APPENDIX E

Answers to Questions in the Exercises

CHAPTER 2

Ex. 2.1. Retrieval and Generalization

Q.2.1.1. Some of Ken's properties are more strongly activated than others because several instance units other than the instance unit for Ken are partially active. These instance units send activation to their property units. In particular, the units for Nick and Neal are both reasonably active. They are active because they share *Single*, *Sharks*, and *HS* with Ken; they in turn support these properties of Ken, reinforcing their activation relative to others.

Q.2.1.2. The instance units for some of the individuals eventually receive considerable excitation because they are supported by several of the active property units. In contrast, the name units for these same individuals only receive excitation from one source—that is, the corresponding instance unit. The inhibition coming from Ken's name unit is stronger than the excitation any of the other name units is receiving.

Q.2.1.3. All of the active instance units have three properties in common with Ken. However, the six active instance nodes differ in the extent to which their properties are shared by the other active instance units. Nick and Neal share *Sharks*, *HS*, and *Single* with

Ken, whereas Earl and Rick share *Sharks*, *HS*, and *Burglar*, and Pete and Fred share *in20s*, *HS*, and *Single*. All the properties Nick and Neal share with Ken are also shared with two of the others, whereas Earl and Rick are the only individuals who share *Burglar* with Ken, and Pete and Fred are the only ones who share *in20s* with Ken. Because the properties that most of this group share receive more top-down support, the individuals who share them get more bottom-up activation. This effect is known as the "gang effect" (McClelland & Rumelhart, 1981) and is discussed again in Chapter 7.

Q.2.1.4. This is a result of hysteresis. The *in20s* unit became active as a result of the activation of the instance unit for Ken. At the same time other property units became active and these eventually began to activate other instance units, which in turn sent excitation to the *in30s* unit. By this point, however, the *in20s* unit was already active, so it continues to send inhibition to the *in30s* unit, keeping it from becoming as active as it would otherwise tend to become.

Q.2.1.5. Among the instance units, only four "competitors" of Ken are active; these are the ones that match one of the two properties of the probe and that share the *HS* and *Single* properties with Ken. Even though Earl and Rick share three properties with Ken, they are not active in this case. Among the property units, the properties shared by all the active instance units are more active than before, whereas the *Burglar* unit is less active; the other occupation units are now above threshold.

Q.2.1.6. In part because the probe directly excites them and in part because they agree on two out of the three properties they share with Ken, the instance units for Pete, Fred, Nick, and Neal are all quite active in this case. Since two of these support *Pusher* and two support *Bookie*, these two units receive more activation than in the previous run. Also, since in the first run the instance unit for Ken was the only instance unit active for some time, the *Burglar* unit was able to establish itself more strongly in the early cycles, thereby helping to keep the other property units from exceeding threshold.

Q.2.1.7. The noisy version of Ken "works" in this case because the probe is still closer to Ken than to any other instance known to the system. In other cases, distorting one property does not work because sometimes it produces a probe that is an equally good match to two or more instances, and sometimes it even produces

a probe that is a perfect match to a different individual from the one that matched the undistorted version of the probe.

Q.2.1.8. Success is due to the fact that several instance units that share several of Lance's properties also happen to share the same occupation. The model generalizes on the basis of similarity, and in this instance it happens to be right. Activation of the *Divorced* unit comes about because two of the instances that are similar to Lance are divorced, rather than married. Guilt by association.

Q.2.1.9. We leave this question to you.

Ex. 2.2. Effects of Changes in Parameter Values

Q.2.2.1. The effect of decreasing these four parameters is simply to slow processing by a factor of 2; it is as if time runs more slowly. One can easily show that the equilibrium activations of units should not be affected by proportional increases in all these parameters since they can be factored out of both the numerator and denominator of Equation 2 (p. 13). The effect of increasing these four parameters, however, is not so simple because it introduces abrupt transitions in the activations of units. Basically, what happens is that a large number of units (for example, at the instance level) suddenly receive enough bottom-up activation to become active. Since none were active before, they were not receiving any inhibition at this point. When they all suddenly become active, however, they all start sending each other inhibition. This causes them all to be shut off on the next cycle, even though all are still receiving bottom-up excitatory input. Since they are all off at this point there is no inhibition, so on the next cycle they all come on. This "ringing" effect eventually sets up an oscillatory pattern that destroys the information content of the pattern of activation.

Q.2.2.2. Increases in *gamma* limit activation of partially matching instances and also limit activation of partially supported properties. The result is a more precise and focused retrieval of the best match to the probe, with less influence from partial matches. In extreme cases this completely eliminates, for example, the "default assignment" process seen previously in the Lance example. Decreases in *gamma* allow more activation of partial matches; in the limit, these partial matches can swamp the best match, destroying the information content of the pattern of activation.

Ex. 2.3. Grossberg Variations

Q.2.3.1. The main point here is that in the standard update rule, the net input to the unit is computed regardless of the current activation of the unit; when the net input is positive it drives the activation of the unit up, and when it is negative it drives the activation of the unit down. In the Grossberg version, the net input is not computed as such; the magnitude of the upward force on the unit's activation (that is, the effect of the excitatory input) is scaled by the distance from the current activation to the maximum, while the effect of inhibitory input is scaled by the distance from the current activation to the minimum. In this regard it is useful to contrast the effect of equal excitatory and inhibitory input in the two cases. In the standard case, these cancel each other and so there is no change in the activation of the unit. In the Grossberg case, the effects are modulated by the current activation of the unit; the effects balance only when the activation is halfway between the maximum and minimum activations. In these simulations this means that the effects will balance at an activation of 0.4, halfway between 1.0 and −0.2. To compensate, one can increase *gamma*. For example, if *gamma* were set to 5 times *alpha*, then equal excitation and inhibition would have canceling effects when the unit's activation was exactly 0. Selecting such a value for *gamma* brings the two versions closer into line. However, this tends to result in lower overall activations, since as units become excited above 0, the power of their inhibitory inputs becomes much more potent.

One advantage of the Grossberg version is that it is in fact more stable; in general it appears that the standard version drifts away from the approximate equilibria we see at around 100 cycles. In contrast, the Grossberg version tends to find very stable fixed points.

CHAPTER 3

Ex. 3.1. The Necker Cube

Q.3.1.1. You should find that each of the two valid interpretations is reached about a third of the time, and local maxima are found on the other third. Of these, you should find some in which one side of cube A is on and the opposite side of cube B is on, and

some in which one edge of one of the cubes is on and the other three edges of the other cube are on.

Q.3.1.2. Generally, what happens is that local minima are reached when the first few units updated are on edges of the two cubes that do not interact. For example, in one minimum, the units on the right surface of cube A are on, together with those on the left surface of cube B. This minimum tends to be reached if units on these two surfaces become active early on. Note that units on these surfaces do not directly inhibit each other, but do inhibit other units in the other's cube. These inhibitory influences prevent either cube from completing itself. In one particular run, of the first four units updated, two were on the left surface of cube A and two were on the right surface of cube B. Four of the next five updates were on these two surfaces. At this point, the bottom-right edge has little activity in either cube, and could still go either way. From this point on, many of the units are being strongly enough inhibited by the activations that have been established that they do not get a chance to come on. Only the units in the two bottom-right edges remain in a situation where they would come on if updated. By chance, one of the units in the bottom right edge of cube A is updated before one of the units in the bottom right edge of cube B. This has the effect of making the input to units in the bottom right edge of cube B inhibitory, thereby determining the solution.

Q.3.1.3. Generally, the larger the value of *istr*, the lower the probability of finding one of the two global minima. There are two related reasons. First, units become active more quickly with larger values of *istr*; second, active units exert stronger effects on other units. This means that within a very small number of updates, activations can be set up in each of the two cubes that effectively block completion of the other cube. With smaller values of *istr*, activation is more gradual, and these kinds of blockage effects are less likely.

Q.3.1.4. Providing external input has an effect, but it is not completely overwhelming, for example, providing external input of strength 1.0 biases the outcome, but only slightly. In one set of 40 runs, with *Afll* receiving external input of 1.0, 15 runs settled to cube A and 15 settled to cube B. The bias toward cube A was only apparent in the mixed solutions. There were five runs with six units on from A, four with four units on from both A and B, and only one with six units on from B. Note that the external input is multiplied by the parameter *estr*, which means that external input of 1.0 only contributes 0.4 to the net input. If you turn on two

units in the same cube, there is a much more consistent shift toward outcomes favoring that cube. In one set of 40 runs, with *Afll* and *Abur* receiving external input of 1.0, 30 runs settled to cube A, and only 2 settled to cube B.

Ex. 3.2. Schemata for Rooms

Q.3.2.1. Somewhat surprisingly, the system tends to settle to almost exactly the same goodness maxima on different runs with the same unit or units clamped. Note that these maxima are not necessarily global maxima. For example, in the bathroom case, the maximum the network finds has window and drapes off; if they are on (and all else is the same) the goodness is greater. In all but one case, the maxima you should reach correspond to those shown on in Figure 6 in *PDP:14* (pp. 26-27); the only discrepancy is the case of the living room where the unit clamped on initially is *sofa*. The state illustrated in *PDP:14* tends to be reached around cycle 25, with a goodness of about 21.5; but sometime between cycle 25 and cycle 50, the network breaks out of this state and moves toward a higher maximum, with a goodness of very close to 27.0 (it may take a total of 75 cycles or so to converge to this point). This maximum seems to be a luxurious office or a living room with a study area in it.

Differences in the maxima reached are attributable to several factors, the most obvious of which is the number of units that are on. Thus the bathroom prototype, in which only nine units are on, has a goodness of only about 8.08, while the living room prototype, in which 21 units are on, has a goodness of about 27. Another factor, of course, is the pattern of the weights; these determine how much each active unit contributes to the overall goodness. We will leave it to you to discover the effects of clamping various single units and pairs of units.

Q.3.2.2. You will find that the office case gives the clearest example of a situation in which it is better to have both drapes or windows than it is to have either without the other. The bathroom prototype shows this same effect but much more weakly, and in the bedroom and the living room cases, the effect disappears; here having neither is worse than having either, and having both is by far the best. What is happening is that drapes and windows each have net positive input from other features of bedrooms and living rooms, so each contributes positively to the goodness, independently of the other. The cases in which the synergy appears are cases in which each separately gets net negative input from other

features that are active, but when the other is active, the balance shifts. You might also notice that drapes without windows generally seems worse than windows without drapes. Can you figure out why? Remember, connections are symmetrical in the schema model!

Ex. 3.5. Simulated Annealing in the Cube Example

Q.3.5.1. You should find a lower density of local maxima—our experience is that they occur only about 10% to 20% of the time. Also, interestingly, only local maxima in which four units are on in each cube seem to "last." That is, the system tends to move away from maxima in which six units are on in one cube and two are on in the other, even at the minimum temperature of 0.5. What would happen if the minimum temperature were reduced—would that have any effect on this tendency?

Q.3.5.2. Initial temperature is pretty much irrelevant in this case unless it is quite low. More important is the gradualness of the annealing. If it is abrupt, there is a very fast transition from approximately random behavior to virtually deterministic behavior; the results in this case (in their most extreme form) approximate the Hopfield net. With slow enough annealing, the network is guaranteed to settle to one of the highest goodness states.

Ex. 3.6. Electricity Problem Solving

Q.3.6.1. Although tne network occasionally seems to be stuck in a local maximum at 200 cycles, given the annealing schedule imposed in the *vir.str* file, our experience is that it will almost always eventually find the best answer if run for 300 or even 400 cycles. In this answer the following knowledge atoms are active:

V1 + V2 = VT:	*d-u-s*
R1 + R2 = RT:	*s-u-u*
I * R1 = V1:	*d-s-d*
I * R2 = V2:	*d-u-u*
I * RT = VT:	*d-u-u*

Other knowledge atoms will occasionally flicker on in this state, and occasionally the active ones will flicker off, but most of the time all of the active ones are on, along with one or two others.

Q.3.6.2. Yes, you can replicate Smolensky's findings, but the model is stochastic, so you may have to average over a number of runs to get reliable results. In general it is not possible to go by the results of one or two runs in these stochastic systems to reach your conclusions.

Q.3.6.3. In general, the network tends to settle first on parts of the answer that most directly depend on the units that have been clamped in the problem specification. Other parts of the answer that depend on these results cannot become firmly established until the conclusions that they depend on are themselves firmly established.

CHAPTER 4

Ex. 4.1. Generalization and Similarity With Hebbian Learning

Q.4.1.1. Each row of the weight matrix is a copy of the input pattern, scaled by the learning rate (0.125), and multiplied by the activation of the corresponding output unit, which is 1 or -1. Row 1, then, is just the input pattern times 0.125 since the corresponding output unit has activation 1. Row 2 is just the input pattern times -0.125 since the corresponding output unit has activation -1.

The columns of the weight matrix can be understood in a similar way. That is, each column is a copy of the output pattern, scaled by the learning rate, and multiplied by the activation of the corresponding input unit. Note that these facts follow directly from the Hebb rule, which states that the weight in row i, column j is equal to the activation of input unit j times output unit i times the learning rate.

Q.4.1.2. When a single association has been stored in a pattern associator, the output of the network is always a scalar multiple of the stored output pattern. The value of that scalar is equal to the normalized dot product of the test input with the stored input pattern. Thus, even when the test input seems very different from the stored input pattern, as long as the normalized dot product of the two is positive, the obtained output will simply be a scaled-down version of the stored output pattern. For example, the test input pattern

$$+ + + + + + - + +$$

has a normalized dot product of 0.25 with the stored input pattern

$$+ - + - + - + +$$

The output obtained with this test pattern is 0.25 times the stored output pattern. Thus the *ndp* of the test output with the stored output is 0.25, and the *nvl* of the obtained output pattern is also 0.25. The *vcor* of the test output with the stored output is 1.0, indicating that the vectors are identical up to a scalar multiple.

A special case of this arises when the test input is orthogonal to the stored input pattern. In this case, the normalized dot product of the test input with the stored input is 0, so the test output vector is 0 times the stored output vector; that is, it is a vector of 0s. When the test input is anticorrelated with the stored input, their normalized dot product is negative. When the stored and test inputs are perfectly anticorrelated, their normalized dot product will be -1.0, so the test output will be -1.0 times the stored output. Note in this case that the magnitude of the output (the *nvl*) is just as great as it is when the test input pattern is the same as the stored input pattern.

One useful way of thinking of what is going on in these tests is to note that the output vector can be thought of as a weighted sum of the columns of the weight matrix, where the weights associated with each column are determined by the activations of the corresponding input units. Each of these column vectors is simply equal to 0.125 times the output pattern times the sign the corresponding input unit had in the stored input pattern. When the test input pattern matches the stored pattern, what we end up with is the sum of eight column vectors, each equal to 0.125 times the stored output pattern. The result is 1.0 times the stored output pattern. Different input patterns simply weight the columns differently, but the result is always some scalar multiple of the basic output pattern column vector.

Ex. 4.2. Orthogonality, Linear Separability, and Learning

Q.4.2.1. Your orthogonal set of vectors will be perfectly learned by the net in the first epoch of training, so that during test, the *ndp*, *nvl*, and *vcor* are all 1.0. In vector terms, each test input produces an output pattern that is equal to the sum of the three stored output patterns, each weighted by the normalized dot product of the test input with corresponding stored input patterns. Since the input vectors form a mutually orthogonal set, input pattern k produces an output that is 1.0 times the corresponding output pattern k, plus 0.0 times each of the other output patterns. Additional epochs of training increase the strengths of the connection weights. One can think of this as a matter of increasing the number of copies of each association that are stored in the net.

Each copy contributes to the output. The result is that the *ndp* and *nvl* grow linearly, even though the *vcor* stays the same.

For the set of vectors that are not orthogonal, you should find that learning is not perfect. Instead, the output pattern produced by each test pattern will be "contaminated" by the other output patterns. The degree of contamination will depend on the similarity relations among the input patterns, as given by their normalized dot product, because the output produced by a test pattern is the sum of the outputs produced by each learned input pattern, each weighted by the normalized dot product of the learned input pattern with the test input pattern. Repeated training epochs will not change this. Though the magnitudes of the output vectors will grow with each training epoch, the *tall* command will show the same degree of contamination by other output patterns, as measured by the *vcor* variable.

Q.4.2.2. When the delta rule is used, the results in the first epoch for the orthogonal patterns are the same as with the Hebb rule. Thereafter, however, no further learning occurs because the learning that occurred in the first epoch reduced the error terms to 0 for all output units. For the linearly independent patterns, the benefit of the delta rule becomes clear: the *vcor*, *ndp*, and *nvl* measures will all eventually converge to 1.0. You will note that the last pattern pair in the training list always produces perfect results when you do a *tall*. This is because, given the value of the *lrate* parameter, the error for the current pattern is always completely reduced to 0. Because the changes that produce this effect generally interfere with previous patterns, however, the results are not so good for earlier patterns on the list. If you watch the *vcor* and *ndp* during training, you will note that they stay less than 1.0 for all of the patterns—unless, of course, the pattern set you are using contains a pattern that is orthogonal to both of the others in the set. The values gradually come closer to 1.0 as the changes to the weights get smaller and smaller.

Q.4.2.3. As we saw at the beginning of the chapter, each weight in the Hebb rule is proportional to the correlation between the corresponding input and output units over the set of patterns. You can retrieve these correlations by dividing each weight by the learning rate times the number of pattern pairs stored. In this instance, given that there are three patterns in the training set, each weight has a value of 0.375, 0.125, −0.125, or −0.125, corresponding to correlations of 1.0, 0.33, −0.33 and −1.0. With the delta rule, the weights will tend to preserve the same sign, although they do not have to. The magnitudes will not in general be the same. Basically, the way to think about what is happening is this. The magnitudes of the weights in each row adjust to

predict the output unit's activation correctly. If a particular output unit is perfectly correlated with only one of the input units over the ensemble of patterns, the corresponding weight will grow quite large. On the other hand, if a particular output unit is perfectly correlated with a number of the input units, they will come to "share the weight." In the patterns in the *li.pat* file, two units are perfectly correlated with output unit 0 and one is perfectly anticorrelated. These develop weights of +0.29 and −0.29, respectively; these weights actually are smaller than they would be in the Hebbian case. On the other hand, output unit 2 correlates perfectly with only one input unit (input unit 6, the next-to-last one) in these patterns. (Actually, the two units are anticorrelated, but that is equivalent to perfect correlation except a sign change, since one is perfectly predictable from the other.) Thus the weight to unit 2 from unit 6 must be much larger (−0.67) because it cannot share the burden of predicting output unit 2's activation with other input units.

Q.4.2.4. The orthogonal pattern is mastered first because the output it produces is not contaminated by any of the other output patterns.

Q.4.2.5. With Hebbian learning, nothing but correlations, scaled by the number of training trials, get stored in the weights. With the delta rule and *lrate* set at 0.125, the weights oscillate back and forth, falling into a pattern that suffices for each pattern pair in turn, but does not work for some or all of the others. If the learning rate is set to a smaller value, say 0.0125 rather than 0.125, the oscillations are smaller. The weights will average out to values that minimize the total sum of squares.

Ex. 4.3. Learning Central Tendencies

Q.4.3.1. Basically, what happens is that the expected value of each weight converges to the value that it achieved with noiseless patterns in one epoch of training. The higher the learning rate, the sooner the expected value converges, but the more variability there is around this expected value as a result of each new learning trial. Smaller learning rates lead to slower convergence but greater fidelity to the central tendency.

Ex. 4.4. Lawful Behavior

Q.4.4.1. In the particular run that we did, the weights at the end of 10 epochs were:

```
 36-28-16  0 -4 -4 -6 -2
-22 38-24 -2  0 -6 -6 -2
-26-22 38  0  0-10 -6 -4
 -2 -8  0 38-24-24 -6 -4
 -8-12  0-34 38-24 -6-14
 -8 -2 -4-22-26 34 -4-10
  0 -2 -4  0 -4 -2-40 34
 -4  4  0  4 -4  0 38-38
```

Given these values, the pattern

```
1 0 0 1 0 0 1 0
```

would produce a net input of 38 to the last output unit, and

```
1 0 0 1 0 0 0 1
```

would produce a net input of −38. Passing these values through the logistic function with temperature 15 produces a probability of .926 that the last output unit will come on in the first instance and a probability of .074 that it will come on in the second instance.

Q.4.4.2. A third of the time each input unit in each of the first two subgroups is on, a particular unit in each of the other subgroups should go on in the output. The other two-thirds of the time the other output unit should go off. Thus, the weights between input units in one of these subgroups and output units in the other tend to be decremented more often than they are incremented, and that is why the weights from units in the first subgroup (units 1-3) to units in the second subgroup (units 4-6) are negative. Each of the two output units in the 7-8 group, however, is on exactly half of the time each of the input units in the other two groups is on. Thus these weights tend to be incremented as often as they are decremented.

Q.4.4.3. The most probable response to *147* at this point will be *148*, although the tendency to make this response will generally be weaker than the tendency to make response *258* to *257*. The weights from units 1 and 4 to unit 7 should be tending to oppose the strong negative weight from input unit 7 to output unit 7. However, this tendency should not be very strong at this point. The weights that were previously involved in producing *147* from *147* have virtually disappeared. This can be seen by comparing the case in which the weights were not reset before training with the *all.pat* set to another run in which the weights were reset before training with the *all.pat* set. The resulting weight matrices

are virtually identical, except for random perturbations due to noise.

Q.4.4.4. It takes so long to get to this point for four reasons. First, the exceptional pattern, *147*, which is the hardest to master, only occurs once per epoch. Second, as the error rate goes down, the number of times the weights get changed also goes down because weight changes only occur when there are errors. Third, several other input patterns (particularly *247*, *347*, *157*, and *167*) tend to work against *147*. Finally, as the weights get bigger, each change in the weights has a decreasing effect on performance because of the nonlinearity of the logistic function.

CHAPTER 5

Ex. 5.1. The XOR Problem

Q.5.1.1. The value of *error* for a hidden unit is the sum of the *delta* terms for each of the units the hidden unit projects to, with each *delta* term weighted by the value of the weight from the hidden unit to the output unit. The *error* for unit 2 (the first hidden unit) is therefore

$$-0.146 \times 0.27 = -0.039.$$

The *delta* for a unit is just its *error* times the derivative of the activation of the unit with respect to its net input, or the activation of the unit times one minus the activation. Thus the *delta* term for unit 2 is

$$-0.039 \times 0.64 \times (1 - 0.64) = -0.009.$$

For unit 3, the *error* is

$$-0.146 \times 0.08 = -0.0117,$$

and the *delta* term is

$$-0.0117 \times 0.40 \times (1 - 0.40) = 0.0028.$$

The values are small in part because the weights from the hidden units to the output units are small. They are also affected by the fact that the *error* is multiplied by the derivative of the activation function with respect to the net input to the unit, first for the output unit, in computing its *delta*, then again for the hidden units, in computing their *delta*s from the *delta*s of the output units.

Each time the error derivative is "passed through" the derivative of the activation function, it is attenuated by at least a factor of 4, since for activations between 0 and 1, the maximum value of *activation* (1 − *activation*) is 0.25. The *delta*s for the hidden units have been subjected to this factor twice.

Q.5.1.2. The network produces an output of 0.60 for input pattern 0 and 0.61 for the rest of the patterns. The results are so similar because the small random starting weights have the effect of making the net input to each hidden unit vary in a rather narrow range (from −0.27 to 0.60 for unit 2 and from −0.44 to −0.36 for unit 3) across the four different patterns. The logistic function squashes these variations considerably since the slope of this function is only 0.25 at its steepest. Thus the activation of unit 2 only ranges from 0.43 to 0.64, while that of unit 3 stays very close to 0.40. This attenuation process repeats itself in the computation of the activation of the output unit from the activations of the hidden units.

Q.5.1.3. The bias term of the output unit and the weights from the hidden units to the output unit all got smaller. There were slight changes to the weights from the input units to the hidden units and in the bias terms for the hidden units but these are less important. Basically, what is happening is that the network has reduced the *tss* somewhat by making each pattern produce an output quite close to 0.50 rather than 0.62. The network achieved this by reducing the weights to the output unit and by reducing its bias term. Note that if the output were exactly 0.50 for each pattern, the *tss* would be 1.0. The *delta*s for the hidden units have gotten even smaller because of the smaller values of the weights from these units to the output unit. Since the gradient depends on these *delta* terms, we can expect learning to proceed very slowly over the next few epochs.

Q.5.1.4. The more responsive hidden unit, unit 2, will continue to change its incoming weights more rapidly than unit 3, in part because its *delta* is quite a bit larger due to the larger weight from this unit to the output unit. Another factor hindering progress in changing the weights associated with unit 3 is the fact that the weight error derivatives for weights both coming into and going out of this unit cancel each other out over the four patterns.

Q.5.1.5. Unit 2 is acting like an OR unit because it has fairly strong positive weights from each input unit, together with a bias near 0. When neither input is on, its activation is close to 0.5. When either unit comes on, its activation approaches 1.0. The other

hidden unit remains rather unresponsive, since the weights from the input units to this unit are very small. It is beginning to develop a negative weight to the output unit. Though it is not doing this very strongly, we can see that it is beginning to serve as a unit that inhibits the output unit in proportion to the number of active input units.

Q.5.1.6. Unit 3 is now in a linear range, allowing its activation to reflect the number of active input units. Because it is developing a strong negative weight to the output unit, the effect is to more and more strongly inhibit the output unit as the number of active input units increases. The other hidden unit no longer differentiates between one and two input units on, since its large positive weights from the input units cause its activation to be pushed against 1.0 whenever any of the input units are on. Thus the inhibition of the output unit by unit 3 increases as the number of active input units increases from one to two, and the excitation of the output unit by unit 2 does not increase.

Q.5.1.7. At epoch 240, the *tss* is mostly due to the fact that the network is treating all of the patterns with one or more hidden units on as roughly equivalent. All three patterns are producing activations greater than 0.5. The pattern in which both input units are on (pattern 3) is producing a larger *pss* than the other two of these patterns—it is exerting a powerful effect. In more detail, the weight from unit 3 to the output unit is pushed to be more positive in the two cases in which one input unit is on but is pushed to be more negative when both input units are on. The effects almost cancel but are slightly negative because the activation of unit 3 is larger when both input units are on and the *delta* for the output unit is larger. Meanwhile, unit 3's *delta* is negative in each of the two cases where one input unit is on and is positive when both input units are on. These almost cancel in their effects on the weight error derivatives for the weights from the input units to unit 3, but again the *delta* term for the pattern in which both input units are on dominates. (Remember that each input unit is on in only one of the two patterns in which only one input unit is on.) Thus the net force on the weight from each input unit to unit 3 is positive. As the weight from unit 3 to the output unit gets larger, *delta* for unit 3 gets larger, so the changes in the weights coming into unit 3 get larger and larger.

Q.5.1.8. One could almost write a book about the process of solving XOR, but the following captures most of the main things that happen. Initially, the network reduces the weights from each hidden unit to the output unit and reduces the bias term of the output unit.

This reduces the *tss* to very close to 1.0. At this point, learning slows to a near standstill because the weight error derivatives are quite small. Gradually, however, the hidden unit that is slightly more responsive to the input (unit 2) comes to act as an OR unit. During this phase the total sum of squares is being reduced because the network is moving toward a state in which the output is 0 for the (0 0) input pattern and is 0.67 for the other three patterns. (If it actually reached this point the *tss* would be 0.67.) At this point, the other hidden unit begins to come into play. The forces on this unit cause its input weights to increase and its weight to the output unit to decrease. It comes to inhibit the output unit if both input units are on, thereby overriding the effects of the other hidden unit in just this case.

There is a long phase during learning in which the weight error derivatives are very small, in part because the weights from the hidden units to the output units are small and in part because the hidden units are not very differentiated in their responses and so components of the the weight error derivatives contributed by each of the four patterns nearly cancel each other out. However, between epoch 140 and epoch 220 or so, the gradient seems to be pretty much in the same direction, as indicated by the *gcor* measure. Learning can be speeded considerably if the *lrate* is increased during this period. In fact with the particular set of starting weights in the *xor.str* file, learning can be speeded considerably by setting *lrate* to a much larger value from the beginning. In general, though, high values of *lrate* throughout can lead to local minima.

Ex. 5.4. A Cascaded XOR Network

Q.5.4.1. In cascade mode, the net input to each hidden unit and to the output unit builds up gradually over processing cycles. At first, the net input to all these units is strongly negative because of the negative bias terms that determine the resting levels of these units. When only one input unit it turned on, at first it produces a weaker input to each hidden unit than is produced when both input units are on. This means that unit 2 comes on more quickly when both input units are turned on than when only one is on. Since it is unit 2 that tends to excite the output unit, the activation of the output unit builds more rapidly at first when both input units are on. Note that early on, it makes little difference to unit 3 whether one or both inputs are on, since this unit's bias is so negative. Gradually, however, as processing continues, the activation of unit 2 begins to saturate whether one or both input

units are on. During this phase, if both input units are on, unit 3's activation will grow. Because of its inhibitory connection to the output unit, the net input to the output unit will gradually become negative in this case.

Ex. 5.5. The Shift Register

Q.5.5.1. In our experience, the network comes up with two different solutions about equally often. In one of these, the network acts as a normal shift register. Each unit has a strong excitatory connection to the unit to its right; otherwise the connections are usually nearly all inhibitory. In this instance, the pattern called *first output* is shifted one bit to the right from the pattern called *input*, and the pattern called *second output* is shifted an additional bit to the right. In the other solution, the network approximates what might be called a shift-and-invert operation at each step. Thus the first output pattern is shifted one bit right from the input pattern and inverted, so that 1s become 0s and 0s become 1s. This description holds only for the patterns in which one or two of the input units is on, however; for the other two patterns, (no units on or all units on) no inversion is performed. Interestingly, this second solution seems to take longer to reach, even though the two are found about equally often. When the constraint that the biases be negative is removed, the shift-and-invert solution is purer and easier to achieve, apparently at the same time making the network slower to converge on the simple shift solution. If the links between the weights are removed, there are a very large number of different solutions. Interestingly, it seems to take longer in general to solve the problem in this case. This is one of many examples in which constraining a **bp** network improves its ability to solve a particular problem.

Ex. 5.6. Plan-Dependent Sequence Generation

Q.5.6.1. The problem is very hard because the input patterns that the net must learn to respond differentially to do not start out very different. As learning proceeds, they gradually differentiate. As this occurs, the network must adjust itself to these changes. You can compare the "copy-back" version of this problem to the case in which the current input is supplied as part of the input by editing the *seq.pat* file so that the negative integers that cause the copy back are replaced by the values of the target from the previous line. In this case the problem is usually solved more quickly.

CHAPTER 6

Ex. 6.1. The Linear Hebbian Associator

Q.6.1.1. The receiving unit for the weight in row 3, column 0 is unit 3; the sending unit is unit 0. The values in row 3 reflect the product of the activation of the receiving unit, the activations of each of the sending units, and the *lrate*, which is 0.125. Since the activation of the receiving unit (unit 3) is −1.0, row 3 is −0.125 times the activation vector.

Q.6.1.2. The two learning patterns *a* and *b* are orthogonal to each other. The internal input to a receiving unit is equal to the dot product of the current activation vector with the vector of weights on the connections to that receiving unit. This vector is equal to *lrate* times pattern *a* times +1 or −1, depending on the particular receiving unit in pattern *a*. Regardless of the sign, the dot product of the current activation vector with the vector of weights is 0 because the two patterns are orthogonal.

Q.6.1.3. The weights in row 2 are 0.125 times pattern *a* plus −0.125 times pattern *b*. Patterns *a* and *b* differ in elements 1, 2, 4, and 7; *a* and $-b$ differ in elements 0, 3, 5, and 6. Row 2 has 0s in every element where the two base patterns agree, or where $a_i - b_i$ is 0.0. In other cells of this row, the values do not cancel out; they are either +0.25 or −0.25.

Q.6.1.4. On cycle 1, the external input is applied, so that the activations of the units at the end of this cycle reflect the external input alone. On cycle 2, these activations, together with the weights, produce an internal input that exactly reproduces the external input pattern. These two patterns together produce the resulting activation, which is 2 times the external input in each case. These results follow from the fact that the response to each external input vector (that is, the internal input generated by it) is equal to *k* (or *nunits* times *lrate*) times the sum of the stored vectors, each weighted by the dot product of itself with the external input.

Q.6.1.5. Before training with the set of four patterns, the two that have previously been learned will produce an internal input that matches the external input, and the two new ones will produce no internal input because they are orthogonal to the stored patterns. After one epoch of training, the two patterns that had previously been presented produce internal input equal to twice the external input, and the patterns that have only been presented once

produce internal input exactly equal to the external input. Essentially, each pattern in the set is an eigenvector of the network. The eigenvalue is equal to the number of presentations times k (or *nunits* times *lrate*). Regarding a negated vector that is -1 times a stored vector, any scalar times an eigenvector of a matrix is still an eigenvector of the matrix, so the magnitude of the response to the negated vector is equal to the magnitude of the response to the vector from which it was derived.

Q.6.1.6. For patterns that have been learned only once, the internal input at time cycle t is equal to the prior activation. When this is added to the continuing external input, the resulting new activation increases by *estr* times the external input. For patterns that have been learned twice, the internal input is twice the previous activation value so activations grow much more quickly. This explosive growth would continue indefinitely except that there is a built in check that stops processing as soon as the absolute value of the activation of any unit exceeds 10.0.

Q.6.1.7. The weight matrix that results from learning these patterns will be the identity matrix. These weights will produce an internal input that matches the current pattern of activation for any pattern of activation. In terms of the weights, this is because the weight from each unit to itself is 1 and the weight from each unit to every other unit is 0. In terms of the eigenvectors, any vector in n-dimensional space can be written as the weighted sum of any set of n orthogonal vectors in that space; such a set of vectors *spans* the space. Since the eigenvalue of each of these vectors is 1.0 for the matrix you have created in this exercise, the vector that comes out is the same weighted sum of these eigenvectors as the vector that went in.

Ex. 6.2. The Brain State in the Box

Q.6.2.1. Each of the trained patterns is an eigenvector of the weight matrix with eigenvalue 2.0, because each pattern was presented to the network twice during learning. At cycle 1, the external input sets the activation of each unit to *estr*, or 0.1 times the external input to each unit. Since the resulting pattern of activation is just 0.1 times the pattern presented, this pattern of activation is an eigenvector, and so, on cycle 2, the internal input pattern will be equal to 2 times the prior activation, or 0.2 times the external input. This product, plus 0.1 times the external input, results in an activation of 0.3 times the external input. On cycle 3, this new

activation pattern is again multiplied by 2 by the matrix, so that now the internal input is 0.6 times the external input; when 0.1 times the external input is added to this, the activation becomes 0.7 times the external input. On the next cycle, the doubling occurs again, but this time, 0.1 times the external input plus the internal input is 1.4 times the external input. This would mean that the activations of the individual elements would go outside the "box" imposed by the BSB model. The model does not allow this to happen; it "clips" the activations at +1.0 or −1.0. The next cycle generates an even larger internal input (2.0 times the external input pattern), but this is again clipped; from this point on, the internal input and activation do not change; the pattern of activation has reached a corner of the box.

Q.6.2.2. The pattern +−+−. . . . has an *ndp* of 0.5 with pattern *a* and an *ndp* of 0.0 with pattern *b*. The pattern of activation this produces on cycle 1 has *ndp* of 0.05 with pattern *a*. Thus, the network responds to this pattern by producing as its output (that is, by setting the internal input to the units) to $(2.0)(0.05)a = 0.1a$. The factor of 2.0 is the eigenvalue of *a*. This internal input, when added to 0.1 times the external input, produces a pattern of activation on cycle 2 that has an *ndp* of 0.15 with pattern *a* and still has an *ndp* of 0.0 with *b*. This in turn produces an internal input pattern equal to $0.3a$. Again the external input is added, but already it is quite clear that the pattern of activation is moving in the direction of *a*. This process continues until the activations of the units reach the boundaries of the box. At this point they are clipped off, and all of the units have activations of equal magnitude and replicate pattern *a* exactly.

Q.6.2.3. At the end of cycle 1 the activations are simply 0.1 times the external input. This pattern has *ndp* of 0.075 with *a* and of 0.025 with *b*. This generates an internal input of $0.15a + 0.05b$. The patterns agree on units 0, 3, 5, and 6 and disagree on the other units. Internal inputs to units where they agree have magnitude 0.20 $(0.15 + 0.05)$, whereas internal inputs to units where they disagree have magnitude 0.10 $(0.15 - 0.05)$. As it happens, the internal input to unit 7 is −0.10, which cancels the external input (when scaled by the *estr* parameter) so that the activation of unit 7 is 0 on this cycle. The activation of unit 6, on the other hand, is −0.3, because the external input and both learned patterns agree on the activation of this unit. The resulting pattern of activation on cycle 2 has an *ndp* of 0.225 with *a* but only 0.075 with *b*. In general, before the clipping threshold is reached, the pattern of activation will have an *ndp* of $(t-1)(0.15) + 0.075$ with *a* but only $(0.05)(t-1) + 0.025$ with *b*; Thus, *a* will always be

three times stronger than *b* in the internal input, with the internal input to unit 6 having twice the magnitude of the internal input to unit 7. The internal input to 6 will always have the same sign as the external input to this unit, but the internal input to 7 will have the opposite sign compared to its external input. On cycle 3 the internal input becomes stronger than *estr* times the external input to 7, so the activation of unit 7 changes in sign at this point. After one more cycle, the activation of unit 6 reaches the clipping threshold, while the activation of unit 7 still lags. After an additional cycle, even unit 7 reaches the clipping threshold. At this point, all of the units have activations equal to their value in pattern *a*.

Q.6.2.4. The network "rectifies" the distorted unit from the prototype after a single pass through the weights, so that by cycle 2, the sign of every unit in the activation vector matches the prototype value. However, the magnitude is less than it would be had the prototype actually been presented, since the distorted unit opposes the other units in the computation of the internal input to each unit. A further factor is that the external input to the distorted unit opposes its rectification directly. The activations of the nondistorted units reflect the first of these two factors, but not the second, so they show a less marked decrement compared to the prototype case. In any case, both inputs end up in the same corner of the box; the network categorizes each distortion by distorting its representation toward the prototype. With regard to the weights, over the ensemble of distortions, each unit correlates perfectly with itself and has a correlation of $+0.5$ or -0.5 with each other unit. These correlations are reflected directly in the values of the weights.

Ex. 6.3. The Delta Rule in a Linear Associator

Q.6.3.1. Essentially, what usually happens is that the delta rule finds, for each unit, the set of other units whose external inputs are perfectly correlated (positively or negatively) with its own. These correlated units eventually get all of the weight; the units whose activations are not perfectly correlated have no influence on each other. This would not happen if the external input to one of the units was not perfectly correlated with any other unit. When there are perfectly correlated units, the weights coming into each unit have absolute values that sum to 1.0 asymptotically because the delta rule is trying to set the internal input to each unit to match the external input, which is $+1.0$ or -1.0. This means that

in cases where a unit's external input is perfectly correlated with the activation of more than one other unit, the correlated units divide the weight equally among themselves. The weight matrix obtained with the Hebb rule is similar, but as usual with Hebbian learning, the weights reflect pairwise correlations only. Thus, the weight does not gravitate toward the inputs that predict each unit perfectly nor is there any restriction that the weights coming into a particular unit sum to 1.0.

Q.6.3.2. The file *imposs.pat* in the **aa** directory contains a set that cannot be learned in an auto-associator. We reproduce these patterns here:

```
xua  +--++--+
xvb  +--++-+-
yub  +-+-+---
yva  +-+-+-++
```

These patterns were set up so that the last unit would not be predictable from any of the others. The first four units were set up as one subset with two alternative patterns (+−−+) and (+−+−) called x and y, respectively. The next three units were set up as a second subset with two alternative patterns (+−−) and (+−−) called u and v. The final unit was set up as a single-unit subpattern with two alternatives (+) and (−) called a and b. The value of the final subpattern cannot be predicted from either of the other subpatterns, because x and y each occur with a and b, as do u and v. It is only the particular conjunction of the first two subpatterns that can predict the correct activation for the final unit, but the delta rule must be able to set the internal input to each unit based on a weighted sum of its inputs.

Q.6.3.3. Two units that predict each other perfectly will develop strong weights, and if they cannot be predicted by the others, their incoming weights from all other units will be 0. This means that although the auto-associator may be able to learn a set of training patterns perfectly, it may not be able to perform pattern completion accurately if the portion of a known pattern that is used as a probe contains any member of a mutually predictive (and not otherwise predictable) pool. This tends to limit the circumstances in which pattern completion will be possible. In fact, if two units perfectly predict each other, the delta rule will pile all the weight coming into each on the connection from the other, even if the activations of these units could be partially predicted by a weighted sum of activations of other units. A similar problem can arise even in auto-associators that use hidden units. To avoid this

problem, it is necessary to introduce noise that keeps activations from being perfectly correlated.

Ex. 6.4. Memory for General and Specific Information

Q.6.4.1. Your weights should look about the same as those in the figure, although, of course, they will differ in detail because you will have used a different random sequence of distortions. You should find that the model responds best to the prototype, and that, on the average, it responds better to the exemplar that it just saw than to new random distortions. Thus, over the short term, it is sensitive to what it has recently seen, but over the longer term, it extracts the prototype and comes to respond best to it.

Q.6.4.2. When you increase *pflip*, you increase the variance of the individual training exemplars from the prototype and from each other, and this increases the extent to which the model acts as though it is retaining information about particular exemplars. The discussion of Whittlesea's (1983) experiments in *PDP:17* (pp. 199-205) draws on this aspect of the model's performance. Increasing the *lrate* parameter also increases sensitivity to recent exemplars at the (slight) expense of the prototype. This is used to account for some aspects of spared learning in amnesia in *PDP:25*.

Q.6.4.3. You should find that the individual weights in your simulation tend to be larger than in the figure. The reason is primarily that the network must rely on the connections internal to the visual pattern, whereas the weights shown in the figure were obtained when it was able to rely on the name pattern as well as the visual pattern.

Q.6.4.4. You should find more adequate completion with *ncycles* equal to 50. If you round off the obtained activations (so that 0.16 rounds up to 0.2, for example), you should get results fairly similar to those shown in Table 1. The model seems to do a good job filling in what it can but does have one limitation that makes it different from human performance: In cases where it is probed with an ambiguous probe, it always fills in a blend of the two responses that would be appropriate rather than selecting one or the other response.

Q.6.4.5. At the end of 50 epochs, it appears that learning has not yet quite finished, and so some of the elements of the retrieved patterns are occasionally incorrect. In our runs we get the correct results after 100 training epochs.

The network has the most trouble with those properties that differ between typical instances and specific familiar examples. We might expect human subjects to have similar problems, confusing the properties of typical and familiar exemplars. However, once again it seems likely that humans would eventually learn to differentiate the specific instance from the prototype somewhat more fully than we seem to observe here. It is likely that stochastic auto-associators with hidden units would be necessary to capture these characteristics adequately in a PDP framework.

Ex. 6.5. Clustering the Jets and Sharks

Q.6.5.1. The typical pattern of weights indicates that each Jet is activating one of the two output units and each Shark is activating the other. The system usually falls into this pattern because it minimizes the differences between the patterns within each cluster. On average, the Jets are more similar to each other than they are to the Sharks. This similarity is in part because the Jets all have the Jets feature and the Sharks all have the Sharks feature.

Q.6.5.2. The pattern of weights over the other features represents the distribution of values for the Jets and the Sharks. Each dimension gets about 20% of the weight and divides this up so that each value on the dimension has a weight whose average is proportional to the frequency that this value occurs. The observed weights will fluctuate a bit since they are adjusted as each pattern is presented. In the particular case of the *Age* units, most Jets are in their 20s and most Sharks are in their 30s, so the 20s connection gets most of the weight to the unit that responds to the Jets and the 30s connection gets most of the weight to the unit that responds to the Sharks.

Q.6.5.3. Many different possible nonoptimal configurations of weights can be observed. Generally, each one involves one or perhaps two dimensions. For example, the following weights were produced on one run:

```
10   9          9 10
 5  11   3      9  8   2
 5   8   6      9  8   2
12   6   1      4  9   5
10   0   9      0 19   0
```

Here the network has partitioned the patterns in terms of whether they are Burglars (second feature on last row) or not. From this we can see that the non-Burglars (who activate the left unit) tend to be in their 30s and single, whereas the Burglars tend to be in their 20s or 30s, with LH or HS educations and with about half of them married. Note that this is a much less optimal partitioning than the typical one.

Reducing the learning rate makes it easier to find an unusual set of weights because the learning process is doing gradient descent in the compactness of the partitions, but each step is being made on the basis of a very small sample (a single pattern). Learning can be thought of as tending downhill, but with noise so that it does not always go downhill. When the *lrate* is higher, that is like more noise, and the network can escape from local minima with greater probability. Note that as the weight step gets infinitesimally small, each weight step will have very little effect and so performance will approximate true gradient descent more and more closely. The nonoptimal states are local minima due to the initial configuration of the weights. As the weights are reorganized, it becomes harder and harder to break out of the current minimum; that is why the pattern of weights eventually gets locked in.

Q.6.5.4. The network no longer categorizes the groups into Jets and Sharks because the Jets and Sharks overlap quite a bit and there are several better ways they might be partitioned into groups. One example case is shown below:

```
 0  0        0  0
21  0  3     0 21  3
12  8  4     6 11  6
14  6  4     9 12  3
 7 10  6     7  8  8
```

Here the network has used age (20s vs. 30s) as the categorization feature, dividing up the 40s individuals in terms of their similarity to the members of the other two age groups. This partition is quite like the Jets vs. Sharks partition, but it groups Ken, a Shark with two of the three typical Jet properties (as many as any Jet has), with most of the Jets, and groups Mike, Al, Doug, and Ralph, the 30s Jets, with most of the Sharks.

Q.6.5.5. When the gang categorization feature is present (Jets vs. Sharks) the network tends to construct one cluster consisting of all the members of one gang, and then tends to divide the other gang into two subgangs. Because there are more Jets than Sharks, it is more typical for the Jets to be divided into two groups, since this

gives a slightly more even distribution of individuals in each of the three groups, but this does not always happen. In fact, the network will occasionally find a group of similar individuals who are about half Jets and half Sharks, putting the rest of the Sharks in one group and the rest of the Jets in the other. There is one sensible pattern that is quite common when there are three output units and the *ajets.pat* patterns are used so that the gang feature is not specified. This is the pattern in which each unit picks up on the individuals with a different occupation:

```
 0  0        0  0        0  0
12  9  2     8 14  1     5 11  7
12 10  2     4 11  8    10  9  5
 4 12  7    13  8  2    19  5  0
 0 24  0    24  0  0     1  0 23
```

This division of the individuals turns out to reveal that there is some systematicity to the distribution of other features, dependent on the occupation feature. For example, the algorithm discovers that the bookies tend to be the least educated.

Ex. 6.6. Graph Partitioning

Q.6.6.1. In 9 out of 10 runs that we tried with the default value of *lrate*, the best solution was found; that is, one unit responded to all patterns involving two units in the left part of the graph, and the other responded to all patterns involving two units in the right part of the graph. The single pattern that connects the two parts of the graph is grabbed by one of the two units. On the one exceptional run, the network found the following local minimum:

```
      7          13              5          12

18        0  0       32       0      21 37        0

     10          18              6          16
```

In this state, the two inputs that have the largest number of connections (those that connect the two graphs) form a strong "bond"—that is, they both activate the same output unit. This leaves the other output unit to pick up the patterns that do not activate either of the linking input units. When smaller *lrate* is used, this and several other local minima can be found more frequently, for the same reasons as discussed in Q.6.5.3.

CHAPTER 7

Ex. 7.1. Using Context to Identify an Ambiguous Character

Q.7.1.1. The letter unit *k* begins to get an edge over *r* at cycle 3, one cycle after *work* exceeds threshold, but the effect at this point is in the third decimal place, since the activation of *work* is very slight (less than 0.01) at the end of cycle 2. A detectable advantage for *k* appears on the screen at the end of cycle 4. The advantage is due to feedback from *work*, and it grows larger and larger as *work* becomes more and more strongly excited. The parameter that determines whether the activation of *r* will decrease as the activation of *k* increases is the letter-to-letter inhibition parameter *gamma/ l−>l*. The response probability for *r* eventually goes down because the response-choice probability for a particular alternative is based on the Luce (1959) ratio, in which each alternative's choice probability is equal to its own response strength divided by the sum of the response strengths of all of the other alternatives. As the response strength for *k* goes up, its contribution to the denominator increases, and therefore the choice probability for *r* goes down. The response probabilities pull apart more slowly because response strengths build up gradually over time, based on the *orate* parameter. If this parameter is set to a larger value than its default value of 0.05, response probabilities shadow activation differences more closely. The asymptotic value is unaffected, however (although with a low *orate* it takes quite a while to reach asymptote).

Ex. 7.2. Simulations of Basic Forced-Choice Results

Q.7.2.1. For the *CAVE* display, the activation of *v* starts at 0 and grows by about 0.06 or 0.07 on each cycle, slowing down at around cycle 8 and reaching a level of 0.73 on cycle 15. When the mask comes on in cycle 16, its effect is sudden, wiping out the activation of *v* in two cycles. The forced-choice results are based on the response probabilities. The response probabilities for *v* and *g* start out equal at .03 (actually .038, or 1/26), since all letters start out with an equal chance of being chosen before the onset of the display. The advantage for *v* begins to appear immediately, but builds up more slowly at first because of the low value of *orate*. By cycle 12 it is growing quite rapidly, however, and reaches a peak of 0.60 on cycle 15. It falls off much more gradually than

the activation of *v*, again because of the low value of *orate*. The forced-choice accuracy at the end of processing is based on the response probabilities at the point in processing where the response probability for the correct alternative is maximal, which is at cycle 15, just before the onset of the mask. The probability of choosing *v* in the forced choice is equal to its response probability at cycle 15 (.60), plus 0.5 times the probability that neither *v* nor *g* is read out (about .38), which comes to .60+.19=.79.

For the *ZXVJ* display, the time course of activation of *v* is very similar at first, but *v* rises more slowly after cycle 3, reaching only 0.44 at cycle 15. The response strength for *v* is correspondingly lower, and the resulting forced-choice accuracy reflects this.

The *ZXVJ* display produces no word-level activations because of letter-to-word inhibition. The letter-to-word inhibition is 0.04 while letter-to-word excitation is 0.07. Given these values, if *ZXVJ* had two letters in common with some word, that word would receive a net excitatory input because two letters in the display would activate the word by 0.07 times their activation and two letters would inhibit it by 0.04 times their activation, for a net input of about 0.06 times the letter-level activation. Since *ZXVJ* has at most one letter in common with any word, the set of active letters produces at best a net inhibitory effect of $0.07-(3\times0.04)=-0.05$ times the letters' activations.

No words other than *cave* achieve substantial activation when *CAVE* is presented because of the strong word-to-word inhibition. The word-to-word inhibition parameter is set at 0.21. At first all words with two or more letters in common with *CAVE* receive net excitatory input from the letter level, but as the activations of these words exceed threshold they inhibit each other. The word unit *cave*, which receives by far the most bottom-up excitation, continues to grow in activation, but the others grow very slowly or not at all. As *cave* gets more active it eventually comes to suppress the activation of the others; they cease inhibiting *cave*, so its activation grows with little restraint, thereby allowing it to increase its domination of the pattern of activation at the word level.

Q.7.2.2. Accuracy for *V* in *MAVE* is almost as good as for *V* in *CAVE* because *MAVE* activates a number of words containing *V* in the third letter position. Though none of these words achieve the level of activation *cave* achieves when that word is presented, all of the active words containing *V* work together, providing feedback that reinforces the activation of *v* in *MAVE*.

Q.7.2.3. The reason the advantage of *have* grows larger and larger is that it exerts more of an inhibitory influence on its neighbors than they

exert on it. The *have* word unit's initial resting-level advantage means that it crosses threshold sooner and has a larger output than other words receiving an equal amount of excitation from the letter level. This means that other words see more inhibition from *have* than *have* sees from them. In McClelland and Rumelhart (1981) this is called the "rich get richer effect." (This effect is similar but not identical to Grossberg's rich get richer effect described in Chapter 2.)

What happens with *move* is even more interesting. Here, though *move* is one of the most frequent words, the letters that it has in common with the display are shared with fewer of the other words that have three letters in common with the display. The words *cave*, *gave*, *have*, *pave*, *save*, and *wave* all have *_ave* in common with the display, while *made*, *make*, *male*, *mare*, *mate*, and *maze* all have *ma_e* in common with the display. The word *move* is the only word that has *m_ve* in common with the display, and no words have *mav_* in common with the display. Thus, 7 words (*move* and the *ma_e* gang) are providing feedback support to *m*, 12 words (the *_ave* gang and the *ma_e* gang) are supporting *a*, 7 are supporting *v* (*move* and the *_ave* gang), and all 13 are supporting the final *e*. The result is that *a* and *e* are more strongly activated at the letter level than *m* or *v*. Thus the members of the *_ave* gang and the *ma_e* gang get more bottom-up activation than *move* as a result of the differential top-down feedback. Word-to-word competition causes *move*'s activation to fall off as these other words gain.

Q.7.2.4. Examples of these effects can be obtained with almost any nonword, but they work better for nonwords that have three letters in common with several words. A nice example of the gang effect can be seen with *RARD*, in which *rare* loses out to the *_ard* gang.

Ex. 7.3. Subtler Aspects of the Word Superiority Effect

Q.7.3.1. High bigram-frequency pseudowords like *TEEL* tend to activate more words than low bigram-frequency pseudowords like *HOET*, but for these examples, both kinds tend to activate only words that have the correct forced-choice alternative; they differ from the word displayed in some other position. Thus at cycle 15, the words activated by *TEEL* are *feel, heel, keel, peel, reel,* and *tell* (*teen* has a tiny residual activation). The summed activation of these words is the feedback support for the *L* in *TEEL*. The sum is about 0.64 (0.62 shows in the display due to truncation). With *HOET*, though fewer words have three letters in common with it,

all end in *T*, and at the end of 15 cycles, they are the only words left active, with a summed activation of about 0.66. Thus the lower bigram-frequency item has, in fact, a bit more support than the higher bigram-frequency item.

Q.7.3.2. McClelland and Johnston got the results that they did (according to the interactive activation model) because their method of constructing pseudowords caused the target letter to benefit from the support of words partially activated by the display. As it happened, both for high and low bigram-frequency words, the number of "friends" of the target letters (i.e., words that had three letters in common with the display, including the target letter) was very high and there were virtually no enemies of any of their low bigram-frequency words. (An "enemy" is a word that has three letters in common with the display, in which the three letters do not include the target letter tested in the forced choice). When an item like *HOEM* or *HOET* is tested in the position of the changed letter, the bias in favor of friends is reversed. Low bigram-frequency items are particularly susceptible to these effects, and so if McClelland and Johnston had tested the changed letter they might have observed a large apparent bigram-frequency effect.

Q.7.3.3. In the bright-target/pattern-mask condition, the more highly constrained word usually has an advantage at the word level due to reduced competition. The correct answer always dominates, however, and the word-level difference makes a smaller difference at the letter level for two reasons. For one thing, the forced choice tends to attenuate word-level differences. More interestingly, words with weaker constraints tend to have more friends as well as enemies. That is, while there are more words that share three letters with the display and differ from it in the tested position, there are also more words that share three letters with the display and match the display in the tested position (e.g., *CAME*, *CARE*, and *CASE* in the case of *CAKE*). These words support the target letter, thereby counteracting some of the disadvantage these words would otherwise have because of reduced feedback support from the target word itself.

Under degraded display conditions, the features extracted are much more likely to be compatible with words other than the word shown if the word shown is a low-constraint word than if it is a high-constraint word. For example, with *CAKE*, the words *LAKE* and *CARE* are frequently consistent with the features detected; *CAFE* and *CAPE* occasionally are as well. With *CLUE*, on the other hand, the only other word that we have ever found consistent with the display is *GLUE*, and this occurred only once

in about 20 runs. Note that, when some other word is consistent with the display, the alternatives that are consistent generally do not have an equal response probability; rather, the more frequent alternative tends to dominate. Thus *CAKE* tends to lose out to *LAKE* and *CARE*, both of which are higher in frequency than *CAKE*. These effects tend to greatly increase the advantage of more constrained words under these visual conditions compared to bright-target/pattern-mask conditions.

Ex. 7.4. Further Experiments With the Interactive Activation Model

Q.7.4.1. Both *SLNT* and *SLET* produce a facilitation for *l* relative to *JLQX* because both activate words that share three letters with them, including the letter *l*. *SLNT* also activates *sent*, which does not support the *l*; this is why there is slightly more facilitation for the *l* in *SLET* than for the *l* in *SLNT*. Over the set of materials used by Rumelhart and McClelland, there was a tendency for the pronounceable items to show an advantage in the second and third positions, both in the experimental data and in the simulations. In the simulations, the effect was due to vowel enemies of the consonant target letter.

What is critical is that the target letter have more friends than enemies. For example, if we test *SLNT* in the third letter position, we get very little facilitation because the *N* has only one friend. The *E* in *SLET* does even worse since it has no friends among the words of four letters.

Q.7.4.2. The model produces a very large contextual enhancement effect for words, but the effect is negligible for pseudowords. What happens with pseudowords is that words that match two or more letters from the preview of the context become activated by the context, and they tend to interfere with the later activation of words that specifically share the target letter with the context. This effect can be particularly severe if (as is not the case with *_hig*) all three letters of the context happen to match some word; in this case the model produces a negative pseudoword context enhancement effect; that is, the model does worse on such items when the context precedes the pseudoword than when it does not.

Q.7.4.3. To produce the context enhancement effect with pseudowords, the main thing we did was lower the resting activation level for words (the minimum activation for words had to be reduced to accommodate this). Lowering the resting level for word units had the effect of allowing the preview to prime all words consistent

with it, without allowing any of them to go above threshold during the preview period. This meant that when the target letter was presented, the words that included this letter would tend to be the first to exceed threshold—this would then allow the target letter to suppress its competition. By itself this manipulation was not quite sufficient; we found we also had to set the threshold for word-to-word inhibition above 0 so that active words could feed activation back to the letter level before they began to inhibit each other. Finally, we also found we needed to reduce the letter-to-word level inhibition. With these changes, we were able to capture the pseudoword enhancement effect as well as all the rest of the findings considered in McClelland and Rumelhart (1981) and Rumelhart and McClelland (1982). These parameters produce a rather exaggerated enhancement effect with this particular item, but over the ensemble of materials used in the pseudoword enhancement effect experiments reported in Rumelhart and McClelland, they produced only a slightly oversized enhancement effect.

Q.7.4.4. To produce the effects that Carr et al. observed, one has to assume that subjects have control over the letter-to-word inhibition parameter *gamma/ l−>w*. If letter-to-word inhibition is strong, then words other than the word actually shown are kept from receiving bottom-up excitation. This would be appropriate if subjects were expecting only words, since it would tend to reduce interference at the word level. This might also be appropriate when expecting only random letter strings, since it would tend to reduce spurious feedback from partially activated words; alternatively, the letter-to-word inhibition might be set high by default. In processing pseudowords, however, a lower setting of letter-to-word inhibition would allow these stimuli to produce partial activations of words, thereby producing a facilitation effect for letters in these stimuli. If the letter-to-word inhibition is assumed to have a value of 0.21 when words or random strings are expected, but is assumed to be reduced to 0.04 only when pseudowords are expected, the model nicely simulates the Carr et al. effects.