COGNITIVE SCIENCE 9, 113–146 (1985)

Putting Knowledge in its Place: A Scheme for Programming Parallel Processing Structures on the Fly

JAMES L. MCCLELLAND
Carnegie Mellon University

This paper introduces a mechanism called CID, the Connection Information Distributor. CID extends connectionism by providing a way to program networks of simple processing elements on line, in response to processing demands. Without CID, simultaneous processing of several patterns has only been possible by prewiring multiple copies of the network needed to process one pattern at a time. With CID, programmable processing structures can be loaded with connection information stored centrally, as needed. To illustrate some of the characteristics of the scheme, a CID version of the interactive activation model of word recognition is described. The model has a single permanent representation of the connection information required for word perception, but it allows several words to be processed simultaneously in separate programmable networks. Multiword processing is not perfect, however. The model produces the same kinds of intrusion errors that human subjects make in processing brief presentations of word-pairs, such as SAND LANE (SAND is often misreported as LAND or SANE). The resource requirements of the mechanism, in terms of nodes and connections, are found to be quite moderate, primarily because networks that are programmed in response to task demands can be much smaller than networks that have knowledge of large numbers of patterns built in.

Connectionism (Feldman, 1981; Feldman & Ballard, 1982) is the idea that the computations performed by a processing system are controlled by the

This research was supported by NSF grant BNS79-24062, ONR contract N00014-82-C-0374, and a Grant from the Systems Development Foundation. The author is the recipient of a Research Scientist Career Development Award 5-K01 MH00385 from the National Institute of Mental Health. This report was prepared while the author was a visitor at Bolt Beranek and Newman, Inc. The author is grateful to Alan Collins, John Frederiksen, Geoff Hinton, David Rumelhart, William Salter, and David Zipser for useful discussions, and encouragement.

Correspondence and requests for reprints should be sent to the author at the Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213.

connections among a large number of simple processing units. The processing units themselves do very simple things—generally, they simply update the strength of the signal they send based on a simple function of signals they receive from other processing elements. The intelligence of the system —what it knows and what it can do with its knowledge—is determined by the interconnections among the elements.

In designing connectionist mechanisms, one is often tempted to hard-wire the knowledge about the objects to be processed directly into the connections between the processing units that process the objects. However, as I shall argue in this paper, there are limitations to this approach. To overcome these limitations, I will propose a way of using connectionist hardware to make *programmable* connectionist mechanisms, in which the connections needed to meet the current processing demands can be set up on line, as the processing demands arise.

The article begins with an example of what a hard-wire connectionist mechanism might look like, and uses the example to illustrate the limitations of the approach. The subsequent sections develop the programmable alternative.

A HARD-WIRED CONNECTIONIST PROCESSING MECHANISM

An example of a model which might be hard wired into connectionist processing structures is the interactive-activation model of word recognition (McClelland & Rumelhart, 1981; Rumelhart & McClelland, 1981, 1982; Figure 1). The model consists of a large number of basic elements or *nodes*. These serve as detectors for visual features, letters and words. Each node corresponds to the assertion that the item the node represents is present in an input pattern being processed by the network, and the *activation* of the node is monotonically related to the strength of this assertion. Nodes are grouped into several levels, with the feature level consisting of the feature nodes, the letter level consisting of the letter nodes, and the word level consisting of the word nodes. Since the inputs presented to the model contain four letters, there are separate sets of feature and letter nodes for each of the four letter positions. Since the model is intended to process only one word at a time, there is only one set of word nodes, with one node assigned to each four-letter word in English.

Processing in the interactive activation model takes place through excitatory and inhibitory interactions between the nodes. Nodes on different levels that are mutually consistent are excitatory. For example, the node for the letter T in the first position excites (and is excited by) the nodes for features of the letter T in the first position, and the nodes for words begin-

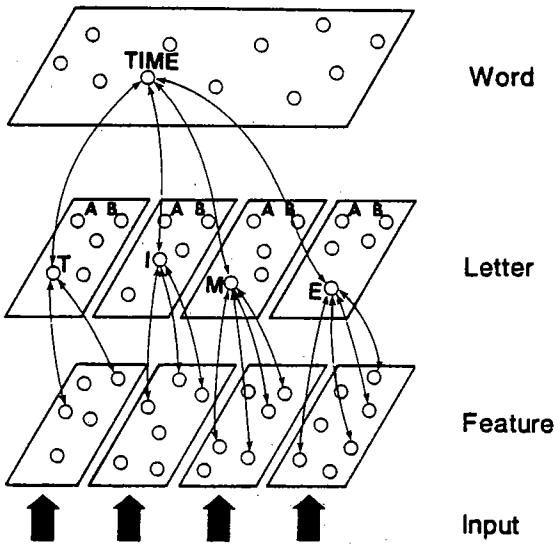

Figure 1. A sketch of some of the nodes in the interactive activation model of word perception (McClelland & Rumelhart, 1981; Rumelhart & McClelland, 1982), illustrating a small fraction of the excitatory and inhibitory influences between a few of the nodes. Nodes within the same rectangle represent mutually exclusive alternatives, and they are all mutually inhibitory; nodes on different levels that are mutually consistent are mutually excitatory. Some of the mutual excitatory interactions are represented by bidirectional arrows.

ning with T, such as TIME or TAPE. In addition, nodes that represent mutually exclusive interpretations of the input in the same position and at the same level are mutually inhibitory. For example, the node for the T in the first position inhibits all of the other first-position letter nodes. At the word level, the word nodes can be thought of as representing alternative interpretations of the whole string, so they are all mutually inhibitory.

One way to view the interactive activation model is as an abstract description of the time course of information accumulation regarding potential hypotheses, without specification of the actual mechanism whereby these interactions take place. On this view, each node represents a hypothesis, and each excitatory or inhibitory connection represents a weighted contingency between hypotheses. However, it is very easy to visualize a connectionist implementation of the model, in which the nodes are physical processing units and the interactions between them are mediated by physical connections between the processing units. Indeed, Figure 1, taken literally, depicts just such an implementation.

This sort of implementation seems very appealing for a number of reasons. The neural hardware of the brain is, after all, apparently well suited to connectionist mechanisms; and the idea that a processing system might be embodied directly in the brain (perhaps with some redundancy of units and connections) gives us the sense that we have begun to reduce cogni-

tion to a level where we might begin to understand its physical basis. Further, systems like Competitive Learning (Grossberg, 1976; Rumelhart & Zipser, 1985; von der Malsberg, 1973) have been proposed which could actually provide mechanisms whereby an unspecialized pool of processing units could learn to behave like the interactive activation model.

There is, however, one difficulty with adopting a hard-wired connectionist implementation of the interactive activation model or any other model involving interactions among large numbers of simultaneous, mutually constraining hypotheses. The difficulty springs from the fact that the knowledge which guides processing is hard-wired into the connections between the processing units in which the processing takes place. This is true, whether the connectionist model is of the *local* variety, in which each hypothesis is represented by a single node, as in the interactive activation model; or of the *distributed* variety, in which each hypothesis is represented by a pattern of activation over a population of nodes, as in the models of Anderson (1983) or Hinton (1981b). In both cases, the knowledge that guides processing is contained in the connections between the processing elements.

Why is this a problem? The reason is that parallel processing of multiple items is purchased at the price of duplication of the knowledge—the connection information—that guides processing. A hard-wired version of the word perception model would be able to process all four letters in a word at one time only because the connections specifying which features make up each letter would be reduplicated in the connections between each of the four banks of feature nodes and the corresponding bank of letter nodes. Only one word could be processed at a time, since there would be only one bank of word nodes. If we wished to process more than one word at a time we could only do so if we were willing to reduplicate the entire model, adding an additional four feature and letter banks and an additional word bank for each additional word we wished to process at the same time.

Even if we were willing to suppose that all of this hardware ought to be dedicated to reading English words, we would have a problem with learning. It would be nice if experience with a pattern when it occurred in one part of the display could result in learning which could be transferred to other parts of the display. But learning in connectionist models amounts to changing the strengths of connections between the nodes, based on what tends to go with what. We would, therefore, need some way of disseminating the changes mandated by events occurring in one bank of detectors to the other banks.

One obvious solution to these problems is just to "go sequential." Rather than reduplicate knowledge, we could put it just in one place—in a single, central connectionist structure—and map inputs into it one at a time. This would solve the learning problem, since patterns arising in different locations would always be processed in the same central set of connections. However, going sequential eliminates the whole point of interactive activa-

tion. A basic tenet of the interactive activation model was that processing occurred in parallel, across all of the letters in a word. This fundamental assumption allowed the model to exploit mutual constraints among the letters. It is precisely the possibility of such mutual constraints between larger, higher level units which makes the approach appear appealing in such domains as speech perception (Elman & McClelland, 1984), sentence analysis (Waltz & Pollack, 1985), and language production (Dell, 1985). It becomes cumbersome, if not impossible, to exploit the mutual constraint between items when they are processed sequentially.

Neither duplication of connection information nor sequential processing seem entirely satisfactory. Indeed, it has seemed to me that connectionism would be an unduly limiting computational framework if we were forced to limit the possibility of exploiting mutual constraints in our models to cases where we are willing to postulate duplication of connection information. Putting the point another way, if we could find a way of permitting parallel processing while still retaining the benefits of a single central representation for learning, we would have achieved an important increase in the computational power of connectionist mechanisms.

The rest of this paper describes a solution to this problem. There are four principle sections. The first describes the basic idea behind the approach, and builds up to an implementation of the interactive activation model of word perception which allows multiple words to be processed at the same time, even though it has a single central representation of the connection information specifying which letters go together to make each word. The second section describes a computer simulation of this model, and shows how the approach can account for some interesting data reported recently by Mozer (1983) on the kinds of errors human subjects make in processing two words at the same time. The third section considers the computational resource requirements of the connection information distribution scheme. In it I indicate that the scheme requires much less hardware than it would seem to require at first glance. The discussion section examines the essential properties of CID, and considers how it might be extended beyond the applications implemented here.

The next two sections focus almost exclusively on the identification of words, based on letter information provided by assumed lower levels of processing. Obviously, the word level is but one layer in a very rich language processing system. I chose to focus on this level because it is concrete, accessible, and familiar (at least to me), and because there is interesting evidence that bears on the model at this particular level. The principles embodied in the mechanisms I describe are obviously applicable at other levels, and to other processing tasks besides language processing. Of course, some new problems do arise at other levels. I will say a bit about extending the connection information distribution scheme to handle some of these problems in the discussion section below.

CID—A CONNECTION INFORMATION DISTRIBUTOR

In the system I propose, information processing takes place in a set of programmable node networks. Each network is a processing system consisting of processing units very similar to those that might be encountered in a direct connectionist implementation of the interactive activation model. Activations in these units stand for hypotheses about what is present where in the input, and information processing unfolds through their excitatory and inhibitory interactions. However, the units are not dedicated permanently to stand for particular hypotheses, and the knowledge that determines the pattern of excitatory and inhibitory interactions is not hard-wired into the connections between them. Rather, the connections in the node 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 hard-wired network. The task of this section is to see how we could replace this hard-wired 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 different from Figure 1 in organization, in order to highlight the excitatory connections between the units and lay them out in a way which will be convenient as we proceed.

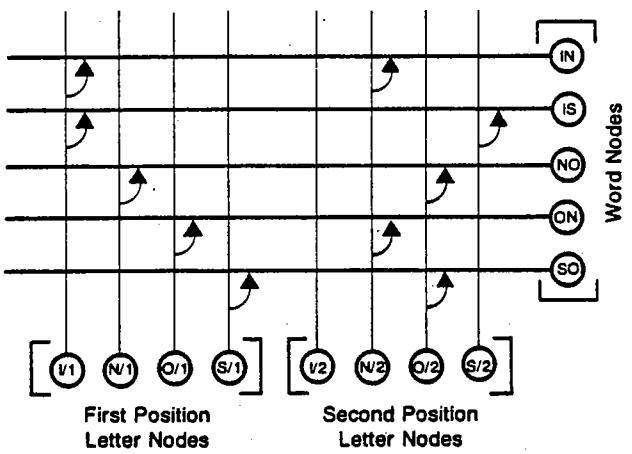

Figure 2. An extremely simple connection mechanism, capable of processing one two-letter string made up of the letters I, N, O, and S. The model knows only the five words that can be made of two of these letters, namely IN, IS, NO, ON, and SO. No top-down connections are included in this simple model. Nodes bracketed together are mutually inhibitory.

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 which 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 which ascend from them. Word units are shown having input lines which run from left to right. Where the output line of each letter node crosses the input line of each word node, 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 nodes. These are represented by the filled triangles in the figure.

Now, we are ready to see how we could build a programmable network, one which we could *instruct* to behave like the hard-wired network shown in Figure 2. Suppose that instead of fixed connections from specific letter nodes to particular word nodes, there is a *potential* connection at the junction between the output line from each letter-level node and the input line to each word-level node. 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 which are hard-wired in Figure 2. This proposal is illustrated in Figure 3.

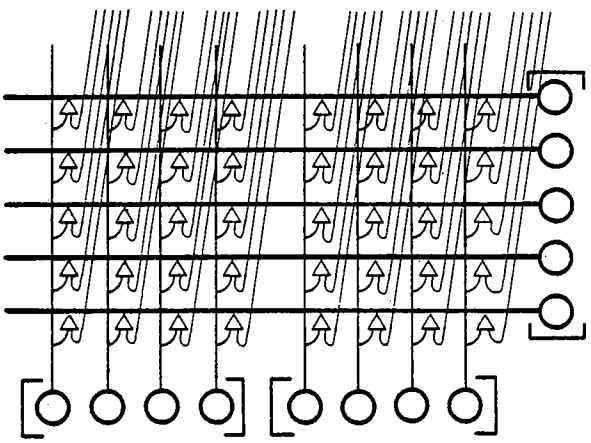

Figure 3. A programmable version of the simplified activation model shown in Figure 2. Each triangle represents a *programmable connection* that can be turned on by a signal coming from the central knowledge store, shown here as lying outside the figure to the upper right. If the triangular connections pass the product of the two signals arriving at their base along to the receiving node, the lines coming into the matrix from above can be thought of as programming the network.

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 be a special kind of unit that can 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 time-varying signal from some node i to some node j is multiplied by the fixed weight or connection stength w_{ij} to determine the value of the input from i to j:

$$\text{input}_{ij}\,(t) = \text{signal}_i\,(t) * w_{ij}.$$

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

$$\text{input}_{ij}\,(t) = \text{signal}_i\,(t) * \text{signal}_k\,(t).$$

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

History, Implementation, and Function of Programmable Connections. The idea of using a second signal to modulate connections has been used in other connectionist models. Hinton (1981a) 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. Feldman and Ballard (1982) have also suggested the idea of making connections contingent on the activation of particular nodes. 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 (1981b).

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 Shepherd (1979) or Crick and Asanuma (in press).

For our purposes, the implementation is less important than the function. In essence, what connections do is specify *contingencies* between *hypotheses*. A positive weight on the connection from unit *i* to unit *j* is like the instruction "if *i* is active, excite *j*." 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.

Let's see what we have achieved so far. 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 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 level nodes would now respond to the two-letter words in some other language. In conjunction with reprogramming the connections from feature level nodes to the letter nodes, we could even assign the network to processing words in a language with a different alphabet, or to processing completely different kinds of patterns.

Programmable networks like the one shown in Figure 3 will be called programmable modules. The input nodes will be called programmable letter nodes, and the output nodes will be called programmable word nodes. Though the nodes could be used for other things besides letters and words, these are the roles they will play in the present model.

Overview of the CID Mechanism

We are now ready to move up to a model containing a number of programmable modules along with the structures required to program them. The system is called a Connection Information Distributor, or CID for short. The basic parts of the model are shown and labeled in Figure 4; they are shown again, with some of the interconnections, in Figure 5.

Basically, CID 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 allows 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 appropriate programmable letter nodes in the left module, and the right one activates appropriate programmable letter nodes in the right module.

The Central Knowledge Store. The knowledge store in CID is shown at the top of Figure 4. This is the part of the model that contains the word-level

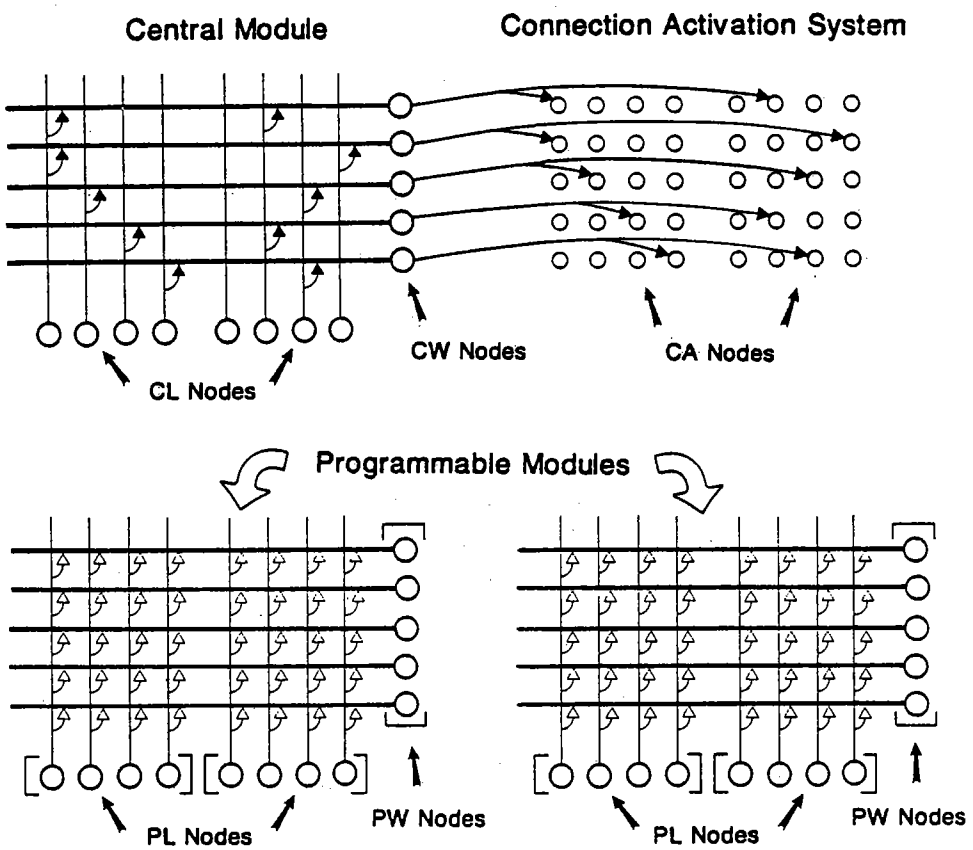

Figure 4. A simplified example of a Connection Information Distributor (CID), sufficient for simultaneous bottom-up processing of two two-letter words. The programmable modules consist of the programmable letter (PL) and programmable word (PW) nodes, and programmable connections between them (open triangles). The central module consists of a set of central letter (CL) nodes, a set of central word (CW) nodes, and hard-wired connections between them (filled triangles). The connection activation system includes the central word nodes, a set of connection activator (CA) nodes, and hard-wired connections between them. Connections between the central knowledge system (central module plus connection activation system) and the programmable modules are shown in Figure 5.

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

The central letter nodes are connected to the central word nodes via connections of the standard connectionist variety. These connections allow patterns of letter-level activation to produce corresponding word level activations, just as in the original interactive activation model. However, it should be noted that the central word node activations are based on a superposition of the inputs to each of the two programmable modules. Thus, the activations in the central letter nodes do not specify which module the letters come from, though relative position within each module is encoded. 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, it 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 nodes and their interconnections. One of these sets of nodes is the central word nodes—they belong both to the central module and to the connection activation system. The other set is the connection activator (CA) nodes. The purpose of the connection activation system is to translate activations of central word nodes into activations of connections appropriate for processing the corresponding words in the local modules. The CA nodes 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 nodes are not strictly necessary computationally, but they serve to maintain the conceptual distinction between that part of the model that contains the knowledge about words, and the parts that simply distribute that knowledge to the local modules). There is one CA node corresponding to the connection between a particular programmable letter node and a particular programmable word node. I have arranged the CA nodes in Figure 4 to bring out this correspondence. Each CA node projects to the corresponding connection in both programmable modules. I have illustrated the projections of two of the CA nodes in Figure 5. For example, the top-left CA node corresponds to the connection between the left-most programmable letter node and the top programmable word node. This CA node 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 node is

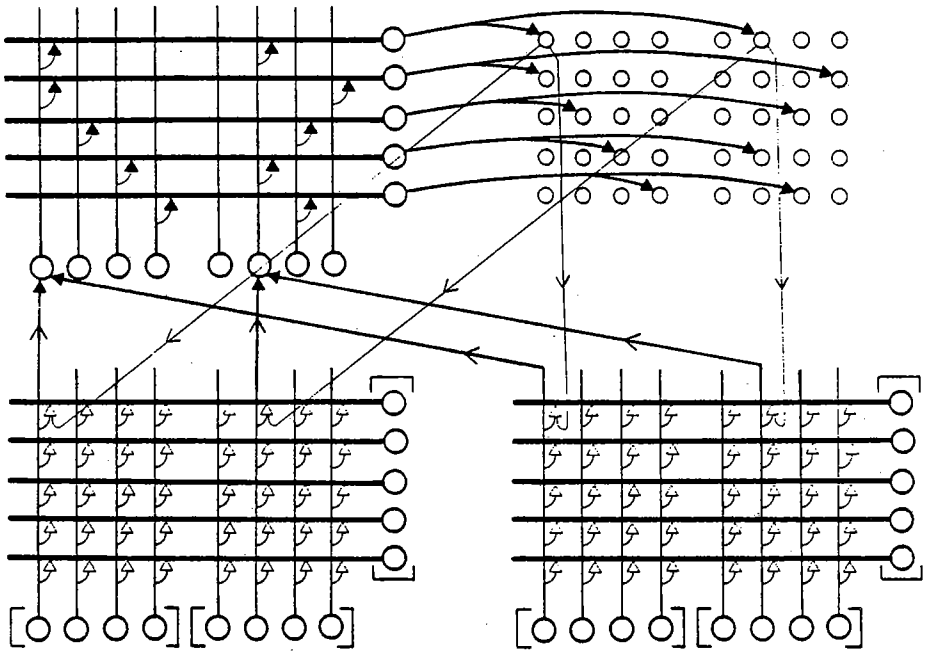

Figure 5. *Each* CA node projects to the corresponding connection in both programmable modules, and *each* central letter node receives projections from the corresponding programmable letter node in both programmable modules. The inputs to two central letter nodes, and the outputs from two CA nodes are shown.

active, it activates the corresponding connection in *all* of the programmable modules. In this way it acts as a sort of master switch.

At a functional level, we can see each CA node as standing for a contingency between two activations. Thus, if we index the programmable letter nodes by subscript *i*, and the programmable word nodes by *j*, the *ij*'th CA node stands for the contingency, "if letter node *i* is active, excite word node *j*." Thus, we can think of the CA nodes as Contingency Activation, as much as Connection Activation nodes. When we activate a CA node (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 nodes, of course, are responsible for activating the CA nodes. There are excitatory connections from each word node to each of the CA nodes for the connections needed to process the word. For example, the central word node for IN activates two CA nodes. One is the CA node for the connection between the left-most programmable letter node and the top-most programmable word node. The other is the CA node for the connection from the sixth programmable letter node from the left to the same programmable word node. These connections effectively assign the top programmable word node to be the detector for IN (assuming, of course, that lower levels of processing have been arranged so that I in the first posi-

tion and N in the second position activate the appropriate programmable letter nodes).

In summary, CID consists of a) the two 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 nodes in one of the programmable modules (say the programmable letter node 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 nodes. From these they activate the central word node for IN. Central word nodes for patterns which 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 nodes pass activation to the CA nodes, 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 nodes—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 nodes. Thus, the appropriate programmable word node will tend to be the most active in each local module. Although the words are not kept straight at the central level, they are kept straight—though with some tendencies to error —down below. We will examine this matter more closely below.

A COMPUTER SIMULATION OF CID

To examine the behavior of the CID scheme in more detail and to compare it to the original interactive activation model, I created a computer simulation. The structure I simulated was scaled up from the version in Figures 4 and 5 so that it would be able to 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 I.

TABLE I
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

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 nodes and 32 programmable word nodes. The programmable letter nodes 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, hard-wired, inhibitory connections. Similarly, all of the programmable word nodes in each module were mutually inhibitory. Each programmable module contained $16*32 = 512$ programmable connections, and there were 512 CA nodes, one for each programmable connection. The central module contained 16 letter and 32 word nodes, like the programmable modules. There were no inhibitory connections either between the central word nodes or between the central letter nodes. The connections between the central letter nodes and the central word nodes, and connections from the central word nodes to the appropriate CA nodes, were hard-wired with the connection information needed to make the central letter nodes activate the right central word nodes and to make the central word nodes activate the right CA nodes.

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

Details of Interaction Dynamics

The programmable letter and word nodes have the same dynamic properties as the letter and word nodes in the original word perception model (McClelland & Rumelhart, 1981). Time is divided into a sequence of discrete processing steps. On each processing step, each programmable node adds up all of its excitatory and inhibitory inputs from all other nodes and from the external input. Then the activation value of each node is updated. If the net

input is excitatory, it will tend to increase the activation of the node; if the net input is inhibitory, it will tend to reduce the activation of the node. The effect is graded and gradual, and activation values are always kept between a maximum and minimum value. There is also a tendency for activations to decay back toward resting level, which is set at an activation of $-.05$ for all nodes. Only positive activation values are transmitted to other nodes, so that nodes with activations below 0 are effectively out of the computation. The model has a global letter-to-word excitation constant, called alpha, and a global word-to-word inhibition constant, called gamma, as well as a global decay called beta. The values of alpha and gamma determine the strength of bottom-up excitation relative to within level inhibition. The values used for these three parameters were taken from the original model.

The only difference between the CID version of the model and the original is in the strengths of excitatory connections between nodes. 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, central word, and CA nodes. The activation of a particular central letter node is simply the number of input nodes projecting to it which have activations greater than 0. Thus, the activation of a particular central letter node just gives a count of the corresponding programmable letter nodes that are active. The activation of a central word node is just the sum of the active central letter nodes which have hard-wired connections to the central letter node. The activation of a CA node is just the activation of the central word node 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 node proportional to the number of active nodes for the letters of the word, summed over both modules. Active letter nodes count only if they stand for letters in appropriate positions, though, within the programmable module of origin.

Output

So far, we have said nothing about how the activations which 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. The readout mechanism can be directed to the word or the letter level of either module, and at the latter 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 relation between activation and response probability is based on the choice model of Luce (1963). The probability of choosing a particular response depends on the strength of the node corresponding to that response, divided by the sum of the strengths of all the relevant alternatives (e.g., nodes for words in the same position). The exact relation of strength and activation is described in McClelland and Rumelhart (1981).

The import of these assumptions is that the probability of a particular response is solely a function of the activations of nodes relevant to the response. All interactions between display items are thus attributed to the node 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 scheme causes the model to behave as though it were sharply tuned to its inputs, thereby eliminating the need for bottom-up inhibition. Second, when processing two words at a time, the connection activation scheme causes the model to make errors similar to those made my human subjects viewing two-word displays. These errors arise as a result of the essential characteristics of the CID scheme.

One Word at a Time: The Poor get Poorer

In the original model, bottom-up inhibition from the letter level to the word level was used to sharpen the net bottom-up input to word nodes. For example, consider a display containing the word SAND. Due to bottom-up inhibition, nodes 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 node 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 node 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 nodes 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 node 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

settled on an overall set of parameters that 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 II). Words with less than two letters in common received net bottom-up inhibition in the old version, whereas in the CID version they simply receive little or no excitation. In both cases, their activation stays below zero due to competition, and thus they have no effect in either case on the behavior of the model.

TABLE II
One Word at a Time:
Bottom-Up Activations of Several Word Nodes in the Original and CID Versions of the Interactive Activation Model

		Input: SAND			
		Original		*CID Version*	
Node	*Letters Shared w/input*	*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 node, divided by the net bottom-up activation of the node for SAND.

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 was ambiguous, however, there was a slight difference in the performance of the two versions of the model. This actually revealed a drawback of bottom-up inhibition that is avoided in CID. Consider the input to a word node from the letter nodes 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 CID, because there is no

bottom-up inhibition. Thus, I found that the CID version could pull a word out of a highly degraded display in which several letters were equally compatible with the feature information presented, while the original model could not. 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 interactive activation 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 CID's structure becomes essential, since simultaneous processing of two patterns introduces considerations which do not arise in the processing of single items.

When letters are presented to both modules, *all* of the letters are combined to turn on connections which 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 node for the word presented to a particular module will generally receive the most activation. However, the activation of programmable word nodes 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 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 III. 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.

TABLE III
Method and Results of Mozer (1983), Experiment 1

Method		
Example Display	SAND	LANE
Target Cue	↑ ↑ ↑ ↑	
Results		
Response Type	% 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 rate*	5.3	

*Psuedo-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.

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 nodes 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 IV
Two Words at a Time: Crosstalk
Relative Bottom-Up Activations Produced by SAND
Presented either Alone or with LANE as Context

	Alone		*with LANE*	
Programmable Word Node	*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 node, divided by bottom-up activation of SAND.

Table IV compares relative programmable word node activations for various words, for two different cases: In one case, the word SAND is presented alone; in the other, it is presented in the context of the word LANE. When SAND is presented alone, all words which share three letters with it receive $(3/4)^2$ or 9/16's as much bottom up activation as the node 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 nodes 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. This is especially true when two or more nodes are working together at the same level of activation. In this case, the nodes for LAND and SANE act together. Neither can beat out the other, and both "gang up" on those receiving slightly less bot-

tom-up activation, thereby pushing these other alternatives out. This "gang effect" was observed in the original version of the model—see McClelland and Rumelhart (1981), for details. This behavior of the model is illustrated in Figure 6. Through the mutual inhibition mechanism, SAND and LANE come to dominate over other words that share three letters in common with the target. Some of these, in turn, dominate words that share but 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 accords with the experimental result that double or "whole-word" migrations are quite rare in Mozer's experiment, as shown in Table III.

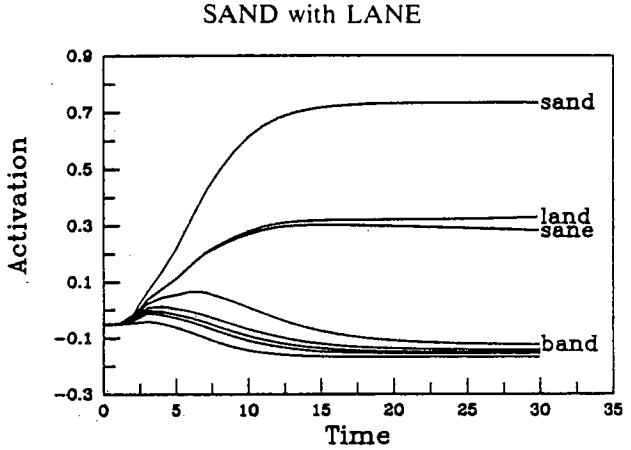

Figure 6. Activation curves for various programmable word nodes in the module to which SAND is shown, when the input to the other module is LANE. The x axis represents time cycles from the onset of the two-word display.

Mozer (1983) reported several additional findings that are consistent with the CID model. First, he showed that letter migrations are more likely to be "copies" than "exchanges." That is, when subjects (in a second experiment) were asked to report both words in the display, the probability of a copy or duplication error, in which both reported items contained an L or an S (as in LAND-LANE or SAND-SANE), was much greater than the probability of an exchange, in which the S and L exchanged places (to make LAND-SANE). The data are not consistent with models which attribute migration errors to a mechanism which conserves the number of occurrences of each letter, such as the feature integration model of Treisman and Gelade (1980). However, if as I assume for the CID model the subject selects the best response independently for each of the two inputs, then we expect the probability of an exchange to be equal to the probability of two independent errors occurring at the same time. That is, the probability of an exchange should be equal to the probability of the left letter turning up on the right

times the probability of the right letter turning up on the left. The expected probability of an exchange based on these considerations is .006 for this experiment, within experimental error of the value of .008 actually obtained.

Second, Mozer showed that migrations are more common when target and context share letters in common than when they do not. Thus, the probability of saying LAND was much higher when SAND was flanked by LANE than when SAND was flanked by LOVE. The relevant data are displayed in Table V.

TABLE V
Percent Correct Migration and Other Responses from Mozer (1983), Experiment 3

	Context Type	
	Common Letters (SAND-LANE) (%)	*No Common Letters (SAND-LOVE) (%)*
Correct response (SAND)	64	74
Single migration (SANE or LAND)	11	6
Other	25	19

Note: Pseudo migration rate (probability of reporting LAND or SANE when context contained no letters which could form a word with the target) was 3%. In this example (Target word LAND), the context string might have been COMB.

At first sight it might appear that the CID model would not expect this difference. When SAND is presented with LANE, connections for all the (S/L)(A)(N)(D/E) words receive 6 units of activation because of the repetition of the A and N in both letter-strings. When SAND is shown in the context of LOVE, connections for these same words all receive 4 units. The ratios of bottom-up activation for correct and single-migration words are the same in both cases. However, once again, this ratio is not the whole story. The absolute *magnitude* of bottom-up activation is greater in the case where there are letters in common than in the case where there are not. The higher the overall magnitude of bottom-up activation, the less difference the ratio makes, due to the tendency of node activations to saturate at high activation levels. When bottom-up activation is reduced overall, sharper differences in the pattern of activation emerge. The result is that there is far more activation of migration words when the two words shown have letters in common than when they do not. When there is less overlap between target and context, the tendency of the correct answer to dominate the pattern of activation is sharply increased. This is illustrated in Figure 7, which shows much less activation for migration error words than Figure 6.

In summary, the CID version of the interactive activation model appears to provide fairly accurate accounts of the intriguing perceptual interactions reported by Mozer. Mozer's basic finding, that letters in one display position tend to show up in reports of the contents of the other position, is a

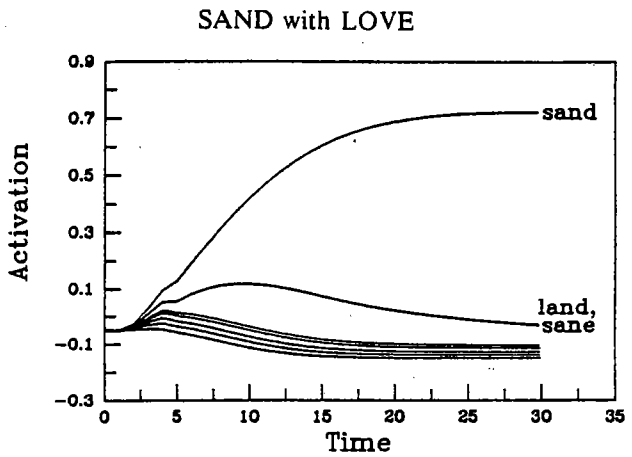

Figure 7. Activation curves for various programmable word nodes in the module to which SAND is shown, when input to the other module is LOVE.

necessary consequence of the CID mechanism. Considerably more empirical and theoretical work is required before we will be in a position to say that the CID version of the interactive activation model provides an adequate account of all aspects of these perceptual interactions. But it appears that the model is on the right track.

THE INFORMATION PROCESSING CAPACITY OF CID

There is one apparent problem with the CID scheme. It appears that it requires a prohibitive number of nodes and connections. To point to the most serious aspect of the problem, the CID model of word perception as I have described it requires one CA node for each potential connection between a letter node and a word node. This number grows as the *product* of the number of letter level node times the number of word-level nodes. A system sufficient to process 50,000 different words of up to 7 letters in length would require 7*26 = 182 letter level nodes, 50,000 word level nodes, and nearly 1 million CA nodes. This seems like an awful lot of nodes. It would appear that the benefits of parallel processing are being purchased at a prohibitive cost.

However, the situation can be improved dramatically by switching over to a *distributed* representation, in which each word (the argument applies to letters, too, or anything else, of course) is represented by a pattern of activation over a set of nodes, rather than by the activation of a single node.

At first glance the switch to distributed representation may appear to be a major change of stance. However, there is less to the choice between

local and distributed representations than meets the eye. All the matter really comes down to is whether we associate conceptual units like words with individual nodes, or with overlapping constellations of nodes. In the CID model I have used local representation for clarity and comparability with the original model, but now that the basic idea of CID is on the table, I will argue that a switch to distributed representation would be of great benefit. Distributed representation has many virtues, several or which are described in Hinton (1984) and in McClelland and Rumelhart (in press). For our present purpose, the great advantage of distributed representation is that it allows us to get by with much smaller programmable modules and far fewer CA nodes. First, I'll briefly explain how distributed representation saves us nodes even with a hard-wired connectionist mechanism. Then I'll show how it pays off in spades in CID.

I'll begin by sketching a simple connectionist mechanism for associating patterns of activity at one level with paired patterns of activity at another level. For word recognition, the levels might be letter or word, but I'll just call them levels A and B for generality. Our goal is to be able to activate the correct B pattern whenever the corresponding A pattern is shown. We assume that each association involves an A pattern and a B pattern each containing some number M of active nodes on each of the two levels. To allow A patterns to retrieve the appropriate corresponding B patterns, we simply imagine that there is an excitatory connection from each node active in the A pattern to each of the nodes active in the corresponding B pattern. Note that a local representation model in which, say, each pattern at the A level is represented by as single active node at the B level, is just a variant of this model, in which the pattern at the B level consists of just a single node.

Essentially, distributed representation can save on nodes because we can use partially overlapping codes for different items: we are no longer required to have at least one node for each alternative pattern. There are, however, limitations on the number of associations that can be stored in an A-B associator: The more patterns we store, the more likely it is that the B pattern retrieved by any A pattern will be contaminated by spurious activations at the B level.

Willshaw (1981) has analyzed the extent of contamination, under a particular set of simplifying assumptions. First, he assumed that the connection between a particular A node and a particular B node has only two states: it is on if the A and B nodes are associated in any of the patterns stored, and off otherwise. This assumption means that when the nodes for the A member of the pair are activated, each of the appropriate B nodes will receive one unit of excitation from each active A unit. Other B nodes not part of the appropriate B pattern may receive excitation from some or all of the A units, via connections that are on because they belong to other stored associations. Second, Willshaw assumed that B nodes remain inactive unless they receive excitation from all of the active A units. (This is imple-

mented by assuming that the unit has a threshold equal to the excitation produced by a complete A pattern). With this assumption, when a known A pattern is presented, the model is able to turn on all the correct B nodes. A spurious B node will only be turned on if *all* active members of the A pattern happen to have excitatory connections to it. Third, Willshaw assumed that the A and B patterns were random selections of some fixed number of A nodes and some fixed number of B nodes.

Given Willshaw's assumptions, it is possible to calculate the average number of B nodes that will be spuriously activated for any combination of Na and Nb, the number of units in each of the two pools, Ma and Mb, the number units active in the A and B members of the pattern, and R, the number of associations known. We then need only adopt an error tolerance criterion to determine how large Na and Nb must be to accommodate R associations of pairs of patterns of size Ma by Mb. Adopting a criterion of an average of one spurious B node activation per retrieval, Willshaw derived the following relationship:

$$\sqrt{Na*Nb} = 1.2\sqrt{Ma*Mb}\ \sqrt{R}$$

These relations only hold if we use distributed representations. The relation holds exactly if the number of units active in the A and B patterns is equal to the $\log_2$ of the number of units in the corresponding pool. For larger fractions of active units, the equation actually underestimates the number of patterns that can be stored for given values of Na and Nb.

The result just presented depends on the fact that, when the associations involve patterns with a reasonable number of active units in both members of the pair, the chances that the connections will result in a spurious B node receiving as much activation as an appropriate B node are remote, as long as the overall proportion of connections that are turned on is reasonable. The proportion we can get away with varies a bit with the exact values of Ma and Mb, but if half or fewer of the connections are turned on, we will generally be quite safe. Roughly speaking, then, the number of patterns we can store is just the number which keeps us from turning on more than half of the connections in the matrix.

We are now ready to examine the implications of distributed representation for CID. Based on the previous equation, we can calculate the number of programmable connections, (and therefore, the number of CA nodes we would need) to program a module to perform like the hard-wired module considered in Wilshaw's analysis. The number of connections between Na A nodes and Nb B nodes is just Na*Nb. Since we need one programmable connection for each connection, we can simply square the previous equation to get an expression for the number of programmable connections (Npc):

$$Npc = Na*Nb = 1.45*Ma*Mb*R.$$

While this represents a slight improvement over the nondistributed case, the number of programmable connections and CA nodes still appears to grow linearly with R, the number of patterns known, and with the size of the patterns.

But this result ignores the fact that in CID, we do not turn on the connections relevant to all the known patterns at one time. Let us just consider what would happen if we were able to avoid activating any unwanted connections, and could turn on only the connections relevant to the particular pattern or patterns we wished to process at one time. In this case, we could get by with far fewer nodes in each of the pools, and therefore far fewer programmable connections. The more patterns we wished to process at one time, of course, the more connections we will have to turn on, and the more risk we will run of spurious activation. Thus, the number of A and B nodes required in each programmable module, and thus the number of programmable connections, is related to the number (S) of associations we wish to be able to process *simultaneously,* rather than by the total number of associations known. The number of nodes we need also depends on the number of A and B nodes active in each member of a pair, as before. Again, allowing an average of one spurious B node activation per retrieval, the following approximate relation holds (the constant will be somewhat larger for values of S less that 3):

$$Npc = Na*Nb = 1.45*Ma*Mb*S.$$

This equation is the same as the one we had for the number of connections needed in a hard-wired associator, except that we now have S, the number of patterns to be processed simultaneously, instead of R, the number known. Replacing R with S makes a huge difference, since it can plausibly be argued that we know something like 50,000 words. We could process up to 5 of these at once (each in a different programmable module), the equation says, with *five orders of magnitude* fewer CA nodes and programmable connections, and two and a half orders of magnitude fewer nodes in each programmable module than we would need in a hard-wired module of the same capacity. Of course, the size of the central module still depends on R, but we pay that price only once, and we incur it with models which do not allow parallel processing, as well as with CID. Relative to the resource requirements of the central module, then, the extra cost of simultaneous processing via connection information distribution is modest, if we are willing to switch to distributed representation.

I have actually overstated my case a little. For one thing, the number of programmable connections required depends on the number of B level patterns fully activated in the central module by the superposition of all of

the A patterns presented for processing at one time. This may be more than the number of patterns actually presented for processing (as when SAND-LANE was presented, their superposition contained all the letters of these two words and LAND and SANE besides), and will depend on the ratio of Ma/Na—for adequate performance, then, Na may have to be bigger than the equation given above suggests. Space prevents a full analysis of these matters here. Suffice it to say for the present that these complications do not eliminate the basic result that with distributed representations we can greatly reduce the resource requirements of CID. Though the degree of the savings previously indicated may be slightly exaggerated, it remains true that distributed representation still brings the resource requirements of CID into reasonable bounds.

DISCUSSION

Thus far, we have explored a particular model embodying the idea of distribution of connection information, and we have seen how this idea provides a way of allowing a single, central representation of knowledge to be made available to each of a number of programmable processing modules, thereby turning the programmable modules into programmable connectionistic information processing structures. We have seen how this scheme provides a natural account of the errors human subjects make in processing two-word displays. Abstracting from this specific application a little, we have explored some aspects of the resource requirements of such a system, and we have found that they are not as exorbitant as one might have feared.

In this section of the paper, I step back even further from particular detailed models, and consider the idea of distributing connection information more generally. First, I discuss the essential properties of the CID mechanism. Then, I discuss possible extensions of the approach to other domains such as sentence processing. Third, I suggest reasons why some sequentiality in programming parallel processing structures might occasionally be a good thing. The paper concludes with a brief examination of the relation between interactive activation processes and connectionist implementations, in light of the CID mechanism.

Essential Characteristics of CID

There is a sense in which CID is not as powerful a mechanism as I had hoped to discover. Although it permits parallel processing to some degree, the performance of the model degrades when multiple items are processed simultaneously. One might well ask questions then: Might another mechanism not do better? Do we need such a complex mechanism to accomplish what

CID has done? Might we not get at least as good behavior with something simpler?

Obviously, there are difficult questions to give definitive answers to. However, I think it worth considering CID somewhat abstractly for a moment, to see what its essential properties are. This will, I think, provide a little insight into these questions.

One essential feature of CID is the superposition of the patterns of activation in each programmable module. Superposition permits simultaneous access to and retrieval from the central module, but what is retrieved is not the response to either or even both individual patterns, but the response to their superposition. This method of simultaneous information retrieval automatically runs the risk of crosstalk because by its very nature it looses track of which letters appeared in each of the programmable modules. Actually, though, the amount of crosstalk depends on the complexity and similarity structure of the patterns we wish to process. In fact, if *all* the patterns stored in the central module of a CID mechanism are maximally dissimilar (that is, orthogonal), there will be no crosstalk. It is only when the known patterns overlap with each other that crosstalk becomes a problem.

The point of this discussion is simply to suggest that the *kind* of limitation we see in CID as a parallel processor is intrinsic to superposition of inputs. This may be intrinsic to parallel retrieval itself—I have not been able to conceive of an alternative (connectionist) scheme for *simultaneous* retrieval of information about two patterns. But the exact *extent* of the limitation depends on the details of the patterns and their similarity structure. Since the similarity structure of patterns assigned to inputs can be affected by the way the inputs are coded, it is possible to manipulate the extent of the crosstalk problem quite easily.

But couldn't we do just as well without CID? Isn't there another, less complex mechanism that could do the same job it does just as well? One difficulty answering this is the amorphous definition of complexity, and the difficulty of specifying in detail what counts as a similar mechanism and what as a different one. However, it is instructive to consider briefly one simplier alternative to CID, in which we distribute *activation* information from the central module instead of *connection* information. Figure 8 shows such a mechanism. It is like CID except that the central word nodes project directly back to the corresponding local word nodes. The idea is that patterns of activation arising in the local letter nodes will be superimposed as inputs to the central module, and the composite output generated by the composite input will be distributed back to the local word nodes. In such a mechanism, if two words are presented at once, the pattern of activation that would appear on the output nodes of each local module would be the same for both of the programmable modules. Indeed, it would be the same as the pattern of activation over the central word nodes. This pattern contains

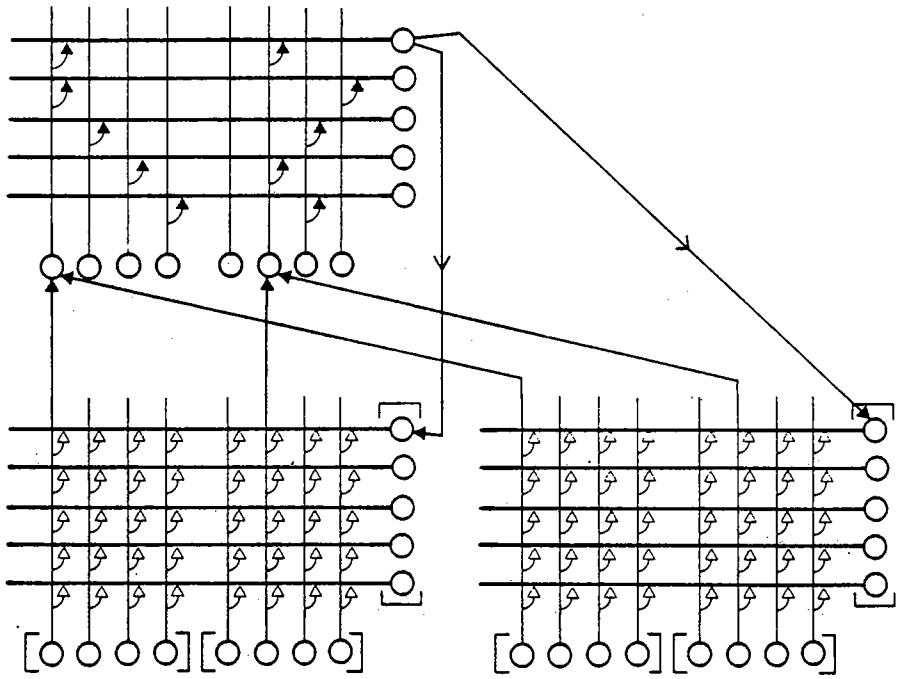

Figure 8. A mechanism that distributes activation information, rather than connection information. It is like CID in that it has local modules and a central module, but what it distributes is the pattern of activation over the central word nodes. Inputs to two central letter nodes and outputs of one central word node are shown.

equal representation of both inputs, as well as any known patterns which can be formed from the superposition of these inputs. With such a mechanism, if SAND LANE were presented, we would have no way of knowing from the activations of the word nodes in the first module whether the input to the module consisted of SAND, LANE, LAND, or SANE. Such a mechanism is slave to the composite output of the central module, and is not much good for processing more than one pattern at a time.

Just like this simpler mechanism, the information CID distributes is based on the composite of the inputs to the two programmable modules. But since CID distributes *connection* information, instead of *activation* information, the pattern at the letter level in each module still influences what the output pattern will be at the word level. As I stressed early on, *connection* information is *contingency* information. It says, if node x is active, let it activate node y. Distributing connection information allows the central module to tell the programmable modules what to do with their inputs, and this allows their output to reflect these inputs, as well as what they are told.

In summary, the essential features of CID are *superposition of inputs* to the connection activation process and *distribution of conditional information*. Some crosstalk is an inevitable byproduct of superposition, but the

amount of it will depend on the amount of hardware used and the number, complexity and similarity structure of the patterns to be processed.

Extension of CID to Other Problems and Other Levels

I claimed in the introduction that CID would be generally useful in extending the computational power of connectionist mechanisms. I have illustrated how it can be applied to the word level in a word perception model, and I generalized the idea in the discussion of the abstract pattern associator introduced in considering the resource requirements of CID. Now it is time to consider the relevance of the approach to computationally more challenging levels, such as syntactic and semantic analysis of sentences.

Sentences, as objects for processing, have one essential characteristic which individual words (at least in English) do not have. In sentences, the same structures can occur at many different levels of the representation of the same sentence. To many, this recursive characteristic of sentences seems to require a recursive processing mechanism, of the sort typically implemented in AI language processing programs.

But recursive structure does not necessarily require recursive—that is sequential—processing. The beauty of recursive processing is that the same knowledge—say, of the constituent structure of a noun phrase—can be made available at multiple levels, because the same subroutine can be called at any level, even inside itself. Connection information distribution provides a way of doing the same thing, in parallel.

CID has already given us a mechanism which makes the same knowledge (connection information) available simultaneously for processing different patterns on the same level without resorting to sequential processing. If access to the same knowledge was possible, not only from different slots on the same level, but from different levels, then recursively structured objects could be processed, in parallel, on several levels at the same time. Of course, crosstalk would still be a problem—it would tend to confound the bindings of things at different levels—but the programmable modules on the same level in our word-perception model are able to keep straight what goes with what in each of the two patterns they are processing at the same level, given sufficient resources and patterns that are sufficiently distinct. The same would be true for programmable modules accessing the same central knowledge system from different levels.

The idea of allowing the simultaneous programming of parallel processing structures at different levels thus preserves the essential positive aspect of recursive processing—access to the same information at different structural levels—without requiring us to resort to seriality.

But there is still a problem, for I seem to be assuming that we have available some sort of stack of levels, all of which can access the same central knowledge. The difficulty is that the number of levels of depth we will

need cannot be specified in advance. Although we would probably be able to get by in almost all practical cases with some adequate fixed number of levels, this assumption does violence to the essential open-endedness of sentences. Any attempt to extend the idea to even more global structures, such as text structures or plans, would be doomed.

But CID provides us, at least in general terms, with a way to get by without any fixed number of processing levels. For levels are defined in terms of connections, and if we can program connections, we can in principle set up the entire hierarchical structure of the processing system on the fly, as well as any specific interactions between units imbedded in that structure.

Of course, we are several steps away from a concrete realization of this idea, and there are many complications that have to be addressed. But, I believe that programming connections will play an important part in the development of interactive activation mechanisms of sentence processing and other higher-level cognitive tasks, and I hope that CID represents a step in this challenging and important direction.

A Little Sequential Programming May Not be a Bad Thing

I have taken the position that parallel processing is important, because it permits the exploitation of mutual constraint. But, this argument does not really apply to the simultaneous *programming* of parallel processing structures. In some cases, it is sufficient to program the processing structures sequentially, so that processing can then occur in parallel. Serial programming could be arranged by projecting from one programmable module at a time to the central module and projecting the output of the connection activation nodes back to the same place. Hinton (1981b) illustrated how this kind of thing can be done, using programmable connections. If connection activations were "sticky," a number of programmable modules could be programmed sequentially, but the resulting activations could continue to interact within (and, through higher-level structures, between) the modules for some time.

There are two advantages to programming parallel processing structures sequentially. One is that we would cut down on crosstalk in the programming process. The less we project onto the central representational structures at one time, the fewer spurious connections will be activated, and the fewer nodes and programmable connections we will need for accurate processing.

The second reason is that crosstalk is especially devastating for learning, since learning takes place in the central knowledge structures. Learning in connectionist models generally involves increasing connection strengths based on simultaneous activation (Ackley, Hinton, & Sejnowski, 1985; Rumelhart & Zipser, 1985). If several patterns are superimposed in the input to

the central module, the learning mechanism would be unable to separate the simultaneous activations which actually came from the same pattern from those which came from different patterns. Thus, serial programming may be particularly important during acquisition of an information processing skill. Indeed, we would probably not expect the central representations to be robust enough to tolerate much crosstalk before they have been well learned in any case. As we learn, we may be forced to proceed sequentially for adequate performance, but this may help us learn better, so that we can eventually process in parallel.

Sequential programming of parallel processing structures allows us most of the benefits of parallel processing without the costs associated with superposition of inputs to the central knowledge system. I would not, however, adopt the view that programming is always sequential. Or rather, I would not suggest that it occurs a single unit at a time at every level. Just how much input can be handled at a time probably changes with practice—but this is a matter to be examined in further research.

Connectionism and Interactive Activation

I began this paper by suggesting that the interactive activation model of word perception had some important limitations as a literal description of a connectionist processing mechanism. If we thought of the model as a description of the mechanism, we would take the present paper as suggesting the replacement of the original model with another kind of model, in which nodes are dynamically assigned to roles and dynamically connected to other nodes, instead of being hard-wired as they were in the original model.

However, as I have already suggested, there is an alternative way of thinking about the interactive activation model and its relation to connectionism. In this alternative approach, we would not view the interactive activation model as a description of a mechanism at all. Rather, we would see it as a functional description of the behavior of a processing system whose actual implementation—connectionistic or otherwise—is not specified.

I believe that it is important to be able to shift between these two perspectives. As important as it is to be clear about implementation, there are two reasons why it is occasionally useful to adopt a more abstract or functional point of view. The first is that it allows us to study interactive activation models of a wide range of phenomena at a psychological or functional level without necessarily worrying about the plausibility of assuming that they provide an adequate description of the actual implementation. On this view, for example, Rumelhart and I would not necessarily be seen as assuming that there "really are" multiple hard-wired copies of each letter node, one for each position within a word, in the original interactive activation model. The existence of CID allows us to be reasonably confident in the belief that a mechanism with the information processing characteristics of a

model which postulated multiple copies of letter nodes could be implemented in connectionist hardware. This would free us to consider the adequacy of the particular dynamic assumptions of the interactive activation model, or more interestingly, its claim that some aspects of apparently rule guided behavior can emerge from the interactions of units standing only for particular exemplars of the rules. Similarly, I think we should be prepared to treat CID in the same way, and examine the adequacy and utility of the functional information processing characteristics it provides. The question of implementation remains an important one, and we would certainly not want a model for which no plausible implementation could be conceived, but it is not the only question which we must consider in trying to understand cognitive processes.

The second reason for taking an abstract view of activation models is to keep in view the fact that we have only begun to scratch the surface of distributed, parallel information processing mechanisms. The original interactive activation model was a step that captured some of the flavor that such a mechanism should have, and I see CID as another. But, we still have a long way to go before we can claim to have done justice to the exquisite information processing mechanism so faintly reflected in the models we have constructed so far.

REFERENCES

Ackley, D., Hinton, G., & Sejnowski, T. (1985). Boltzmann machines: Constraint satisfaction networks that learn. *Cognitive Science, 9,* 147–169.

Anderson, J. A. (1983). Cognitive and psychological computation with neural models. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13,* 799–815.

Crick, F., & Asanuma, C. (in press). Certain aspects of the anatomy and physiology of the cerebral cortex. In J. L. McClelland & D. E. Rumelhart (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. II: Applications.* Cambridge, MA: Bradford.

Dell, G. S. (1985). Positive feedback in hierarchical connectionist models: Applications to language production. *Cognitive Science, 9,* 3–23.

Elman, J. L., & McClelland, J. L. (1984). Speech perception as a cognitive process: The interactive activation model. In N. Lass (Ed.), *Speech and language: Vol. X.* Orlando, FL: Academic.

Feldman, J. A. (1981). A connectionist model of visual memory. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum.

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6,* 205–254.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors. *Biological Cybernetics, 23,* 121–134.

Hinton, G. E. (1981a). A parallel computation that assigns canonical object-based frames of reference. *Proceedings of the Seventh International Joint Conference in Artificial Intelligence,* Vol 2. Vancouver, BC, Canada.

Hinton, G. E. (1981b). Implementing semantic networks in parallel hardware. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum.

Hinton, G. E. (1984). *Distributed representations.* (Tech. Rep. No. CMU-CS-84-157). Pittsburgh, PA: Department of Computer Science, Carnegie-Mellon University.

Luce, R. D. (1963). Detection and recognition. In R. D. Luce, R. R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology: Vol. I* New York: Wiley.

McClelland, J. L., & Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: Part I. An account of basic findings. *Psychological Review, 88,* 375-407.

McClelland, J. L., & Rumelhart, D. E. (in press). Distributed memory and the representation of general and specific information. *Journal of Experimental Psychology: General.*

Mozer, M. C. (1983). Letter migration in word perception. *Journal of Experimental Psychology: Human Perception and Performance, 9,* 531-546.

Poggio, T., & Torre, V. (1978). A new approach to synaptic interactions. In R. Heim & G. Palm (Eds.), *Approaches.to complex systems.* Berlin: Springer-Verlag.

Rumelhart, D. E., & McClelland, J. L. (1981). Interactive processing through spreading activation. In A. M. Lesgold & C. A. Perfetti (Eds.), *Interactive processes in reading.* Hillsdale, NJ: Erlbaum.

Rumelhart, D. E., & McClelland, J. L. (1982). An interactive activation model of context effects in letter perception: Part 2. The contextual enhancement effect and some tests and extensions of the model. *Psychological Review, 89,* 60-94.

Rumelhart, D. E., & Zipser, D. (1985). Competitive learning. *Cognitive Science, 9,* 75-112.

Sejnowski, T. (1981). Skeleton filters in the brain. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum.

Shepherd, G. M. (1979). *The synaptic organization of the brain* (2nd ed.). New York: Oxford University Press.

Treisman, A., & Gelade, G. (1980). A feature integration theory of attention. *Cognitive Psychology, 12,* 97-136.

von der Malsburg, C. (1973). Self-Organizing of orientation sensitive cells in the striate cortex. *Kybernetik, 14,* 85-100.

Waltz, D. L., & Pollack, J. B. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science, 9,* 51-74.

Willshaw, D. (1981). Holography, associative memory, and inductive generalization. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory.* Hillsdale, NJ: Erlbaum.