CHAPTER 2

A General Framework for Parallel Distributed Processing

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

In Chapter 1 and throughout this book, we describe a large number of models, each different in detail—each a variation on the parallel distributed processing (PDP) idea. These various models, and indeed many in the literature, clearly have many features in common, but they are just as clearly distinct models. How can we characterize the general model of which these specific models are instances? In this chapter we propose a framework sufficiently general so that all of the various models discussed in the book and many models in the literature are special cases. We will proceed by first sketching the general framework and then by showing properties of certain specific realizations of the general model.[1]

The General Framework

It is useful to begin with an analysis of the various components of our models and then describe the various specific assumptions we can

[1] We are, of course, not the first to attempt a general characterization of this general class of models. Kohonen (1977, 1984), Amari (1977a), and Feldman and Ballard (1982) are papers with similarly general aims.

make about these components. There are eight major aspects of a parallel distributed processing model:

- A *set of processing units*
- A *state of activation*
- An *output function* for each unit
- A *pattern of connectivity* among units
- A *propagation rule* for propagating patterns of activities through the network of connectivities
- An *activation rule* for combining the inputs impinging on a unit with the current state of that unit to produce a new level of activation for the unit.
- A *learning rule* whereby patterns of connectivity are modified by experience
- An *environment* within which the system must operate

Figure 1 illustrates the basic aspects of these systems. There is a set of processing units generally indicated by circles in our diagrams; at each point in time, each unit u_i has an activation value, denoted in the diagram as $a_i(t)$; this activation value is passed through a function f_i to produce an output value $o_i(t)$. This output value can be seen as passing through a set of unidirectional connections (indicated by lines or arrows in our diagrams) to other units in the system. There is associated with each connection a real number, usually called the *weight* or *strength* of the connection designated w_{ij} which determines the amount of effect that the first unit has on the second. All of the inputs must then be combined by some operator (usually addition)—and the combined inputs to a unit, along with its current activation value, determine, via a function F, its new activation value. The figure shows illustrative examples of the function f and F. Finally, these systems are viewed as being plastic in the sense that the pattern of interconnections is not fixed for all time; rather, the weights can undergo modification as a function of experience. In this way the system can evolve. What a unit represents can change with experience, and the system can come to perform in substantially different ways. In the following sections we develop an explicit notation for each of these components and describe some of the alternate assumptions that have been made concerning each such component.

A set of processing units. Any parallel activation model begins with a set of processing units. Specifying the set of processing units and what they represent is typically the first stage of specifying a PDP model. In some models these units may represent particular conceptual objects such as features, letters, words, or concepts; in others they are

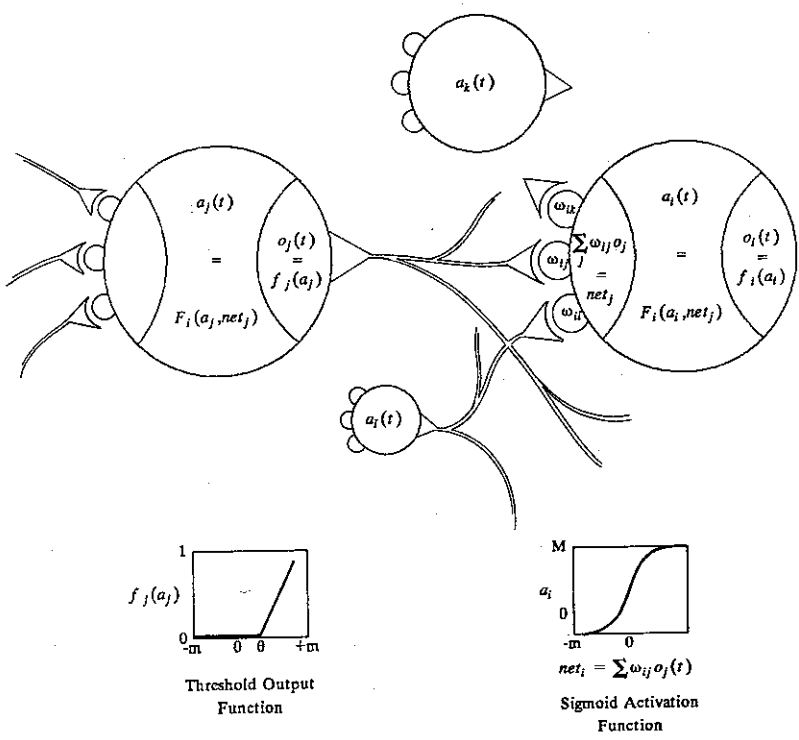


FIGURE 1. The basic components of a parallel distributed processing system.

simply abstract elements over which meaningful patterns can be defined. When we speak of a distributed representation, we mean one in which the units represent small, feature-like entities. In this case it is the pattern as a whole that is the meaningful level of analysis. This should be contrasted to a *one-unit–one-concept* representational system in which single units represent entire concepts or other large meaningful entities.

We let N be the number of units. We can order the units arbitrarily and designate the ith unit u_i. All of the processing of a PDP model is carried out by these units. There is no executive or other overseer. There are only relatively simple units, each doing it own relatively simple job. A unit's job is simply to receive input from its neighbors and, as a function of the inputs it receives, to compute an output value which it sends to its neighbors. The system is inherently parallel in that many units can carry out their computations at the same time.

Within any system we are modeling, it is useful to characterize three types of units: *input, output,* and *hidden.* Input units receive inputs from sources external to the system under study. These inputs may be either sensory input or inputs from other parts of the processing system in which the model is embedded. The output units send signals out of the system. They may either directly affect motoric systems or simply influence other systems external to the ones we are modeling. The hidden units are those whose only inputs and outputs are within the system we are modeling. They are not "visible" to outside systems.

The state of activation. In addition, to the set of units, we need a representation of the state of the system at time t. This is primarily specified by a vector of N real numbers, $\mathbf{a}(t)$, representing the pattern of activation over the set of processing units. Each element of the vector stands for the activation of one of the units at time t. The activation of unit u_i at time t is designated $a_i(t)$. It is the pattern of activation over the set of units that captures what the system is representing at any time. It is useful to see processing in the system as the evolution, through time, of a pattern of activity over the set of units.

Different models make different assumptions about the activation values a unit is allowed to take on. Activation values may be continuous or discrete. If they are continuous, they may be unbounded or bounded. If they are discrete, they may take binary values or any of a small set of values. Thus in some models, units are continuous and may take on any real number as an activation value. In other cases, they may take on any real value between some minimum and maximum such as, for example, the interval [0,1]. When activation values are restricted to discrete values they most often are binary. Sometimes they are restricted to the values 0 and 1 where 1 is usually taken to mean that the unit is active and 0 is taken to mean that it is inactive. In other models, activation values are restricted to the values $\{-1,+1\}$ (often denoted simply $\{-,+\}$). Other times nonbinary discrete values are involved. Thus, for example, they may be restricted to the set $\{-1,0,+1\}$, or to a small finite set of values such as $\{1,2,3,4,5,6,7,8,9\}$. As we shall see, each of these assumptions leads to a model with slightly different characteristics. It is part of the program of research represented in this book to determine the implications of these various assumptions.

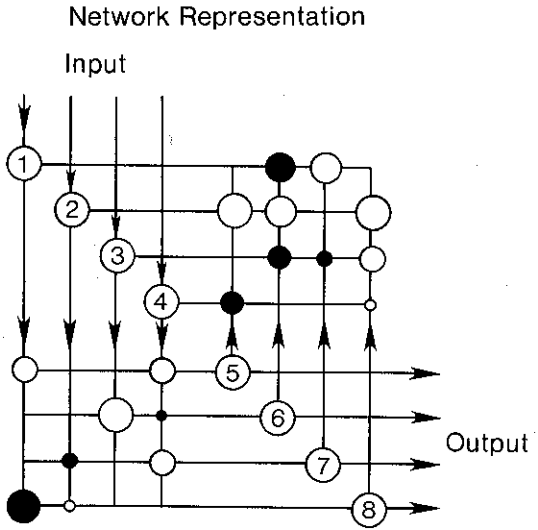
Output of the units. Units interact. They do so by transmitting signals to their neighbors. The strength of their signals, and therefore the degree to which they affect their neighbors, is determined by their degree of activation. Associated with each unit, u_i, there is an output function, $f_i(a_i(t))$, which maps the current state of activation $a_i(t)$ to

an output signal $o_i(t)$ (i.e., $o_i(t) = f_i(a_i(t))$). In vector notation, we represent the current set of output values by a vector, $\mathbf{o}(t)$. In some of our models the output level is exactly equal to the activation level of the unit. In this case f is the identity function $f(x)=x$. More often, however, f is some sort of threshold function so that a unit has no affect on another unit unless its activation exceeds a certain value. Sometimes the function f is assumed to be a stochastic function in which the output of the unit depends in a probabilistic fashion on its activation values.

The pattern of connectivity. Units are connected to one another. It is this pattern of connectivity that constitutes what the system knows and determines how it will respond to any arbitrary input. Specifying the processing system and the knowledge encoded therein is, in a parallel distributed processing model, a matter of specifying this pattern of connectivity among the processing units.

In many cases, we assume that each unit provides an additive contribution to the input of the units to which it is connected. In such cases, the total input to the unit is simply the weighted sum of the separate inputs from each of the individual units. That is, the inputs from all of the incoming units are simply multiplied by a weight and summed to get the overall input to that unit. In this case, the total pattern of connectivity can be represented by merely specifying the weights for each of the connections in the system. A positive weight represents an excitatory input and a negative weight represents an inhibitory input. As mentioned in the previous chapter, it is often convenient to represent such a pattern of connectivity by a weight matrix $\mathbf{W}$ in which the entry w_{ij} represents the strength and sense of the connection from unit u_j to unit u_i. The weight w_{ij} is a positive number if unit u_j excites unit u_i; it is a negative number if unit u_j inhibits unit u_i; and it is 0 if unit u_j has no direct connection to unit u_i. The absolute value of w_{ij} specifies the *strength of the connection.* Figure 2 illustrates the relationship between the connectivity and the weight matrix.

In the general case, however, we require rather more complex patterns of connectivity. A given unit may receive inputs of different kinds whose effects are separately summated. For example, in the previous paragraph we assumed that the excitatory and inhibitory connections simply summed algebraically with positive weights for excitation and negative weights for inhibition. Sometimes, more complex inhibition/excitation combination rules are required. In such cases it is convenient to have separate connectivity matrices for each kind of connection. Thus, we can represent the pattern of connectivity by a set of connectivity matrices, $\mathbf{W}_i$, one for each *type* of connection. It is common, for example, to have two types of connections in a model: an



Matrix Representation

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
u_1	0	0	0	0	0	−5	−5	0
u_2	0	0	0	0	+6	+5	0	+6
u_3	0	0	0	0	0	−3	−2	+3
u_4	0	0	0	0	−4	0	0	+1
u_5	+4	0	0	+4	0	0	0	0
u_6	0	0	+6	−1	0	0	0	0
u_7	0	−2	0	+4	0	0	0	0
u_8	−6	+1	0	0	0	0	0	0

FIGURE 2. The connectivity of a network represented by a network drawing and in a matrix. The figure shows an eight-unit network with units numbered from 1 to 8. Units 1 to 4 are input units. They receive inputs from the outside world and feedback from the output units—units 5 through 8. The connections among the units are indicated by the open and filled disks. The size of the disk indicates the strength of connection. Thus, the large black disk on the line connecting unit 1 to unit 8 indicates a strong inhibitory connection from 1 to 8. Similarly, the large open disk on the output line from unit 8 to unit 2 indicates that unit 8 strongly excites unit 2. The same connections are shown in the matrix representation on the left. The +6 in the column for u_8 and the row for u_2 indicates that unit 8 strongly excites unit 2. It should be noted that whenever there is a disk on a line connecting the output of one unit to the input of another in the network diagram there is a corresponding nonzero entry in the matrix. If the disk is filled, the entry in the matrix is negative. If the disk is open, the entry is positive. The larger the disk the greater the magnitude of the entry in the matrix. It might also be noted that the connections in the network have been laid out to correspond to the entries of the matrix. The black disk in the upper left corner of the network corresponds to the −6 in the upper left corner of the matrix. Each disk in the network is in the corresponding position of its location in the matrix. The network would not have to be drawn in this way, of course, and the matrix would still capture all of the connectivity information in the network. In general, because network drawings are difficult to work with we will often simply use the matrix representation to specify the pattern of connectivity.

inhibitory connection and an excitatory connection. When the models assume simple addition of inhibition and excitation they do not constitute different *types* of connections in our present sense. They only constitute distinct types when they combine through some more complex rules.

The pattern of connectivity is very important. It is this pattern which determines what each unit represents. As we shall see below, many of the issues concerning whether *top-down* or *bottom-up* processing systems are correct descriptions or whether a system is hierarchical and if so how many levels it has, etc., are all issues of the nature of the connectivity matrix. One important issue that may determine both how much information can be stored and how much serial processing the network must perform is the *fan-in* and *fan-out* of a unit. The fan-in is the number of elements that either excite or inhibit a given unit. The fan-out of a unit is the number of units affected directly by a unit. Note, in some cases we need more general patterns of connectivity. Specifying such a pattern in the general case is complex and will be addressed in a later section of this chapter.

The rule of propagation. We also need a rule which takes the output vector, $\mathbf{o}(t)$, representing the output values of the units and combines it with the connectivity matrices to produce a *net input* for each type of input into the unit. We let net_{ij} be the net input of type i to unit u_j. Whenever only one type of connectivity is involved we suppress the first subscript and use net_j to mean the net input into unit u_j. In vector notation we can write $\mathbf{net}_i(t)$ to represent the net input vector for inputs of type i. The propagation rule is generally straightforward. For example, if we have two types of connections, inhibitory and excitatory, the net excitatory input is usually the weighted sum of the excitatory inputs to the unit. This is given by the vector product $\mathbf{net}_e = \mathbf{W}_e\mathbf{o}(t)$. Similarly, the net inhibitory effect can be written as $\mathbf{net}_i = \mathbf{W}_i\mathbf{o}(t)$. When more complex patterns of connectivity are involved, more complex rules of propagation are required. We treat this in the final section of the chapter.

Activation rule. We also need a rule whereby the net inputs of each type impinging on a particular unit are combined with one another and with the current state of the unit to produce a new state of activation. We need a function, $\mathbf{F}$, which takes $\mathbf{a}(t)$ and the vectors $\mathbf{net}_j$ for each different type of connection and produces a new state of activation. In the simplest cases, when $\mathbf{F}$ is the identity function and when all connections are of the same type, we can write $\mathbf{a}(t+1) = \mathbf{W}\mathbf{o}(t) = \mathbf{net}(t)$. Sometimes $\mathbf{F}$ is a threshold function so that the net input must exceed some value before contributing to the new state of activation. Often,

the new state of activation depends on the old one as well as the current input. In general, however, we have

$$\mathbf{a}(t+1) = \mathbf{F}(\mathbf{a}(t), \mathbf{net}(t)_1, \mathbf{net}(t)_2, \ldots);$$

the function **F** itself is what we call the activation rule. Usually, the function is assumed to be deterministic. Thus, for example, if a threshold is involved it may be that $a_i(t) = 1$ if the total input exceeds some threshold value and equals 0 otherwise. Other times it is assumed that **F** is stochastic. Sometimes activations are assumed to decay slowly with time so that even with no external input the activation of a unit will simply decay and not go directly to zero. Whenever $a_i(t)$ is assumed to take on continuous values it is common to assume that **F** is a kind of sigmoid function. In this case, an individual unit can *saturate* and reach a minimum or maximum value of activation.

Perhaps the most common class of activations functions is the *quasi-linear* activation function. In this case the activation function, **F**, is a nondecreasing function of a single *type* of input. In short,

$$a_i(t+1) = \mathbf{F}(net_i(t)) = \mathbf{F}(\sum_j w_{ij} o_j).$$

It is sometimes useful to add the constraint that **F** be a *differentiable* function. We refer to differentiable quasi-linear activation functions as *semilinear* functions (see Chapter 8).

Modifying patterns of connectivity as a function of experience. Changing the processing or knowledge structure in a parallel distributed processing model involves modifying the patterns of interconnectivity. In principle this can involve three kinds of modifications:

1. The development of new connections.
2. The loss of existing connections.
3. The modification of the strengths of connections that already exist.

Very little work has been done on (1) and (2) above. To a first order of approximation, however, (1) and (2) can be considered a special case of (3). Whenever we change the strength of connection away from zero to some positive or negative value, it has the same effect as growing a new connection. Whenever we change the strength of a connection to zero, that has the same effect as losing an existing connection. Thus, in this section we will concentrate on rules whereby *strengths* of connections are modified through experience.

Virtually all learning rules for models of this type can be considered a variant of the *Hebbian* learning rule suggested by Hebb in his classic book *Organization of Behavior* (1949). Hebb's basic idea is this: If a unit, u_i, receives a input from another unit, u_j; then, if both are highly active, the weight, w_{ij}, from u_j to u_i should be *strengthened.* This idea has been extended and modified so that it can be more generally stated as

$$\Delta w_{ij} = g(a_i(t), t_i(t)) h(o_j(t), w_{ij}),$$

where $t_i(t)$ is a kind of *teaching* input to u_i. Simply stated, this equation says that the change in the connection from u_j to u_i is given by the product of a function, $g()$, of the activation of u_i and its teaching input t_i and another function, $h()$, of the output value of u_j and the connection strength w_{ij}. In the simplest versions of Hebbian learning there is no teacher and the functions g and h are simply proportional to their first arguments. Thus we have

$$\Delta w_{ij} = \eta a_i o_j,$$

where η is the constant of proportionality representing the learning rate. Another common variation is a rule in which $h(o_j(t), w_{ij}) = o_j(t)$ and $g(a_i(t), t_i(t)) = \eta(t_i(t) - a_i(t))$. This is often called the *Widrow-Hoff* rule (Sutton & Barto, 1981). However, we call it the *delta rule* because the amount of learning is proportional to the *difference* (or delta) between the actual activation achieved and the target activation provided by a teacher. (The delta rule is discussed at length in Chapters 8 and 11.) In this case we have

$$\Delta w_{ij} = \eta(t_i(t) - a_i(t)) o_j(t).$$

This is a generalization of the *perceptron* learning rule for which the famous *perception convergence theorem* has been proved. Still another variation has

$$\Delta w_{ij} = \eta a_i(t)(o_j(t) - w_{ij}).$$

This is a rule employed by Grossberg (1976) and a simple variant of which has been employed in Chapter 5. There are many variations on this generalized rule, and we will describe some of them in more detail when we discuss various specific models below.

Representation of the environment. It is crucial in the development of any model to have a clear model of the environment in which this model is to exist. In PDP models, we represent the environment as a time-varying stochastic function over the space of input patterns. That

is, we imagine that at any point in time, there is some probability that any of the possible set of input patterns is impinging on the input units. This probability function may in general depend on the history of inputs to the system as well as outputs of the system. In practice, most PDP models involve a much simpler characterization of the environment. Typically, the environment is characterized by a stable probability distribution over the set of possible input patterns independent of past inputs and past responses of the system. In this case, we can imagine listing the set of possible inputs to the system and numbering them from 1 to M. The environment is then characterized by a set of probabilities, p_i for $i = 1, \ldots, M$. Since each input pattern can be considered a vector, it is sometimes useful to characterize those patterns with nonzero probabilities as constituting *orthogonal* or *linearly independent* sets of vectors.[2] Certain PDP models are restricted in the kinds of patterns they are able to learn: some being able to learn to respond correctly only if the input vectors form an orthogonal set; others if they form a linearly independent set of vectors; and still others are able to learn to respond to essentially arbitrary patterns of inputs.

CLASSES OF PDP MODELS

There are many paradigms and classes of PDP models that have been developed. In this section we describe some general classes of assumptions and paradigms. In the following section we describe some specific PDP models and show their relationships to the general framework outlined here.

Paradigms of Learning

Although most learning rules have roughly the form indicated above, we can categorize the learning situation into two distinct sorts. These are:

- *Associative learning,* in which we learn to produce a particular pattern of activation on one set of units whenever another particular pattern occurs on another set of units. In general, such a learning scheme must allow an arbitrary pattern on one set of

[2] See Chapter 9 for explication of these terms.

units to produce another arbitrary pattern on another set of units.

- *Regularity discovery,* in which units learn to respond to "interesting" patterns in their input. In general, such a scheme should be able to form the basis for the development of feature detectors and therefore the basis for knowledge representation in a PDP system.

In certain cases these two modes of learning blend into one another, but it is valuable to see the different goals of the two kinds of learning. Associative learning is employed whenever we are concerned with storing patterns so that they can be re-evoked in the future. These rules are primarily concerned with storing the relationships among subpatterns. Regularity detectors are concerned with the *meaning* of a single units response. These kinds of rules are used when *feature discovery* is the essential task at hand.

The associative learning case generally can be broken down into two subcases—pattern association and auto-association. A *pattern association* paradigm is one in which the goal is to build up an association between patterns defined over one subset of the units and other patterns defined over a second subset of units. The goal is to find a set of connections so that whenever a particular pattern reappears on the first set of units, the associated pattern will appear on the second set. In this case, there is usually a *teaching input* to the second set of units during training indicating the desired pattern association. An *auto-association* paradigm is one in which an input pattern is associated with itself. The goal here is pattern completion. Whenever a *portion* of the input pattern is presented, the remainder of the pattern is to be filled in or completed. This is similar to simple pattern association, except that the input pattern plays both the role of the teaching input and of the pattern to be associated. It can be seen that simple pattern association is a special case of auto-association. Figure 3 illustrates the two kinds of learning paradigms. Figure 3A shows the basic structure of the pattern association situation. There are two distinct groups of units—a set of input units and a set of output units. Each input unit connects with each output unit and each output unit receives an input from each input unit. During training, patterns are presented to both the input and output units. The weights connecting the input to the output units are modified during this period. During a test, patterns are presented to the input units and the response on the output units is measured. Figure 3B shows the connectivity matrix for the pattern associator. The only modifiable connections are from the input units to the output units. All other connections are fixed at zero. Figure 3C shows the basic

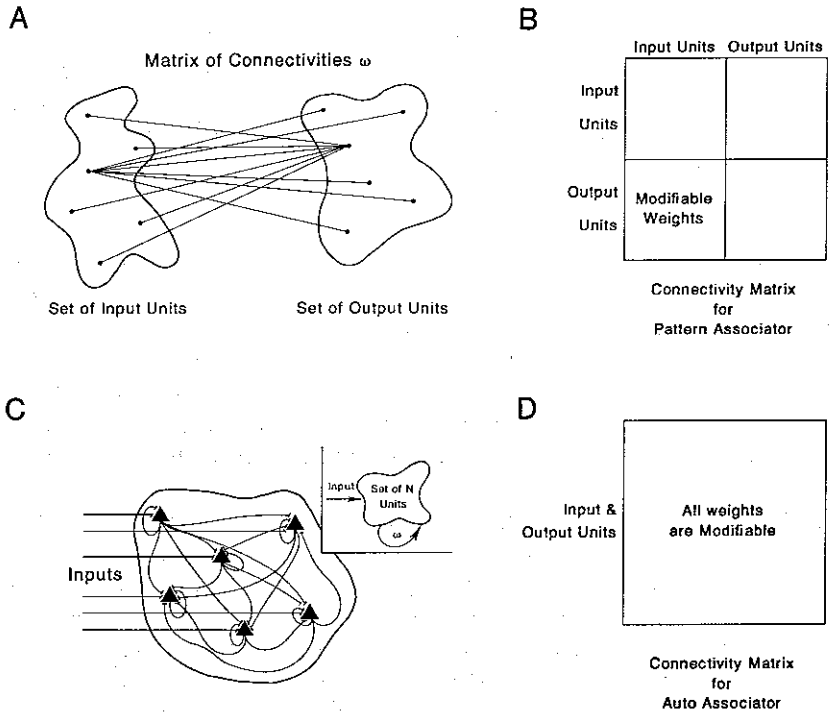


FIGURE 3. *A:* The basic structure of the pattern association situation. There are two distinct groups of units—a set of input units and a set of output units. Each input unit connects with each output unit and each output unit receives an input from each input unit. During training, patterns are presented to both the input and output units. The weights connecting the input to the output units are modified during this period. During a test, patterns are presented to the input units and the response on the output units is measured. (After Anderson, 1977.) *B:* The connectivity matrix for the pattern associator. The only modifiable connections are from the input units to the output units. All other connections are fixed at zero. *C:* The basic structure of the auto-association situation. All units are both input and output units. The figure shows a group of 6 units feeding back on itself through modifiable connections. Note that each unit feeds back on itself as well as on each of its neighbors. (After Anderson, Silverstein, Ritz, & Jones, 1977.) *D:* The connectivity matrix for the auto-associator. All units connect to all other units with modifiable weights.

structure of the auto-association situation. All units are both input and output units. The figure shows a group of 6 units feeding back on itself through modifiable connections. Note that each unit feeds back on itself as well as on each of its neighbors. Figure 3D shows the connectivity matrix for the auto-associator. All units connect to all other units with modifiable weights. In the case of auto-association, there is

potentially a modifiable connection from every unit to every other unit. In the case of pattern association, however, the units are broken into two subpatterns, one representing the input pattern and another representing the teaching input. The only modifiable connections are those from the input units to the output units receiving the teaching input. In other cases of associative learning the teaching input may be more or less indirect. The problem of dealing with indirect feedback is difficult, but central to the development of more sophisticated models of learning. Barto and Sutton (1981) have begun a nice analysis of such learning situations.

In the case of regularity detectors, a teaching input is not explicitly provided; instead, the teaching function is determined by the unit itself. The form of the internal teaching function and the nature of its input patterns determine what features the unit will learn to respond to. This is sometimes called unsupervised learning. Each different kind of unsupervised learning procedure has its own evaluation function. The particular evaluation procedures are mentioned when we treat these models. The three unsupervised learning models discussed in this book are addressed in Chapters 5, 6, and 7.

Hierarchical Organizations of PDP Networks

It has become commonplace in cognitive science to describe such processes as *top-down*, *bottom-up*, and *interactive* to consist of many stages of processing, etc. It is useful to see how these concepts can be represented in terms of the patterns of connectivity in the PDP framework. It is also useful to get some feeling for the processing consequences of these various assumptions.

Bottom-Up Processing

The fundamental characteristic of a bottom-up system is that units at level i may not affect the activity of units at levels lower than i. To see how this maps onto the current formulation, it is useful to partition the coalitions of units into a set of discrete categories corresponding to the levels their inputs come from. There are assumed to be no coalitions with inputs from more than one level. Assume that there are L_i units at level i in the system. We then order the units such that those in level L_1 are numbered $u_1, \ldots, u_{L_1}$, those in level L_2 are numbered $u_{L_1+1}, \ldots, u_{L_1+L_2}$, etc. Then, the constraint that the system be a pure

bottom-up system is equivalent to the constraint that the connectivity matrix, **W**, has zero entries for w_{ij} in which u_j is the member of a level no higher than u_i. This amounts to the requirement that the upper right-hand region of **W** contains zero entries. Table 1 shows this constraint graphically. The table shows an example of a three-level system with four units at each level.[3] This leads to a 12×12 connectivity matrix and an **a** vector of length 12. The matrix can be divided up into 9 regions. The upper-left region represents interactions among Level 1 units. The entries in the left-middle region of the matrix represents the effects of Level 1 units on Level 2 units. The lower-left region represents the effects of Level 1 units on Level 3 units. Often bottom-up models do not allow units at level i effect units at level $i+2$. Thus, in the diagram we have left that region empty representing no effect of Level 1 on Level 3. It is typical in a bottom-up system to assume as well that the lowest level units (Level 1) are input units and that the highest level units (Level 3) are output units. That is, the lowest level of the system is the only one to receive direct inputs from outside of this module and only the highest level units affect other units outside of this module.

TABLE 1

		Level 1 Input Units u1 u2 u3 u4	Level 2 Hidden Units u5 u6 u7 u8	Level 3 Output Units u9 u10 u11 u12
Level 1 Units	u1 u2 u3 u4	within Level 1 effects		
Level 2 Units	u5 u6 u7 u8	Level 1 affecting Level 2	within Level 2 effects	
Level 3 Units	u9 u10 u11 u12		Level 2 affecting Level 3	within Level 3 effects

[3] In general, of course, we would expect many levels and many units at each level.

Top-Down Processing

The generalization to a hierarchical top-down system should be clear enough. Let us order the units into levels just as before. A top-down model then requires that the lower-left regions of the weight matrix be empty—that is, no lower level unit affects a higher level unit. Table 2 illustrates a simple example of a top-down processing system. Note, in this case, we have to assume a top-down input or "message" that is propagated down the system from higher to lower levels as well as any data input that might be coming directly into Level 1 units.

Interactive Models

Interactive models are simply models in which there can be both top-down and bottom-up connections. Again the generalization is straightforward. In the general interactive model, any of the cells of the weight matrix could be nonzero. The more restricted models in which information flows both ways, but in which information only flows between adjacent levels, assume only that the regions of the matrix more than one region away from the main diagonal are zero. Table 3 illustrates a simple three-level interactive model with both top-down and bottom-up input. Most of the models that actually have been suggested count as interactive models in this sense.

TABLE 2

		Level 1 Input Units u1 u2 u3 u4	Level 2 Hidden Units u5 u6 u7 u8	Level 3 Output Units u9 u10 u11 u12
Level 1 Units	u1 u2 u3 u4	within Level 1 effects	Level 2 affecting Level 1	
Level 2 Units	u5 u6 u7 u8		within Level 2 effects	Level 3 affecting Level 2
Level 3 Units	u9 u10 u11 u12			within Level 3 effects

TABLE 3

		Level 1 Input Units u1 u2 u3 u4	Level 2 Hidden Units u5 u6 u7 u8	Level 3 Output Units u9 u10 u11 u12
Level 1 Units	u1 u2 u3 u4	within Level 1 effects	Level 2 affecting Level 1	
Level 2 Units	u5 u6 u7 u8	Level 1 affecting Level 2	within Level 2 effects	Level 3 affecting Level 2
Level 3 Units	u9 u10 u11 u12		Level 2 affecting Level 3	within Level 3 effects

It is sometimes supposed that a "single level" system with *no hierarchical structure* in which any unit can communicate with any other unit is somehow less powerful than these multilevel hierarchical systems. The present analysis shows that, on the contrary, the *existence* of *levels* amounts to a *restriction*, in general, of free communication among all units. Such *nonhierarchical* systems actually form a superset of the kinds of *layered* systems discussed above. There is, however, something to the view that having multiple levels can increase the power of certain systems. In particular, a "one-step" system consisting of only input and output units and no communication between them in which there is no opportunity for feedback or for hidden units is less powerful than systems with hidden units and with feedback. Since, in general, hierarchical systems involve many hidden units, some intralevel communication, and some feedback among levels, they are more powerful than systems not involving such hidden units. However, a system with an equal number of hidden units, but one not characterizable as hierarchical by the communication patterns is, in general, of more potential computational power. We address the issue of hidden units and "single-step" versus "multiple-step" systems in our discussion of specific models below.

Synchronous Versus Asynchronous Update

Even given all of the components of the PDP models we have described so far, there is still another important issue to be resolved in the development of specific models; that is the timing of the application of the activation rule. In some models, there is a kind of central timing pulse and after each such clock tick a new value is determined simultaneously for all units. This is a *synchronous update* procedure. It is usually viewed as a discrete, difference approximation to an underlying continuous, differential equation in which all units are continuously updated. In some models, however, units are updated *asynchronously* and at random. The usual assumption is that at each point in time each unit has a fixed probability of evaluating and applying its activation rule and updating its activation value. This later method has certain theoretical advantages and was developed by Hopfield (1982) and has been employed in Chapters 6, 7, and 14. The major advantage is that since the units are independently being updated, if we look at a short enough time interval, only one unit is updating at a time. Among other things, this system can help the stability of the network by keeping it out of oscillations that are more readily entered into with synchronous update procedures.

SPECIFIC VERSIONS OF THE GENERAL PARALLEL ACTIVATION MODEL

In the following sections we will show how specification of the particular functions involved produces various kinds of these models. There have been many authors who have contributed to the field and whose work might as well have been discussed. We discuss only a representative sample of this work.

Simple Linear Models

Perhaps the simplest model of this class is the simple linear model. In the simple linear model, activation values are real numbers without restriction. They can be either positive or negative and are not bounded. The output function, $f(a_i)$, in the linear model is just equal to the activation level a_i. Typically, linear models consist of two sets of units: a set of *input* units and a set of *output* units. (As discussed

below, there is no need for hidden units since all computation possible with a multiple-step linear system can be done with a single-step linear system.) In general, any unit in the input layer may connect to any unit in the output layer. All connections in a linear model are of the same type. Thus, only a single connectivity matrix is required. The matrix consists of a set of positive, negative, and zero values, for excitatory values, inhibitory values, and zero connections, respectively. The new value of activation of each unit is simply given by the weighted sums of the inputs. For the simple linear model with connectivity matrix $\mathbf{W}$ we have

$$\mathbf{a}(t+1) = \mathbf{W}\mathbf{a}(t).$$

In general, it can be shown that a linear model such as this has a number of limitations. In particular, it can be shown that nothing can be computed from two or more steps that cannot be computed by a single step. This follows because the above equation implies

$$\mathbf{a}(t) = \mathbf{W}^t \mathbf{a}(0).$$

We can see this by proceeding step by step. Clearly,

$$\mathbf{a}(2) = \mathbf{W}\mathbf{a}(1) = \mathbf{W}(\mathbf{W}\mathbf{a}(0)) = \mathbf{W}^2\mathbf{a}(0).$$

It should be clear that similar arguments lead to $\mathbf{a}(t) = \mathbf{W}^t \mathbf{a}(0)$. From this, it follows that for every linear model with connectivity matrix $\mathbf{W}$ that can attain a particular state in t steps, there is another linear model with connectivity matrix $\mathbf{W}^t$ that can reach the same state in one step. This means, among other things, that there can never be any computational advantage in a linear model of multiple-step systems, nor can there ever be any advantage for allowing feedback.

The pattern association paradigm is the typical learning situation for a linear model. There is a set of input units and a set of output units. In general, each input unit may be connected to any output unit. Since this is a linear network, there is no feedback in the system nor are there hidden units between the inputs and outputs. There are two sources of input in the system. There are the input patterns that establish a pattern of activation on the input units, and there are the teaching units that establish a pattern of activation on the output units. Any of several learning rules could be employed with a linear network such as this, but the most common are the simple Hebbian rule and the delta rule. The linear model with the simple Hebbian rule is called the simple *linear associator* (cf. Anderson, 1970; Kohonen, 1977, 1984). In this case, the increment in weight w_{ij} is given by $\Delta w_{ij} = \eta a_j t_i$. In matrix notation, this means that $\Delta\mathbf{W} = \eta\mathbf{T}\mathbf{a}^T$. The system is then tested by presenting an input pattern without a teaching input and

seeing how close the pattern generated on the output layer matches the original teaching input. It can be shown that if the input patterns are orthogonal,[4] there will be no interference and the system will perfectly produce the relevant associated patterns exactly on the output layer. If they are not orthogonal, however, there will be interference among the input patterns. It is possible to make a modification in the learning rule and allow a much larger set of possible associations. In particular, it is possible to build up correct associations among patterns whenever the set of input patterns are linearly independent. To achieve this, an error correcting rule must be employed. The delta rule is most commonly employed. In this case, the rule becomes $\Delta w_{ij} = \eta(t_i - a_i)a_j$. What is learned is essentially the difference between the desired response and that actually attained at unit u_i due to the input. Although it may take many presentations of the input pattern set, if the patterns are linearly independent the system will eventually be able to produce the desired outputs. Kohonen (1977, 1984) has provided an important analysis of this and related learning rules.

The examples described above were for the case of the pattern associator. Essentially the same results hold for the auto-associator version of the linear model. In this case, the input patterns and the teaching patterns are the same, and the input layer and the output layer are also the same. The tests of the system involve presenting a portion of the input pattern and having the system attempt to reconstruct the missing parts.

Linear Threshold Units

The weaknesses of purely linear systems can be overcome through the addition of nonlinearities. Perhaps the simplest of the nonlinear system consists of a network of linear threshold units. The linear threshold unit is a binary unit whose activation takes on the values $\{0,1\}$. The activation value of unit u_i is 1 if the weighted sum of its inputs is greater than some threshold θ_i and is 0 otherwise. The connectivity matrix for a network of such units, as in the linear system, is a matrix consisting of positive and negative numbers. The output function, f, is the identity function so that the output of a unit is equal to its activation value.

[4] See Chapter 9 for a discussion of orthogonality, linear independence , etc.

It is useful to see some of the kinds of functions that can be computed with linear threshold units that cannot be computed with simple linear models. The classic such function is the *exclusive or* (XOR) illustrated in Figure 4. The idea is to have a system which responds {1} if it receives a {0,1} or a {1,0} and responds {0} otherwise. The figure shows a network capable of this pattern. In this case we require two

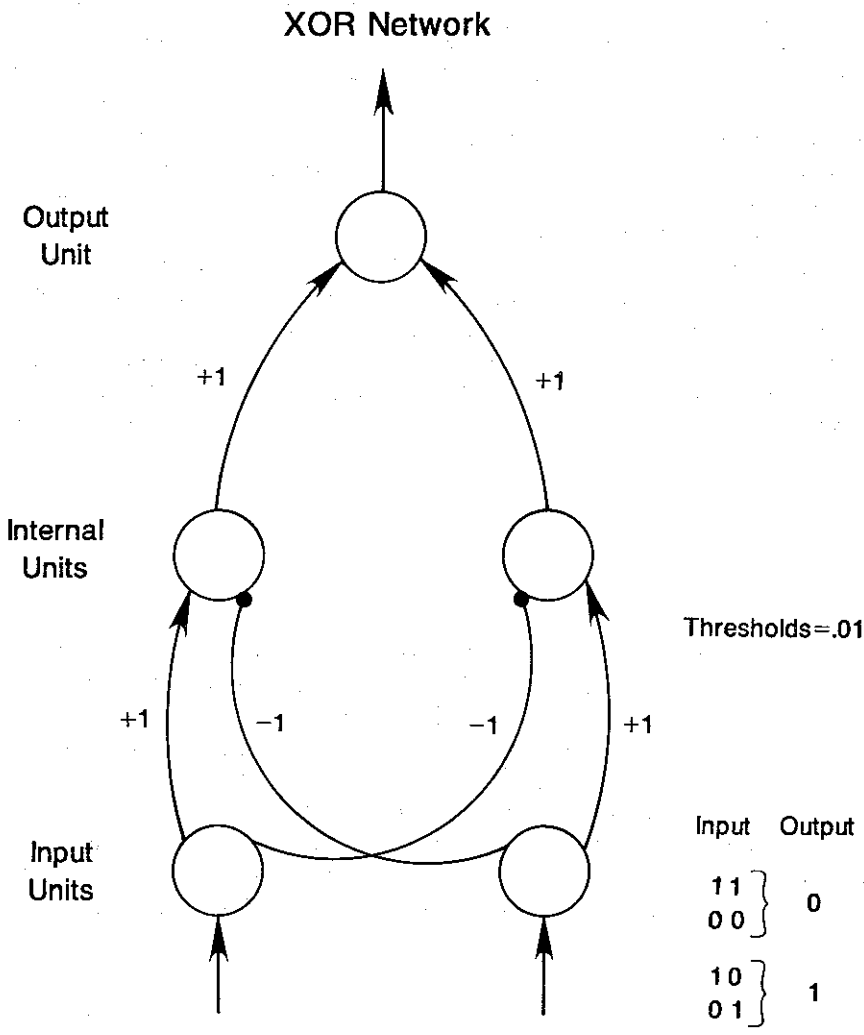


FIGURE 4. A network of linear threshold units capable of responding correctly on the XOR problem.

layers of units. Each unit has a zero threshold and responds just in case its input is greater than zero. The weights are ± 1. Since the set of stimulus patterns is not linearly independent, this is a discrimination that can never be made by a simple linear model and cannot be done in a single step by any network of linear threshold units.

Although multilayered systems of linear threshold units are very powerful and, in fact, are capable of computing any boolean function, there is no generally known learning algorithm for this general case (see Chapter 8). There is, however, a well-understood learning algorithm for the special case of the *perceptron*. A perceptron is essentially a single-layer network of linear threshold units without feedback. The learning situation here is exactly the same as that for the linear model. An input pattern is presented along with a teaching input. The perceptron learning rule is precisely of the same form as the delta rule for error correcting in the linear model, namely, $\Delta w_{ij} = \eta(t_i - a_i)a_j$. Since the teaching input and the activation values are only 0 or 1, the rule reduces to the statements that:

1. Weights are only changed on a given input line when that line is turned on (i.e., $a_j = 1$).

2. If the system is correct on unit i (i.e., $t_i = a_i$), make no change on any of the input weights.

3. If the unit j responds 0 when it should be 1, increase weights on all active lines by amount η.

4. If the unit j responds 1 when it should be 0, decrease weights on all active lines by amount η.

There is a theorem, the perceptron convergence theorem, that guarantees that if the set of patterns are learnable by a perceptron, this learning procedure will find a set of weights which allow it to respond correctly to all input patterns. Unfortunately, even though multilayer linear threshold networks are potentially much more powerful than the linear associator, the perceptron for which a learning result exists can learn no patterns not learnable by the linear associator. It was the limitations on what perceptrons could possibly learn that led to Minsky and Papert's (1969) pessimistic evaluation of the perceptron. Unfortunately that evaluation has incorrectly tainted more interesting and powerful networks of linear threshold and other nonlinear units. We have now developed a version of the delta rule—the generalized delta rule—which is capable of learning arbitrary mappings. It does not work for linear threshold units, but *does work* for the class of *semilinear* activation

functions (i.e., differentiable activation functions). See Chapter 8 for a full discussion. As we shall see in the course of this book, the limitations of the one-step perceptron in no way apply to the more complex networks.

Brain State in a Box

The brain state in a box model was developed by J. A. Anderson (1977). This model too is a close relative of the simple linear associator. There is, however, a maximum and minimum activation value associated with each unit. Typically, units take on activation values in the interval $[-1,1]$. The brain state in a box (BSB) models are organized so that any unit can, in general, be connected to any other unit. The auto-associator illustrated in Figure 3 is the typical learning paradigm for BSB. Note that with this pattern of interconnections the system feeds back on itself and thus the activation can recycle through the system in a positive feedback loop. The positive feedback is especially evident in J. A. Anderson and Mozer's (1981) version. Their activation rule is given by

$$a_j(t+1) = a_j(t) + \sum w_{ij} a_i(t)$$

if a_j is less than 1 and greater than -1. Otherwise, if the quantity is greater than 1, $a_j = 1$ and if it is less than -1, $a_j = -1$. That is, the activation state at time $t+1$ is given by the sum of the state at time t and the activation propagated through the connectivity matrix provided that total is in the interval $[-1,1]$. Otherwise it simply takes on the maximum or minimum value. This formulation will lead the system to a state in which all of the units are at either a maximum or minimum value. It is possible to understand why this is called a brain state in a box model by considering a geometric representation of the system. Figure 5 illustrates the "activation space" of a simple BSB system consisting of three units. Each point in the box corresponds to a particular value of activation on each of the three units. In this case we have a three-dimensional space in which the first coordinate corresponds to the activation value of the first unit, the second coordinate corresponds to the activation value of the second unit, and the third coordinate corresponds to the activation value of the third unit. Thus, each point in the space corresponds to a possible state of the system. The feature that each unit is limited to the region $[-1,1]$ means that all points must lie somewhere within the box whose vertices are given by the points $(-1,-1,-1)$, $(-1,-1,+1)$, $(-1,+1,-1)$, $(-1,+1,+1)$, $(+1,-1,-1)$, $(+1,-1,+1)$, $(+1,+1,-1)$, and $(+1,+1,+1)$. Moreover, since the

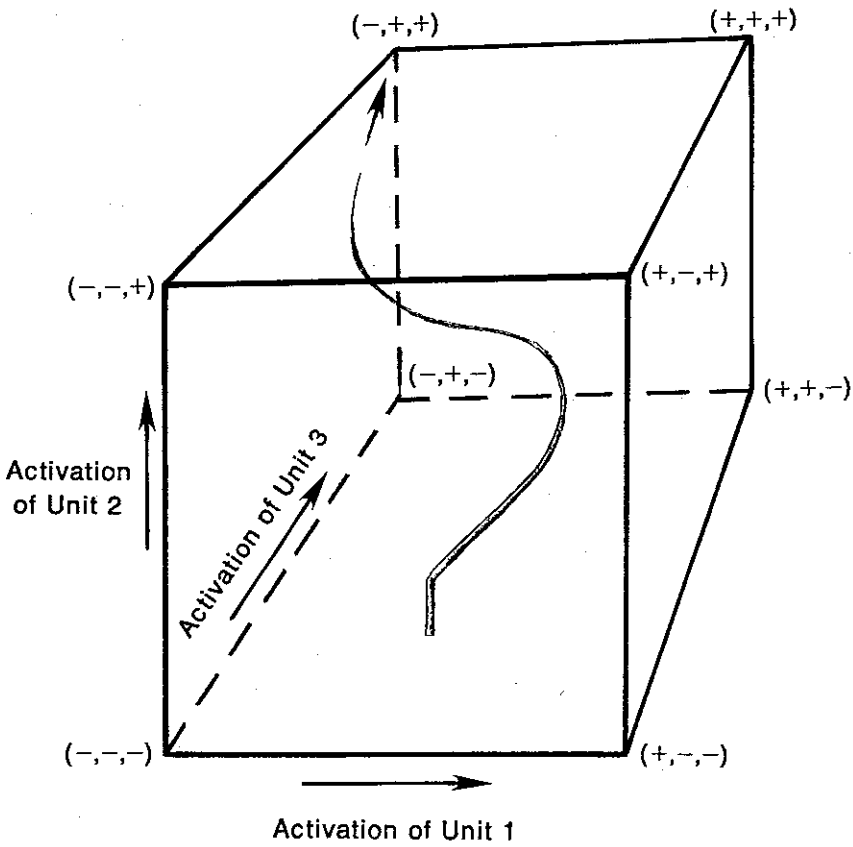


FIGURE 5. The state space for a three-unit version of a BSB model. Each dimension of the box represents the activation value of one unit. Each unit is bounded in activation between [−1,1]. The curving arrow in the box represents the sequence of states the system moved through. It began at the black spot near the middle of the box and, as processing proceeded, moved to the (−,+,+) corner of the box. BSB systems always end up in one or another of the corners. The particular corner depends on the start state of the network, the input to the system, and the pattern of connections among the units.

system involves positive feedback, it is eventually forced to occupy one of these vertices. Thus, the state of the system is constrained to lie within the box and eventually, as processing continues, is pushed to one of the vertices. Of course, the same geometric analogy carries over to higher dimensional systems. If there are N units, the state of the system can be characterized as a point within this N-dimensional hypercube and eventually the system ends up in one of the 2^N corners of the hypercube.

Learning in the BSB system involves auto-association. In different applications two different learning rules have been applied. J. A. Anderson and Mozer (1981) applied the simplest rule. They simply allowed the system to settle down and then employed the simple Hebbian learning rule. That is, $\Delta w_{ij} = \eta a_i a_j$. The error correction rule has also been applied to the BSB model. In this case we use the input as the teaching input as well as the source of activation to the system. The learning rule thus becomes $\Delta w_{ij} = \eta (t_i - a_i) a_j$ where t_i is the input to unit i and where a_i and a_j are the activation values of the system after it has stabilized in one of the corners of the hypercube.

Thermodynamic Models

Other more recent developments are the thermodynamic models. Two examples of such models are presented in the book. One, *harmony theory*, was developed by Paul Smolensky and is described in detail in Chapter 6. The other, the Boltzmann machine, was developed by Hinton and Sejnowski and is described in Chapter 7. Here we describe the basic idea behind these models and show how they relate to the general class of models under discussion. To begin, the thermodynamic models employ binary units which take on the values {0,1}. The units are divided into two categories: the *visible* units corresponding to our input and output units and the *hidden* units. In general, any unit may connect to any other unit. However, there is a constraint that the connections must be symmetric. That is, the $w_{ij} = w_{ji}$. In these models, there is no distinction between the output of the unit and its activation value. The activation values are, however, a stochastic function of the inputs. That is,

$$p(a_i(t)=1) = \frac{1}{1 + e^{-(\sum_j w_{ij} a_j + \eta_i - \theta_i)/T}}$$

where η_i is the input from outside of system into unit i, θ_i is the threshold for the unit, and T is a parameter, called *temperature,* which determines the slope of the probability function. Figure 6 shows how the probabilities vary with various values of T. It should be noted that as T approaches zero, the individual units become more and more like linear threshold units. In general, if the unit exceeds threshold by a great enough margin it will always attain value 1. If it is far enough below threshold, it always takes on value 0. Whenever the unit is above threshold, the probability that it will turn on is greater than 1/2.

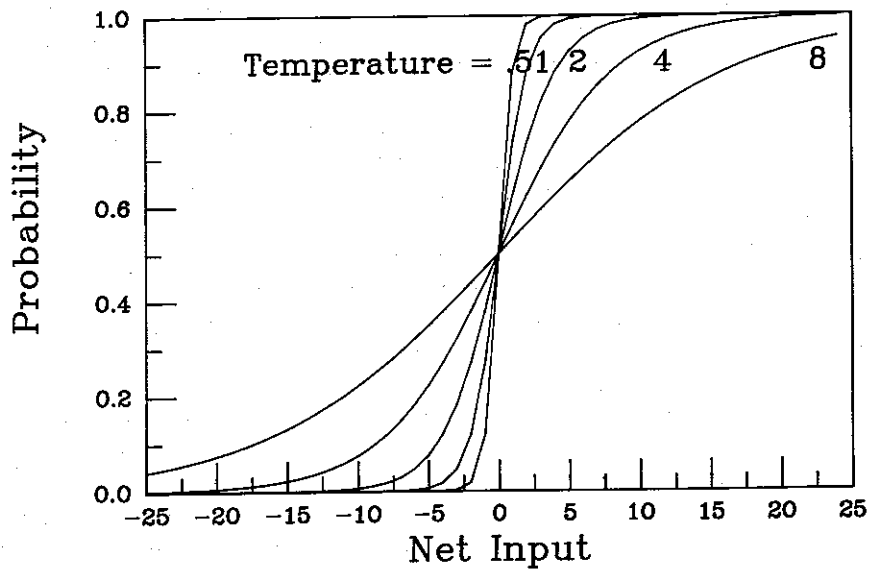


FIGURE 6. Probability of attaining value 1 as a function of the distance of the input of the unit from threshold. The function is plotted for several values of T.

Whenever it is below threshold, the probability that it will turn off is greater than 1/2. The temperature simply determines the range of uncertainty as to whether it will turn on or off. This particular configuration of assumptions allows a formal analogy between these models and thermodynamics and allows the proof of theorems concerning its performance as a function of the temperature of the system. This is not the place to discuss these theorems in detail, suffice it to say that this system, like the BSB system, can be viewed as attaining states on the corners of a hypercube. There is a global measure of the degree to which each state of the system is consistent with its input. The system moves into those states that are maximally consistent with the input and with the internal constraints represented by the weights. It can be shown that as the temperature approaches 0, the probability that the system attains the maximally consistent state approaches 1. These results are discussed in some detail in Chapters 6 and 7.

There is a learning scheme associated with the Boltzmann machine which is somewhat more complex than the others. In this case, the learning events are divided into two phases. During one phase, a set of patterns is randomly presented to the visible units and the system is allowed to respond to each in turn. During this phase of learning, the system is environmentally driven; a simple Hebbian rule is assumed to

apply so that $\Delta w_{ij} = \eta a_i a_j$. Note, since activations take on values of 0 and 1 this says that the weight is incremented by an amount η whenever unit i and j are on, otherwise no change occurs. During the second phase of learning, the system is allowed to respond for an equal period of time in a so-called free-running state in which no inputs are presented. Since the system is stochastic, it will continue to respond even though no actual stimuli are presented. During this phase, a simple anti-Hebbian rule is employed, $\Delta w_{ij} = -\eta a_i a_j$. The intuition is roughly that the performance during the environmentally driven phase is determined by both the pattern of interconnections and by the environment. The performance during the free-running phase is determined only by the internal set of connections. To correctly reflect the environment, we should look at its performance due to the environment plus internal structure and then subtract out its performance due to internal structure alone. This is actually quite a powerful learning scheme. It can be shown that if a portion of the input units are turned on after the system has learned, it will complete the remaining portion of the visible units with the probability that those units had been present in the stimulus patterns given the subpattern that had been turned on. These issues are again addressed in Chapter 7.

Grossberg

Stephen Grossberg has been one of the major contributors to models of this class over the years. His work is complex and contains many important details which we cannot review here. We will instead describe some of the central aspects of his work and show how it relates to the general framework. Perhaps the clearest summary of Grossberg's work appears in Grossberg (1980). Grossberg's units are allowed to take on any real activation value between a minimum and a maximum value. The output function is, in many of Grossberg's applications, a threshold function so that a given unit will affect another unit only if its activation level is above its threshold. Moreover, Grossberg argues that the output function must be a *sigmoid* or S-shaped function of the activation value of the unit. Grossberg's activation rule is rather more complex than the others we have discussed thus far in that excitatory and inhibitory inputs don't simply sum, but appear separately in the activation rule. Grossberg has presented a number of possible activation rules, but they typically have the form

$$a_j(t+1) = a_j(t)(1-A) + (B-a_j(t))net_{ej}(t) - (a_j(t)+C)net_{ij}(t)$$

where A is the decay rate, B represents the maximal degree of excitation of the unit, and C is much smaller in magnitude than B and represents the maximal amount the unit can be inhibited below the resting value of 0. Grossberg generally assumes that the inhibitory inputs come from a kind of recurrent inhibitory field in which the unit is embedded and the excitatory inputs come from the unit itself and from another level of the system.

Grossberg has studied learning in these networks over a number of years and has studied several different learning schemes. The learning rule he has studied most, however, is similar to the one analyzed in Chapter 5 and is given by

$$\Delta w_{ij} = \eta a_i (o_j - w_{ij}).$$

Grossberg has applied this and similar learning rules in a number of cases, but a review of these applications is beyond the scope of the present discussion.

Interactive Activation Model

The interactive activation model of McClelland and Rumelhart (1981) and Rumelhart and McClelland (1982) had units which represented visual features, letters and words. Units could take on any value in the range $[min,max]$. The output function was a threshold function such that the output was 0 if the activation was below threshold and was equal to the difference of the activation value and the threshold if the activation was above threshold. The interactive activation model involves a connectivity pattern in which units are organized in layers, such that an element in a layer connects with excitatory connections with all elements in the layers above and below that are consistent with that unit, and connects negatively to all units in the layers above and below that are inconsistent with that unit. In addition, each unit inhibits all units in its own layer that are inconsistent with the unit in question. Thus, the interactive activation model is a kind of positive feedback system with maximum and minimum values for each unit, like the BSB model. The information coming into each unit is weighted (by the interconnection strengths) and summed algebraically to yield a "net input" to the unit. Let $net_j = \sum w_{ij} a_i$ be the net input to unit j. This net input is then combined with the previous activation

value to produce the new activation value according to the following activation rule:

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

where Θ is the decay rate of the activation given no input. In other words, the new activation value is given by the old activation value properly decayed, plus (or minus) a factor that pushes toward the minimum or maximum value depending on the magnitude of the net input into the unit. This activation rule is similar to that employed by Grossberg, except in this formulation the excitation and inhibition are algebraically combined.

The interactive activation model was designed as a model for a processing system and our goals were to show how we could account for specific aspects of word perception. Thus, there was no specific model of learning proposed to explain where the particular network we assumed came from. As we shall see, much of the work on learning reported in this book has been aimed at giving plausible accounts of how such a network might have been learned. (See especially Chapters 5 and 6.)

Feldman and Ballard

Feldman and Ballard (1982) have proposed a framework they call *connectionist modeling*. The units have continuous activation values, which they call *potential* which can take on any value in the range $[-10,10]$. Their output function is a kind of threshold function which is allowed to take on a small number of discrete integer values $(0 \leqslant o_i \leqslant 9)$. They have proposed a number of other unit types each with a somewhat different activation rule. Their simplest unit type is what they call the P-unit. In this case the activation rule is given by

$$a_j(t+1) = a_j(t) + \beta net_j(t).$$

Once the activation reaches its maximum or minimum value it is simply pinned to that value. Decay is implemented by self inhibition. Feldman and Ballard also have a *conjunctive* unit similar to our *sigma-pi* units described below. Feldman (1981) has also considered learning. In general, the approach to learning offers more machinery than is available within our current framework. In practice, however, the learning rules actually examined are of the same class we have already discussed.

SIGMA-PI UNITS

Before completing our section on a general framework, it should be mentioned that we have sometimes found it useful to postulate units that are more complex than those described up to this point in this chapter. In our descriptions thus far, we have assumed a simple additive unit in which the net input to the unit is given by $\sum w_{ij} a_i$. This is certainly the most common form in most of our models. Sometimes, however, we want multiplicative connections in which the output values of two (or possibly more) units are multiplied before entering into the sum. Such a multiplicative connection allows one unit to *gate* another. Thus, if one unit of a multiplicative pair is zero, the other member of the pair can have no effect, no matter how strong its output. On the other hand, if one unit of a pair has value 1, the output of the other is passed unchanged to the receiving unit. Figure 7 illustrates several such connections. In this case, the input to unit A is the weighted sum of the products of units B and C and units D and E. The pairs, BC and DE are called *conjuncts.* In this case we have conjuncts of size 2. In general, of course, the conjuncts could be of any size. We have no applications, however, which have required conjuncts larger than size 2. In general, then, we assume that the net input to a unit is given by the weighted sum of the products of a set of individual inputs. That is, the net input to a unit is given by $\sum w_{ij} \prod a_{i_1} a_{i_2} \cdots a_{i_k}$ where i indexes the conjuncts impinging on unit j and $u_{i_1}, u_{i_2}, \ldots, u_{i_k}$ are the k units in the conjunct. We call units such as these *sigma-pi units.*

In addition to their use as gates, sigma-pi units can be used to convert the output level of a unit into a signal that acts like a *weight* connecting two units. Thus, assume we have the pattern of connections illustrated in the figure. Assume further that the weights on those connections are all 1. In this case, we can use the output levels of units B and D to, in effect, set the weights from C to A and E to A respectively. Since, in general, it is the weights among the units that determine the behavior of the network, sigma-pi units allow for a dynamically programmable network in which the activation value of some units determine what another network can do.

In addition to its general usefulness in these cases, one might ask whether we might not sometime need still more complex patterns of interconnections. Interestingly, as described in Chapter 10, we will never be forced to develop any more complex interconnection type, since sigma-pi units are sufficient to mimic any function monotonic of its inputs.

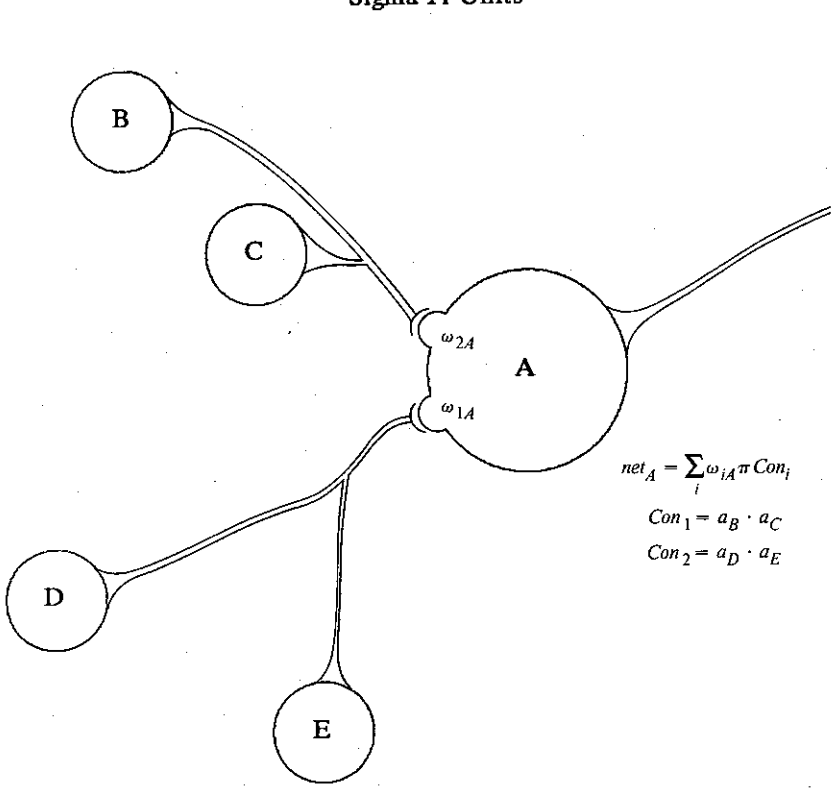


FIGURE 7. Two conjunctive inputs to unit A from the conjunct B and C and D and E. The input to unit A is the sum of the product of the outputs of units BC and DE.

CONCLUSION

We have provided a very general mathematical and conceptual framework within which we develop our models. This framework provides a language for expressing PDP models, and, though there is a lot of freedom within it, it is at least as constrained as most computational formalisms, such as production systems or high-level languages such as Lisp.

We must take note of the fact, however, that the framework does not specify *all* of the constraints we have imposed on ourselves in our model building efforts. For example, virtually any computing device, serial or parallel, can be described in the framework we have described here.

There is a further set of considerations which has guided our particular formulations. These further considerations arise from two sources: our beliefs about the nature of the hardware available for carrying out mental processes in the brain and our beliefs about the essential character of these mental processes themselves. We discuss below the additional constraints on our model building which arise from these two beliefs.

First, the operations in our models can be characterized as "neurally inspired." We wish to replace the "computer metaphor" as a model of mind with the "brain metaphor" as model of mind. This leads us to a number of considerations which further inform and constrain our model building efforts. Perhaps the most crucial of these is time. Neurons are remarkably slow relative to components in modern computers. Neurons operate in the time scale of milliseconds whereas computer components operate in the time scale of nanoseconds—a factor of 10^6 faster. This means that human processes that take on the order of a second or less can involve only a hundred or so time steps. Since most of the processes we have studied—perception, memory retrieval, speech processing, sentence comprehension, and the like—take about a second or so, it makes sense to impose what Feldman (1985) calls the "100-step program" constraint. That is, we seek explanations for these mental phenomena which do not require more than about a hundred elementary sequential operations. Given that the processes we seek to characterize are often quite complex and may involve consideration of large numbers of simultaneous constraints, our algorithms *must* involve considerable parallelism. Thus, although a serial computer could be created out of the kinds of components represented by our units, such an implementation would surely violate the 100-step program constraint for any but the simplest processes.

A second consideration differentiates our models from those inspired by the computer metaphor: that is, the constraint that all the knowledge is *in the connections.* From conventional programmable computers we are used to thinking of knowledge as being stored in the state of certain units in the system. In our systems we assume that only very short term storage can occur in the states of units; long term storage takes place in the connections among units. Indeed, it is the connections—or perhaps the rules for forming them through experience—which primarily differentiate one model from another. This is a profound difference between our approach and other more conventional approaches, for it means that almost all knowledge is *implicit* in the structure of the device that carries out the task rather than *explicit* in the states of units themselves. Knowledge is not directly accessible to interpretation by some separate processor, but it is built into the processor itself and directly determines the course of

processing. It is acquired through tuning of connections as these are used in processing, rather than formulated and stored as declarative facts.

In addition to these two neurally inspired working assumptions, there are a number of other constraints that derive rather directly from our understanding of the nature of neural information processing. These assumptions are discussed more fully in Chapter 4.

The second class of constraints arises from our beliefs about the nature of human information processing considered at a more abstract, computational level of analysis. We see the kinds of phenomena we have been studying as products of a kind of constraint satisfaction procedure in which a very large number of constraints act simultaneously to produce the behavior. Thus, we see most behavior not as the product of a single, separate component of the cognitive system, but as the product of large set of interacting components, each mutually constraining the others and contributing in its own way to the globally observable behavior of the system. It is very difficult to use serial algorithms to implement such a conception, but very natural to use highly parallel ones. These problems can often be characterized as *best match* or *optimization* problems. As Minsky and Papert (1969) have pointed out, it is very difficult to solve best match problems serially. However, this is precisely the kind of problem that is readily implemented using highly parallel algorithms of the kind we consider in this book. See Kanerva (1984) for a discussion of the best match problem and its solution with parallel processing systems.

To summarize, the PDP framework consists not only of a formal language, but a perspective on our models. Other qualitative and quantitative considerations arising from our understanding of brain processing and of human behavior combine with the formal system to form what might be viewed as an aesthetic for our model building enterprises. The remainder of our book is largely a study of this aesthetic in practice.

ACKNOWLEDGMENTS

This research was supported by Contract N00014-79-C-0323, NR 667-437 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 second author.