered on the fovea. For simplicity, the entire display was assumed to fall within the center of the spotlight so that each letter in the display was given equal weight.

Details of activation dynamics. The rules for updating programmable letter and word unit activations were the same as in CID. However, the introduction of coarse coding and overlapping slots necessitated a number of changes in the quantitative parameters. The values I used were chosen so that PABLO would produce approximately the same behavior with single four-letter words as CID.

The activation assumptions for units in the connection programming loop (the central letter and word units, and the connection activator units), and for the programmable connections themselves, were kept simple as before. Two modifications were introduced, however. First, letter-level activations in the penumbra of attention were treated as weaker inputs to the central letter units than letter-level activations in the center: They only turned on the corresponding central letter units half as much as those in the center of attention. Second, activations of central word units were normalized by the length of the word. This was done so that that the connections for a word would be turned on to the same extent, regardless of word length, whenever all the letters of the word were present in the center of attention. For example, the activation of the central word unit for the word *CAT* is just a count of the active letter/role units for *CAT* (*_C*, *Cx*, *xA*, *Ax*, *xT*, and *T_*) in the center of the illuminated zone, plus .5 times a count of the active letter/role units in the penumbra, divided by 6. With the display *_CAT_*, centered in the fovea, this number would therefore be 1.0.

As before, each CA unit simply multiplies the activation value it receives from its CW unit by the appropriate global excitatory constant,

[4] An alternative conception would be to imagine that the contents of the fixation are always projected directly to the central module and to the illuminated region of the programmable blackboard. There are reasons for preferring this view, one of which is that it opens up the possibility that the information projected into the central module might be of higher grain than that projected to the programmable blackboard. This is desirable because higher fidelity is needed in the connection activation process than in the programmable blackboard once the correct connections have been activated. (See Chapter 12.)

depending on whether the connection is letter-to-word or word-to-letter, and passes the result on as the activation of the programmable connections it projects to in each local matrix. Projections into the penumbra were assumed to be half strength.

Finally, as noted above, the model assumes that programmable connection activations are "sticky"—they tend to retain the values programmed into them when the spotlight of attention moves on. For simplicity, I adopted the most radical version of this assumption. Input from the central module can increase the activation of a programmable connection, and, once activated to a level, it stays at that level while the rest of the line of text is processed. At the beginning of a new line, of course, the slate is wiped clean.

One Fixation at a Time

Simultaneous resolution of two tokens of the same ambiguous character. Consider the ambiguous character in *THE CHT*. Let us see what happens when this display is presented to PABLO. We assume that the whole display, with blanks on both sides, fits inside the center of attention, with the first *T* lined up with the second input position, so that each letter activates all of the appropriate letter/role units in the appropriate positions. For the two slots to which the *H* character is presented, we assume that *xH* and *Hx* and *xA* and *Ax* are all activated to equal degrees. PABLO will end up selecting *xH* and *Hx* for the first *H* and *xA* and *Ax* for the second *H*, in parallel.

Once the letter/role units have been activated in the appropriate letter level slots, they will activate the central word units for various words to various degrees. For example, the central word unit for the word *THE* will receive eight excitatory inputs: from the *_T* and *Tx* at the beginning of *THE*, from the *xH* and *Hx* in *THE*, from the *E_* and *xE* at the end of *THE*, and from the *xH* and *Hx* in the middle of *CHT*. The central word unit for the word *CAT* will receive eight excitatory inputs as well: two from the initial *C* of *CHT*, two from the final *T* of *CHT*, and two from each (possible) medial *A*. Normalizing for length, the central word units for both words end up getting an activation value of 8/6 or 1.33.

The central word units for other words will also be turned on to some extent. For example, the unit for the word *HAT* will receive eight excitatory inputs as well. The two *Hx* units activated by the *H* figures will each contribute one input, the two *xA* and the two *Ax* activated by the two *H* figures will each contribute one input, and the *xT* and *T_* occurring at the end of *CHT* will each contribute one input.

Note that the medial *H* turns on the weights for *HAT* half as much as an initial *H* would because it only has one letter-form feature in common with the initial *H* pattern of the word *HAT*. In any case, when we normalize again for length, the CW unit for *HAT* ends up getting an activation value of 8/6 or 1.33 as well.

The CW units for *THAT* and *CHAT* will be set at activation values of 1.25. The display contains a total of 10 tokens of the eight letter/role elements of each of these two words, and the denominator for each is 8. It is important to remember that the activation of central word units do not directly determine what is seen; they simply turn on connections in the programmable blackboard and thereby permit local letter units to interact with local word units. We now examine the activations that result from these interactions.

Letter and word activations resulting from *THE CHT* after 10 cycles of processing are shown in Figure 11. The model has successfully completed the first word as *THE* and the second as *CAT* by the end of 10

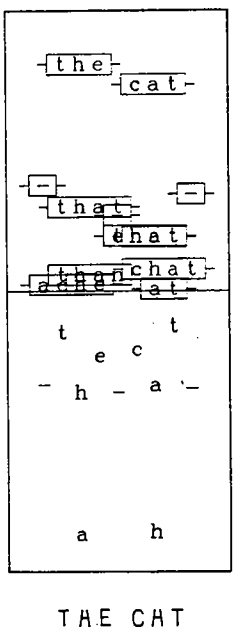

FIGURE 11. Word- and letter-level activations in PABLO in response to *THE CHT*. At the word level (upper panel), each rectangle represents a word unit. The vertical position of the rectangle indicates the activation of the unit, and the horizontal placement relative to the input shown at bottom indicates the portion of the input that the unit spans. At the letter level (lower panel), the height of each square represents the average activation of the two active letter/role units representing the letter inscribed in the square.

cycles and is in the process of reinforcing the correct interpretation of the H figure in each position through feedback. Words such as *HAT* whose weights are strongly activated play a stronger role in the competition than words such as *CAN*, which contain letters that do not occur anywhere in the input string, but even *HAT*, as well as *THAT* and *CHAT*, are eventually driven out. After 30 cycles (not shown) *THE*/2 and *CAT*/6 are the only word units left active, and, at the letter level, the *H* has beaten *A* in *THE* while *A* has beaten *H* in *CHT*.

Conspiracies of Words of Different Lengths

So far what we have illustrated could have been done in a three-letter word version of CID, though the details about what words got activated would have been somewhat different. Now, however, we can present the words starting in any letter position—as long as they do not fall off either end of our series of overlapped modules—and they can be of any length up to the maximum length allowed by the structural parameters of the model. An even more important advantage of PABLO is that, when nonwords are shown, they can profit from conspiracy effects arising from words of different lengths. The mechanism causes the activations of the conspirators to be aligned appropriately with the input.

PABLO has been tested with each of the words that it knows and with pseudowords made from each word by replacing either a vowel with a vowel or a consonant with a consonant. In all cases, the letter string was presented alone, surrounded by blanks. The model works well with either words or pseudowords. When a word of any length is shown surrounded appropriately by blanks, the appropriate word unit always wins the competition. When an nonword is shown, both shorter and longer strings can participate, as shown in Figure 12. In the figure, several representative conspiracies are shown for nonwords of four, three, and two letters in length. Though in some cases one word the model knows is a much better match than all the others, and therefore dominates the competition, in other cases there is a much more homogeneous conspiracy. Generally speaking, words both longer and shorter than the item shown get into the act—though of course no words longer than four letters are activated, because they have not been included in the model.

Within-word crosstalk. One idiosyncrasy of PABLO needs to be mentioned. It has a tendency to activate words that have repetitions of the same letter more strongly than words that do not, even when only one copy of the letter is shown. For example, *DAT* activated *DATA*

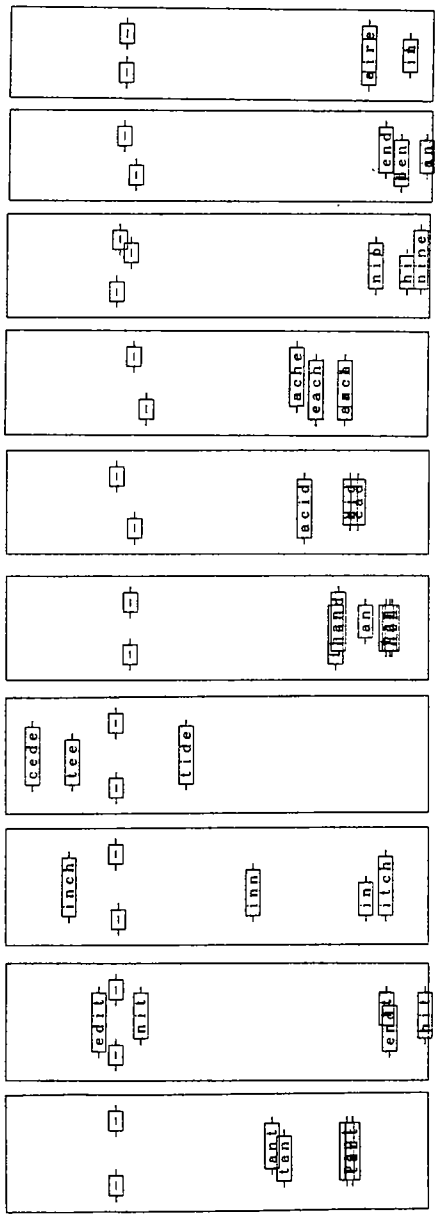

FIGURE 12. The sets of conspirators active at the word level 10 cycles after presentation of the display shown at the bottom of each column. Relative height corresponds to activation. There are several cases in which two or more words are overprinted on each other, particularaly in the cases of *HAN* (where *RAN, TAN, PAN, CAN, HAT, HAD,* and *HEN* are all active) and *TANT* (where *RANT, CANT, PANT, TART, TACT, TENT,* and *TINT* are all active).

more strongly than it activated *DATE*. The reason is that the *xA* unit activated by *DAT* turns on the weights for *DATA* two times—that is, there is a double excitatory link from this letter/role unit to the CW unit for *DATA*. At the present, I know of no data to indicate whether this characteristic is consistent with human behavior or not.

There is another kind of within-word crosstalk that occurs in PABLO for which there is evidence. Connections for words that are anagrams of the display are strongly activated, particularly if they involve switches of the two internal letters. For example, the display *CLAM* activates the connections for *CALM* as much as the connections for *CLAM*, and the display *BCAK* activates the weights for *BACK* as much as *BACK* itself does. These connection activations make *CALM* a reasonably likely error response when *CLAM* is shown and make *BACK* a very likely error response when *BCAK* is shown. Johnston, Hale, and van Santen (1983) have reported evidence of such "transposition" errors in word perception. Generally, such errors are attributed to imprecise positional specificity, so that letters in one position activate detectors for words having these same letters in adjacent positions. In PABLO, such effects occur for a slightly different reason: It is not that the model does not keep straight which letter occurred where, it is just that the description of a letter that is projected into the central module is very similar to the description that would be projected by the same letter in a similar role. The results of simulations illustrating these effects are shown in Figure 13. One feature of these data was originally puzzling when Johnston et al. first reported them. On the one hand, it seemed that subjects had very accurate letter position information, since they only rarely rearranged letters in real words to make other real words. On the other hand, they acted as though they had poor letter position information when the rearrangement of the string formed a word but the display itself did not. In PABLO, the reason for the difference is simply that in the first case, there is a word that matches the input better than the transposition word; due to competitive inhibition, the correct word dominates the transposition word. When there is no correct word, the transposition word is not dominated, so it may become quite strongly activated. This effect is clearly illustrated in the simulation shown in Figure 13.

Effects of Display Length on Amount of Feedback

One additional feature of PABLO is that shorter displays receive less benefit from feedback than longer displays. This is true, even though I have normalized the activation of central word units, and therefore of

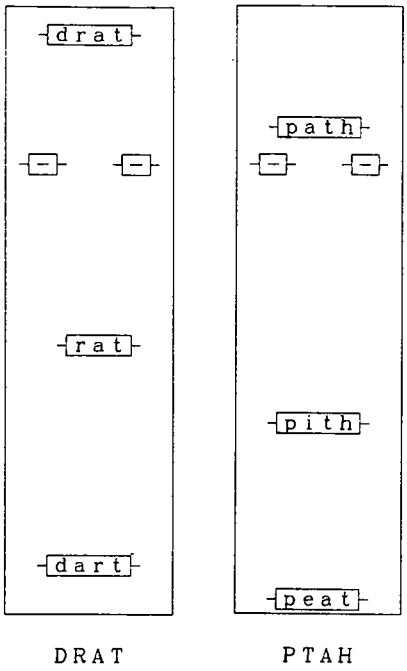

FIGURE 13. Word-level activations produced by two displays which can be made into words by rearranging the central two letters.

connection strengths, for word length. The reason for this is simply that shorter displays activate fewer letter/role units than longer ones, and therefore provide less excitatory input to the word level than longer displays. Word units are less strongly activated than they are by longer displays. Consequently, less feedback is generated. This occurs both for word and for pronounceable nonword displays. The activations shown in Figure 12 illustrate this aspect of the model in the case of pronounceable nonword displays.

At first I found this characteristic of the model undesirable. However, recent evidence collected by Samuel, van Santen, and Johnston (1982) is consistent with the model as it is. Samuel et al. examined accuracy of letter identification for letters in one-, two-, three-, and four-letter words and random letter strings and found that letter identification accuracy increased as a function of word-length for letters in words but not for letters in random strings. The advantage for letters in words over letters in random strings was either very small or nonexistent for one-letter words and grew as the words grew longer, up to about four or five letters. Thus, the data appear to support this aspect of the model's behavior. It should be noted that this aspect of the

model does not depend on the connection activation aspects of PABLO but would occur in a hardwired version of the original model, as long as the net excitatory input to the word unit from a complete specification of the word varied linearly with the length of the word.

Sequences of Fixations

Up to now I have examined the behavior of PABLO within single fixations. Now, we will examine the model as it reads along a line of text. The present version of the model is fairly rudimentary, and there are a lot of important properties of human reading that it does not capture. However, it does read in a limited sense: It makes a series of fixations, and builds up a representation of the letters and words in its programmable blackboard as it moves along, as illustrated in Figure 14. PABLO's reading in this simulation is not affected by semantic and syntactic constraints since it lacks these higher levels. Furthermore, as it presently stands, PABLO does not even monitor its own performance. Instead, each fixation lasts a fixed number of cycles. PABLO does adjust the point of its fixation for the word it is viewing, however: It fixates on the center of words containing an odd number of letters, or the middle of the letter before the center for words containing an even number of letters.[5]

Although it lacks many of the features of Thibadeau, Just, and Carpenter's (1982) READER model, PABLO does show one property that READER does not: It integrates information over successive fixations. That is, while it is looking at one word in a particular fixation, it is also gathering some information about the next word. This is, of course, something that human readers do also (Rayner, 1975; for a dissenting view, see McConkie, Zola, Blanchard, & Wolverton, 1982).

We can see that PABLO is picking up information from peripheral vision in Figure 14, but it is hard to tell from the figure how much difference this makes. To get a handle on this matter, I repeated with PABLO the same experiment Rayner (1975) did to demonstrate the pickup of peripheral information in human readers. That is, I adjusted the display during PABLO's saccade, so that what the model saw in the periphery and what it saw when it later fixated the previously peripheral location was not necessarily the same. Rayner showed that when the

[5] The reader should be clear about the fact that PABLO's fixation shifting apparatus is *not* simulated using interactive activation processes, though I think it could be. A saccade for PABLO simply amounts to moving a pointer into the array of input letters and another into the programmable blackboard.

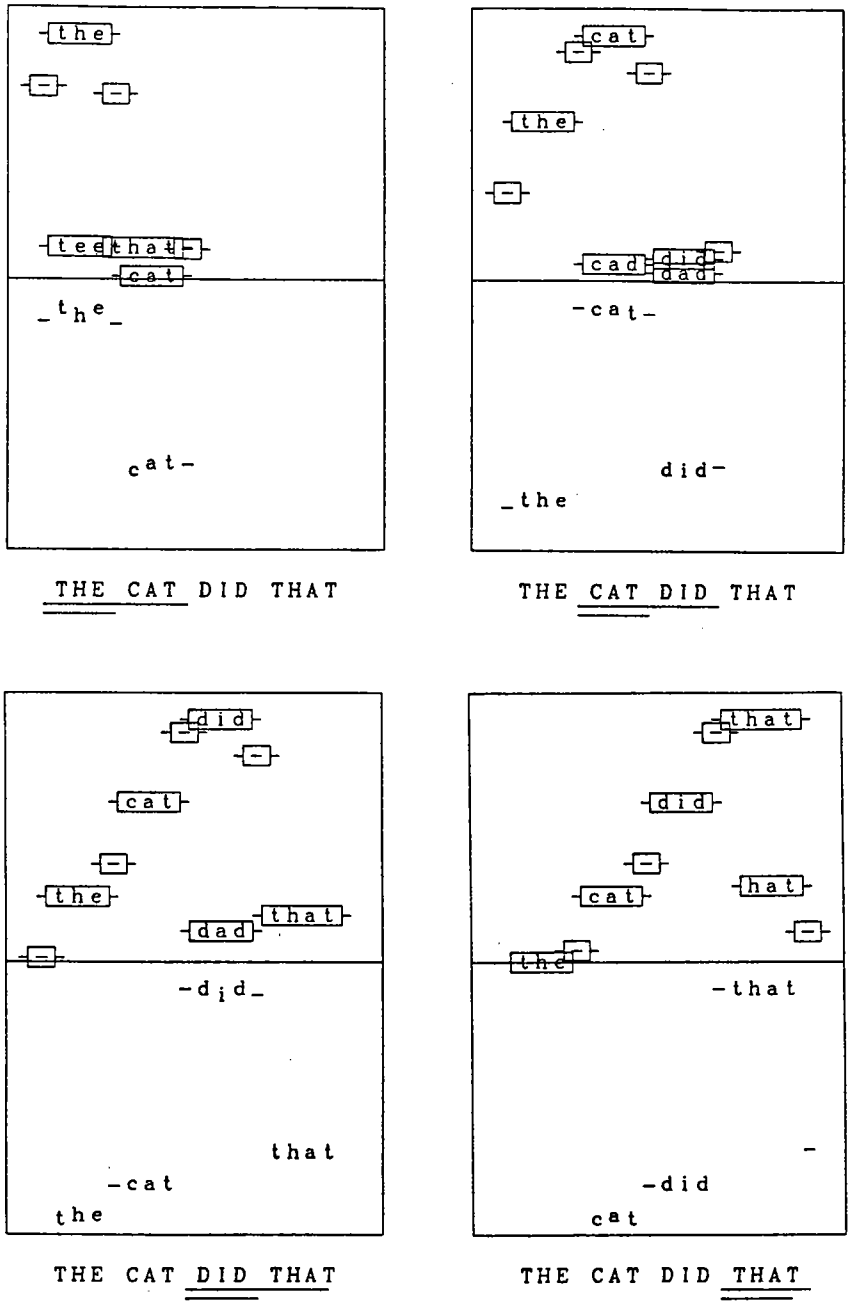

FIGURE 14. Pattern of activation at the letter and word level in PABLO, at the end of each fixation, as the model reads along a line of text. Location of the fovea and penumbra of the spotlight of attention is indicated with underbars.

information in peripheral vision differed from the information later fixated directly, subjects spent longer fixating the word than they did in the normal case in which the preview and the foveal view were the same. Other experiments have shown that subjects can name the fixated word more quickly if it matches the preview than if the preview is different (Rayner, McConkie, & Ehrlich, 1978).

Figure 15 illustrates PABLO's behavior in the following experiment. The model was given a two-word phrase and read it in two successive fixations. During the second fixation, when the fixation point was centered on the second word, the two-word phrase was always *CAR RIDE*. During the first fixation, however, there were different versions of the second word. In one case, the second word was intact on the first fixation. In the second case, the two internal letters were replaced with other letters having the same general shape. In the third case, all four letters were changed.

The figure shows that the activation of the word *RIDE* is greater at the end of the second fixation in the first case than in the other two cases. In the third case, the activation of *RIDE* is considerably less at the end of the second fixation than it would be without any preview of the second word at all (indicated by the curve marked *XXXX* in the figure). Generally, PABLO shows a facilitation, as it does in this example, when the preview matches the target. It also generally shows an interference effect when there is a gross mismatch between preview

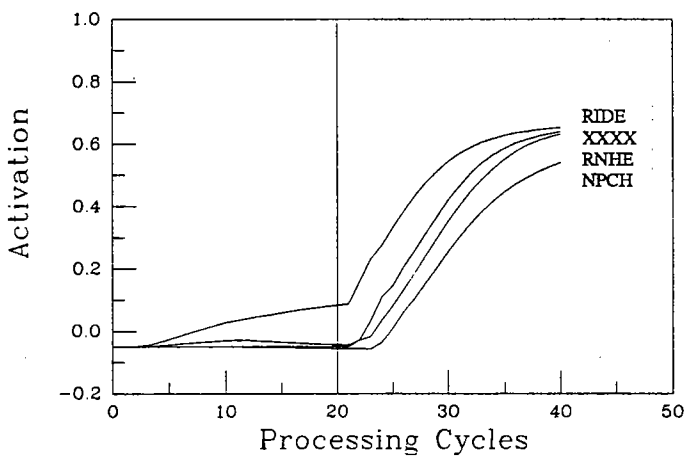

FIGURE 15. Time course of activation of the programmable word unit for the word *RIDE* in the sixth word channel of the programmable blackboard when the preview was *RIDE*, *XXXX*, *RNHE*, or *NPCH*. The *XXXX* marks the case of no preview information at all, as though there had never been a first display.

and target. For intermediate cases, it can show either a facilitation or an interference, depending on the target and the exact distortion of it. On the average, the intermediate condition shows a slight interference effect when two letters are changed, as in the example shown in the figure.

To relate these results to Rayner's, we would begin by assuming that human readers, unlike PABLO, move on to the next fixation when their certainty of the contents of one fixation reaches a satisfactory level. In the simulation, PABLO has a fixed duration for each fixation, but if the fixation was terminated when the activation of one of the words reached some criterion, then the duration of the fixation on the target would have been shortest in the case where the preview matches the target and longest in the case where the two differ the most, just as Rayner found.

Thus, PABLO integrates information over successive fixations, and conforms to the main result of Rayner's experiment. Though PABLO does not capture all the details, I would expect it to capture more of them if some of the simplifying assumptions were replaced with more realistic ones. For example, end letters appear to be more critical to human readers than to PABLO, and people are sensitive to the similarity of shape between original and replacement letters. A more elaborate version of PABLO, including a real feature level, would surely show these effects.

DISCUSSION

I have described two models that rely on the same basic principles of connection information distribution from a central knowledge source to local, programmable modules. CID, the first model, illustrated some basic properties of the connection information distribution scheme, but it had serious limitations because of its fixed-length, nonoverlapping programmable modules. PABLO overcame this deficiency of CID and provided a framework for beginning to think about the problem of reading by a series of fixations and integrating information from one fixation to the next.

Benefits of Connection Information Distribution

At this point, it is appropriate to step back from these particular models and take a look at the more general implications of the idea of connection information distribution. I believe this is quite a powerful

idea. Connection information distribution allows us to instruct parallel processing structures from outside the network, making their behavior contingent on instructions originating elsewhere in the network. This means, for example, that the way a network responds to a particular input can be made contingent on the state of some other network in the system, thereby greatly increasing the flexibility of parallel processing mechanisms.

Perhaps the most general way of stating the benefit of connection information distribution is to note that it is analogous, in a way, to the invention of the stored program. [6] Before programs were invented, special-purpose logic circuits were built to carry out particular tasks; the invention of the computer program made it possible for a single, general-purpose machine to carry out any one of a huge number of tasks. The use of centrally stored connection information to program local processing structures is analogous. This allows the very same processing structures to be programmed to perform a very wide range of different tasks.

More specifically, there are two computational problems facing PDP models to which connection information distribution can be beneficially addressed. Connection information distribution allows multiple tokens of the same type to be active at once. PABLO, for example, would have no difficulty processing the same word twice if it occurred in two different locations in the programmable blackboard, and higher levels would be able to make different choices from alternative word activations in different places, just as the word level in PABLO was able to make different choices between *A* and *H* for the two instances of the *H* character in *THE CAT*. The ability to treat different tokens of the same type separately is, of course, very important in such domains as sentence processing, in which several tokens of the same kind of object (e.g., noun phrase, sentence, etc.) must be kept straight and assigned to appropriate roles in larger structures.

Connection information distribution also carries out a form of what is known in production systems as "resolution," binding the right tokens in the blackboard together into higher-order structural patterns. In PABLO, we have the following situation: Activations from everywhere in the programmable blackboard (analogous for present purposes to the working memory in production system models) are projected into the central knowledge system (analogous to the production memory). Central word units (analogous to productions) are activated to the extent the input from anywhere in the programmable blackboard activates them. The problem faced in production systems is to avoid spuriously

[6] I thank Gary Cottrell for pointing out this analogy.

firing productions when the activations in working memory lack the proper overall organization and to bind the correct pieces together, so that each production has the correct set of arguments rather than a mélange of arguments coming from different places. If spurious activations occur and the bindings are done improperly, the consequence will be that the wrong things will be added to the working memory. The ultimate problem faced in PABLO is analogous. We need to avoid allowing the wrong word units to get activated in the programmable blackboard by spurious conjunctions of letters. PABLO solves this problem, not by explicitly binding productions to particular elements and checking that they occur in the right conjunctions, but by distributing connection information so that when and if the right combination of elements occurs, the appropriate word unit will be activated. This allows different elements to be bound to different higher-order structures in parallel, as we saw in the case of *THE CHT*; one of the *H*'s was "bound" to an appropriate "instantiation" of *THE*, and the other to an appropriate instantiation of *CAT*. This scheme does involve some crosstalk, as we saw, but the crosstalk is of about the same kind and magnitude as the crosstalk shown by human subjects when they have to process more than one pattern in parallel. In short, it looks as though the CID model captured pretty well some of the features of the way elements are bound together in human information processing.

Costs of Connection Information Distribution

At first glance, it may seem that the ability to program parallel processing structures is being purchased at a very high price. For it appears that, wherever we had a connection in a hardwired PDP model, we now need a lot of extra circuitry to program that connection. The cost appears particularly high in terms of connection activation units and programmable connections. If there are $n \times n$ units in a programmable module, it appears that we need n^2 programmable connections and the same number of CA units to activate the programmable connections. This can get very expensive very quickly as processing structures get to realistic sizes.

Actually, however, the situation is not as bleak as this observation seems to suggest. It turns out that with programmable modules, it is possible to process patterns with far fewer units than we would need in a hardwired parallel processing structure. The reason is that the size of a module required to process a pattern without error depends on the number of patterns the module must be prepared to process at one time. To see this, consider a module which must either be hardwired

to process three different languages or which can be programmed to process any one of them. Clearly, we can get by with a smaller module in the latter case. More generally, if we use distributed representations, simple mathematical analyses show that the more patterns a module is set up to process, the larger it must be to avoid errors in processing any one pattern that might be presented on a particular occasion. The reason why this fact is relevant is that in conventional, hardwired PDP mechanisms, the model is programmed to process all of the patterns that it knows, all of the time. With programmable modules, on the other hand, the module only gets loaded with the connection information relevant to processing the patterns whose central representations are active. Since this number is vastly smaller than the number of total patterns a person might know, the size of the programmable processing structure can be very much smaller than a hardwired processing structure would have to be to do the same work. A full discussion of these issues is presented in Chapter 12 on the resource requirements of PDP models.

One interesting result of the analysis of the resource requirements of connection information distribution is the fact that the required number of units and connections varies with the number of different patterns the network is being programmed to process at a particular time. This fact means that for a CID mechanism of a given size, there is a limit of the amount of parallelism it can handle (quite independently of the number of programmable modules). But the limit is in programming the modules, not in their actual parallel operation once they have been set up. Thus if we can program each module separately, we can let them run in parallel; though we are forced to go sequential at the level of programming the modules, they can still be allowed to process simultaneously and to mutually constrain each other though interactions with higher-level processing structures. We therefore get the main benefit we wanted parallel distributed processing for—simultaneous, mutual constraint—without reduplicating hardwired knowledge or paying an undue price in terms of programmable connections and connection activation units.

Extensions of Connection Information Distribution

A programmable version of TRACE. In PABLO, we programmed the programmable blackboard sequentially, in fixation-sized blocks. In extending the basic ideas inherent in PABLO to speech, we notice immediately that the speech signal forces seriality on us by its very sequential nature. We argued, though, in Chapter 15, for a dynamic

parallel processing structure in which we could exploit both forward and backward interactions, simply allowing new constraints to be added from the input at the right edge of the Trace, as the speech input unfolded in time. Now, to make the Trace into a programmable parallel processing structure, we can use the seriality of speech to program the Trace sequentially, as the speech comes in, thereby minimizing the resource requirements of the connection information distribution apparatus.

There would be several advantages to implementing a programmable version of TRACE. One is that retuning, priming, and learning effects could be accounted for by adjustment of the strengths of connections in the central knowledge source. Another is that the rather high cost of the connection modulation scheme for tuning feature-phoneme connections based on phoneme activations in the context could be born just once in the central knowledge structures. A third is that the central knowledge source could be retuned by global variables such as rate, etc., thereby allowing it to create in the Trace just the right pattern of interconnections to fit the expected form different possible utterances might take in the given situation. Such a model might be seen as performing an analysis of the speech signal by synthesis of the appropriate connection information, capturing some of the flexibility and context sensitivity which Halle and Stevens (1964) noted that speech perception calls for in their seminal article on "Analysis by Synthesis."

Processing syntactic structures. Introducing programmable connections into the TRACE model of speech perception will be a challenging task. Equally challenging will be the problem of extending the approach described here to higher levels of language structure. At higher levels, we must come to grips with the recursive structure of sentences, and the unbounded dependencies among elements of sentences. Some ideas about how this might be done are described in Chapter 19.

CONCLUSION

In this chapter, I have tried to argue that connection information distribution provides a way of overcoming some apparent limitations of parallel distributed processing mechanisms. Using connection information distribution, we can create local copies of relevant portions of the contents of a central knowledge store. These local copies then serve as the basis for interactive processing among the conceptual entities they program local hardware units to represent. With this mechanism, PDP

models can now be said to be able to create multiple *instantiations* of the same schema, bound appropriately to the correct local variables, though subject to just the kinds of binding errors human subjects seem to make.

Perhaps the main lesson is that some of the limitations of PDP mechanisms that connection information distribution has been proposed to overcome are more apparent than real. For I have not really done anything more than show how existing tools in the arsenal of parallel distributed processing mechanisms can be used to create local copies of networks. This is not to say that all the challenges facing PDP models have suddenly vanished. Obviously there is a long way still to go in the development of computationally adequate models of reading, speech perception, and higher levels of language processing. I hope this chapter helps to indicate that PDP mechanisms will help us get there in a way that captures the flexibility and interactive character of human processing capabilities.

ACKNOWLEDGMENTS

This work was supported by ONR Contract N00014-82-C-0374, NR 667-483, by a grant from the System Development Foundation, by a grant from the National Science Foundation (BNS-79-24062), and by a NIMH Research Scientist Career Development Award. Special thanks for useful ideas and suggestions are due to Gary Cottrell, Geoff Hinton, Dave Rumelhart, and David Zipser.