CHAPTER 7

Modeling Cognitive Processes: The Interactive Activation Model

In this chapter our goal is to consider the application of PDP modeling techniques to the task of accounting for human cognitive processes, as revealed through psychological experimentation. As our example for this, we've chosen the interactive activation model of word perception (McClelland & Rumelhart, 1981; Rumelhart & McClelland, 1982). This model exemplifies our approach to modeling psychological processes, and it is of tractable size for running on smaller machines.

BACKGROUND

Our initial interest in parallel distributed processing mechanisms grew out of an attempt to capture our ideas about continuous, interactive processes, particularly as they applied to the problems of visual word recognition and reading. Both of us had already done both experimental and theoretical work in this area, but without the benefit of simulations (see McClelland 1976, 1979; Rumelhart, 1977; Rumelhart & Siple, 1974).

Our primary aim was to account for contextual influences on perception. These influences have been described since psychologists first began to present visual or auditory stimuli under controlled conditions (Bagley, 1900; Cattell, 1886). Among the early observations was the fact that subjects could identify far more letters from a single brief flash if the letters fit together to form a word than if the letters made a random string. Context could override the sensory input too, as in the cases where subjects

reported the strong impression that they saw all the letters in the word *FOREVER* when in fact *FOYEVER* was shown (Pillsbury, 1897).

In early studies the experimenter relied on what is generally called a free report of the contents of the briefly displayed stimulus, and many researchers pointed out that there were serious methodological problems with this. It has often been pointed out that subjects might see as much in the two cases, but forget less or correctly guess more when the stimuli form words.

An experiment that controlled for both guessing and forgetting at once was carried out by Reicher (1969) and followed up by a number of investigators, including Wheeler (1970), Johnston and McClelland (1973, 1974, 1980; Johnston, 1978; McClelland & Johnston, 1977), and several others (Baron & Thurston, 1973; Manelis, 1974; Massaro, 1973; Massaro & Klitzke, 1979; Spoehr & Smith, 1975).

In Reicher's experiments, a target was presented (e.g., *E*) either in a word (e.g., *READ*), in a scrambled letter string (e.g., *AEDR*), or in isolation. The presentation was followed by a masking stimulus, which consisted of a jumbled array of letter parts, and a pair of letters, which was keyed to the position occupied by the target letter. One of the letters was the target letter itself, and the other was another letter that fit the context (if any) to make an item of the same type. For the displays *READ, AEDR*, and *E* in isolation, the pair could be *E* and *O*, presented with a row of dashes to indicate which display location was being tested:

```
  E
- - - -
  O
```

The subject's task was to choose which of the two letters had appeared in the indicated position. The target could appear in any of the four positions, and subjects did not know in advance which position would be tested on a given trial.

Reicher's test is called the *forced choice test*. Using this test, he found that subjects were more accurate when the letters occurred in words than in either of the other two conditions. This finding is called the *word superiority effect*.

Reicher's finding is important because it indicates that the advantage for words is not simply a matter of guessing letters that fit the context better from fragmentary cues. Rather, it appears that the perceptual system is better able to use the information in the display when the letters form a word with their context. The fact that the advantage holds for words over single letters makes it difficult to view the phenomenon as a result simply of forgetting, since a single letter surely places a very light load on memory.

Reicher's findings, backed up by a large literature of further experimental tests, seemed to us to be a very clear demonstration that context plays a role in perception. We therefore set out to model this phenomenon, basing our approach on a number of basic assumptions.

Basic Assumptions of the Interactive Activation Model

Here we describe each of the basic tenets of the interactive activation model and explain why we adopted each one.

Perception occurs in a multilevel processing system. This assumption is nearly ubiquitous, and so we will give it little discussion; surely there are separate levels of representation for visual features, for words, and for larger wholes such as sentences. For our model of the processing of individual words or strings of letters, we assumed that there are at least three levels: a visual feature level, a letter level, and a word level.

Deeper levels of processing are accessed via intermediate levels. This assumption has often seemed contentious, particularly with respect to visual word recognition. We have assumed that a letter level is interposed between the feature and the word level because words appear to be defined, not in terms of their particular visual configurations, but in terms of the sequences of letters that they contain. Thus *READ*, *read*, and *READ* are all recognizable as words, and letters in such stimuli are all perceived better than letters in unrelated context (e.g., the *E* in *READ* is perceived better than the *E* in *AEDR*; cf. Adams, 1979; McClelland, 1976). Thus it would appear that readers can use their knowledge of words to perceive sequences of letters, even if the visual configuration of the input is highly novel.

If, as we assume, access to the word level is via the letter level, then sequences of letters should be more effective as masks for words than sequences of feature bundles that do not form letters. This prediction was confirmed by Johnston and McClelland (1980).

Processing is interactive. By this we mean that processing involves the simultaneous consideration of both bottom-up input information and top-down knowledge-based constraints. Our principle reason for this belief was the well-known and ubiquitous role of contextual factors in perceptual processing already alluded to above. We take the role of word context in letter perception as one example of this kind of interactive processing. Models that captured the *outcome* of the simultaneous consideration of bottom-up and top-down information had been developed by others (particularly Morton, 1969), but we wished to embody this assumption in a dynamic processing model.

Information flow is continuous. At the time we began to consider interactive processing, the predominant view among psychologists working in perceptual information processing was that information processing occurred through a sequence of discrete steps. Each step took a certain amount of time and resulted in a discrete output. However, alternatives to

this view were developed during the course of the 1970s (cf. McClelland, 1979; Norman & Bobrow, 1976; Turvey, 1973). In fact, the utility of continuous information flow was pointed out quite early in the *pandemonium* model of Selfridge (1955), an early AI model designed to account for the role of context in letter recognition. For us, the assumption of continuity seemed to be required in order to capture contextual influences in word recognition (McClelland, 1976; Rumelhart, 1977). The reason is that if the word level is to influence processing at the letter level, then the letter level must be making information available to the word level before processing at the letter level is complete.

PDP models as a way of capturing these basic assumptions. We turned to PDP models because they provided a simple and direct way of making our basic assumptions about continuous, interactive processing explicit in a computational model. By assuming a processing unit for each possible hypothesis about the input at each of the three levels of processing, by allowing each unit to be working continually, updating its own activation and sending activation to other units, and by allowing units to influence each other via simple excitatory and inhibitory interactions, we found we were able to capture our basic assumptions in a simulation model and explore how well these assumptions could account for contextual influences in letter perception.

Central Questions

In developing the interactive activation model, there were several basic questions:

- Could we make a PDP embodiment of our basic assumptions account for the basic fact that word context facilitates letter perception, as established by Reicher and others?

- Could we account for the fact that subjects perceive letters in pronounceable nonwords (e.g., *REAT*) more accurately than letters in random or scrambled strings and, under some conditions, more accurately than single letters (Johnston & McClelland, 1973; Wheeler, 1970)?

- Could we apply the model to the large body of existing data and show that we could really account for the existing findings? The most important facts we considered were these: (a) The perceptual advantage for letters in words is shared with pronounceable nonwords; that is, letters in words and in pronounceable nonwords

show a sizeable advantage over single letters or letters in random strings. (b) Within pronounceable words and nonwords, there is no consistent advantage for strings containing frequent letter clusters (e.g., *PEEP* or *TEEP*) compared to those containing much less frequent letter clusters (e.g., *POET* or *HOET*). Though apparent letter-cluster effects are found in some studies (see Rumelhart & McClelland, 1982, Experiment 9), other studies did not show these effects (McClelland & Johnston, 1977). Our hope was to account for both patterns of results, based on detailed aspects of the particular materials used in different experiments. (c) For letters in words, under the visual conditions in which Reicher's word superiority effect was obtained, there is no advantage for letters occurring in contexts that strongly constrain the identity of the letter (e.g., the *C* in *CLUE*: only three letters make words in the context *_LUE*) compared to letters in context that exert much weaker constraints (e.g., the *C* in *CAKE*; 10 letters make words in the context *_AKE*; Johnston, 1978). Again, however, such effects do occur in other studies. Our hope was to use the model to understand and account for these differences.

- Could we account for a set of new findings from our own laboratory? These findings were based on the use of a new technique for visual presentation, in which the letters in a four-letter string could be started and ended at different times. We found (as reported in Rumelhart & McClelland, 1982) that subjects perceived a particular letter better when the other letters with which it occurred were presented for a longer time. This was true both when the letter formed a word with the context and when it formed a pronounceable nonword with the context, but not when the letter was embedded in a random-letter string.

The approach that we took in trying to answer these questions was to begin by trying to develop a model that produced the basic perceptual facilitation advantage for letters in words when compared to letters in nonword strings; this turned out to be one of the hardest parts of the project. We tried a number of variants on the basic activation equation, as well as a wide range of different parameter values before we developed enough understanding for what we were doing to find a combination of assumptions that worked. Once we accomplished this, we began to consider the list of phenomena we wanted to account for, working our way more or less down the list just given. Our goal was to find a single formulation, together with a single set of parameter values, that would allow us to give a fairly close account of the findings discussed earlier.

Two further aspects of our approach are worth mentioning. First, we endeavored to keep the model as simple as possible, within the constraint that we preserved sufficient structure to capture our basic assumptions. For

example, we used a highly simplified representation of the visual forms of letters, and we assumed that each letter in a visual display came rigidly channeled into one of four letter positions. Second, we did not attempt to obtain detailed quantitative fits of the model to the results of particular experiments. Rather, we attempted to come as close as we could to producing results that captured the major qualitative features of the phenomena.

We do not mean to suggest that detailed quantitative fits are not appropriate in many cases. Rather, we want to suggest that in this and many other cases, detailed quantitative fits may require very detailed and specific assumptions (for example, about the confusability of particular pairs of letters) that fall outside of the basic assumptions that are at issue. Attention to such assumptions may in certain instances interfere with the search for understanding of the basic principles that transcend such details. Under these circumstances, a simplified model may yield a more satisfying explanatory account, even when it is known to be wrong in some details, if it provides an explanation for the qualitative patterns observed in the empirical phenomena. (More discussion of this point may be found in Sejnowski's discussion of the role of computational models in *PDP:21*, pp. 387-389.)

We were also concerned with making sure we understood exactly what was going on in the model that allowed it to account for the phenomena. To this end, we spent a great deal of time studying the processing of individual examples so that the details of what was happening in the simulations would be clear. This kind of detailed study of individual items will be the focus of this chapter. In assessing the model, we supplemented this individual example approach by running large simulations with lists of items taken from the original experiments we were trying to understand. We omit these kinds of simulations here, although they played a central role in evaluating how well the model could account for the facts reported in particular experiments.

When we felt we had been reasonably successful in accounting for the phenomena that we wanted to account for, we began to consider whether there might not be additional experiments that we might do to test the principles underlying the approach. We were able to come up with one such experiment (Rumelhart & McClelland, 1982, Experiment 10). It will be described in more detail in one of the exercises.

THE INTERACTIVE ACTIVATION MODEL

In this section we describe the essential features of the interactive activation (IA) model. Some additional details may be found in McClelland and Rumelhart (1981).

Network Architecture

The model consists of units at each of three processing levels: the feature level, the letter level, and the word level (see Figure 1). At the feature level, there is a set of units that serves to detect features in each of four letter positions. Within each set, there is a unit for the *presence* of each of the line segments in the simple font used by Rumelhart and Siple (1974) (shown in Figure 2) and another unit for the *absence* of each such line segment. These units are said to be detectors for the different values (present, absent) of each of the possible line-segment features. This assumption allows the model to distinguish between not knowing whether a line segment is present (both units off) and knowing that a segment is not present (the absence unit on and the presence unit off). At the letter level, there are four sets of letter units, one for each position. Each set contains a unit for each of the 26 letters of the English alphabet. At the word level, there is a single set of detectors for each word in a list of 1179 four-letter words taken from the word list of Kucera and Francis (1967). The set

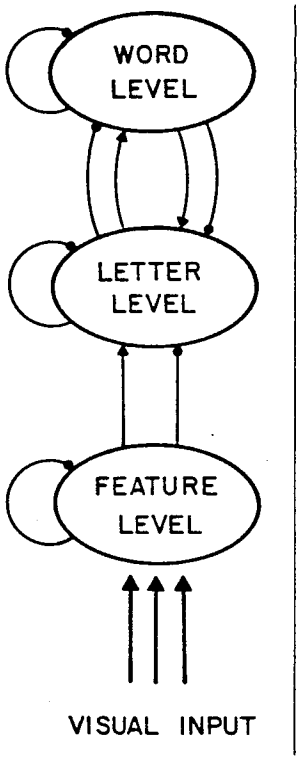


FIGURE 1. The basic architecture of the interactive activation model. (From "An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings" by J. L. McClelland and D. E. Rumelhart, 1981, *Psychological Review*, *88*, 375-407. Copyright 1981 by the American Psychological Association. Reprinted by permission.)

FIGURE 2. The Rumelhart-Siple letter font used by the interactive activation model. (From "Process of Recognizing Tachistoscopically Presented Words" by D. E. Rumelhart and P. Siple, 1974, *Psychological Review*, *81*, 99-118. Copyright 1974 by the American Psychological Association. Reprinted by permission.)

includes all the words with frequencies of two or more per million, excluding proper names, contractions, abbreviations, and foreign words that crept into the count.

Each unit can be taken as representing the hypothesis that the entity it stands for—feature, letter, or word—is present in the input. The activations of the units are monotonically related to the strengths of these hypotheses, according to a function that will be described later.

Connections

The connections among the units are intended to encode the mutual constraints among hypotheses about the possible contents of a four-letter display. The overall framework allows excitatory connections between units on different levels that are mutually consistent and allows inhibitory connections between units on different levels that are mutually inconsistent. For example, *T* in the first letter position is mutually consistent with all words beginning with *T*; these units therefore have mutually excitatory connections. To simplify matters, some of these connections are left out of the model: First, there are no feedback connections that are inhibitory. Second, there is no feedback at all from the letter level to the feature level. This leaves the following sets of between-level connections:

- *Feature-to-letter excitation.* Feature units have excitatory connections to all the letter units in the same spatial position that contain the feature. Thus, the presence unit for a horizontal bar at the top

of a letter position is connected to all the letters in this same position that have this feature.

- *Feature-to-letter inhibition.* Feature units have inhibitory connections to all of the letter units in the same spatial position that do not contain the feature. Thus each feature unit, when active, excites some of the letters and inhibits all the others.

- *Letter-to-word excitation.* Each letter unit has excitatory connections to each of the units standing for a word that contains the letter in the appropriate position. Thus the unit for *T* in the first position excites *TAKE* but not *CART*.

- *Letter-to-word inhibition.* Each letter unit has inhibitory connections to each of the units standing for a word that does not contain the letter in the appropriate position. Thus the unit for *T* in the first position inhibits all words that do not begin with *T*, including *CART*, *STOP*, and so on.

- *Word-to-letter excitation.* Each word unit has excitatory connections to the units for all of the letters in the word. Thus, there is an excitatory connection from *TAKE* to *T* in the first position, to *A* in the second, and so on.

In addition to these between-level connections, there are also within-level connections. These are exclusively inhibitory and are used to implement competition among mutually exclusive interpretations of the same portion of the input. All of the units for letters in the same spatial position are mutually inhibitory; and at the word level, all of the word units are mutually inhibitory.

The Concept of a Trial

In the IA model, we simulate trials from psychological experiments. A trial consists of a sequence of fields presented one after the other for processing. Before the trial, network activations are reset to baseline levels, and the cycle number is set to 0. The first field is presented at the beginning of cycle 1, followed by the second field, if any, at some later time, and so on. The last field is assumed to stay on until after a response occurs. Note that it is possible to present a field of blanks, if this is desired.

During processing, interactive activation processes govern the activations of units in the network. In addition, a readout process is assumed to be operating concurrently with the processing activity itself. The results of this readout process are assumed to be available for overt report.

After the processing of the sequence of fields has gone on for some chosen number of cycles, there may be a two-alternative forced-choice test of the kind used by Reicher and others; that is, the model may be confronted with a position cue and two letters, with the task of indicating which had actually occurred in the corresponding letter position.

Input to the Model

The inputs to the model—the contents of the fields presented during the processing trials—consist of specifications of which feature-level units should be turned on. This turning on may be deterministic (i.e., all features specified may be turned on) or it may be stochastic (specified features may be "detected" by the model with some probability). This probability can vary from field to field to allow for the possibility that the display is more or less legible in different cases.

Processing in the Model

Processing in the model occurs via the interactive activation and competition mechanisms discussed in Chapter 2. For simplicity, however, the feature units are treated as binary. Each feature unit is turned on or off by external input, and none of the feature units receive any input from any other units.

Letter and word units are activated according to the IAC equations given in Chapter 2. We show them again here for convenience. The net input to each unit is given by

$$net_i = \sum_j w_{ij} o_j. \tag{1}$$

Recall that o_j, the output of unit j, is equal to the extent to which the activation of the unit exceeds its threshold $(a_j - t_j)$, or 0 if $a_j \leqslant t_j$. The net input acts as a force that drives the activation of the unit upward if the net input is excitatory and downward if it is inhibitory:

If $(net_i > 0)$,
$$\Delta a_i = (max - a_i) netinput_i - decay\,(a_i - rest).$$

Otherwise,
$$\Delta a_i = (a_i - min) netinput_i - decay\,(a_i - rest).$$

Readout From the Model

Readout from the model is thought of as a separate process that integrates the activations of the units over time to assess the *response strength* of each unit. The response strength is defined to be the model's measure of the strength of the hypothesis that the entity a unit stands for is present in the input. The readout process chooses one of the units probabilistically, based on its response strength relative to the strength of other units.

The response strength of each unit is given by

$$s_i(t) = e^{k\bar{a}_i(t)} \tag{2}$$

where k is a scale factor and where $\bar{a}_i$, the running average of the activation of unit i, is given by

$$\bar{a}_i(t) = (orate)a_i(t) + (1 - orate)\bar{a}_i(t-1). \tag{3}$$

(Initially or when the model is reset, $\bar{a}_i(0) = a_i(0) = rest_i$, the resting activation of unit i.) Following Luce's (1959) choice model, the probability of choosing a particular item i as the response at time t is simply

$$p(r_{i,t}) = \frac{s_i(t)}{\sum_{j \in C} s_j(t)}. \tag{4}$$

This probability is called the *response probability* of response i at time t. Here, C is the set of competing alternatives (all letters in the same position for letter responses; all words for word responses), including unit i itself. Note that this response choice rule has the effect of ensuring that the response probabilities always sum to 1 for each set of competing alternatives.

The Forced-Choice Test

In simulating what happens in the forced-choice test, we first made the assumption that choices were based on responses read out from the letter level only. Under this assumption, the word superiority effect is due to the feedback from the word level to the letter level.

Because the choice alternatives appear after the target display has been replaced by the mask, we assumed that readout from the network had to occur without regard to the alternatives. That is, we assumed that the subjects did their best to identify all the letters in the target display following

the response choice assumptions described above, and only after doing so did they consult the forced-choice alternatives.

The determination of response probabilities in the model is complicated by the fact that the response strengths on which they are based rise and fall with time. Some assumptions must be made about the timing of readout. We assumed that subjects chose a time after onset of the target display that optimized their probability of choosing correctly. In practice, this means that readout occurs just before the onset of the postdisplay mask, if there is one. When no mask is used, readout is assumed to occur after activation and response strengths reach their asymptotic values.

Once readout has occurred, the rule for making choices is very simple: If the response letter chosen for the target position matches one of the alternatives, that alternative is chosen; otherwise, the choice is a random guess. From this rule, the probability of correctly choosing the target letter is the probability that this letter was chosen by the response readout process, plus 0.5 times the probability that neither the correct nor the incorrect alternative was chosen by the readout process.

Although the readout process is assumed to be stochastic, we do not actually simulate the probabilistic choice between alternatives. We calculate the probability of reading out the correct and incorrect alternatives and use these probabilities to calculate the probability correct in the forced choice.

Parameters

Here we discuss the parameters of the interactive activation model. There are parameters that influence the internal processing dynamics, parameters that reflect assumptions about the input, and parameters that influence the assignment of response probabilities during the readout process. The list of parameters of the model is rather long, but the majority are fixed at rather arbitrary values. The main parameters modified to capture the basic experimental effects we wished to account for are the excitatory and inhibitory strength parameters, *alpha* and *gamma*. However, in keeping with our philosophy of allowing users the greatest degree of control over the programs as possible, we have made all of the following parameters modifiable, as we shall see later.

alpha and *gamma*

The excitatory and inhibitory connection strength parameters, *alpha* and *gamma*, respectively, depend only on the processing levels of the units in question. This means, for example, that the strengths of the excitatory connections from feature units to letter units are the same for all such excitatory connections. The model has a separate parameter for feature-to-letter excitation, another for

letter-to-word excitation, and another for word-to-letter excitation, as well as parameters for feature-to-letter inhibition, letter-to-letter inhibition, letter-to-word inhibition, and word-to-word inhibition. In our simulations, these parameters were subject to tuning, with the goal of obtaining the best possible fit to the entire ensemble of experimental data we were considering. The values we settled on in this process are shown in Table 1.

decay

The model provides separate parameters for the rate of decay, at both the letter and the word levels. In practice, however, the decay parameters were both set to the same value (0.07) at a fairly early point in our simulations, and then were left at this value for the remainder of our experiments with the model. This value was chosen because it seemed to be the largest value that would nevertheless allow the model to settle smoothly to a target pattern of activation. With larger values the activations of units can start to oscillate wildly from cycle to cycle.

threshold

The model provides separate parameters for the output thresholds of units at the letter and word levels. In fact, at the word level there are separate threshold values for output to the letter level and for inhibitory output to other words. These thresholds were generally left at 0, except during our simulations of the contextual enhancement effect in pronounceable nonwords (see Ex. 7.4).

max and *min*

The model also provides separate parameters for the maximum and minimum activations units are allowed to have at the letter and word levels. We have always left the maximum activations at 1.0.

TABLE 1

DEFAULT VALUES FOR THE ALPHA AND GAMMA PARAMETERS USED IN THE IA MODEL

Excitation parameters (*alpha*):	
feature to letter	0.005
letter to word	0.07
word to letter	0.30
Inhibition parameters (*gamma*):	
feature to letter	0.15
letter to word	0.04
word to word	0.21
letter to letter	0.00

We did, however, experiment with different values for the minimum, settling on −0.20 for both the letter and the word levels.

rest and *fgain*

The model provides separate parameters for the resting activation levels of units at the letter and word levels. After some experimentation, the resting levels were generally left at 0. However, in the case of words, it should be noted that the value of the word-level resting activation parameter was not 0 for all units, but was offset downward from 0 depending on the word's frequency. The most frequent word known to the model was *that*, which was assigned a resting activation offset of 0.00. Other words were given resting activation offsets ranging between −0.92 and −0.01, according to a function that assigned offsets proportional to the log of the frequency of the word, subtracted from the log of the frequency of *that*. The model multiplies these offsets by the value of a frequency-scale parameter, *fgain*, and subtracts the result from 0. Throughout the simulations we used a value of 0.05 for *fgain*. Thus, the resting levels of words actually ranged from 0.00 to nearly −0.05.

oscale

This parameter corresponds to the parameter k in Equation 2. The model provides separate parameters for scaling the output strengths of units at the letter and word levels. A larger value of *oscale* is needed at the word level to compensate for the fact that there are more competitors at this level of processing, even though most of them usually are assigned highly negative activation values by the model. We recommend that the user keep the given values of 10.0 for letter-level output and 20.0 for word-level output.

fdprob

For each display field the user wishes to present, it is possible to set a separate value for the probability that features are detected from the input. By default, *fdprob* is set to 1.0, which means that all of the features of the input are detected. However, this parameter can be set to a lower value to simulate the effects of degraded visual presentation.

estr

Many experiments find that end letters are perceived more accurately than letters internal to a word. To accommodate this, we provide separate parameters for the strength of feature-level activations for each letter position. By default, these parameters are set to 1.0, and we recommend that users leave them at these values unless they specifically wish to explore these positional differences.

orate

This last parameter determines the rate of accumulation of activation for the purpose of determining response strength. Its default

value is 0.05. Generally, we have not found the value to be terribly critical, although we have generally assumed that it is small so that activations are translated into outputs only gradually.

Processing Under Different Visual Display Conditions

In our simulations, we tended to lump visual display conditions used in different experiments into two categories, based on the different conditions used by Johnston and McClelland (1973). One condition was called the *bright-target/pattern-mask* condition and the other was called the *dim-target/blank-mask* condition.

For the bright-target/pattern-mask condition we assumed that the target display was bright and clear enough so that all visual features of the display were detected and that the same applied to the patterned masking stimulus that followed the target. We further assumed that the effect of the mask was to quickly clear out the pattern of activation at the letter level, replacing it with a new pattern.

These assumptions led us to assume that the feature-to-letter inhibition was very strong, so that features of the mask would quickly inhibit letter activations that had been produced by the target display. A side effect of this was that no letters received net bottom-up excitatory input unless they were consistent with all of the features in a particular display position. As a result, under bright-target/pattern-mask conditions, only the letter actually displayed ever became activated on the basis of bottom-up information. Therefore, we found that we had little need for letter-to-letter inhibition. Consequently, though the model provides for the possibility of such inhibitory influences, we set the letter-to-letter inhibition parameter to 0.

Given that all the features of the display are detected, why is it that performance is less than perfect in the forced-choice test? The answer is simply that it takes time for activation to build up and be read out. The role of feedback is to enhance the activation of letters and, therefore, to increase the probability of correct read out from the letter level.

For the dim-target/blank-mask conditions, we assumed that the temporal brightness summation between the target and the blank mask operated so that the display was approximately equivalent to a very low-contrast, and hence degraded, input. In this situation, we assumed that visual feature information could only be detected imperfectly. The trial is simulated as a single display of letters with a feature detection probability considerably less than 1. The effect of this is that several letters generally are consistent with the detected features in each letter position. Under these conditions, the role of feedback from the word level is to selectively enhance the activations of units at the letter level that fit together with active letters in other positions to form words or to activate groups of words.

IMPLEMENTATION

Data Structures

The implementation of the interactive activation model in the **ia** program is similar to the implementation of the IAC model, although it differs from it in many details. One important difference is that the connections among the units are not in fact specified in a connection matrix. Instead, they are determined by table look-up. There are two relevant tables: the *word* table and the *uc* table.

The *word* table, as its name implies, contains a list of all of the words known to the model, stored as a sequence of four lowercase ASCII characters. To determine whether a particular letter unit should activate a particular word, the model checks to see if the letter is in the word in the appropriate position. We do not, of course, assume that activation in the mind is actually done by table look-up.

The *uc* table contains a list of all the features of the letters. The table is called *uc* to remind the user that the model only knows one alphabet and that is the uppercase Rumelhart-Siple alphabet shown in Figure 2. Each row of the *uc* table consists of fourteen 1s and 0s, indicating whether the corresponding character does or does not have the corresponding segment from the Rumelhart-Siple font in it. For example, the row corresponding to the letter *A* is

1,1,1,1,1,0,1,0,1,0,0,0,0

The exact arrangement of the features will be described when we explain the use of the program. The table also contains several special, nonletter characters, in addition to the uppercase letters. These will be described later.

The *uc* table is used both to specify the set of input features, given a display specification consisting of a sequence of letters entered by the user, and to determine which letter units should be activated when a particular feature unit is activated.

One other important data structure is the *trial* data structure. This is simply a list of field onset times and their contents. This information is specified via the *trial* command, which will be described in the section on using the program.

Processing

As in the **iac** program, processing is controlled by the *cycle* routine. Here is what it looks like:

```
cycle() {

  for (cyc = 0; cyc < ncycles; cyc++) {
    cycleno++;
    if (cycleno == ftime[tt]) {
      setinput();
    }
    interact();
    wupdate();
    lupdate();
    update_out_values();
    update_display();
  }
}
```

The *setinput* routine is called when each new field is scheduled for presentation; it first zeros the activations of all of the feature units, then sets them to the new values dictated by the new input. If *fdprob* (the probability of feature detection) for the present input is less than 1, then input units are turned on only if the value of a random number returned by the random number generator is less than the value of *fdprob*. The detected input feature activations are stored in an array called *dinput*, with indexes for the feature value (absent or present), the feature (0-13), and the position (0-3).

The *interact* routine is responsible for the excitatory and inhibitory interactions between units on different levels; as we shall see, the inhibitory interactions between units on the word and letter levels are handled in the *wupdate* and *lupdate* routines. The *interact* routine has three separate parts: one for the letter-to-word interactions, one for the word-to-letter interactions, and one for the feature-to-letter interactions. For completeness we show all three portions of the routine:

```
interact() {

/* letter -> word */
  for (pos = 0; pos < WLEN; pos++) {
    for (ln = 0; ln < NLET; ln++) {
      out = l[pos][ln] - t[LU];
      if (out > 0) {
        for (wn = 0; wn < NWORD; wn++) {
          if (ln == (word[wn][pos] - 'a'))
            ew[wn] += alpha[LU] * out;
          else
            iw[wn] += gamma[LU] * out;
        }
      }
    }
  }
```

```
/* word -> letter */
  for (wn = 0; wn < NWORD; wn++) {
    out = wa[wn] - t[WD];
    if (out > 0) {
      for (ln = 0; ln < NLET; ln++) {
        for (pos = 0; pos < WLEN; pos++) {
          if (ln == (word[wn][pos] - 'a'))
            el[pos][ln] += alpha[WD] * out;
        }
      }
    }
  }

/* feature -> letter */
  for (pos = 0; pos < WLEN; pos++) {
    for (fet = 0; fet < LLEN; fet++) {
      for (val = 0; val < NFET; val++) {
        out = dinput[val][fet][pos];
        if (out > 0) {
          for (ln = 0; ln < NLET; ln++) {
            /* 0th line of table is 'A' */
            if (val == uc[ln + 'A'][fet])
              el[pos][ln] += estr[pos]*alpha[FU]*out;
            else
              il[pos][ln] += estr[pos]*gamma[FU]*out;
          }
        }
      }
    }
  }
}
```

The letter-to-word and word-to-letter portions are quite similar; we discuss just the first of these. In the letter-to-word portion, the model cycles through all the letters in each letter position. If the output of the letter unit (that is, its activation minus the letter-level threshold) is greater than 0, then the program searches through all of the words. For each word, if the word contains the letter in question in the appropriate position, then the excitation of the word is increased by the output of the unit times the letter-to-word excitation parameter; otherwise, the inhibition of the word is increased by the output of the unit times the letter-to-word inhibition parameter.

The feature-to-letter portion of *interact* is a bit different. Here, the program cycles through each of the 14 features for each letter position, first checking the absence unit (indexed by $val = 0$) for that feature, then checking the presence unit (indexed by $val = 1$). For each such unit, if it is on, the program scans through the letter table, incrementing the excitatory

input to the letter units that have this feature and incrementing the inhibitory input to letter units that do not have this feature.

The Update Routines

The two update routines, *wupdate* and *lupdate*, are nearly identical; we will go through *wupdate* because it is slightly simpler—it does not have to loop separately through each of the four pools of letter-level units. The routine is as follows:

```
wupdate() {
  ss = sum;
  prsum = sum = 0;
  tally = 0;

  for (i = 0; i < NWORD; i++) {
/* word -> word inhibition */
    if (wa[i] > t[W])
      iw[i] += g[W] * (ss - (wa[i] - t[W]));
    else
      iw[i] += g[W] * ss;
/* now compute net input and update */
    net = ew[i] - iw[i];
    if (net > 0)
      effect = (max[W] - wa[i]) * net;
    else
      effect = (wa[i] - min[W]) * net;
    wa[i] += effect - decay[W]*(wa[i] - wr[i]);
    if (wa[i] > 0) {
      if (wa[i] > max[W]) wa[i] = max[W];
    }
    else {
      if (wa[i] < min[W]) wa[i] = min[W];
    }
    if (wa[i] > t[W])
      sum += wa[i] - t[W];
/* take running average for readout */
    ow[i] = ow[i] * (1 - outrate) + wa[i] * outrate;
/* then zero arrays for next cycle */
    ew[i] = iw[i] = 0;
  }
}
```

The only thing that is different about this routine compared to the *update* routine in the **iac** program is that the inhibition is handled in a way that is more efficient. Since each word unit inhibits every other word unit, the inhibitory input to a word unit *i* can be determined from the summed

outputs of all the word units, less the output of word unit *i*. The inhibitory input to word unit *i* is then simply this difference times the word-level inhibition parameter *gamma[W]*.

On each sweep through the *wupdate* routine, the summed output of the word units from the previous cycle is used to determine the inhibitory input to each unit for the current cycle. At the same time, the output of all of the word units is accumulated for use on the next pass through the update routine.

RUNNING THE PROGRAM

The use of the **ia** program is much like the use of the other programs described in this book. Its main differences are the way inputs are indicated to the model and there are more parameters and more units than in most other models.

Trial and Forced-Choice Specifications

Trials are specified by entering the *trial* command to the **ia:** prompt; details for using this command are given in the "New Commands" section. A separate command, called *fcspec*, is used to specify the position and alternatives to be tested in the forced choice.

Screen Displays

The program has far too many units to display all their activation values on the screen at once. To compensate for this, the program keeps track of summary information about the activations in the network, as well as a *display list* of units for display to the screen.

The summary information consists of the current cycle number, the number of active units at the word level and in each letter position at the letter level, and the summed activation of all the active units at the word level and in each position at the letter level. An active unit is defined to be a unit whose activation is greater than 0.

The display list is a list of units whose activations are to be displayed. This list can be specified by the user, using the *get/ dlist* command. Alternatively, the program will compute its own display list at the end of each processing cycle. In this case, the display list consists of all of the units

whose activation exceeds the values of the *dthresh/ word* and *dthresh/ letter* parameters, up to 15 letters in each position and up to 30 words.

New Commands

The **ia** program introduces only a small number of new commands:

fcspec

Allows the user to specify a forced-choice test for the simulation, much as Reicher did in his experiments. The command first prompts for a position (enter 0, 1, 2, or 3) followed by a pair of letters, the first of which is the correct alternative and the second of which is the incorrect alternative. When a pair of choice alternatives has been specified, at the end of each cycle, the program will compute the probability that each would be chosen in a two-alternative forced-choice test.

print

Allows the user to inspect the activations of each of the letter and word units and to inspect the response probability values associated with each letter unit. The command responds with a **print words?** prompt. If the user enters a string beginning with *y*, the program prints a screen full of words and their activations, and then presents the **p to push /b to break /<cr> to continue:** prompt. Responses of *p* and *b* have the usual effects; *return* causes the next screen of words and activations to appear. After finishing with the words, the program then prompts **print letters?** If the answer is yes, the activations of all of the letter units are presented for all four positions. Next, the command prompts **print letter resp-probs?** If the answer to this is yes, then for each letter, the probability that the letter would be given as the model's response in each position is displayed. Since this display would be overwritten by the top-level menu, the program prompts for a *return* before returning to the top level.

trial

Allows the user to specify a sequence of *fields*, each containing an onset time and a field specifications. After the command is entered, the program presents a series of prompts of the form:

field #N: time:

Here *N* is the ordinal number of the field. To the first such prompt, the user enters the time for field 0, which is usually cycle 1, followed by the field specification. (A prompt for the field specification is provided if the cycle number is followed by a

return.) The program then prompts for the next time-specification pair. When all the desired fields have been specified, enter *end* or type an extra *return*. Note that the times must be strictly increasing and that the time for field 0 must be 1 or greater. If the time for field *N* is less than or equal to the time for any preceding field, the program will never move on to field *N*. The last field encountered is just left on indefinitely, as is the case in most experiments. The field specification itself consists of a sequence of four characters. Allowable characters, and their meanings, are as follows:

Letters Letters are translated into the feature specifications of the corresponding uppercase letter as found in the Rumelhart-Siple font. Letters may be entered either in uppercase or lowercase.

\# Specifies the mask character, which is formed by turning on all the features that are present in either an *X* or an *O*.

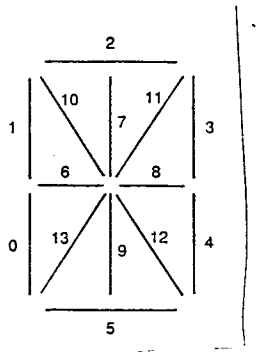
? Question mark requests a random feature array; the random number generator is consulted to determine whether, for each feature, the presence or the absence unit should be on.

_ The underscore character is used for blanks; it requests that neither presence nor absence units be turned on in the corresponding position.

. The period character turns on all of the absence features and turns off all of the presence features; not to be confused with "_" above.

* The asterisk sets up the input array with the set of features common to the letters *K* and *R*, leaving both the presence and absence units off for the features that distinguish these two characters in the Rumelhart-Siple font.

" The double quote character informs the program that you wish to specify exactly which presence and which absence units should be turned on manually. You are prompted to supply these specifications immediately after entering the specification string containing this character. You are first prompted to specify which presence units should be turned on, then prompted to specify which absence units should be turned on. Each specification is given as an uninterrupted string of 0s and 1s. The indexes of the features in the Rumelhart-Siple font are as follows:



Thus, the letter *R* is designated by the presence specification:

11110010100010

and the absence specification:

00001101011101

disp/ opt/ lthresh
disp/ opt/ wthresh

Allow the user to set the minimum activation required for letters and words to be entered in the display list when the list is being generated automatically by the program; these variables have no effect when the display list is entered manually using the *get/ dlist* command. These commands can be useful when a large number of units are activated, particularly at the word level, because of the length limitations of the *dlists*.

get/ dlist

Allows the user to specify the display list or to clear one if a list has previously been specified. In response to this command, the program prompts:

enter words or – for dynamic specification:

Typing a "–" at this point will clear the old specification, if any, and return to the command level; if this is done, the program will construct a display list for each trial, as it does before a *dlist* is specified by the user. Entering *end* or typing *return* will set up a specification that specifies no words; any other strings are taken to be words. The program looks up the string in the list of word-unit names, and if the string is found, it is placed on the word display list. The program then prompts for subsequent words. The "–" option is no longer available at this point, but *end* or *return* can be given to end

the list of words. After the end of the word list, the program prompts for letters for each of the four letter positions, starting with position 0. The list of letters for each position is terminated by *end* or an extra *return*. Due to limitations of screen space, the display lists are limited to 30 words and 15 letters per position.

Variables

The **ia** program has no configuration or environment variables; it has only one state variable, *cycleno*, and one mode variable, *comprp*. However, it has a large number of parameters. In fact, the parameters tend to be organized into parameter arrays, consisting of from two to as many as seven separately specifiable parameter values. For example, there are three different between-level excitation parameters. These are called *alpha/ f–>l, alpha/ l–>w,* and *alpha/ w–>l*, for feature-to-letter, letter-to-word, and word-to-letter excitation, respectively. All of the *alpha* parameters are grouped together under the specifier *alpha*, under *set/ param/*. Thus, to set *alpha/ l–>w* to 0.01, you would enter

set param alpha l–>w 0.01

or, more compactly,

se p a l 0.01

The following list indicates all of the new or changed variables accessible through the *set* and *exam* commands. When there is a parameter array, the array name is followed by a slash and the names of the members are given in {} after the slash.

stepsize
: Determines the frequency of screen updating and/or pausing as in other programs. Available values are *cycle* and *ncycles*; by default the value is *cycle*.

mode/ comprp
: When set to 1, enables the computation of response probabilities for letter units. When set to 2, enables computation of response probabilities for word units as well. The default value is 1 since, typically, responses are assumed to be read from the letter level.

param/ alpha/ {f–>l, l–>w, w–>l}
: Feature-to-letter, letter-to-word, and word-to-letter excitation parameters.

param/ beta/ {letter, word}
: Decay parameters for units at the letter and word levels.

param/ estr/ {p0-p3}
Relative strength of the external input in each letter position. These are set to 1.0 by default.

param/ fdprob/ {f0-f6}
For each field (f0-f6) specified in the trial command, the probability that each feature specified in the field specification is in fact detected. These are set to 1.0 by default.

param/ fgain
The scale factor for setting frequency-based resting activation levels for word units; larger values accentuate differences in resting levels as a function of word frequency. Defaults to 0.05.

param/ gamma/ {f->l, l->l, l->w, w->l, w->w}
Inhibition parameters for feature-to-letter, letter-to-letter, letter-to-word, word-to-letter, and word-to-word connections.

param/ max/ {letter, word}
Maximum activation parameters for letter and word units.

param/ min/ {letter, word}
Minimum activation parameters for letter and word units.

param/ orate
The rate of accumulation of the time-averaged activation used in computing response strengths.

param/ oscale/ {letter, word}
Multipliers used in scaling the time-averaged activations of units to obtain response strengths. Larger values produce larger differences in probability for a given difference in degree of activation.

param/ rest/ {letter, word}
Resting activation levels for letter-level and word-level units. For word-level units, this is the base resting activation; frequency-based offsets are multiplied by the *fgain* parameter and then subtracted from the base to obtain the true resting level.

param/ thresh/ {letter, w->l, w->w}
Output thresholds for letter-level and word-level units. The activation a unit passes to other units is equal to its activation minus its threshold, or 0, whichever is larger. For word units, there are separate values for outputs from words to other words and for output from words to letters.

OVERVIEW OF EXERCISES

The exercises we will propose here allow you to explore the behavior of the interactive activation model, using selected example stimuli. Generally, we have chosen the same items that were used in the examples described in McClelland and Rumelhart (1981) and Rumelhart and McClelland (1982).

Ex. 7.1. Using Context to Identify an Ambiguous Character

The first exercise gives you the opportunity to simulate the role of word-to-letter feedback in resolving ambiguity in visual input using the *R-K* example from McClelland and Rumelhart (1981). To run this exercise, get into the working directory you have set up for the **ia** program and enter

ia ia.tem ia.str

This sets up the display, and it results in the presentation of the **ia:** prompt along with a list of available menu options.

To set up the program to run this example, you must use the *trial* command to indicate the input to the program. To display the string *WOR** where the * stands for a character that is ambiguous between *R* and *K* enter:

trial 1 WOR end*

This indicates that at the beginning of cycle 1, the display designated by the string *WOR** should be presented. The word *end* indicates that no further fields are to be specified. Thus the display is turned on and left on indefinitely by this *trial* command. If you just enter *trial* by itself, the program will prompt you, first for a *field time* and then for *field contents*. Alternatively, you can enter everything on a single line as we have shown. (As in other programs, the *end* can be replaced with an extra *return*.)

Note that the display specification given here consists of uppercase letters and an asterisk. Lowercase letters may also be entered; they are treated as equivalent to uppercase. The "*" is one of several characters that have special meaning to the *trial* command, as described above; this one is specific to this particular example since it specifies the ambiguous *R-K* character, as shown in Figure 3.

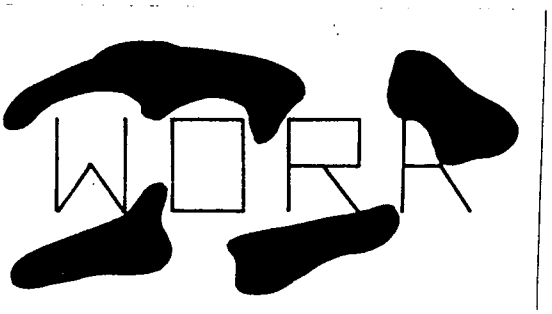


FIGURE 3. The inputs specified by the field specification *WOR**. (From "An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings" by J. L. McClelland and D. E. Rumelhart, 1981, *Psychological Review*, *88*, 375-407. Copyright 1981 by the American Psychological Association. Reprinted by permission.)

Now that you have specified a display, you can begin the process of simulating the perception of the display. Simply enter *cycle*. The program will run for 10 cycles, since this is the number of cycles specified in the *ia.str* file.

The display that you will see at the end of 10 cycles is shown in Figure 4. At the top of the display area (below the command information) the current cycle number is shown, followed by the number of active units and their summed activation for each pool of letter units and for the word units. Below this information are three fields: two smaller ones, labeled TRIAL and CHOICE, and a larger, unlabeled one, which is used for displaying the activations of units on the *dlist*.

The *dlist* field displays the activations and response probabilities of units on the *dlist*. Since no fixed *dlist* has been specified, the units that are displayed are those whose activations are greater than the values of the display options *lthresh* (for letters) and *wthresh* (for words). There is a separate column for each letter position, and room for two columns for words, though in this instance only one of these is used. Within each of these columns there are three subcolumns. The first contains an identifier for each item, the second indicates its activation (in hundredths, so that 0.15 is given as 15 and 0.07 is given as 7), and the third gives the response probabilities.

An exception is made in the case of the word response probabilities. These take a tremendous amount of time to compute, and since for the moment we are focusing on context effects on letter perception and we are not concerned with word response probabilities, these are neither computed nor displayed. (If you wish to compute and display these probabilities, set the *comprp* mode variable to 2 and the *dlevel* to 3.)

```
ia:
disp/  exam/  get/  save/  set/  clear  cycle  do  fcspec  log  newstart  print
quit  reset  run  trial


cycle  10     letter 0   letter 1   letter 2   letter 3     word
              num    1   num    1   num    1   num    2   num      5
              sum   62   sum   62   sum   62   sum   71   sum     52
-------------------------------------------------------------------------------
  TRIAL    |  w  62 26   o  62 25   r  62 26   k  44 15   word  8
  1 WOR*   |                                   r  27 12   wore  6
           |                                              work 32
           |                                              worm  0
           |                                              worn  4
           |
           |
           |
-----------|
 CHOICE    |
pos      0 |
-     0  0 |
-     0  0 |
fca      0 |
```

FIGURE 4. The state of the **ia** program after 10 cycles processing *WOR**.

Since both *lthresh* and *wthresh* default to 0, the items whose activations exceed 0 are displayed in alphabetical order until the available display space runs out; if the activation is between 0.00 and 0.01, a 0 is displayed. The display area allows 15 letters in each position and 30 words. Thus, if more than 30 words are activated (as indicated in the *word num* field in the upper portion of the display), only the activations of the 30 words that come earliest in the alphabet will be displayed in the *dlist* display. If this ever happens in a simulation, you might want to increase the value of *wthresh*, which is accessed by the *disp/ opt/ wthresh* command.

It is worth reiterating the meaning of the response probabilities. These numbers are computed using the formulas given earlier in the "Readout" section under the description of the model; they can increase or decrease through the course of processing. What these numbers represent at any given moment is the probability that the corresponding item *would* be chosen as a response, *if* readout were to occur at the present moment. Readout occurs at the end of processing when the display is degraded and is not followed by a mask; for trials in which the target display is followed by a mask, we assume readout occurs at the optimal time. This is fairly constant across different displays and is mostly affected by the timing of the onset of the mask field.

To the left of the main display area is the trial field. This simply indicates the onset times and contents of the fields that have been specified in the *trial* command. Below this is the choice field, which is not relevant in this exercise.

To summarize what can be seen on the screen after 10 cycles, only one letter is activated in each of positions 0, 1, and 2. In position 3 (the last position) the letters *k* and *r* are both active; *k* is beginning to get stronger than *r* because of feedback from the word level. At the word level, we can see that the words *word, wore, work, worm*, and *worn* are all active to some degree, although *work* is clearly most active.[1]

Now that the display has been explained, reinitialize the program (*reset* and *newstart* are equivalent for this exercise because there is no element of randomness), set *single* to 1, and step the program along, observing on each cycle the build-up of activation at the letter and word levels. Run about 40 cycles.

Q.7.1.1 When does *k* start to gain an edge over *r* in activation? Why? Even though *k* becomes more active, *r* is not suppressed. What parameter controls this? Explain why the response probability for *r* eventually does go down, even though its activation does not. Also explain why the response probabilities for *r* and *k* tend to pull apart more slowly than their activations.

---

[1] We will follow the convention of indicating visual displays and choice alternatives in uppercase and indicating unit names and activations in lowercase. Thus the display *WOR** activates the word unit *work*.

Ex. 7.2. Simulations of Basic Forced-Choice Results

In this exercise we will give you the opportunity to simulate some basic experimental phenomena that we considered in McClelland and Rumelhart (1981); in particular, we'll examine the forced-choice advantage of letters in words over letters in unrelated context and the fact that the word advantage holds for letters in pronounceable pseudowords as well as actual words.

First simulate the advantage of letters in words over letters in unrelated context, using as your example the letter *V* in *CAVE* compared to the letter *V* in *ZXVJ*. Assume that the target display is presented at the beginning of cycle 1, followed by a mask containing a sequence of mask characters, appearing at the beginning of cycle 16. Assume that the forced choice tests for the letter *V* in position 2 (counting from 0) and that the incorrect alternative is the letter *G*.

The trial specification for *CAVE* is

trial 1 CAVE 16 #### end

The string #### specifies a string of mask characters (each consisting of an *X* superimposed on an *O*). For the *ZXVJ* trial specification, simply replace *CAVE* with *ZXVJ*.

For this exercise, we introduce the *fcspec* command and the choice field of the display, which has thus far remained blank. To set up the choice field, you must specify a letter position to test and a pair of choice alternatives using the *fcspec* command. For this exercise, the position to test is position 2 (since we count from 0), the correct alternative is *V*, and the incorrect alternative is *G*, so the following command specification should be entered:

fcspec 2 V G

You will now see the choice field in the lower left-hand corner of the screen filled in with the information you have specified. This field contains:

- The letter position specified in *fcspec*, if any (defaults to 0).
- The correct alternative (or "–" if none has been specified), its activation, and its current choice probability.
- The incorrect alternative (or a "–" if none), its activation, and its current choice probability.
- The forced-choice accuracy (labeled *fca*) that would result if processing were terminated at this point.

The *current choice probability* is the probability that the alternative would be chosen if the forced choice were to be based on readout of letter activations on the current cycle. The *forced-choice accuracy* is the maximum value thus far attained for the current choice probability of the correct alternative. The value of this number at the end of the trial is taken to be the probability that the model would make the correct forced-choice response.

To run these examples, you will probably want to run about 20 cycles for each item. By cycle 16, the response probabilities will have peaked, so additional cycles will have no effect on the forced-choice accuracy, but it is sometimes interesting to watch letter- and word-level activations begin to drop off again as the mask takes effect.

For this simulation it is also useful to specify a *dlist* so that you can see the activation and response probability for the incorrect choice alternative as well as for the correct response. The file *cave.dli* is useful in this regard. It contains a *get/ dlist* command that specifies that the *dlist* should contain *cave* plus all the other words that the model knows that have three letters in common with *CAVE*, and that the letters displayed should be *C* in position 0, *A* in position 1, *V* and *G* in position 2, and *E* in position 3. You can leave the same *dlist* in place for the *ZXVJ* example since this one activates no words. This command file is read in by using the *do* command:

do cave.dli 1

(the 1 indicates that the commands in this file are to be executed just once).

Finally, the parameters you need to be aware of to understand the word-level activations are the *alpha* and *gamma* parameters given previously in Table 1.

Q.7.2.1. Describe the time course of activation of the unit for *V* in position 2 in each of the two cases, and explain how this is translated into the forced-choice advantage for *V* in *CAVE* over *V* in *ZXVJ*. Also, consider what happens at the word level. Why does the string *ZXVJ* produce no word-level activations, given the parameters of the model? Why do no word units other than *cave* achieve substantial activation values when *CAVE* is presented?

Next we consider the processing of *pseudowords*—pronounceable nonwords such as *MAVE* or *REAT*. Experimentally, subjects perform almost as well with these stimuli as with real words, and they show a considerable advantage over single letters and letters in unrelated strings (Baron & Thurston, 1973; Johnston & McClelland, 1977).

At first glance, we might not expect perceptual facilitation with pseudowords in the model, since it contains units only for real words. However, as we shall see, the model does in fact replicate the pattern of human performance. To see this, compare the model's performance on the *V* in

MAVE to the results you have already obtained with *CAVE* and *ZXVJ*. Keep everything else the same as before.

The file *mave.dli* contains a useful *dlist* specification for this simulation. It sets up the letters *m,a,v,e*, the alternative *g*, and all the words that have three letters in common with *MAVE*. Read this file in with the command *do mave.dli 1* followed by *reset*. During the simulation runs, study the pattern of activation at the letter and word levels.

Q.7.2.2. Explain why forced-choice accuracy for *V* in *MAVE* is almost as good as for *V* in *CAVE*.

Q.7.2.3. Study the word-level activations that are produced when the item *MAVE* is shown. See if you can explain why the initial advantage of *have* is amplified through the course of processing so that by cycle 15 it is considerably more strongly activated than *gave* and *save*. Also, try to explain why *move* shows less activation than *save*, even though they have about the same frequency and therefore start out at the same resting level.

Hints. The fact that *have* is more active than *gave* or *save* after 15 cycles is influenced by the resting level; however, your job is not to explain simply why *have* starts out a bit higher; you have to say why its advantage is amplified.

Q.7.2.4. Choosing another pronounceable nonword, study the pattern of activation it produces and see if you can observe word-level effects similar to those you observed with *MAVE*.

Ex. 7.3. Subtler Aspects of the Word Superiority Effect

This exercise allows you to go beyond the basic word and pseudoword superiority effects, going more deeply into the accounts offered by the interactive activation model for some of the detailed findings reported in the literature on letter and word perception. In both parts of the exercise you will see that it often requires a careful consideration of the inner workings of the model as well as the characteristics of particular experiments to understand why particular results were obtained. The model serves not only to explain what often looks like a confusing pattern of results, but also helps pinpoint which aspects of the experiments are responsible for the findings obtained.

Bigram frequency effects. One surprising finding in the word superiority effect literature is the apparent absence of effects of bigram frequency on the size of the word superiority effect obtained in the Reicher paradigm

(McClelland & Johnston, 1977). To test for an effect of bigram frequency, McClelland and Johnston examined forced-choice accuracy of letters in high bigram-frequency words (such as *PEEL-PEEP*) and pseudowords (such as *TEEL-TEEP*) and in low bigram-frequency words (*POET-POEM*) and pseudowords (*HOET-HOEM*). They found a slight advantage for words over pseudowords, but no effect of bigram frequency either for the word or for the pseudoword stimuli. These results are particularly surprising in view of the fact that other studies have found effects that can be interpreted as supporting the notion that items of higher bigram frequency lead to more accurate perception (Rumelhart & McClelland, 1982, Experiment 9). Here we explore what light the interactive activation model can shed on these findings.

To see what happens in the interactive activation model with stimuli of the type used by McClelland and Johnston, run the **ia** program with the items *TEEL* and *HOET*, testing the last position (position 3) in each case, with *P* as the incorrect alternative on *TEEL* and *M* as the incorrect alternative on *HOET*. Set up the trial specifications as in the previous exercise, so that the letter string is displayed on cycle 1 with the mask coming on cycle 16.

Q.7.3.1. Describe the results you obtain in these tests, and explain why the forced choice comes out actually slightly favoring the low bigram-frequency item.

As the examples used above illustrate, McClelland and Johnston made up their stimuli according to the following procedure. They first constructed a set of high bigram-frequency word pairs like *PEEL-PEEP* and a set of low bigram-frequency pairs like *POEM-POET*. Then they derived high and low bigram-frequency pseudoword pairs from the word pairs by changing one of the context letters from each pair. Thus *TEEL-TEEP* was derived from *PEEL-PEEP* and *HOET-HOEM* was derived from *POET-POEM*.

Q.7.3.2. Try to explain why McClelland and Johnston's procedure for constructing pseudoword pairs by changing nontarget letters in word pairs may have prevented them from finding a difference between high and low bigram-frequency items.

Hints. You might consider what would happen if the pseudowords *HOEM* and *HOET* were used, testing performance on the *H* rather than the final letter. You can use *L* as the forced choice alternative.

In their Experiment 9, Rumelhart and McClelland (1982) report results that look at first sight to be inconsistent with those of McClelland and Johnston (1977); that is, Rumelhart and McClelland found that forced-

choice performance on pseudowords did depend on a measure very closely related to bigram frequency. However, the materials used by Rumelhart and McClelland were not constructed from word pairs at all. Rather, all possible pseudoword items were constructed and separate sets of pairs were made from those having high and low letter-transition probabilities (a measure highly correlated with bigram frequency). For these items, both the experiment and the simulation obtained an advantage for the high transitional probability items.

Effects of contextual constraint. In the 1960s and 1970s, several studies were carried out examining the role of contextual constraint in word recognition. Contextual constraint refers to the degree to which the context a target letter occurs in restricts the possible identity of the target letter, based on which letters form words with the context. Thus *_LUE* is a relatively constraining context, because only three letters (*B*, *C*, and *G*) make words in this case;[2] whereas *_AKE* is much less constraining, since 10 letters can fit with it to make a word. The results of the experiments (most importantly, those of Broadbent and Gregory, 1968, and Johnston, 1978) presented a fairly complex picture. Effects of contextual constraints appear to be obtained under conditions in which subjects are shown a visually dim or degraded display with no patterned mask and are asked to give a free report. However, these effects do not appear when subjects receive a brief, masked presentation of the target word followed by a forced choice. The interactive activation model appears to explain this pattern of results.

This section describes how you can explore the account of these effects of contextual constraint using the **ia** program. Before we proceed further, however, we will note that these simulations are rather computationally intensive. Users who do not have a floating-point coprocessor and/or a lot of time may want to skip this section.

For these simulations, we will use the target words *CLUE* and *CAKE*. The forced choice (where applicable) will be applied to the letter *C*, with *B* serving as the incorrect alternative. The example is carefully chosen: *CLUE* and *CAKE* have about the same frequency and therefore the same resting level in the model. Forced-choice performance is modeled as before. For the free report, it is assumed that subjects read out their response from the word level, timing the readout to achieve optimal performance.

To begin examining the effects of contextual constraint, present the items *CLUE* and *CAKE*, using the same display conditions we have been using, in which the word is presented at the beginning of cycle 1 and is followed by the mask at cycle 16. Compare the forced-choice accuracy for both items, as well as the response probability for each target word at the word level. (To examine word-level response probabilities, you will need

---

[2] We leave out the low-frequency word *FLUE*, which is not in the word list used by the IA program.

to set the *comprp* mode variable to 2 and the *dlevel* variable to 3.) Note that response probabilities build up very slowly at the word level and depend on the persistence of activations of words beyond the onset of the mask, so you will want to run about 32 cycles of processing to get a good sense of the probability of identifying the target word based on readout from the word level.

Now repeat the above experiment, using display conditions intended to simulate a visually dim or degraded display with no patterned mask. To do this, use the command

set param fdprob f0 0.65

to set the probability of feature detection in field 0 of the trial sequence to 0.65. With this setting, the model will only detect 65% of the features in the display, simulating the effects of visual degradation. Now give the trial specification, simply presenting the target string at cycle 1. For example, for *CLUE* the specification would be

trial 1 CLUE end

Run several runs with each target, letting processing go on for 50 cycles so that word-level activations and response probabilities reach asymptotic levels. Use the *newstart* command to reinitialize between runs to obtain a new random set of features on each run.[3]

Q.7.3.3. Try to explain why effects of contextual constraint emerge in the free report with dim target conditions, and why they do not occur with the forced choice under bright-target/pattern-mask conditions.

Ex. 7.4. Further Experiments With the Interactive Activation Model

This set of exercises is based on material from Rumelhart and McClelland (1982). The first part explores a prediction of the model that was verified by an experiment reported in that paper. The second part applies the model to a variant of the word superiority effect called the *contextual enhancement effect*. The last part considers how expectations might

[3] Actually, after a few runs you will begin to get a feel for what happens with the response probabilities. To speed things up, you can set the *comprp* mode variable to 1 or even 0 (eliminating computation of letter-level response probabilities) and just watch the activations of the units.

influence processing. These exercises are rather specialized and are provided for readers with a particular interest in further exploration of the IA model and the phenomena of word perception.

Facilitation effects with unpronounceable nonwords. In Rumelhart and McClelland (1982), we predicted that the perceptual facilitation for letters in pronounceable nonwords should also occur with letters in a class of unpronounceable nonwords that were made up specifically to have several "friends"; that is, that were made up so that they had three letters in common with several words, even though they were not pronounceable. An example is *SLNT*, with a forced choice between *L* and *P*. As predicted, we found roughly the same forced-choice advantage for such items as for items like *SLET*, compared to random strings like *JLQX*.

In this exercise, explore the model's handling of the items *SLNT* and *SLET* compared to *JLQX*, using the display conditions of Ex. 7.2. Then consider the following question:

Q.7.4.1. Why does the model predict roughly equal facilitation for the *L* in *SLNT* as for the *L* in *SLET*? What characteristics of the items appear to be critical? Do simulations with other selected items to see if you are correct.

The contextual enhancement effect. Rumelhart and McClelland (1982) reported a phenomenon called the contextual enhancement effect. This is the finding that letters are perceived better when the *context* they are in is enhanced by presenting it for a longer time. This effect is demonstrated in experiments using trial sequences in which the context is presented before the target letter is shown, as in this trial specification:

```
 1 _HIP
13 SHIP
25 ####
```

Here the subject is given a forced choice between *S* and *W* for letter position 0. Forced-choice accuracy is greater when the context is presented before the full target field than in a control condition in which there is no preview of the context:

```
 1 SHIP
13 ####
```

The effect can be obtained both with words and with pronounceable nonwords like *SHIG*. In the experiments, the context boosted accuracy about 8% for both kinds of materials; there was an advantage for words, both with and without a preview of context, of about 4%.

For this exercise, simulate the context enhancement effect, with both *SHIP* and *SHIG*. That is, run trials with and without a preview for both *SHIP* and *SHIG*.

Q.7.4.2. Describe how well the model does at accounting for the contextual enhancement with words and pseudowords. See if you can explain the results that you observe.

In answering Q.7.4.2 you will have discovered that the model does not produce an enhancement effect with the pseudoword *SHIG* when the default parameters are in effect. This item is typical of most pseudowords. After considerable exploration, we were able to find a new set of parameters that did allow us to accommodate the context enhancement effect with pseudowords. The file *pwcee.par* contains commands that will produce the required modifications to the parameters. These commands can be executed using the command *do pwcee.par 1*. You may use these parameters, or (if you really want a challenge) you may come up with a set of your own that works. (As a hint, we will note that the modifications that are necessary have more to do with resting levels and thresholds than with excitation and inhibition parameters.)

Q.7.4.3. Using either the parameters in *pwcee.par* or your own parameters, simulate the context enhancement effect with pseudowords. Explain why the modifications work.

Expectation effects. Carr, Davidson, and Hawkins (1978) reported an interesting finding on the role of subjects' expectations in perceiving letters in words, pseudowords, and random-letter strings. They set up their subjects to expect either words, pseudowords, or random letters to be presented, and then, using a small number of trials of other types surreptitiously included in their materials, they were able to assess the advantage for words and pseudowords in each of these three expectation conditions. Their results are shown in Table 2.

Basically, what they found is that there was a word advantage over random letters, regardless of the subject's expectations, but that there was a pseudoword advantage over random letters only if the subjects were actually expecting pseudowords to be shown. Try to figure out for yourself whether the interactive activation model can account for these results, assuming that subjects are able to exert strategic control over one or more parameters of the model.

In simulations for this exercise, it is sufficient to use a single word, a single pseudoword, and a single random letter item to study these effects; for example, *CAVE*, *MAVE*, and *ZXVJ* will do. Use the standard parameters and set up trial specifications with the mask coming at cycle 16 as usual.

TABLE 2

EFFECT OF EXPECTED STIMULUS TYPE ON THE WORD AND PSEUDOWORD ADVANTAGE OVER RANDOM LETTERS

	Expectation		
Target	Word	Pseudoword	Random letters
Word	0.15	0.15	0.16
Pseudoword	0.03	0.11	−0.02

Note. From "Perceptual Flexibility in Word Recognition: Strategies Affect Orthographic Computation but not Lexical Access" by T. H. Carr, B. J. Davidson, and H. L. Hawkins, 1978, *Journal of Experimental Psychology: Human Perception and Performance, 4,* 674-690. Copyright 1978 by the American Psychological Association. Reprinted by permission.

Q.7.4.4 Experiment with different parameters of the model and see which ones can produce the effects of Carr et al. Describe each change you made, indicate how well it worked, and tell why it had the effects that it had.

Our answer to this question involved control over one of the excitation or inhibition parameters (we will not say which). Perhaps if you change other parameters you could obtain some or all of the same effects.