PART IV

PSYCHOLOGICAL PROCESSES

The chapters in this section are designed to show how parallel distributed processing has changed the way we think about the mechanisms of cognition. Each chapter describes a model of some mental process. Each shows how the model captures some basic aspects of the process. And each offers an alternative to other, earlier ways of thinking about aspects of the process.

The models described in these chapters were motivated by a number of different considerations. Quite often, they were motivated by the feeling that parallel distributed processing mechanisms provide a very natural and appropriate set of mechanisms for dealing with some aspect of cognition that has not been dealt with successfully by other approaches. Quite often, they were motivated by an attempt to meet computational challenges posed by the processes they address. And quite often, they were motivated by attempts to extend the domain of PDP models, to encompass processes and phenomena we did not know how to model at the outset.

The first chapter of the section, Chapter 14, explains how parallel distributed processing can take us beneath the surface of schemata, to a level of description that allows us to see how we can preserve the desirable characteristics of schemata and at the same time make them more flexible, more sensitive to context, more adaptable. This chapter also begins to show how PDP mechanisms can be used as the building blocks from which we may construct sequential thought processes, such as problem solving.

The next two chapters consider issues in perception. Chapter 15 presents a model of speech perception and shows how it provides a unified framework for capturing a number of aspects of speech. Here, the goal was to develop a model that accounts in detail for psychological data on the process of speech perception and, at the same time, to begin to deal with several computational problems that make the extension of PDP models to speech a challenging and stimulating task. The model introduces a processing structure called *the Trace*, a dynamic working memory, in which units that stand for hypotheses about the contents of an utterance at different points in time can interact. The parallel distributed processing that occurs in the Trace allows the model to account for contextual influences on phoneme identification, and the simple competitive interactions among hypotheses representing competing interpretations of the same portion of an utterance allow the model to segment and identify the words in an utterance in a simple and integrated fashion. The model accounts for a wide range of data in a direct and coherent way, and shows how PDP mechanisms offer new ways of interpreting several phenomena, such as categorical perception and the perception of phonologically regular nonwords.

The model described in Chapter 15 purchases its parallel processing capabilities by duplicating the same hardware to cover each time-slice of the Trace. This idea seems incorrect; in general, it seems more plausible to view the Trace not as a fixed processing structure, but as one that is dynamically configured in the course of processing, using knowledge of contingencies between hypotheses to construct the Trace on the fly. Chapter 16 develops a model called the Programmable Blackboard Model of Reading (PABLO for short) that does just this, though for printed, as opposed to spoken input. This model is based on the idea of *connection information distribution*—roughly, the idea is to use information stored in one part of a processing system to *set* or *program* connections in another part of the same system. Two related simulation models based on this idea are applied to a number of aspects of reading that could not be addressed by the interactive activation model of word recognition, which was described in Chapter 1.

Neither TRACE nor PABLO really extend above the word level to deal with the larger syntactic and semantic structures that organize words into sentences; these levels are considered in Chapter 19, which we will describe a bit more below.

Chapters 17 and 18 describe distributed models of different aspects of learning and memory. Chapter 3 (on Distributed Representations) provides useful background for these chapters, as well as for Chapter 19. Chapter 17 considers an existing dilemma for models of memory—whether to store summary representations in memory or whether to store an enumeration of specific experiences. The chapter

points out that with distributed representations, you can have it both ways. In the model, the (long-term) memory trace of an event is the change or *increment* to the connections that results from the event. Functional equivalents of summary representations (e.g., prototypes) emerge naturally from the superimposition of memory traces of specific events. Traces of recent or often-repeated events can coexist with the summary representation. The chapter also illustrates that the same composite memory trace can learn several different prototypes from exemplars, without ever being informed—and indeed, without having to "figure out"—which exemplars belong to each category. The model uses the delta rule (examined in Chapters 2, 8, and 11) for adjusting connection strengths and has some advantages over some distributed memory models, but it contains no hidden units, and so is not capable of overcoming what we call the "linear predictability constraint" on the set of patterns it can learn. At the end of the chapter, we illustrate with a simple example simulation how the model can be extended with hidden units that are trained with the aid of the generalized delta rule discussed in Chapter 8.

Chapter 18 draws out some other implications of distributed models of learning and memory. It considers how knowledge underlying the lawful use of language might be represented in a PDP model and how that knowledge might be acquired. More generally, it shows how distributed representations provide an alternative to the conventional view that linguistic knowledge is represented in the form of explicit (though inaccessible) rules. The chapter considers a paradigm case of rule learning from the language acquisition literature—the acquisition of the past tense by children acquiring English as their first language. This case is often cited as an instance of rule acquisition par excellence because of the fact that children "regularize" irregular verbs at one stage, often saying "goed," for example. The model we describe in this chapter exhibits this and many other aspects of the acquisition process, and it does this by using very simple learning mechanisms. Lawful behavior emerges from the superposition of changes to connection strengths. The representation of the rules of past-tense formation is implicit in the resulting connection strengths and is acquired without the aid of any device that relies on the formulation and testing of explicit but inaccessible rules.

Many of the chapters we have been describing deal with aspects of language, but none of them get much beyond the processing of individual words. Does this indicate that PDP models are inappropriate for capturing higher levels of language structure and processing? We think not. Indeed, one of our goals has been to work toward the development of PDP models of sentence processing. As Chapter 19 indicates, much of the groundwork has now been laid. That chapter describes a

distributed model that brings the benefits of parallel distributed processing to the processing of simple sentences, and shows how a PDP network can be configured to produce a representation that can capture the underlying case structure of simple sentences. The model exhibits a number of very nice properties. It can assign arguments of sentences to the appropriate case roles based on word-order information, based on the mutual selectional constraints imposed by the different words in the sentence, and based on word-order and mutual constraints working together; it can choose the appropriate "case frame" for a verb on the basis of the content and configuration of the arguments in the sentence; it can choose the appropriate reading of a semantically ambiguous word based on constraints imposed by the other arguments in the sentence; it can fill in default values for missing arguments; and it can generalize its case-role assignments to novel verbs if it is given a representation of some of the semantic features of the verb. The model has not yet reached the stage where it can process sentences with embedded clauses, but we suggest three ways in which it might be extended to do so, including one that relies on the use of the connection information distribution mechanism described in Chapter 16 and one that involves "true" recursion. These suggestions indicate that PDP mechanisms are capable of processing recursively defined structures, contrary to prevalent belief.

Similarities and Differences Between Models

Each of the models described in these chapters differs in detail from all of the others. One model uses continuous-valued, asynchronous units; several others use continuous, sychronous units; still others use stochastic units. In some models, activation values can range from 1 to -1; in others, they can only range from 1 to 0, or slightly below 0. All the models, however, are examples of the class of PDP models, as described in Chapter 2, and their minor differences are in most cases incidental to their behavior. Where these detailed assumptions seem to make a difference, we point it out, but in general, we do not think that much hinges on these differences.

One characteristic that differentiates some of the models from the others requires some comment: The models in Chapters 15 and 16 use local representations, while the models in the other chapters use distributed representations. Even this does not reflect a fundamental difference in the philosophy of the models; rather, they reflect differences in the points we wanted to make and the issues we wanted to raise in the different chapters. In fact, we believe that both the TRACE model and

the models described in Chapter 16 could be made more efficient by the use of distributed rather than local representations, for reasons discussed in Chapters 3 and 12. It is also true that nearly all of the models can be seen as either local or distributed, depending on one's point of view. They are local, in the sense that each unit generally stands for a particular conceptual object or feature, or perhaps a (coarse coded) conjunction of features. They are distributed, in the sense that any given stimulus object—a phoneme, a word, a sentence, a scene—produces a pattern of activation over a very large number of units. Whether we see the models as distributed or local is primarily a matter of whether we are looking at the forest or the trees.

Stepping Stones

Most of the models we present represent the result of a long series of explorations. In some cases, we learned a great deal from our earlier attempts, and the problems we had with them lead natually to better formulations. In other cases, our early attempts were failures we did not know at first how to overcome. For example, an early version of the model of past tense learning could be made to produce overgeneralization errors, but it could only process verbs consisting of a consonant, a vowel, and a consonant. It took us two years to formulate a better representation.

We think the models we describe in these chapters demonstrate progress in the development of PDP models of psychological processes. But none of the models is *the final word.* Each model has its weaknesses, as well as its strengths. We have not solved all the problems; rather, we have suggested an approach that shows promise of leading us closer to their solution. We offer these models in that spirit: as stepping stones along the way to a deeper understanding of the microstructure of cognition.

pg 6
intentionally
blank

CHAPTER 14

Schemata and Sequential Thought Processes in PDP Models

D. E. RUMELHART, P. SMOLENSKY, J. L. McCLELLAND
and G. E. HINTON

One of our goals for this book is to offer an alternative framework for viewing cognitive phenomena. We have argued that talk at the level of units and activations of units is the preferable way to describe human thought. There is, however, already an established language for discussing cognitive phenomena. In this chapter we wish to address the relationship between some of the key established concepts and our parallel distributed processing models. There are many important concepts from modern cognitive science which must be explicated in our framework. Perhaps the most important, however, is the concept of the *schema* or related concepts such as scripts, frames, and so on. These large scale data structures have been posited as playing critical roles in the interpretation of input data, the guiding of action, and the storage of knowledge in memory. Indeed, as we have argued elsewhere (cf. Rumelhart, 1980), the schema has, for many theorists, become the basic building block of our understanding of cognition. Yet, the PDP language we are proposing is devoid of terms such as schemata, scripts, frames, and so forth. Instead, we have proposed building blocks at a much more microlevel—at the level of units, activations, and similar "low-level" concepts. Interestingly, it was struggling with the concept of the schema and some of its difficulties that led one of us (DER) to an exploration of PDP models to begin with. It was therefore with

some priority that we began to develop an interpretation of the schema in the language of parallel distributed processing.[1]

Perhaps the first thought that comes to mind is to map the notion of the schema onto the notion of the unit. This does, indeed, capture some of the important aspects of the schema. In particular, the unit is an element, like the schema, which monitors its inputs searching for a good fit and takes on a value which represents how well its inputs fits its own internal criteria. However, such an identification misses much of what makes the schema a powerful conceptual tool. In particular, there is no analog to the *variable* or *default values.* There is no notion of the internal structure of the schema nor many of the other important aspects of schemata. Moreover, the scale is wrong. Schema theorists talk of schemata for rooms, stories, restaurants, birthday parties, and many other high-level concepts. In our parallel distributed processing models, units do not tend to represent such complex concepts. Instead, units correspond to relatively simple features or as Hinton (1981a) calls them *microfeatures.* If we are to do justice to the concept of the schema, we are going to have to look beyond the individual unit. We are going to have to look for schemata as properties of entire networks rather than single units or small circuits. In the following sections we show how features of networks can capture the important features of schemata. Since our interpretation is clearest in the subset of PDP models that can be characterized as *constraint satisfaction* networks, it will be useful to first describe that class of models and provide a language for talking about their properties.

PARALLEL DISTRIBUTED PROCESSING MODELS AS CONSTRAINT SATISFACTION NETWORKS

It is often useful to conceptualize a parallel distributed processing network as a *constraint network* in which each unit represents a hypothesis of some sort (e.g., that a certain semantic feature, visual feature, or acoustic feature is present in the input) and in which each connection represents constraints among the hypotheses. Thus, for example, if feature B is expected to be present whenever feature A is,

---

[1] All of the authors have contributed to the ideas expressed in this chapter. Smolensky's slightly different framework is sketched in Chapter 6. Hinton's view of the microstructure of symbols is sketched in J. A. Anderson and Hinton (1981, pp. 29-32), and McClelland (1981) shows how PDP networks can be employed to fill default values (see the discussion in Chapter 1). While we all agree with the flavor of the current discussion not all of us endorse the exact details.

there should be a positive connection from the unit corresponding to the hypothesis that A is present to the unit representing the hypothesis that B is present. Similarly, if there is a constraint that whenever A is present B is expected *not* to be present, there should be a negative connection from A to B. If the constraints are weak, the weights should be small. If the constraints are strong, then the weights should be large. Similarly, the inputs to such a network can also be thought of as constraints. A positive input to a particular unit means that there is evidence from the outside that the relevant feature is present. A negative input means that there is evidence from the outside that the feature is not present. The stronger the input, the greater the evidence. If such a network is allowed to run it will eventually *settle* into a locally optimal state in which as many as possible of the constraints are satisfied, with priority given to the strongest constraints. [2] The procedure whereby such a system *settles* into such a state is called *relaxation.* We speak of the system *relaxing* to a solution. Thus, a large class of PDP models, including the interactive activation model of word perception, are constraint satisfaction models which settle on locally optimal solutions through the process of relaxation.

Figure 1 shows an example of a simple 16-unit constraint network. Each unit in the network represents a hypothesis concerning a vertex in a line drawing of a Necker cube.[3] The network consists of two interconnected subnetworks—one corresponding to each of the two global interpretations of the Necker cube. Each unit in each network is assumed to receive input from the region of the input figure—the cube—corresponding to its location in the network. Each unit in the Figure is labeled with a three letter sequence indicating whether its vertex is hypothesized to be front or back (F or B), upper or lower (U or L), and right or left (R or L). Thus, for example, the lower left-hand unit of each subnetwork is assumed to receive input from the lower left-hand vertex of the input figure. The unit in the left-hand network represents the hypothesis that it is receiving input from a lower left-hand vertex in the front surface of the cube (and is thus labeled FLL), whereas the one in the right subnetwork represents the hypothesis that it is receiving input from a lower left vertex in the back surface (BLL).

---

[2] Actually, these systems will in general find a locally best solution to this constraint satisfaction problem. It is possible under some conditions to insure that the "globally" best solution is found through the use of stochastic elements and a process of annealing (cf. Chapters 6 and 7 for a further discussion).

[3] J. A. Feldman (1981) has proposed an analysis of the Necker cube problem with a somewhat different network. Although the networks are rather different, the principles are the same. Our intention here is not to provide a serious account of the Necker cube phenomena, but rather to illustrate constraint networks with a simple example.

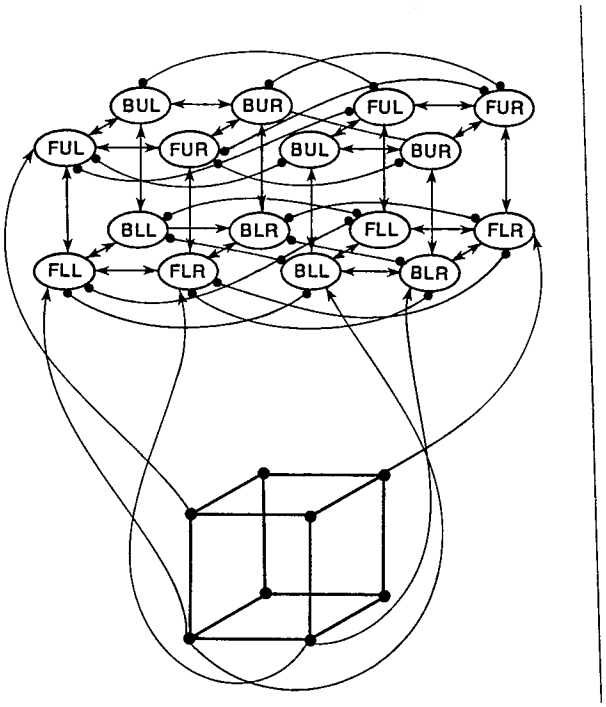


FIGURE 1. A simple network representing some of the constraints involved in perceiving the Necker cube.

Since there is a constraint that each vertex has a single interpretation, these two units are connected by a strong negative connection. Since the interpretation of any given vertex is constrained by the interpretations of its neighbors, each unit in a subnetwork is connected positively with each of its neighbors within the network. Finally, there is the constraint that there can only be one vertex of a single kind (e.g., there can only be one lower left vertex in the front plane FLL). There is a strong negative connection between units representing the same label in each subnetwork. Thus, each unit has three neighbors connected positively, two competitors connected negatively, and one positive input from the stimulus. For purposes of this example, the strengths of connections have been arranged so that two negative inputs exactly balance three positive inputs. Further, it is assumed that each unit receives an excitatory input from the ambiguous stimulus pattern and that each of these excitatory influences is relatively small. Thus, if all three of a unit's neighbors are on and both of its competitors are on, these effects would entirely cancel out one another; and if there was a small input from the outside, the unit would have a tendency to come on. On the other hand, if fewer than three of its neighbors were on and both of its

competitors were on, the unit would have a tendency to turn off, even with an excitatory input from the stimulus pattern.

In the last paragraph we focused on the individual units of the networks. However, it is often useful to focus not on the units, but on entire *states* of the network. In the case of binary (on-off or 0-1) units, there is a total of 2^{16} possible states in which this system could reside. That is, in principle, each of the 16 units could have the value either 0 or 1. In the case of continuous units, in which each unit can take on any value between 0 and 1, the system can, in principle, take on any of an infinite number of states. Yet, because of the constraints built into the network, there are only a few of those states in which the system will settle. To see this, consider the case in which the units are updated asynchronously, one at a time. During each time slice, one of the units is chosen to update. If its net input exceeds 0 its value will be pushed toward 1, otherwise its value will be pushed toward 0, using the activation rule from the word perception model:

$$a_j(t+1) = a_j(t) + \begin{cases} net_j(1 - a_j(t)) & net_j > 0 \\ net_j a_j(t) & \text{otherwise.} \end{cases}$$

Here, $a_j(t)$ stands for the activation of unit j at time t, and $net_j(t)$ stands for the net input to unit j at t. $net_j(t)$ is simply the sum of the excitatory and inhibitory influences on unit j:

$$e_j(t) \sum_{i \neq j} w_{ji} a_i(t)$$

where $e_j(t)$ is the external input to unit j at t and w_{ji} is the weight on the connection to unit j from unit i.

Imagine that the system starts with all units off. A unit is then chosen at random to be updated. Since it is receiving a slight positive input from the stimulus and no other inputs, it will be given a positive activation value. Then another unit is chosen to update. Unless it is in direct competition with the first unit, it too will be turned on. Eventually, a coalition of neighboring units will be turned on. These units will tend to turn on more of their neighbors in the same subnetwork and turn off their competitors in the other subnetwork. The system will (almost always) end up in a situation in which all of the units in one subnetwork are fully activated and none of the units in the other subnetwork are activated. That is, the system will end up interpreting the Necker cube as either facing left or facing right. Whenever the system gets into a state and stays there, the state is called a *stable state* or a *fixed point* of the network.

Figure 2 shows the output of three runs of a simulation based on this network. The size of the square indicates the activation value of each

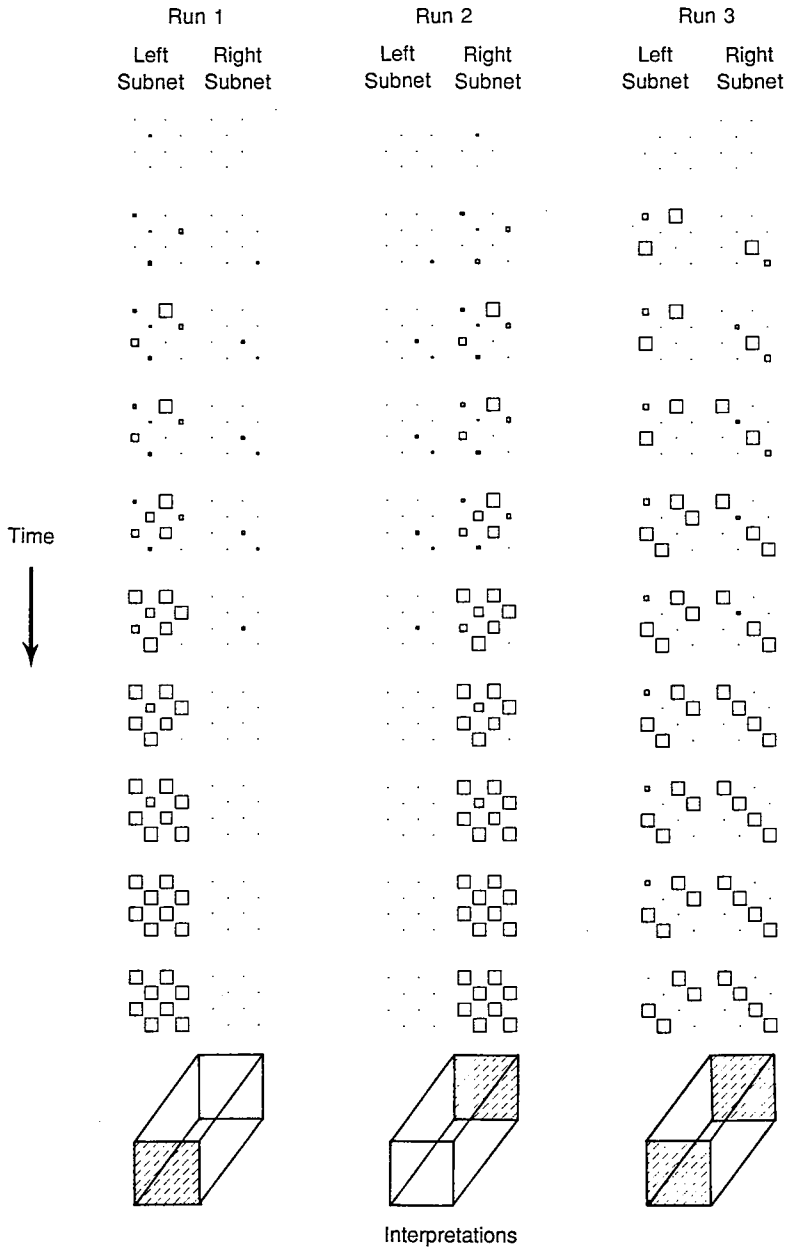


FIGURE 2. Three runs of a simulation based on this network. The size of the square indicates the activation value of each unit. The units are arranged in the shape of the subnetwork with each square shown in its position corresponding to the vertex of the cube from which it is receiving input. The states are shown after every second update.

unit. The units are arranged in the shape of the subnetwork with each square shown in its position corresponding to the vertex of the cube from which it is receiving input. The system begins with a zero activation value on all units—represented by single dots. Then, once each time slice, at most one unit is changed. On each run the system winds up in a state in which each unit has a value of either 0 or 1 (designated by a large square). The first two runs are most typical of the system. In this case, the inputs are low relative to the strength of the constraints among units. When low inputs are involved, the system virtually always winds up either in the state in which all of the units in the left-hand network are turned on and all of the units in the right-hand are off or vice versa. These final stable states correspond to the interpretations of a left-facing and right-facing cube as illustrated in the figure for the first and second run respectively. The third example of simulation results is much more aberrant and was generated with a high input value. With a high input value, the system can occasionally get a third interpretation of the Necker cube. This is the "impossible" cube with two front faces illustrated in the figure. Thus, of the 2^{16} possible states of the system, only two are ever reached with low input values and only three are ever reached at all. The constraints implicit in the pattern of connections among the units determines the set of possible stable states of the system and therefore the set of possible interpretations of the inputs.

Hopfield (1982) has shown that it is possible to give a general account of the behavior of systems such as this one (with symmetric weights and asynchronous updates). In particular, Hopfield has shown that such systems can be conceptualized as minimizing a global measure which he calls the *energy* of the system through a method of *gradient descent* or, equivalently, maximizing the constraints satisfied through a method of *hill climbing*. In particular, Hopfield has shown that the system operates in such a way as to always move from a state that satisfies fewer constraints to a state that satisfies more constraints, where the measure of constraint satisfaction is given by[4]

$$G(t) = \sum_i \sum_j w_{ij} a_i(t) a_j(t) + \sum_i input_i(t) a_i(t).$$

[4] Note, the question of what to call this constraint satisfaction function is difficult. Hopfield uses the negation of this function and, by analogy to thermodynamics, calls it *energy*. This system can thus be said to settle into states of minimum energy. Similarly, Hinton and Sejnowski (Chapter 7) use the same terminology. Smolensky (Chapter 6) has a similar function which he calls *harmony* to emphasize that increasing values correspond to more harmonious accounts of the inputs. In this chapter we have chosen to use the language of constraint satisfaction and call the function G for measure of the goodness-of-fit of the state to its constraints.

Essentially, the equation says that the overall goodness-of-fit is given by the sum of the degrees to which each pair of units contribute to the goodness plus the degree to which the units satisfy the input constraints. The contribution of a pair of units is given by the product of their activation values times the weights connecting them. Thus, if the weight is positive, each unit wants to be as active as possible—that is, the activation values for these two units should be pushed toward 1. If the weight is negative, then as least one of the units should be 0 to maximize the pairwise goodness. Similarly, if the input constraint for a given unit is positive, then its contribution to the total goodness-of-fit is maximized by being the activation of that unit toward its maximal value. If it is negative, the activation value should be decreased toward 0. Of course, the constraints will generally not be totally consistent. Sometimes a given unit may have to be turned on to increase the function in some ways while decreasing it in other ways. The point is that it is the sum of all of these individual contributions that the system seeks to maximize. Thus, for every state of the system—every possible pattern of activation over the units—the pattern of inputs and the connectivity matrix **W** determines a value of the goodness-of-fit function. The system processes its input by moving upward from state to adjacent state until it reaches a state of maximum goodness. When it reaches such a *stable state* or *fixed point* it will stay in that state and it can be said to have "settled" on a solution to the constraint satisfaction problem or alternatively, in our present case, "settled into an interpretation" of the input.

It is important to see, then, that entirely *local* computational operations, in which each unit adjusts its activation up or down on the basis of its net input, serve to allow the network to converge towards states that maximize a *global* measure of goodness or degree of constraint satisfaction. Hopfield's main contribution to our present analysis was to point out this basic fact about the behavior of networks with symmetrical connections and asynchronous update of activations.

In general, since there are so many states, it is difficult to visualize the goodness-of-fit function over which the system is moving. In the present case, however, we can get a reasonably good image of this landscape. To begin, we can limit our consideration to those states in which a particular unit is either on or off since the system always ends up in such states. We can consider the states arrayed along two dimensions. One dimension corresponds to the number of units turned on in the left subnetwork and the other dimension corresponds to the number of units turned on in the right subnetwork. Thus, at (0,0) we locate the state in which no units are turned on. Clearly, by the above

equation such a state will have zero goodness of fit.[5] At (8,8) we have the state in which all of the units are turned on. At location (8,0) we have the state in which the units on the left network are all turned on and those on the right network are all off. At position (0,8) we have the state in which those in the left network are all off and those in the right network are all on. Each of those locations contain unique states. Now, consider the location (1,0) in which one unit from the left subnetwork and zero units in the right subnetwork are turned on. There are eight different states, corresponding to the eight different units in the left subnetwork that might have been turned on. In order to plot the goodness-of-fit landscape for this state space, we have plotted only the states at each location of the two-dimensional space with highest goodness-of-fit—i.e., the best state at each location. Figure 3 shows the landscape. In the figure, we are viewing the goodness landscape from about the (0,0) corner, the start state. Thus, the peak to the right corresponds to the goodness of the state in which all of the units in the left subnetwork are turned on and all in the right subnetwork are turned off. The peak at the upper left portion of the figure

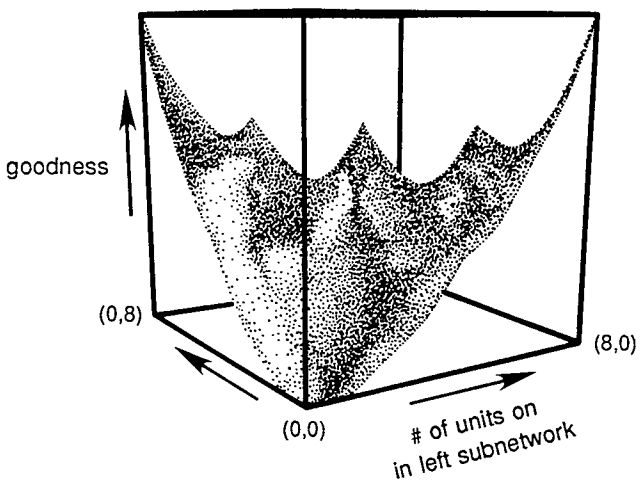


FIGURE 3. The goodness-of-fit surface for the Necker-cube network. The low point at the (0,0) corner corresponds to the start state. The peaks on the right and left correspond to the standard interpretations of the Necker cube, and the peak in the center corresponds to the impossible Necker cube illustrated in the previous figure.

[5] Note, zero goodness-of-fit is *not* the minimum goodness-of-fit attainable. In general, goodness-of-fit can be negative as well as positive. When there is negative goodness-of-fit, the system can always be made better by turning off all of the units.

corresponds to the state (0,8). The two peaks in the graph at (8,0) and (0,8) correspond to the two primary interpretations of the Necker cube. It should be clear that if we start a system at (0,0) and allow it to "hill climb" it will almost always end up at one of these two peaks. It might be noted, that there are three smaller peaks right in the middle of the surface. These local peaks are very hard to get to because the system is almost always swept from the start state uphill to one of the two major peaks. It is possible, by having large input values, to reach location (4,4). This peak corresponds to the impossible Necker cube illustrated in the previous figure.

The input to the system can be conceptualized as systematically modifying or *sculpting* the goodness landscape. This effect is illustrated in Figure 4. In this case, the same landscape has been plotted, except the units corresponding to the interpretation of the Necker cube as facing to the left receive more input than the corresponding units on the other subnetwork. (This could perhaps be done by slightly shading that face of the Necker cube.) What we see is a "sloping" goodness surface with the peak associated with the interpretation of the Necker cube as left facing.

To summarize, then, there is a large subset of parallel distributed processing models which can be considered constraint satisfaction models. These networks can be described as carrying out their information processing by climbing into states of maximal satisfaction of the

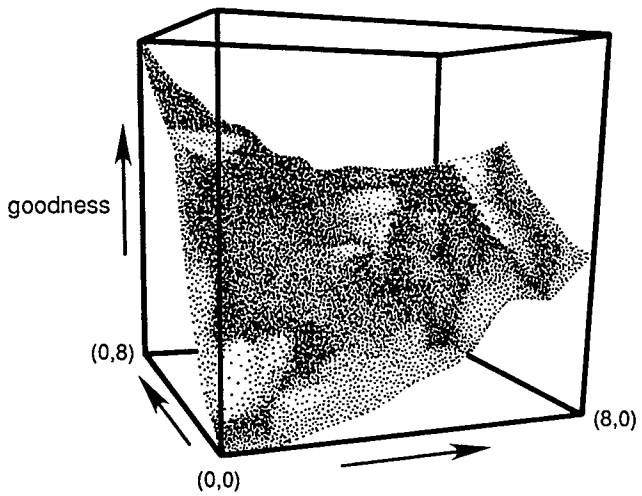


FIGURE 4. The distortions of the goodness landscape when a large input is given to the units corresponding to the front face of a left-facing cube. The figure shows only one major peak corresponding to the view of the left-facing cube.

constraints implicit in the network. A very useful concept that arises from this way of viewing these networks is that we can describe the behavior of these networks, not only in terms of the behavior of individual units, but in terms of properties of the network itself. A primary concept for understanding these network properties is the *goodness-of-fit landscape* over which the system moves. Once we have correctly described this landscape we have described the operational properties of the system—it will process information by moving uphill toward goodness maxima. The particular maximum that the system will find is determined by where the system starts and by the distortions of the space induced by the input. One of the very important descriptors of a goodness landscape is the set of maxima which the system can find, the size of the region that feeds into each maximum, and the height of the maximum itself. The states themselves correspond to possible interpretations, the peaks in the space correspond to the best interpretations, the extent of the foothills or skirts surrounding a particular peak determines the likelihood of finding the peak, and the height of the peak corresponds to the degree that the constraints of the network are actually met or, alternatively, to the goodness of the interpretation associated with the corresponding state.

CONSTRAINT SATISFACTION AND SCHEMATA

In the previous section we recounted a perspective on parallel distributed processing systems. In this section we address, again, the nature of the schema and relate it to constraint satisfaction systems and PDP models. We will proceed by first recounting some of the history of the concept of schemata, then by offering an interpretation of the schema in terms of PDP models, by giving a simple example, and finally showing how the various properties attributed to schemata are, in fact, properties of the PDP networks of the kind we have been discussing.

The schema, throughout its history, has been a concept shrouded in mystery. Kant's (1787/1963) use of the term has been provocative but difficult to understand. Bartlett's (1932) usage has long been decried for its vagueness. Piaget (1952) used the term schema, but it was difficult to come up with a consistent interpretation of Piaget's own views on the matter. Throughout most of its history, the notion of the schema has been rejected by mainstream experimental psychologists as being too vague. As a result, the concept of the schema was largely shunned until the mid-1970s. The concept was then revived by an attempt to offer a more clearly specified interpretation of the schema in

terms of explicitly specified computer implementations or, similarly, formally specified implementations of the concept. Thus, Minsky (1975) postulated the concept of the frame, Schank and Abelson (1977) focused on the concept of the script, and Bobrow and Norman (1975) and Rumelhart (1975) developed an explicit notion of the schema. Although the details differed in each case, the idea was essentially the same. Perhaps Minsky (1975) was clearest in the motivation:

> It seems to me that the ingredients of most theories both in artificial intelligence and in psychology have been on the whole too minute, local, and unstructured to account—either practically or phenomenologically—for the effectiveness of common sense thought. The "chunks" of reasoning, language, memory, and "perception" ought to be larger and more structured, and their factual and procedural contents must be more intimately connected in order to explain the apparent power and speed of mental activities. (p. 211)

Minsky and the others argued that some higher-level "suprasentential" or, more simply, conceptual structure is needed to represent the complex relations implicit in our knowledge base. The basic idea is that schemata are data structures for representing the generic concepts stored in memory. There are schemata for generalized concepts underlying objects, situations, events, sequences of events, actions, and sequences of actions. Roughly, schemata are like models of the outside world. To process information with the use of a schema is to determine which model best fits the incoming information. Ultimately, consistent configurations of schemata are discovered which, in concert, offer the best account for the input. This configuration of schemata together constitutes the *interpretation* of the input.

Different theorists have proposed more or less concrete specifications of the exact nature of these higher-level structures, but somehow none of them has ever really been adequate. None of them ever captured all of the qualitative characteristics that schemata were supposed to have. For example, a schema is supposed to be a kind of generative thing, which is flexible but which can produced highly structured interpretations of events and situations. Many representational formats have been proposed in an attempt to meet these criteria. For example, Rumelhart (1975) chose as a representation for the schema, a notation rich in generative capacity, namely, the rewrite rules from generative linguistics. Although the generativity of the rewrite rules and the idea that the structure is "constructed" in the process of interpretation is well captured by the rewrite rules, the nonprocedural character of such a system seems wrong. Some more active representation seems

necessary. Moreover, the important notions of "default values," variables, and so forth are poorly represented by the rewrite notation. Minsky (1975) and Schank and Abelson (1977) employed passive data structures with slots and explicit default values. These representations are better but are not active and seem to lack the flexibility and generativity that the schema requires. Rumelhart (1977) proposed a representation in which schemata are special kinds of procedures. This view was most completely explicated in Rumelhart (1980). Attempts to build explicit models employing this view, however, have proven unsuccessful. The representation is simply too unwieldy.

It should be clear from the foregoing that there are two distinct ways in which the term schema can be used. On the one hand, it is used to refer to an idea which is common to the work of Kant, Bartlett, Piaget, Minsky, Schank and Abelson, Norman and Bobrow, Rumelhart and Ortony, and many others. This is an idea that has evolved over the years and through the eyes of many different theorists. Many people have sought to clarify and further develop the idea. On the other hand, the term schema is used to refer to one of a large number of instantiations of the general idea of the schema. These explicit schema models are always only pale representations of the underlying intuitions. Whenever a new instantiation of the schema idea is developed, a new perspective is offered on the underlying idea. What we hope to do in this chapter is to propose an alternative to the conventional representation of the schema and at the same time, through the development of a new perspective on schemata, sharpen the idea and develop a system which better captures our intuitions of the nature of the human information-processing system.

One important feature of schemata proposed by Rumelhart and Ortony (1977) has never actually been included in any implementation of the idea. This involves the nature of variable constraints and the filling of default values. The variable constraints associated with each variable serve two functions. On the one hand, they are important for determining whether a particular candidate is an allowable assignment for a variable and, if the variable remains unfilled, are used in the assignment of a default value. These constraints should not be considered absolute. Rather it was proposed that variable constraints should be considered as distributions of possible values. The nearer to the mode of the distribution, the better the variable filler. Moreover, the mode could itself be considered the default value. Importantly, however, there are interdependencies among the possible slot fillers. If one variable is filled with a particular value then it changes the default for the other variables. It was therefore proposed that the variable constraints (and the fillers of the default values) should be considered *multivariate distributions* in which the default value for a particular

variable is determined by the values filling the other slots. This idea was difficult to integrate with any of the conventional semantic networks or similar representational formats for schemata. As we shall see, this is a central feature of the PDP analog to schemata.

If schemata are to work as a basis for models of cognitive processing, they must be very flexible objects—much more flexible than they really ever have been in any actual implementations. This is a sort of dilemma. On the one hand, schemata are the structure of the mind. On the other hand, schemata must be sufficiently malleable to fit around most everything. None of the versions of schemata proposed to date have really had these properties. How can we get a highly structured schema which is sufficiently rich to capture the regularities of a situation and to support the kinds of inferences that schemata are supposed to support and at the same time is sufficiently pliable to adapt to new situations and new configurations of events?

On our current view, the answer is simple. Schemata are not "things." There is no representational object which is a schema. Rather, schemata emerge at the moment they are needed from the interaction of large numbers of much simpler elements all working in concert with one another. Schemata are not explicit entities, but rather are implicit in our knowledge and are created by the very environment that they are trying to interpret—as it is interpreting them.[6] Roughly, the idea is this: Input comes into the system, activating a set of units. These units are interconnected with one another, forming a sort of constraint satisfaction network. The inputs determine the starting state of the system and the exact shape of the goodness-of-fit landscape. The system then moves toward one of the goodness maxima. When the system reaches one of these relatively stable states, there is little tendency for the system to migrate toward another state.

The states themselves are the product of the interaction among many groups of units. Certain groups, or subpatterns of units tend to act in concert. They tend to activate one another and, when activated, tend to inhibit the same units. It is these coalitions of tightly interconnected units that correspond most closely to what have been called schemata. The stable pattern as a whole can be considered as a particular configuration of a number of such overlapping patterns and is determined by

---

[6] Hofstadter (1979) expresses essentially the same view in his book *Gödel, Escher, Bach* when the Anteater says:

> My "symbols" are ACTIVE SUBSYSTEMS of a complex system, and they are composed of lower-level active subsystems . . . They are therefore quite different from PASSIVE symbols, external to the system, such as letters of the alphabet of musical notes, which sit there immobile, waiting for an active system to process them. (p. 324)

the dynamic equilibrium of all of these subpatterns interacting with one another and with the inputs. Thus, the maxima in the goodness-of-fit space correspond to interpretations of the inputs or, in the language of schemata, configurations of instantiated schemata. In short, they are those states that maximize the particular set of constraints acting at the moment. Depending on the context and the inputs, the system will be closer to one or another of the peaks in the goodness-of-fit function at the outset and will usually find the closest one. This interpretation, we believe, captures almost all of the important aspects of the schema with a view that is at once more flexible than the previous interpretations and yet highly structured. The degree of structure depends on the tightness of the coupling among the coalitions of units which correspond to the schemata in question. Thus, the language of schemata and schema theories should be considered an approximation to the language of PDP. In those cases in which there are coalitions of units that tend to work together, we have a rather close correspondence to the more conventional notion of a schema. In those cases in which the units are more loosely interconnected, the structures are more fluid and less schema-like. Often, knowledge is structured so that there are relatively tight connections among rather large subsets of units. In these cases, the schema provides a very useful description.

One important difference between our interpretation of schemata and the more conventional ones is that in the conventional story, schemata are stored in memory. Indeed, they are the major *content of memory.* In our case, *nothing stored corresponds very closely to* a schema. What is stored is a set of connection strengths which, when activated, have implicitly in them the ability to generate states that correspond to instantiated schemata. This difference is important—especially with regard to learning. There is no point at which it must be decided to create this or that schema. Learning simply proceeds by connection strength adjustment, according to some simple scheme such as those we discuss in various places in this book. As the network is reorganized as a function of the structure of its inputs, it may come to respond in a more or less schema-like way.

We now turn to an example to illustrate the various aspects of these PDP networks and show that many of those features that prompted the invention of schemata in the first place are present in these networks. At the same time, we show that certain features that are problematic with conventional representations of schemata are better dealt with in the PDP language.

An Example

Consider our knowledge of different kinds of rooms. We all have a clear idea of what a typical kitchen or bathroom or living room or bedroom or office looks like. We know that living rooms have sofas and easy chairs, but they don't usually have ovens or bathtubs and that offices have desks and typewriters, but they don't usually have beds. On the other hand, kitchens, living rooms, and offices might all very well have telephones, carpets, etc. Our default bathroom is very small, our default kitchen is somewhat larger but still probably small relative to our default living room. We chose our knowledge of rooms and types of rooms as the primary example to illustrate the PDP representation of schemata. To begin, we need a constraint network that embodies the constraints implicit in our knowledge of rooms. We built our constraint network in the following way. We chose a set of 40 descriptors of rooms. These descriptors are listed in Table 1. We asked two subjects to imagine an office and then, for each of the 40 descriptors asked if the descriptor was accurate of that office. We then asked subjects to imagine a living room and asked about the 40 descriptors again. We then asked about a kitchen, a bathroom, and a bedroom. After finishing these five types of rooms we asked subjects to imagine another office, etc. We collected a total of sixteen judgments of the 40 descriptors on each of the five room types. This data served as the basis for creating our network.[7] In principle, we could imagine presenting each of these 80 room descriptions to the system and have it

TABLE 1

THE FORTY ROOM DESCRIPTORS

ceiling	walls	door	windows	very-large
large	medium	small	very-small	desk
telephone	bed	typewriter	bookshelf	carpet
books	desk-chair	clock	picture	floor-lamp
sofa	easy-chair	coffee-cup	ashtray	fireplace
drapes	stove	coffeepot	refrigerator	toaster
cupboard	sink	dresser	television	bathtub
toilet	scale	oven	computer	clothes-hanger

[7] This was not designed to be a formal experiment of any kind. Rather it was conceptualized as a method of quickly getting a reasonable data base for building an example. Some slight modifications in the data base were made in order to emphasize certain points in our example.

learn according to one or another learning rule we have discussed. Rather than doing that, however, we simply set the weights according to the following equation:

$$w_{ij} = -\ln \frac{p(x_i = 0 \;\&\; x_j = 1)p(x_i = 1 \;\&\; x_j = 0)}{p(x_i = 1 \;\&\; x_j = 1)p(x_i = 0 \;\&\; x_j = 0)}.$$

This equation is derived from a Bayesian analysis of the probability that unit x_i should be on given unit x_j is on and vice versa (see Hinton & Sejnowski, 1983). Four aspects of the weight equation should be noted:

- If the two units tend to be on and off together (i.e., the probability that $x_i = x_j$ is much greater than the probability that $x_i \neq x_j$), then the weight will be a large positive value.
- If, on the other hand, the probability that the two units take on different values (i.e., $x_i \neq x_j$) is much greater than the probability that they take on the same values (i.e., $x_i = x_j$), then the weight takes on a large negative value.
- If the two units come on and off independently (i.e., if $p(x_i = v_1 \;\&\; x_j = v_2) = p(x_i = v_1)p(x_j - v_2)$), then the weight between the two units is zero.
- The weights are symmetric (i.e., $w_{ij} = w_{ji}$).

In addition, each unit has a bias (constant input) which is given by

$$bias_i = -\ln \frac{p(x_i = 0)}{p(x_i = 1)}.$$

Note that if the unit is usually off, it has a negative bias; if it is usually on, it has a positive bias; and if it is equally often on or off, it has a zero bias. [8] The weight matrix estimated by this means is shown in Figure 5. The figure uses the method of Hinton and Sejnowski (Chapter 7) to display the weights. Each unit is represented by a square. The name below the square names the descriptor represented by each square. Within each unit, the small black and white squares represent the weights from that unit to each of the other units in the

[8] With a finite data base some of the probabilities mentioned in these two equations might be 0. In this case the values of weights are either undefined or infinite. In estimating these probabilities we began by assuming that everything occurs with some very small probability (.00001). In this way the equation led to finite values for all weights.

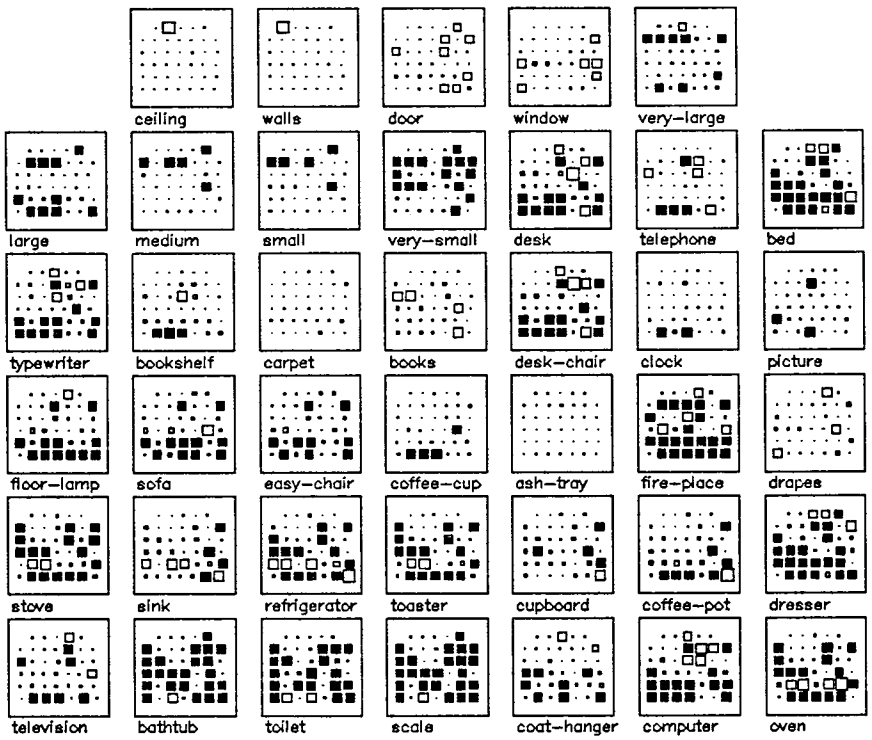


FIGURE 5. The figure uses the method of Hinton and Sejnowski (Chapter 7) to display the weights. Each unit is represented by a square. The name below the square names the descriptor represented by each square. Within each unit, the small black and white squares represent the weights from that unit to each of the other units in the system. The relative position of the small squares within each unit indicates the unit with which that unit is connected.

system. The relative position of the small squares within each unit indicates the unit with which that unit is connected. For example, the white square on the lower right-hand portion of the *refrigerator* units represents the strength of the connection between *refrigerator* and *oven*. White squares represent positive connections and black squares represent negative connections. The size of the square represents the strength of the connection. Thus, the fact that the square representing the connection from the *refrigerator* unit to the *stove* unit is large and white represents the fact that there is a strong positive weight between the two.

It should be noted that each of the units in this example is a visible unit in the sense that each can directly receive inputs from outside the network. There are no hidden units receiving inputs only from other

units in the network. We consider this example to be a simplified case. It is possible to imagine hidden units which respond to patterns among the input units. In the general case, we, of course, recognize that hidden units would be required to give different coalitions enough coherence. As we have pointed out elsewhere in the book (cf. Chapter 2 and Chapter 8), multilayer systems containing hidden units are sometimes required to carry out certain computations. In the present instance, however, the existence of hidden units would not change the basic properties of the network which we wish to illustrate. Such higher-level units are not required for the basic schema-like behavior of these networks and, in no case should such a unit be confused with a schema.

It should also be noted that we have chosen a rather high level of abstraction for this example. We have taken such features as *has television* as a *microfeature*. In a more realistic example, we would expect *television* to itself be a particular pattern over a set of units that are used to represent many different varieties of television. There might be many variations on the *television* pattern corresponding to variations among televisions. Moreover, since televisions in bedrooms may be systematically different (perhaps smaller) than televisions in living rooms, we would expect that these correlations would be picked up and there would be a context dependency between the particular version of *television* and the remaining objects in the room. In such a case the units that participate in the representation of television would play the role of a slot in a schema, and the particular pattern of activation on these units would represent the characteristics of the slot filler.

Figure 6 shows several examples of the processing of this network. These runs started by "clamping" one of the descriptors on (that is, by setting the value to 1 and not letting it change) and then letting the system find a goodness-of-fit maximum. In the first example, the descriptor *oven* was clamped on. In such a case, we expect that the system will bring those units most tightly bound to the *oven* unit on and turn off those units negatively correlated to *oven* or other units that it turns on. On the assumption that *oven* is a central feature of the *kitchen* schema, the pattern the system eventually turns on is just that which might be said to correspond to the default *kitchen* schema. The strengths of each of the 40 units is shown along with the "goodness-of-fit" of the state after every 20 updates. The system begins with *oven* and *ceiling* on and then adds *coffee-cup* (weakly), then *sink* and *refrigerator*, concludes that the room is *small*, adds *toaster* and *coffeepot* and finally ends up at a maximum with *ceiling*, *walls*, *window*, *small*, *telephone*, *clock*, *coffee-cup*, *drapes*, *stove*, *sink*, *refrigerator*, *toaster*, *cupboard*, *coffeepot*, and *oven*. In other words, it finds the default or prototype kitchen. Similarly, runs of the system starting with *desk*, *bathtub*, *sofa*, or *bed* clamped lead to goodness maxima corresponding to the prototype or default office,

bathroom, living room, or bedroom, respectively. It is, as previously noted, these maxima that we believe correspond roughly to instantiations of schemata for kitchens, offices, bathrooms, living rooms, and bedrooms. The system receives input in the form of having some of the descriptors clamped from the outside. It then finds the best interpretation of the input through this process of hill climbing. As it climbs, the system "fills in" the relevant descriptors of the scene in question.

In the case of the network we created for this example, there are essentially five maxima—one corresponding to each of the different room types. There are 2^{40} possible binary states in which the system could potentially settle, but, in fact, when it is started by clamping exactly one descriptor, it will only settle into one of five states. This roughly corresponds to the view that this data base contains five schemata defined over the set of 40 descriptors. There are, as we shall see, numerous subschemata which involve subpatterns within the whole pattern.

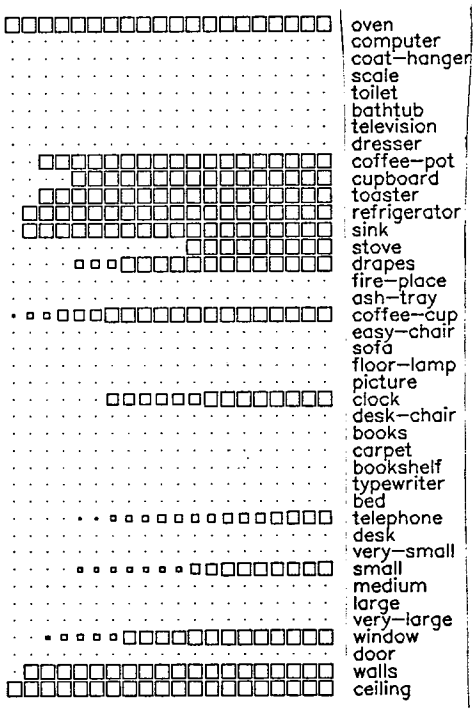
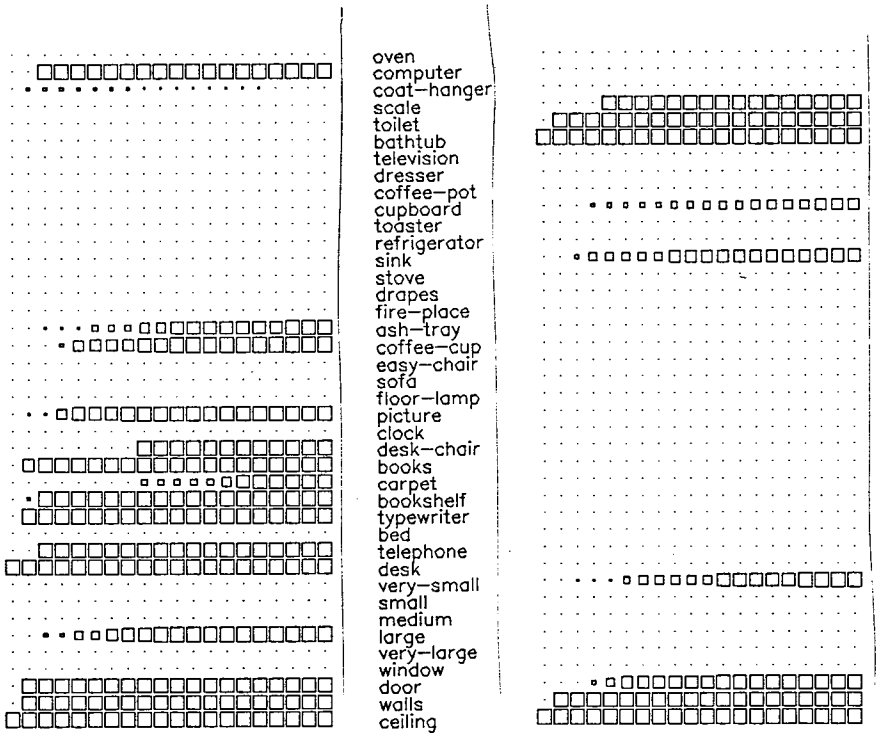
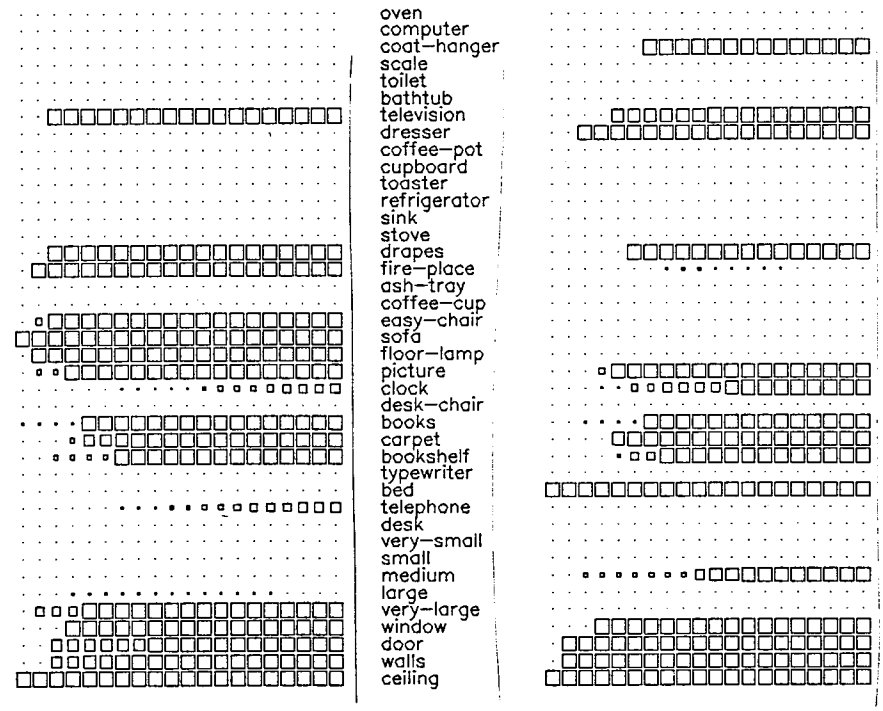


FIGURE 6. Five runs of the network from five different starting places. In each case, the unit *ceiling* and one other is clamped on. The clamping of *ceiling* represents information indicating that *room* is the domain of discussion. The other clamped units are *oven*, *desk*, *bathtub*, *sofa*, and *bed* in the five runs. In each case, the system settles on a prototype for the type of room most closely related to the clamped units.


oven
computer
coat–hanger
scale
toilet
bathtub
television
dresser
coffee–pot
cupboard
toaster
refrigerator
sink
stove
drapes
fire–place
ash–tray
coffee–cup
easy–chair
sofa
floor–lamp
picture
clock
desk–chair
books
carpet
bookshelf
typewriter
bed
telephone
desk
very–small
small
medium
large
very–large
window
door
walls
ceiling


oven
computer
coat–hanger
scale
toilet
bathtub
television
dresser
coffee–pot
cupboard
toaster
refrigerator
sink
stove
drapes
fire–place
ash–tray
coffee–cup
easy–chair
sofa
floor–lamp
picture
clock
desk–chair
books
carpet
bookshelf
typewriter
bed
telephone
desk
very–small
small
medium
large
very–large
window
door
walls
ceiling

If it was difficult to visualize the landscape for the case of the Necker Cube model described above with 2^{16} states, it is even more difficult with the 2^{40} states of this example. It is, however, possible to get some idea of the landscape in the region of the maxima by plotting the goodness function over a small subset of the goodness landscape. It should be recalled that the states of a system with 40 units can be considered to be a vector in 40-dimensional space. If, as in the present case, the units have a minimum and maximum value, then each point in the 40-dimensional hypercube is a possible state of the system. The states of the system in which all units are at their minimum or maximum values (the binary states of the system) correspond to the corners of the hypercube. Now, each maximum of our network falls at one corner of this 40-dimensional cube. The intersection of a plane and the hypercube will pick out a two-dimensional subset of all of the possible states of the system. Finally, since three points determine a plane, we chose subsets of three of the maxima and plotted the goodness-of-fit landscape for those states falling on the plane passing through those three points. Figure 7 shows the landscape for the plane passing through the maxima for *bedroom*, *office*, and *kitchen*. Note that there are three peaks on the graph, one corresponding to each of the maxima on the plane. Note also that there are "ridges" connecting the two maxima. These correspond to simple mixtures of pairs of concepts. The fact that the ridge connecting *bedroom* and *office* is higher than those connecting *kitchen* to either of the others indicates that *kitchen* is more

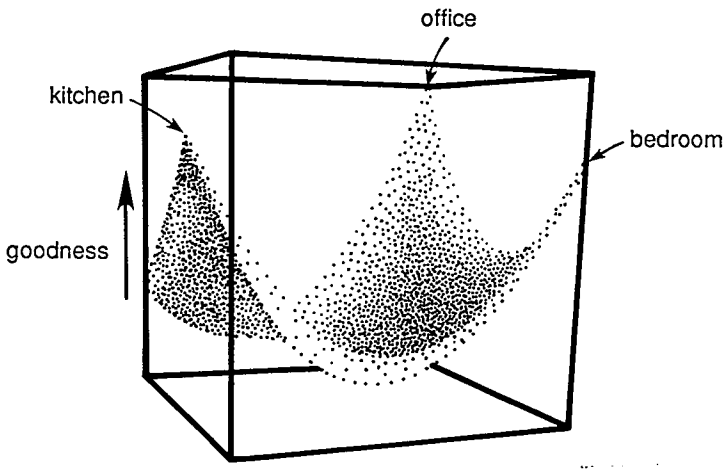


FIGURE 7. The value of the goodness function for the states on the plane passing through the three goodness maxima corresponding to the prototypes for *kitchen*, *bedroom*, and *office*.

distinctive from the other two than they are from each other. Figure 8 shows the plane containing *office*, *bedroom*, and *bathroom*. In this case we see that the goodness function sinks much lower between the *office-bedroom* axis and *bathroom*. This occurs because *bathroom* is much more different from the other two than is *kitchen*. In any of these cases, it should be recognized that given the starting configuration of the system it will simply find one of these maxima and thereby find one of these interpretations of the input. By contrast, Figure 9 shows the goodness-of-fit landscape on the plane passing through *bedroom*, *office*, and *living-room*. In order to get a clearer perspective on the surface, the angle of viewing was changed so we are looking at the figure from between *bedroom* and *office*. Clearly, the whole graph is greatly elevated. All points on the plane are relatively high. It is a sort of goodness plateau. These three points are essentially three peaks on a much larger mountain containing all three maxima. Finally, Figure 10 shows the goodness landscape on the plane containing the three most distinct prototypes—*bedroom*, *kitchen*, and *bathroom*. The goodness function dips well below zero in this plane. Mixtures of *kitchens*, *living-rooms*, and *bathrooms* are poor rooms indeed.

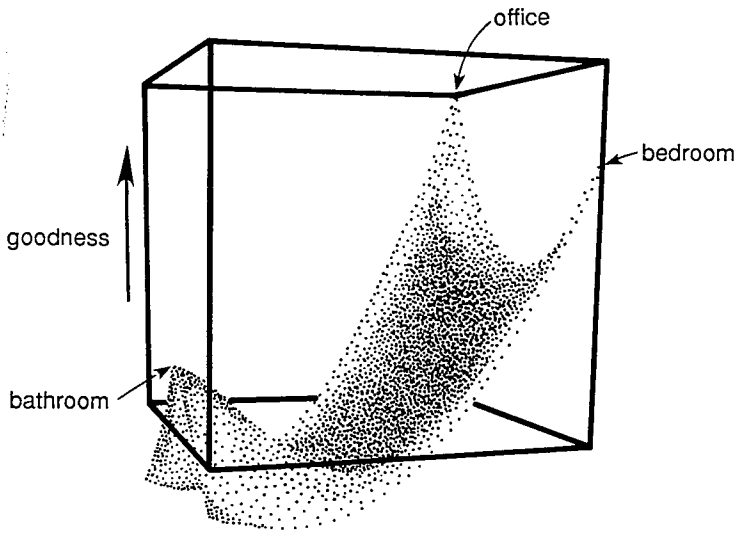


FIGURE 8. The value of the goodness function for the states on the plane passing through the three goodness maxima corresponding to the prototypes for *bathroom*, *bedroom*, and *office*.

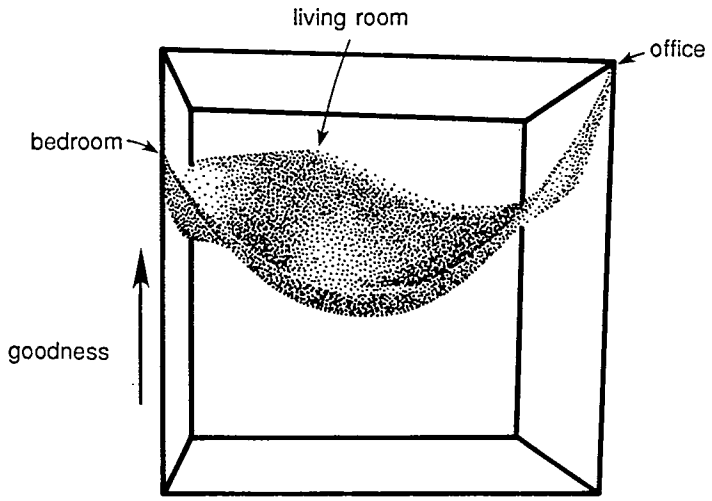


FIGURE 9. The value of the goodness function for the states on the plane passing through the three goodness maxima corresponding to the prototypes for *living-room*, *bedroom*, and *office*.

It should be mentioned that there are essentially two assumptions that can be made about the input. Under one assumption, inputs are clamped to either their minimum or maximum value and aren't allowed to move. That was the way inputs were treated in the present examples. Other times, it is convenient to imagine that inputs are merely biases feeding input into certain of the units. This is the way inputs were treated in the Necker cube example. These two ways of viewing inputs can be combined by assuming that the case of clamping corresponds to very, very strong biasing. So strong that internal constraints can *never* overcome the external evidence. The case of clamping is simpler, however. In this case there is no distortion of the goodness-of-fit landscape; certain states are simply not available. The system is forced to move through a different part of the state space. In addition to its effects on the region of the state space that is accessible, the input (along with the context) determines where the system begins its processing and therefore, often, which of the maxima it will find. Figure 11 illustrates this point. The figure shows the goodness function on the set of states on the plane passing through the start state, the *bedroom* maximum, and the *kitchen* maximum for two different inputs. In Figure 11A we have clamped the *bed* unit to be on. In Figure 11B we have clamped the *oven* unit on. It should be noted that to move from the start state to the *kitchen* peak in the first case involves climbing through a dip in goodness-of-fit. Since the system strictly goes

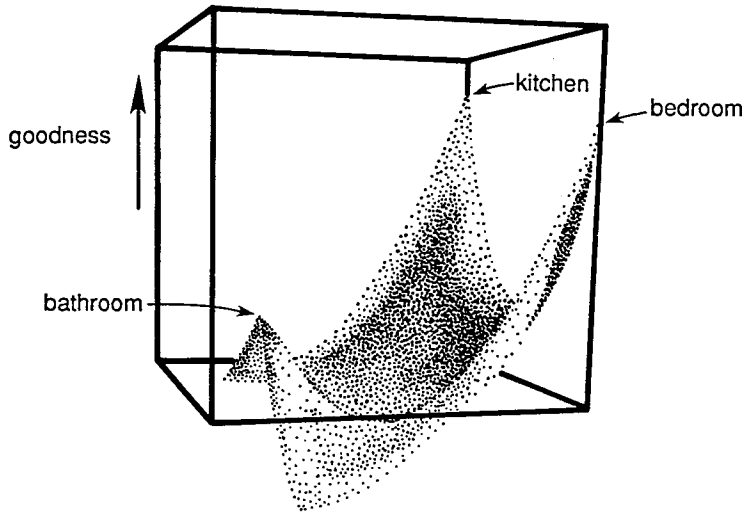


FIGURE 10. The value of the goodness function for the states on the plane passing through the three goodness maxima corresponding to the prototypes for *kitchen*, *bedroom*, and *bathroom*.

"uphill" it will be unable to reach the *kitchen* maximum and will move instead monotonically uphill toward the *bedroom* peak. Similarly, in Figure 11B with *oven* clamped there is a dip separating the start state from the *bedroom* maximum but a monotonically increasing slope flowing into the *kitchen* peak. Figure 11C shows, for comparison, the landscape from the start state when no units are clamped on. In this case, there is no dip separating the start state from either peaks, so the system can move to either maximum.

To summarize, we have argued that the maxima in the goodness-of-fit landscapes of our networks correspond to configurations of instantiated schemata. We have shown how these maxima are determined by coalitions among units and how the inputs determine which of the maxima the system will find. It should be clear that the multivariate distributions proposed by Rumelhart and Ortony are readily captured in the PDP framework. The values of each variable determine what values will be filled in for the other variables. We have yet to show that the kind of PDP system we have been describing really has all of the important properties of schemata. In the following section, we use the present example to illustrate these properties and discuss some of the advantages of our formulation over previous formulations of the schema idea.

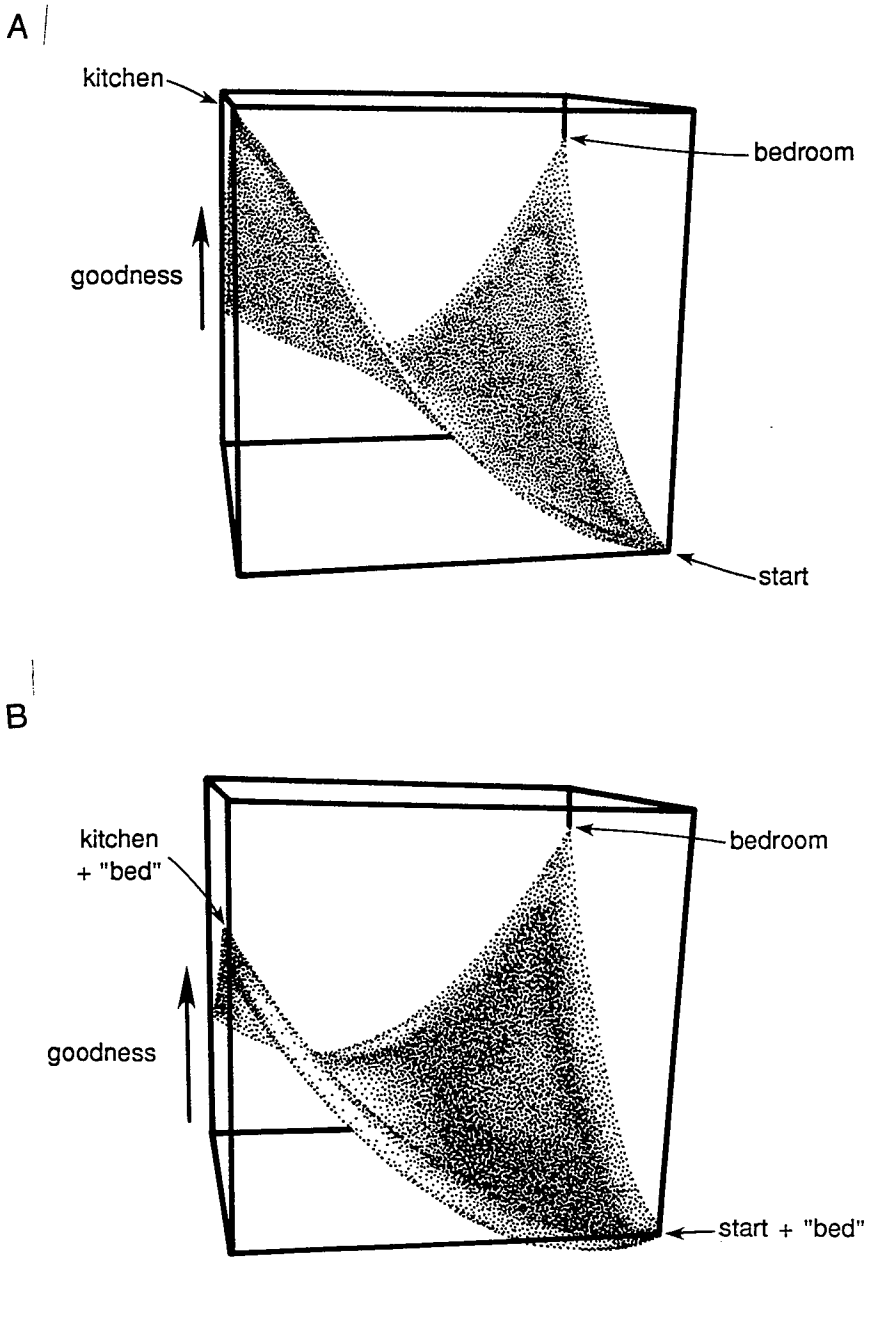
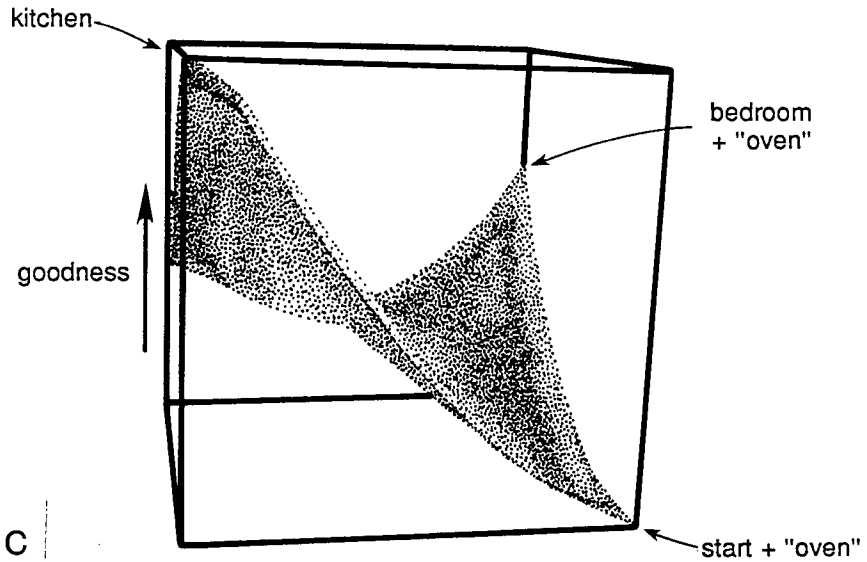

A
kitchen
bedroom
goodness
start
B
kitchen
+ "bed"
bedroom
goodness
start + "bed"

FIGURE 11.



C

FIGURE 11. The goodness function over the set of states on the plane passing through the start state, the *bedroom* maximum, and the *kitchen* maximum for two different inputs. In *A* we have clamped the *bed* unit to be on. In *B* we have clamped the *oven* unit on. *C* shows the landscape from the start state when no units are clamped on.

Properties of Schemata and Maxima in Constraint Satisfaction Networks

Rumelhart and Ortony (1977; Rumelhart, 1980) have outlined a set of properties which characterize schemata. In this section we consider each of these properties and show how they map onto features of the PDP networks we are now outlining.

Schemata have variables. Essentially, the variables of a schema correspond to those parts of the pattern that are not completely determined by the remainder of the structure of the pattern itself. It is these parts that vary from one situation to another in which, nevertheless, the bulk of the units corresponding to the schema is active. On this account, the binding of a variable amounts to filling in a variable subpattern. Default values represent variable subpatterns that tend to get filled-in in the absence of any specific input. Since patterns tend to complete themselves, default values tend to be automatically filled in by the process of settling into an interpretation. In some cases, there are sets of units that are mutually inhibitory so that only one can be active at a time but any of which could be combined with most other units. Such a set of units can be considered to constitute a *slot* which is

filled in in the processing of the input. Perhaps the best example from our current data base is what might be called the *size slot.* In this case, the *very-large*, *large*, *medium*, *small*,and *very-small* units are all mutually inhibitory. (See the weight matrix in Figure 6). The different maxima have different default values for these slots. The *bathroom* has a default value of *very-small*, the *kitchen* has a default value of *small*, the *bedroom* has a default value of *medium*, the *office* is *large*, and the default *living-room* is *very-large*. Interestingly, when the input contains information that descriptors other than the default descriptors apply, the default size changes as well. For example, Figure 12 shows a case in which *bed* and *sofa* were both clamped. What we get in such a case is a room which might best be described as a large, fancy bedroom. The size variable is filled in to be *large*, it also includes an *easy-chair*, a

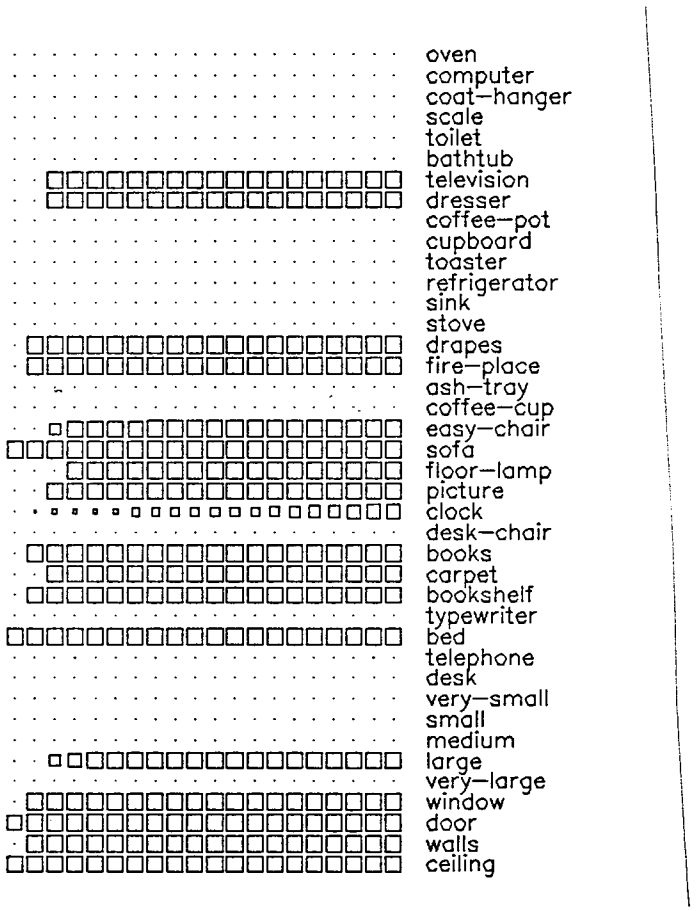


FIGURE 12. The output of the room network with *bed*, *sofa*, and *ceiling* initially clamped. The result may be described as a large, fancy bedroom.

floor-lamp, and a *fireplace*. Similarly, the setting of the size variable modifies the default values for the other descriptors. In this case, if we set the size variable to *large* we get a bedroom with a *fireplace*.

Schemata can embed. In structural interpretations of schemata, it is useful to think of a schema as a kind of tree structure in which subschemata correspond to subtrees that can fill variable slots. Under our interpretation, subschemata correspond to small configurations of units which cohere and which may be a part of many different stable patterns (and therefore constitute a schema on their own right). Each stable subset of cohering units can be considered a schema. Large schemata will often consist of patterns of coherence among these coherent subsets. There are several instances of this in our example. For example, the *easy-chair* and *floor-lamp* constitute a subschema, the *desk* and *desk-chair*, the *window* and *drapes*, and other similar combinations constitute small schemata that can be either present or absent in several different configurations. Consider, for example, the case of *window* and *drapes*. These two elements almost always appear together and either both appear or neither appear. We will refer to this pattern as the *window* schema. Figure 13 shows the effect of adding *drapes* and/or *window* to the *office* schema. The default value for this schema involves no windows. The highest peak, at the origin of the graph, corresponds to the *office* maximum. One axis corresponds to the

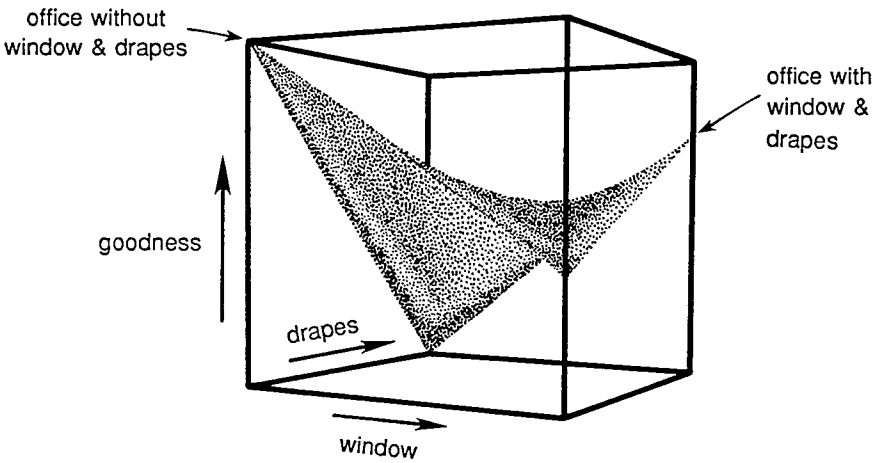


FIGURE 13. The goodness landscape for *office* as a function of the activation value of the *drapes* unit and the *window* unit. The function shows that it is maximum when either both are 0 or both are 1. This pattern of interaction is consistent with the view that the combination *window-drapes* form a subschema.

amount of *drapes* added to the schema (i.e., to the activation value for the *drapes* unit). The second axis corresponds to the amount of *window* added. The third axis, of course, corresponds to the goodness-of-fit for each of the states. It should be noted that the low points on the graph correspond to those cases in which one of the two units of the window subschema is on and the other is off. The high points on the graph, corresponding to goodness maxima, occur when either neither (at the origin) or both of the units are on. The case where neither is on corresponds to a slightly higher peak than when both are on. Thus, the default *office* probably doesn't have a window, but if the input indicates that either one of the units (*window* or *drapes*) is on, turning the other one on is best. To conclude, large schemata such as the *office* schema can be conceptualized as consisting, in part, of a configuration of subschemata which may or may not be present as wholes. Having parts of these subschemata is worse than having either the entire subschema or none of it.

Schemata represent knowledge at all levels. They should represent encyclopedic knowledge rather than definitional information. This amounts to the claim that all coherent subpatterns should be considered schemata as well as the whole stable pattern. It also suggests that knowledge of all sorts should be represented in the interconnections among the constituent units.

Schemata are active processes. This is obviously true of the the PDP system we are describing. They are a kind of organic element which grows and fulfills itself within its environment.

Schemata are recognition devices whose processing is aimed at the evaluation of their goodness-of-fit to the data being processed. This feature is obviously also a part of the idea outlined here. The goodness-of-fit is roughly determined by the height of the peak in goodness space. The processing of the system is aimed toward climbing uphill along the goodness-of-fit gradient. The stable points correspond to local maxima in this space. The height of the peak corresponds to the goodness-of-fit.

Some additional features of our interpretation. There are three major difficulties with the conventional representation of schemata that are naturally overcome in the PDP approach. In the conventional approaches, decisions must be made about which aspects of a given

schema are constant and which are variable. The PDP solution is essentially that all aspects are variable; some aspects are simply more tightly constrained than others. Secondly, in a conventional representation, one has to decide exactly which aspects of the situation are part of the schema and which are not. In our PDP approach, units may cohere more or less strongly to their mates and in this sense be more or less a part of the schema. Finally, on the conventional view a decision must be made about whether a certain set of relationships should be put together to form a schema at all. Again, in the PDP formulation no such decision needs to be made. One can have schemata of varying degrees of existence. The rigidity of the schema is determined by the tightness of bonding among the units that constitute the schema. The tighter the bond, the more strongly the constituent elements activate one another, and the more rigid the structure. The weaker the bonds, the more fluid the structure, and the more easily a system can flow among states. This degree of fluidity depends on the shape of the goodness-of-fit landscape. Tightly rigid schemata have sharp peaks in goodness space; fluid schemata with many variable parts correspond to rounded hilltops. The goodness landscape, in turn, depends on the knowledge base that lies beneath it. If the knowledge is tightly interconstrained so that one part strongly predicts other parts, then we have a rigid schema. We can't easily get just part of it active. If part of it becomes active, the part will pull in the whole and suppress the activity of aspects that are not part of it. On the other hand, if the knowledge is only loosely interrelated, the schema will be a relatively weak organizer of the information and will be a pressure for structuring the input, but it will flow easily from one pattern to another. Moreover, within a schema itself, some aspects will be tightly bound together while other aspects will be only more loosely tied to the body of the schema. Input situations that demand an interpretation that breaks up the tightly bound clusters are going to be more difficult for the system to attain than those that require breaking up much more loosely interconnected elements.

Finally, we point out one way in which these ideas about schemata might be elaborated to overcome one apparent deficiency of the network we have thus far been considering. The network uses a fixed set of units to represent each type of object that might be present in a particular instantiation of a schema. This is clearly an oversimplification since it is often necessary to be able to think about two different instantiations of the same subschema within a larger overall schema—for example, there is often more than one chair in a living room. To capture such situations, it is necessary to imagine that the network may contain several subsets of units, each capable of representing a different possible chair. The subsets would correspond to different roles the

different chairs might play in the overall room schema. This would also allow the representation to capture the assignment of a particular object to a particular role.

PARALLEL DISTRIBUTED PROCESSING MODELS AND THINKING

In the previous section we offered an interpretation of the schema in terms of the emergent properties of simple PDP networks. From there, we believe that we can make contact with much of the cognitive science literature. There are central issues, however, which remain difficult to describe within the PDP framework. We have particularly in mind here, the process of thinking, the contents of consciousness, the role of serial processes, the nature of mental models, the reason for mental simulations, and the important synergistic role of language in thinking and in shaping our thought. These issues are especially important because these are issues that PDP approaches do not, on first blush, seem to have much to say about. In this section we attack some of those problem areas. We don't claim to have solutions to them, rather we can outline a story that represents our current understanding of these processes. The story is overly simplistic but it does give an idea of where we are in our thinking on these critical issues.

It should be noted here that the account of mental processing we have been developing offers an interesting perspective on the relations between parallel and serial processing. The "parallel" in "parallel distributed processing" is intended to indicate that, as a basic architectural design principle, processing is carried out, in so far as possible, in parallel. Parallel algorithms are employed rather than serial ones. At the same time, however, the "distributed" in "parallel distributed processing" brings a serial component to PDP systems. Since it is patterns of activations over a set of units that are the relevant representational format and since a set of units can only contain one pattern at a time, there is an enforced seriality in what can be represented. A given set of units can, however, be seen as representing a sequence of events. Since we assume that the system is moving toward a maximum goodness solution every time a new input comes into the system, the system operates in the following way. An input enters the system, the system relaxes to accommodate the new input. The system approaches a relatively stable state which represents the interpretation of the input by the system. The system then occupies this state until the stimulus conditions change. When a new input arrives, the system relaxes to a new state. Looking at the system over a short time frame, it is dominated

by the relaxation process in which all units work cooperatively to "discover" an interpretation of a new input. Looking over a somewhat longer time frame, we see the system as sequentially occupying a series of relatively stable states—one for each change in input. Roughly, if we imagine that the relaxation process takes on the order of a few tenths of a second and that the time spent in essentially the same stable state is on the order of a half of a second or so, we could see events requiring less than about a half second to be essentially a parallel process, and those requiring several seconds to involve a series of such processes and therefore to have a serial component.

The Contents of Consciousness

It isn't necessary for the arguments that follow, but for the sake of concreteness, we suppose that there is a relatively large subset of the total units in the system whose states of activity determine the contents of consciousness. We imagine that the time average of the activities of these units over time periods on the order of a few hundred milliseconds correspond to the contents of consciousness. Since we imagine that our systems must be such that they reach equilibrium in about this amount of time, the contents of consciousness are dominated by the relatively stable states of the system. Thus, since consciousness is on the time scale of sequences of stable states, consciousness consists of a sequence of interpretations—each represented by a stable state of the system. Typically, consciousness contains a single interpretation (i.e., a single pattern representing its inputs) and consists of a sequence of such interpretations. On occasions in which the relaxation process is especially slow, consciousness will be the time average over a dynamically changing set of patterns and thus would be expected to lead to "fuzzy" or unclear phenomenal impressions.

The Problem of Control

One common critique of the kind of model we have sketched so far is that it can't really change without external prodding. Suppose that we are in a fixed stimulus environment. In this case, the system will relax into an interpretation for that environment and stay there. Our conscious experience will be of a fixed interpretation of a fixed stimulus. Until the world changes there is no change to the system nor to the contents of our consciousness. Obviously this is an incorrect

conclusion. How can such a system do something? Perhaps the first thing that comes to mind is that the environment never really is fixed. It is always changing and therefore the contents of our consciousness must always be changing to interpret the current state of affairs. A good example of this might be the movies. We sit in the movies and watch. Our system reaches a sequence of interpretations of the events on the screen. But, since the movie is always continuing, we are driven to continue to interpret it. Surely, what is true of a movie is also true of life—to some extent. This may be part of the story, but it would appear to be rather more passive than we might want. We don't just sit passively by and let the world change and then passively monitor it. Rather we act on the world.

A second answer to the problem of a system fixated on a particular interpretation becomes apparent in realizing that our interpretation of an event often dictates an action which, in turn, changes the environment. The environmental change can then feed back into the system and lead to another interpretation and another action. Figure 14 illustrates how this feedback loop can continuously drive the system from state to state. A paradigm case for this is playing a game. We can imagine that we are playing a game with someone; our input consists of a board configuration, and we settle into a state which includes a specification of a response. It would be quite easy to build a relaxation network that would take as input a description of a current board situation and produce, as part of the state to which it relaxed, a specification of the response. It would simply require that, for each game situation, the system relaxes to a particular state. Certain units of the state represent the action (or class of actions) that should be taken. Upon taking these actions, the opponent makes a play which in turn leads to a new set of constraints to which the system relaxes. In this way, the system can make a sequence of moves. Indeed, as we describe below, we have built such a network that can play tic-tac-toe. Other more complex games, such as checkers or chess require rather more effort, of course, but can, in principle, be dealt with in the same way. Although this is a much more activist view, it is still a "data-driven" view. The system is entirely reactive—given I am in this state, what should I do?

Mental Models

Suppose, for arguments sake, that the system is broken into two pieces—two sets of units. One piece is the one that we have been discussing, in that it receives inputs and relaxes to an appropriate state that includes a specification of an appropriate action which will, in turn,

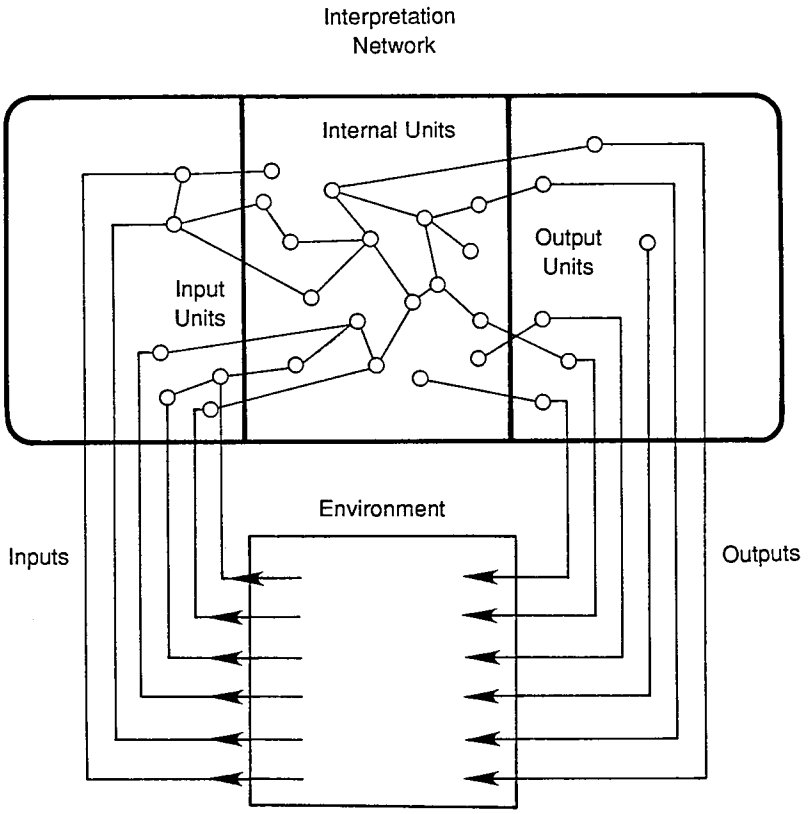


FIGURE 14. The inputs to the PDP system should be considered as partially due to the effects that the output of the system has had on the environment. In this way, the output of the system can drive it from state to state.

change the inputs to the system. The other piece of the system is similar in nature, except it is a "model" of the world on which we are acting. This consists of a relaxation network which takes as input some specification of the actions we intend to carry out and produces an interpretation of "what would happen if we did that." Part of this specification would be expected to be a specification of what the new stimulus conditions would be like. Thus, one network takes inputs from the world and produces actions; the other takes actions and predicts how the input would change in response. This second piece of the system could be considered a mental model of the world events. This second portion, the mental model of the world, would be expected to be operating in any case, in as much as it is generating expectations

about the state of the world and thereby "predicting" the outcomes of actions.

Now, suppose that the world events did not happen. It would be possible to take the output of the mental model and replace the stimulus inputs from the world with inputs from our model of the world. In this case, we could expect that we could "run a mental simulation" and imagine the events that would take place in the world when we performed a particular action. This mental model would allow us to perform actions entirely internally and to judge the consequences of our actions, interpret them, and draw conclusions based on them. In other words, we can, it would seem, build an internal control system based on the interaction between these two modules of the system. Indeed, as we shall show, we have built a simple two-module model of tic-tac-toe which carries out exactly the process and can thereby "imagine" playing tic-tac-toe. Figure 15 shows the relationships between the interpretation networks, the inputs, the outputs, and the network representing a model of the world and the process of "mental simulations."

Mental Simulations and Mental Practice

One nice feature of this model is that it ties into the idea of mental simulations and learning through mental practice. Performance in the task involves two parts—a system that determines what to do in any given situation and a system that predicts what will happen if any given action is carried out. If we have a reasonably good model of the "world" we could learn from our model the various consequences of our actions—just as if we were carrying them out in the real world. It may very well be that such a feature accounts for the improvement that occurs in mentally practicing complex motor tasks.

Conversations: Actual and Imagined

Imagine a situation in which we had a relaxation network which would take as input a sentence and produce an interpretation of that sentence as well as the specifications for a response to that input. It is possible to imagine how two individuals each with such a network could carry out a conversation. Perhaps, under appropriate circumstances they could even carry out a logical argument. Now, suppose that we don't actually have another participant, but instead have a mental model of the other individual. In that case, we could imagine carrying

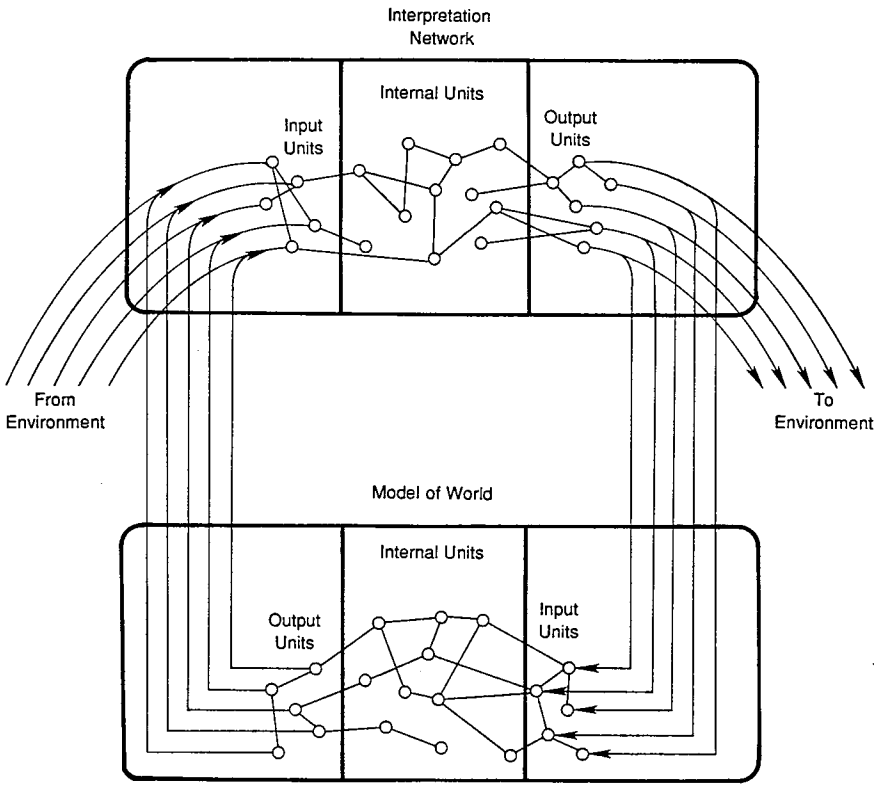


FIGURE 15. The relationships among the model of the world, the interpretation network, the inputs, and the outputs for the purpose of mental simulations.

out a conversation with someone else. We could hold an imaginary argument with someone else and perhaps even be convinced by it! Indeed, this brings up the last move that we wish to suggest. Suppose that we don't have a model of another person at all. Suppose that instead we simply use our one single model to both produce and react to imagined linguistic inputs. This description is, it would seem, consistent with Vygotsky's (1934/1962) view of thinking and is consistent with the introspections about a certain kind of thinking and "internal speech." Note, this does *not* suggest that thinking is simply internal speech. More generally, thinking, as we have argued, involves a sequence of states of consciousness. There are a number of ways of controlling that sequence. One way involves running "mental simulations." Another way involves recycling linguistic inputs. Note, this gives language an interesting, almost Whorfian, role however. Suppose that the interpretation that led to the production of the internal speech was much richer than the linguistic forms could possibly suggest. Thus,

the linguistic forms pick out aspects of the entire interpretation to emphasize. Once this emphasis has taken place and the new input has been processed, the next state will be strongly affected by the new input and our new interpretation will be shaped, to some extent, by the words we chose to express our first idea. Thus, our thinking about a topic will be, sometimes strongly, affected by the language tools we have for expressing our ideas.

External Representations and Formal Reasoning

If the human information-processing system carries out its computations by "settling" into a solution rather than applying logical operations, why are humans so intelligent? How can we do science, mathematics, logic, etc.? How can we do logic if our basic operations are not logical at all? We suspect the answer comes from our ability to create artifacts—that is, our ability to create physical representations that we can manipulate in simple ways to get answers to very difficult and abstract problems.

The basic idea is that we succeed in solving logical problems not so much through the use of logic, but by making the problems we wish to solve conform to problems we are good at solving. People seem to have three essential abilities which together allow them to come to logical conclusions without being logical. It is these three abilities that have allowed us to accomplish those uniquely human achievements of formal reasoning. These abilities are:

- We are especially good at pattern matching. We seem to be able to quickly "settle" on an interpretation of an input pattern. This is an ability that is central to perceiving, remembering, and comprehending. Our ability to pattern match is probably not something which sets humans apart from other animals, but is probably *the* essential component to most cognitive behavior.

- We are good at modeling our world. That is, we are good at anticipating the new state of affairs resulting from our actions or from an event we might observe. This ability to build up expectations by "internalizing" our experiences is probably crucial to the survival of all organisms in which learning plays a key role.

- We are good at manipulating our environment. This is another version of man-the-tool-user, and we believe that this is perhaps the crucial skill which allows us to think logically, do mathematics and science, and in general build a culture.

> Especially important here is our ability to manipulate the environment so that it comes to represent something. This is what sets human intellectual accomplishments apart from other animals.

Roughly speaking, the view is this: We are good at "perceiving" answers to problems. Unfortunately, this is not a universal mechanism for solving problems and thinking, but as we become more expert, we become better at reducing problem domains to pattern-matching tasks (of the kind best accomplished by PDP models).[9] Thus, chess experts can look at a chess board and "see" the correct move. This, we assume, is a problem strictly analogous to the problem of perceiving anything. It is not an easy problem, but it is one that humans are especially good at. It has proven to be extraordinarily difficult to duplicate this ability with a conventional symbol-processing machine. However, not all problems can be solved by immediately "seeing" the answer. Thus, few (if any) of us can look at a three-digit multiplication problem (such as 343 times 822) and see the answer. Solving such problems cannot be done by our pattern-matching apparatus, parallel processing alone will not do the trick; we need a kind of serial processing mechanism to solve such a problem. Here is where our ability to manipulate our environment becomes critical. We can, quite readily, learn to write down the two numbers in a certain format when given such a problem.

```
343
822
---
```

Moreover, we can learn to see the first step of such a multiplication problem. (Namely, we can see that we should enter a 6 below the 3 and 2.)

```
343
822
---
  6
```

We can then use our ability to pattern match again to see what to do next. Each cycle of this operation involves first creating a representation through manipulation of the environment, then a processing of this (actual physical) representation by means of our well-tuned perceptual apparatus leading to a further modification of this representation.

---

[9] As we have argued before, it is because experts have such a powerful pattern-matching capability that expert systems that rely only on pattern matching (albeit not nearly as powerful as the human system) are as successful as they are.

By doing this we reduce a very abstract conceptual problem to a series of operations that are very concrete and at which we can become very good. Now this applies not only to solving multiplication problems. It applies as well to solving problems in logic (e.g., syllogisms), problems in science, engineering, etc. These dual skills of manipulating the environment and processing the environment we have created allow us to reduce very complex problems to a series of very simple ones. This ability allows us to deal with problems that are otherwise impossible. This is *real* symbol processing and, we are beginning to think, the primary symbol processing that we are able to do. Indeed, on this view, the external environment becomes a key extension to our mind.

There is one more piece to the story. This is the tricky part and, we think, the part that fools us. Not only can we manipulate the physical environment and then process it, we can also learn to internalize the representations we create, "imagine" them, and then process these imagined representations—just as if they were external. As we said before, we believe that we are good at building models of our environment so that we can anticipate what the world would be like after some action or event takes place. As we gain experience with the world created by our (and others') actions we develop internal models of these external representations. We can thus imagine writing down a multiplication problem and imagine multiplying them together. If the problem is simple enough, we can actually solve the problem in our imagination and similarly for syllogisms. Consider, for example, a simple syllogism: All A are B and no C are B. We could solve this by drawing a circle for A, a larger circle including all of the A's around the first circle to represent the B's, and a third disjoint circle standing for the C's. We could then "see" that no A's are C. Alternatively, we need not actually draw the circles, we can merely imagine them. We believe that this ability to do the problem in our imagination is derivative from our ability to do it physically, just as our ability to do mental multiplication is derivative from our ability to do multiplication with pencil and paper. The argument that external representations play a crucial role in thought (or, say, in solving syllogisms) is sometimes challenged on the ground that we don't really *have* to draw Venn diagrams (or whatever) to solve them since we *can* solve them in our head. We argue that the major way we can do that is to imagine doing it externally. Since this imagination is, we argue, dependent on our experience with such representations externally, the argument that we *can* solve them mentally loses its force against the view that external symbols are important for thought processes. Indeed, we think that the idea that we reason with mental models is a powerful one precisely because it is about this process of imagining an external representation and operating on that.

It is interesting that it is apparently difficult to invent new external representations for problems we might wish to solve. The invention of a new representation would seem to involve some basic insight into the nature of the problem to be solved. It may be that the process of inventing such representations is the highest human intellectual ability. Perhaps simply creating an external representation sufficient to support problem solving of a particular kind is evidence of a kind of abstract thinking outside of the simple-minded view sketched here. That may be, but it seems to us that such representational systems are not very easy to develop. Usually they are provided by our culture. Usually they have evolved out of other simpler such systems and over long periods of time. Newer ones, when they are developed, usually involve taking an older system and modifying it to suit new needs. One of the critical aspects of our school system would seem to be teaching such representational schemes. The insights into the nature of the problem become embedded in the representations we learn to use to solve the problems.

Language plays an especially tricky and interesting role in all of this. Perhaps the internal/external issue is not too important with language. The notion we have here is one of "self-instruction." This follows Vygotsky's (1934/1962) view, we believe. We can be instructed to behave in a particular way. Responding to instructions in this way can be viewed simply as responding to some environmental event. We can also remember such an instruction and "tell ourselves" what to do. We have, in this way, internalized the instruction. We believe that the process of following instructions is essentially the same whether we have told ourselves or have been told what to do. Thus, even here, we have a kind of internalization of an external representational format (i.e., language). We don't want to make too much of this point since we recognize that the distinction between external and internal when we ourselves produce the external representation is subtle at best, but we don't really think it differs too much from the case in which we write something down and therefore create a real, physical, viewable representation. Saying something out loud creates a hearable representation. There are interesting cases in which people talk to themselves (for example, solving difficult problems in noisy environments leads people to literally talk to themselves and instruct themselves on the problems they are solving).

Before leaving this topic, one more important aspect of external representations (as opposed to internal representations) should be noted. External representations allow us to employ our considerable perceptual/motor abilities in solving abstract problems. This allows us to break problems into a sequence of relatively simple problems. Importantly, once an external representation is created, it can be

reinterpreted without regard to its initial interpretation. This freedom allows us to discover solutions to problems without "seeing" our way to the end. We can inspect intermediate steps and find alternative solutions which might be better in some ways. In this way, we can discover new features of our representations and slowly extend them and make them more powerful.

Goal Direction in Thinking

Our discussion thus far has left one central issue undiscussed, namely, the role of goals in thought and problem solving. Clearly it is not the case that the same perceptual stimulus always drives the system to react in a consistent way. Rather, our goals or intentions interact with the stimuli (internal and external) that provide the inputs to the thinking process. Further, goals organize whole sequences of thoughts into a coherent problem-solving activity, and the notion that there is a hierarchy of goals is certainly important for understanding these coherent sequences.

While we have not stressed the importance of goals, it is not difficult to see how they could be incorporated into our framework. Goals can be explicitly represented as patterns of activation and can thus provide one source of input to the thinking process. Nor is it difficult to imagine how a PDP network could learn to establish specific subgoal patterns in response to particular superordinate goals and inputs.

Summary

These ideas are highly speculative and detached from any particular PDP model. They are useful, we believe, because they suggest how PDP models can be made to come into contact with the class of phenomena for which they are, on the face of it, least well suited—that is, essentially sequential and conscious phenomena. Even in these cases, however, they lead us to view phenomena in new ways.

An Example

In these last few sections, we have been talking at a very general level. We often find it useful to be concrete about our ideas.

Therefore, to illustrate the notion of thought as mental simulation more concretely, we created two relaxation networks that can be connected together to mentally simulate playing a game of tic-tac-toe. The two networks are very similar. The first is a system which, given a pattern representing the board of a tic-tac-toe game, will relax to a solution state that fills in an appropriate response. The second module is nearly identical to the first; it takes as input a board position and a move and settles to a prediction of the opponent's responding move. In short, it is a "mental model" of the opponent. When the output of the first is fed, as input, to the second and the output to the second is fed, as input, to the first, the two networks can simulate a game of tic-tac-toe.

Figure 16 illustrates the basic structure of the tic-tac-toe playing network. The network consists of a total of 67 units. There are nine units representing the nine possible responses. These are indicated by the nine dots at the top of the figure. There is one unit for each of the nine possible moves in tic-tac-toe. Since only one response is to be

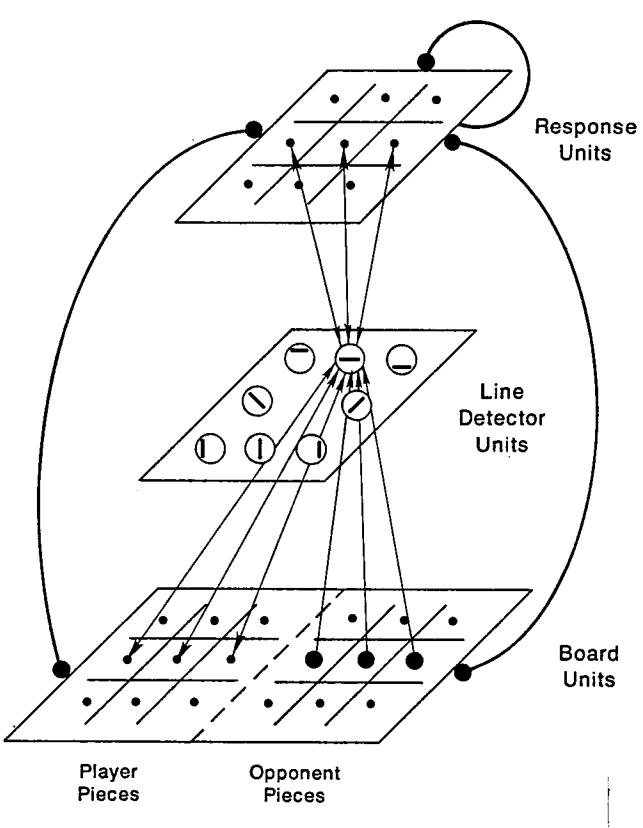


FIGURE 16. The basic structure of the tic-tac-toe playing network.

made at a time, these units are mutually inhibitory. This is indicated by the heavy black line feeding back from the top plane in the figure to itself. There are a total of 18 units representing the board configuration. These are divided into two groups of nine: one group for the positions of the friendly or player pieces on the board and one group representing the positions of the opponent's pieces. Since if any square is occupied, it is not a possible move, each board unit strongly inhibits its corresponding output unit. This strong inhibition is indicated by the heavy black lines connecting the board units to the response units. In addition to these 9 output units and 18 input units, there is a total of 40 hidden units which detect patterns in the board units and activate the various response alternatives. These 40 units can be divided into eight classes corresponding to the eight possible ways of getting three ×'s or ○'s in a row. In the figure, one of each such category of units is illustrated. The receptive field of each unit is indicated by the line inside the circle representing that unit. Thus, there is one unit for each of the three horizontal lines, one for each of the three vertical lines, and one for each of the two possible diagonal lines. For each of the eight classes of hidden units, there are five different pattern types that different units are responsive to. For example, some units are responsive to empty regions. That is, they respond just in case none of the board units from which it receives inputs is turned on. This is implemented by making them be inhibited by any activity in their receptive field and by giving them a negative threshold. We call this an *empty line detector.* All things being equal, it is better to move into an empty row, column, or diagonal; therefore these units weakly activate their respective output units. At the start of the game these are the only units which are active and therefore the sole criterion for the first move is the number of possible strings of three the square is a member of. Since the center square intersects the largest number, it will usually be chosen by the system for its first move. On later moves, there are other units feeding into the decision, but these units also contribute. Another unit type will respond whenever two or more units of the same kind occur in its regions. This is the kind of unit illustrated in the figure. It receives strong inhibitory input from one set of board units (in this case the opponent's pieces) and excitatory inputs from the other set. It has a rather high positive threshold so that it will not come on until at least two units are on in its row. We call this a *friendly doublet detector.* Whenever this unit comes on it means that the system can make a winning move by playing in that row. Therefore, it is strongly positively connected to its respective output units. If such a move is possible, the system will make it. There are similar units which respond to two or more units from the representation of the opponent's pieces are active in its receptive field. We call this an *opponent doublet detector.* If such a

unit comes on, it means that the opponent could win by a move in this region, so it excites its response units very strongly as well. Unless there is a winning move somewhere else, the blocking response will become the most active. Finally, there are units which respond to one or more friendly pieces or one or more opponent pieces in an otherwise open line. We call these *friendly singleton* and *opponent singleton detectors.* It is generally good to extend your own singleton or block an opponent's singleton if there is nothing more pressing, so these units also activate their respective output units, only rather more weakly than the units detecting doublets. Thus, the net input arriving at any given unit is the weighted sum of all of these urgencies detected by the hidden units. Because of the direct inhibition from the board units, only those response units corresponding to open squares receive positive input. The mutual inhibition insures that the strongest unit will usually win. In order that this system truly climb in overall goodness-of-fit, all weights are symmetric and the update is done asynchronously at random. This means that when there isn't much difference between the possible response alternatives, a weaker one will sometimes get the upper hand and win the competition. This never happens, however, when a unit is receiving very strong input, as with the case of an open double for a win or to be blocked.

The simplest case is when the system simply is given a board position, settles on a move, is given the next board position incorporating the opponent's response, settles on a move for that position, etc., until a game is finished. This involves only one network and presumably would be the basic modality for the system to run in. Figure 17 shows a sample run of such a situation. The activation levels of each of the 67 units is shown in each column. Successive activation states are shown from left to right. At the start, it should be noted that there is a friendly piece in the center square and two enemy pieces, one in each corner of the upper row. It is the system's move. The system starts with the units corresponding to the board position clamped on and all other units off. The figure shows the strengths for each of the units after every 50 asynchronous updates. By the second time slice, the system is beginning to extract the relevant features of the board position. The five groups of eight units shown in each column following the response units and board position display the line-detector units. Each group of eight is laid out with three units across the top corresponding to the top, middle, and bottom lines from left to right. The second row of the eight corresponds to the left and right diagonals, and the bottom row of each eight corresponds to the left, middle, and right columns. Thus, we see that the system has, by the second time slice, begun to discover that the bottom row is empty, that it has a singleton in the middle row, that the opponent has a singleton in the left column, and

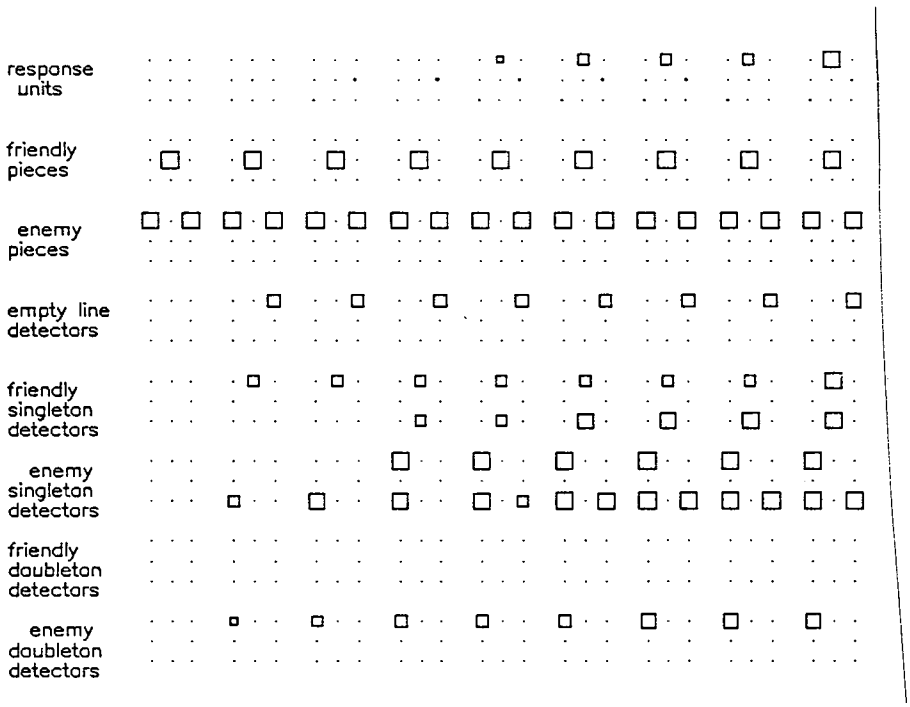


FIGURE 17. A sample run of the tic-tac-toe playing network.

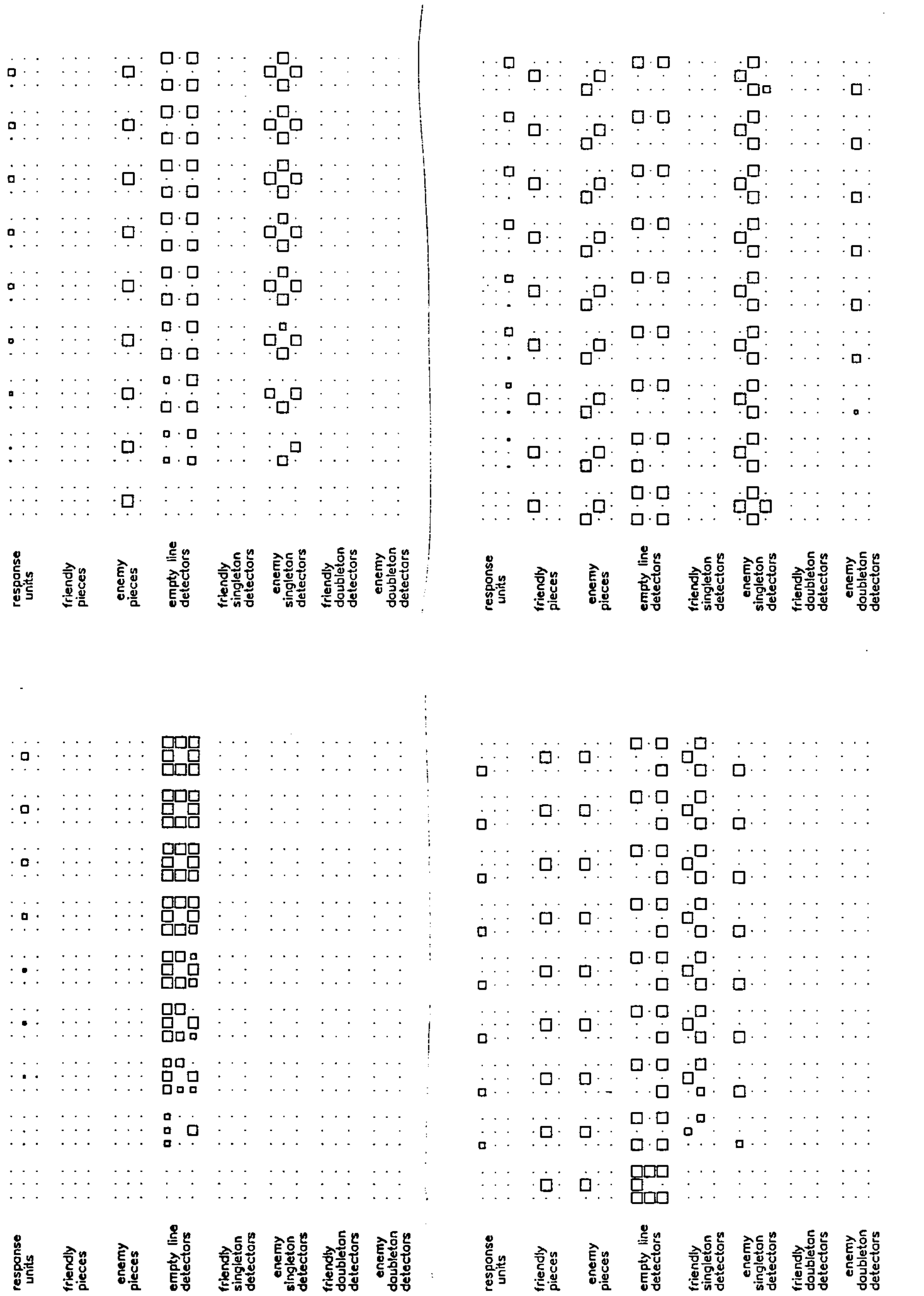
that the opponent has a doubleton in the top row. Then the system discovers a friendly singleton in the middle column, an enemy singleton in the top row, and an enemy singleton in the right column. Based on this, the blocking response in the top-middle position begins to gain strength. As the observation that the opponent has an open double increases in strength, the activation level of the blocking response gains strength even faster. Finally, the strength of a response unit exceeds criterion and the system detects that it has settled on a move. At this point, we assume a motor system (or the like) would be invoked and the move actually made. Upon having made the move, the world has changed and (we can imagine) the opponent makes a response. In this case, a new board position could be presented to the system and the process could begin again.

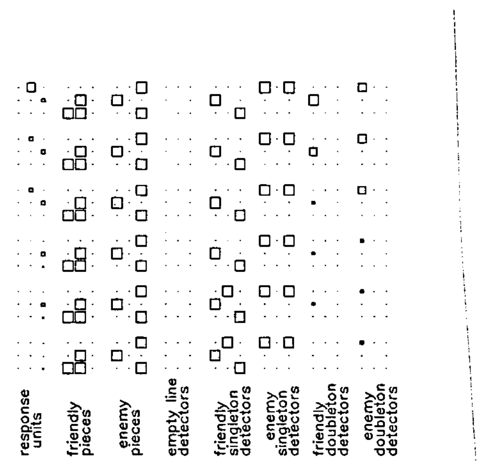
Here we have a system that can take a board position and decide on a move. When we make our move and present the system with the new board position, it can make its next move, etc., until the game is finished. Here we see a kind of sequential behavior of our relaxation system driven by changes in the outside which are, in turn, the result of the activity of the system itself. Nothing is different; we see that sequential processing results simply from the addition of the systems

own responding to the environmental process of generating a sequence of stimuli. This is a very common form of control of sequential behavior and involves no special mechanism. The case of "mental simulations" involves a bit more machinery. We need a "model of the world" to predict the environment's response to any action that might be taken. In the case of tic-tac-toe, this involves a network which models the opponent. In general, the model of the opponent may be arbitrarily different from that used by the system to make its own response. In our case, however, it is sufficient to build an identical network for the opponent. The only difference is that in the opponent's network, the board is interpreted differently: The opponent's board position of the original network setup drives the friendly board position in the model of the opponent, and the friendly board position in the original network drives the opponent board position in the model of the opponent. Figure 18 shows a run of the system "mentally simulating" the play of a game. The state of the units are shown after every 100 updates. First, the state of the original network is shown as it settles on its move, then the state of the model of the opponent is shown as it settles on its move. This continues until the game reaches a conclusion. In this instance, the "simulated player" makes a "mistake" in its response to the system's opening. Successive moves show the system taking advantage of its own mistake and winning the game.

CONCLUSION

We have argued in this book that the analysis of psychological phenomena in terms of units, activations of units, connections among units, and the action of large coalitions of such units leads us to many insights that have not been possible in terms of the language that has recently been more popular in cognitive science. In that sense, we may be perceived as throwing out the insights gained from the more conventional language and concepts. We are not throwing out such insights. In this chapter, we attempt to show the relationship between two such insights and our models. At start, we argue that the concept of the schema has a correspondence in the PDP framework. In particular, we argue that a schema is best viewed as a coalition of units which cohere and that configurations of such coalitions determine the interpretations that the system can attain. These stable states correspond to instantiated configurations of schemata that can be characterized in terms of goodness-of-fit maxima that the system can move into. Such processing systems, we argue, have all of the features of schemata and more. Among the advantages of this view is that the


response units
friendly pieces
enemy pieces
empty line detectors
friendly singleton detectors
enemy singleton detectors
friendly doubleton detectors
enemy doubleton detectors



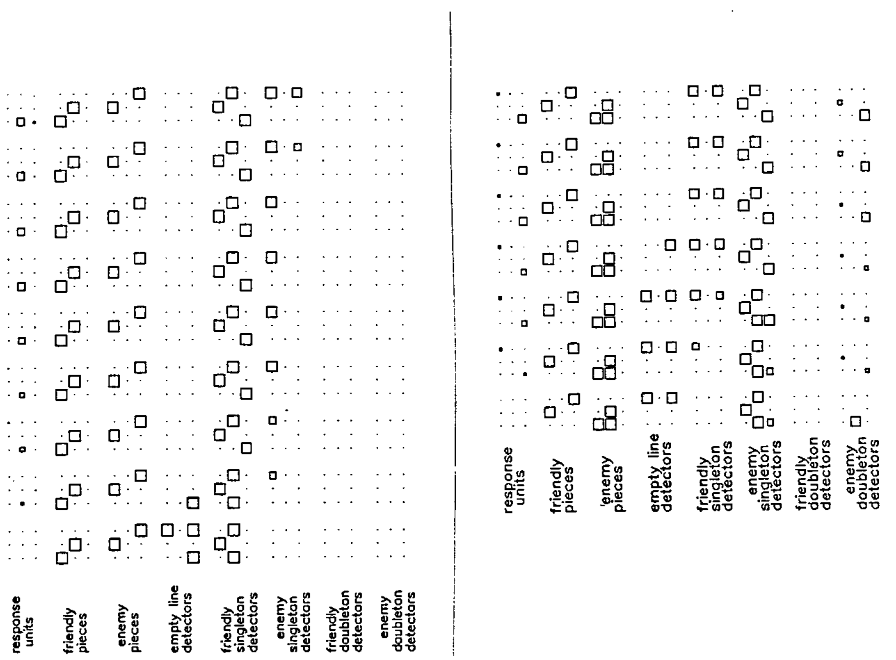


FIGURE 18. The tic-tac-toe system mentally simulating the play of a game.

schema becomes much more fluid and flexible and able to accommodate itself to inputs. In more conventional representations there is a strong distinction between variables and slots and the bulk of the schema. Under our interpretation an aspect of the schema is more or less a variable. Even central aspects of the schema can be missing and the system can still find a reasonably low energy stable state. If very rigid schemata are implicit in the knowledge base, this will show up as narrow peaks in the goodness landscape. If more fluid and variable schemata are required, the landscape will contain broad plateaus which allow for a good deal of movement in the region of the maximum.

We see the relationship between our models and schema theory as discussed by other researchers as largely a matter of levels of analysis. This is roughly analogous to the relationship between the level of discussion of fluid dynamics and an underlying level of description involving statistical mechanics. It is often useful to theorize at the level of turbulence and different kinds of turbulence, and such a description will do for many purposes. However, we can often run up against phenomena in which our high-level descriptions will not do, we must describe the system in terms of the underlying processes in order to understand its behavior. Another feature of this example is our understanding of the phenomena of emergent properties. Turbulence is not predicted by the knowledge of the elements of the system; it is inherent in the interactions among these elements. Similarly, we do not believe that single-unit activity is the appropriate level of analysis. Properties of networks "emerge" from the interactions of the elements. Indeed, such properties as goodness maxima, etc., are emergent in just this way. In general, we see cognitive phenomena as emergent from the interactions of many units. Thus, we take the symbolic level of analysis to provide us with an approximation to the underlying system. In many cases these approximations will prove useful; in some cases they will be wrong and we will be forced to view the system from the level of units to understand them in detail.

We also discussed the relationship between PDP models and the more conventional sequential processing systems. We believe that processes that happen very quickly—say less than .25 to .5 seconds—occur essentially in parallel and should be described in terms of parallel models. Processes that take longer, we believe, have a serial component and can more readily be described in terms of sequential information-processing models. For these processes, a process description such as a production would, we imagine, provide a useful approximate description. We would caution, however, that when one chooses a formalism such as production systems and attempts to use it not only to describe the conscious sequential processes that occur at this slow time scale, it is important not to fall into the trap of assuming that the

microstructure of these sequential processes should also be described in the same terms. Production systems have the power of Turing machines and people often attempt to describe phenomena at this faster time scale in terms of the same sequential formalism that seems appropriate for the slower time scale. We believe that it will turn out that this approach is wrong, that the power of the formalism has led us astray. In these cases we suspect that the unit level of analysis will be required.

Finally, we showed how the important notion of mental models and the related notion of mental simulations play important roles in the sequential behavior of a PDP system. We illustrated this point with a system which could use a "model of its opponent" to "mentally simulate" a tic-tac-toe game. We suspect that this process will turn out to be important when we begin to apply our models to temporally extended reasoning and problem-solving tasks.

ACKNOWLEDGMENTS

This research was supported by Contracts N00014-79-C-0323, NR 667-437 and N00014-85-K-0450, NR 667-548 with the Personnel and Training Research Programs of the Office of Naval Research, by grants from the System Development Foundation, and by a NIMH Career Development Award (MH00385) to the third author.