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 the 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 input 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