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.

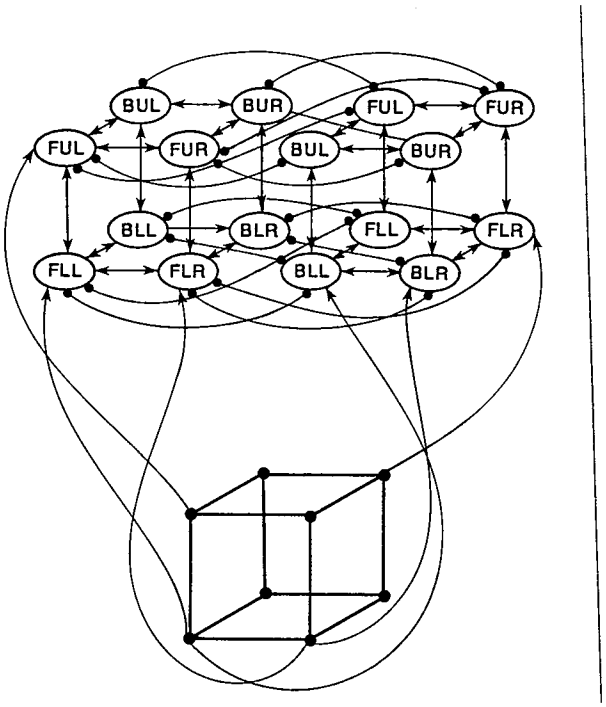

FIGURE 1. A simple network representing some of the constraints involved in perceiving the Necker cube.

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

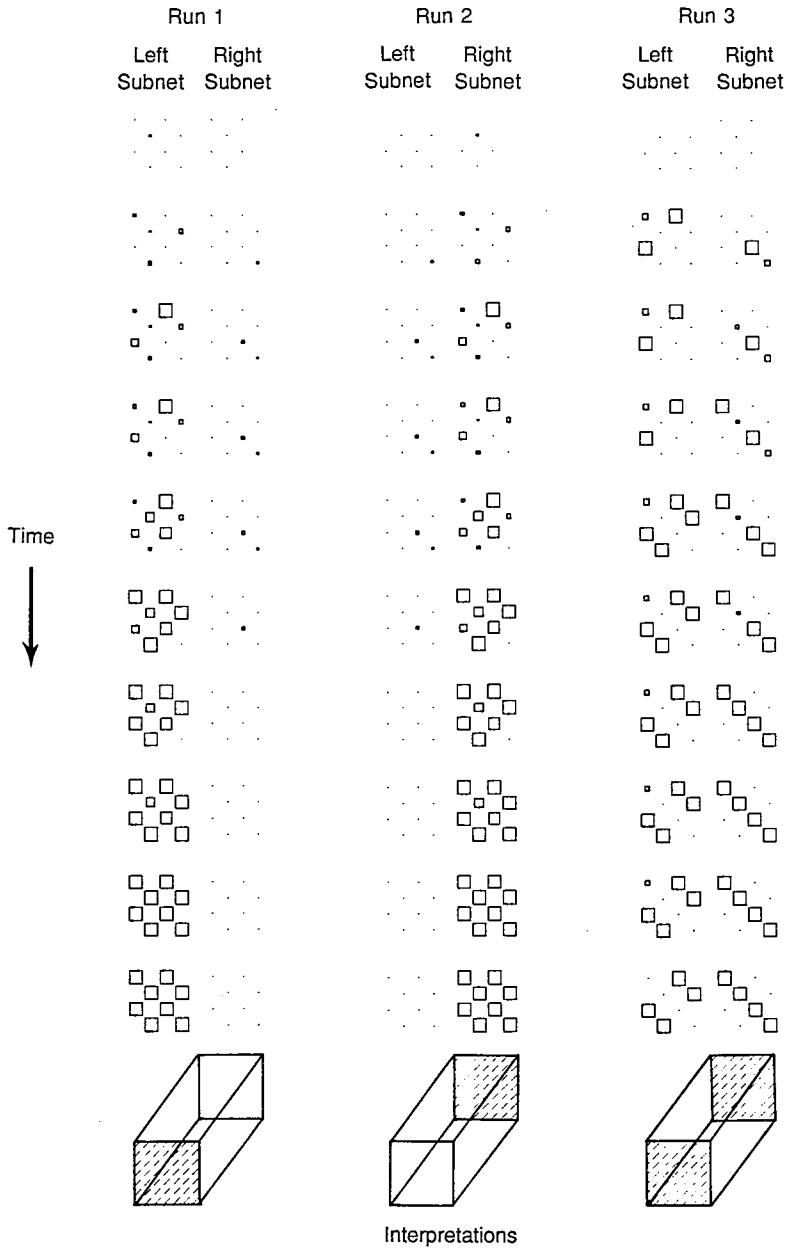

FIGURE 2. Three runs of a simulation based on this network. The size of the square indicates the activation value of each unit. The units are arranged in the shape of the subnetwork with each square shown in its position corresponding to the vertex of the cube from which it is receiving input. The states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

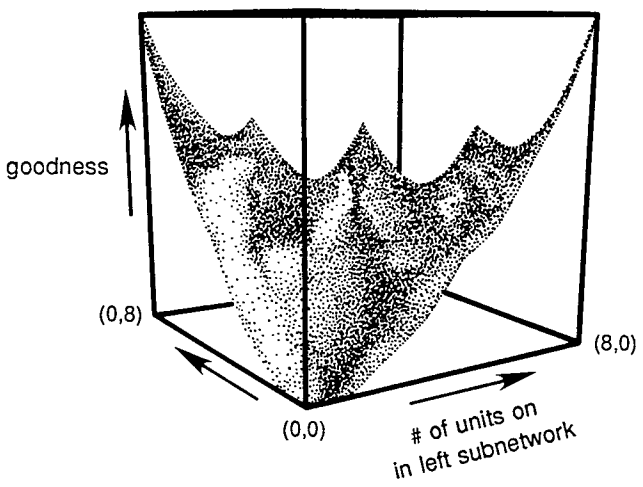

FIGURE 3. The goodness-of-fit surface for the Necker-cube network. The low point at the (0,0) corner corresponds to the start state. The peaks on the right and left correspond to the standard interpretations of the Necker cube, and the peak in the center corresponds to the impossible Necker cube illustrated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

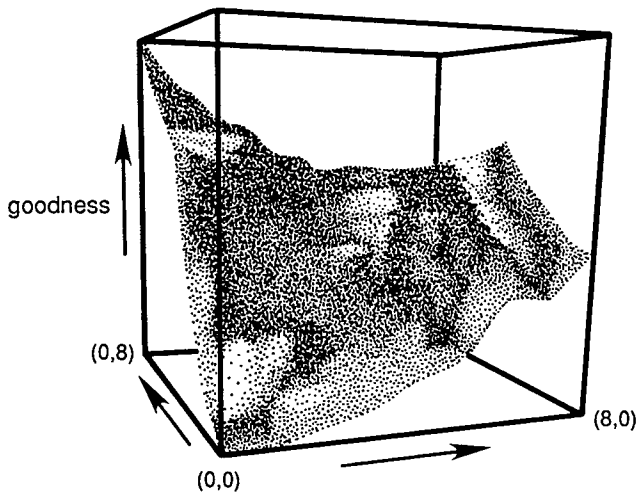

FIGURE 4. The distortions of the goodness landscape when a large input is given to the units corresponding to the front face of a left-facing cube. The figure shows only one major peak corresponding to the view of the left-facing cube.

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)