



**Computational Models of Cognition and Perception**

Editors

Jerome A. Feldman  
Patrick J. Hayes  
David E. Rumelhart

*Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, by David E. Rumelhart, James L. McClelland, and the PDP Research Group

*Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models*, by James L. McClelland, David E. Rumelhart, and the PDP Research Group

*Neurophilosophy: Toward a Unified Science of the Mind-Brain*, by Patricia S. Churchland

*Qualitative Reasoning About Physical Systems*, edited by Daniel G. Bobrow

*Visual Cognition*, edited by Steven Pinker

---

**PARALLEL DISTRIBUTED  
PROCESSING**  
**Explorations in the Microstructure  
of Cognition**

**Volume 1: Foundations**

---

**David E. Rumelhart    James L. McClelland  
and the PDP Research Group**

Chisato Asanuma	Alan H. Kawamoto	Paul Smolensky
Francis H. C. Crick	Paul W. Munro	Gregory O. Stone
Jeffrey L. Elman	Donald A. Norman	Ronald J. Williams
Geoffrey E. Hinton	Daniel E. Rabin	David Zipser
Michael I. Jordan	Terrence J. Sejnowski	

Institute for Cognitive Science  
University of California, San Diego

A Bradford Book  
The MIT Press  
Cambridge, Massachusetts  
London, England

## CHAPTER 8

Learning Internal Representations  
by Error Propagation

D. E. RUMELHART, G. E. HINTON, and R. J. WILLIAMS

## THE PROBLEM

We now have a rather good understanding of simple two-layer associative networks in which a set of input patterns arriving at an input layer are mapped directly to a set of output patterns at an output layer. Such networks have no *hidden* units. They involve only *input* and *output* units. In these cases there is no *internal representation*. The coding provided by the external world must suffice. These networks have proved useful in a wide variety of applications (cf. Chapters 2, 17, and 18). Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in a PDP system is determined by their overlap. The overlap in such networks is determined outside the learning system itself—by whatever produces the patterns.

The constraint that similar input patterns lead to similar outputs can lead to an inability of the system to learn certain mappings from input to output. Whenever the representation provided by the outside world is such that the similarity structure of the input and output patterns are very different, a network without internal representations (i.e., a

network without hidden units) will be unable to perform the necessary mappings. A classic example of this case is the *exclusive-or* (XOR) problem illustrated in Table 1. Here we see that those patterns which overlap least are supposed to generate identical output values. This problem and many others like it cannot be performed by networks without hidden units with which to create their own internal representations of the input patterns. It is interesting to note that had the input patterns contained a third input taking the value 1 whenever the first two have value 1 as shown in Table 2, a two-layer system would be able to solve the problem.

Minsky and Papert (1969) have provided a very careful analysis of conditions under which such systems are capable of carrying out the required mappings. They show that in a large number of interesting cases, networks of this kind are incapable of solving the problems. On the other hand, as Minsky and Papert also pointed out, if there is a layer of simple perceptron-like hidden units, as shown in Figure 1, with which the original input pattern can be augmented, there is always a recoding (i.e., an internal representation) of the input patterns in the hidden units in which the similarity of the patterns among the hidden units can support any required mapping from the input to the output units. Thus, if we have the right connections from the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from input to output through these hidden units. In the case of the XOR problem, the addition of a feature that detects the conjunction of the input units changes the similarity

TABLE 1

Input Patterns	Output Patterns
00	0
01	1
10	1
11	0

TABLE 2

Input Patterns	Output Patterns
000	0
010	1
100	1
111	0

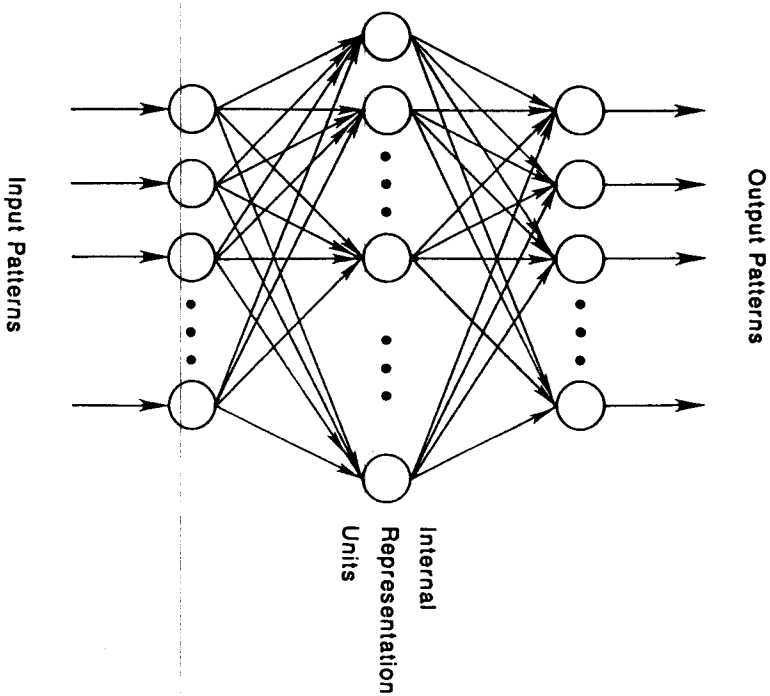


FIGURE 1. A multilayer network. In this case the information coming to the input units is *re-coded* into an internal representation and the outputs are generated by the internal representation rather than by the original pattern. Input patterns can always be encoded, if there are enough hidden units, in a form so that the appropriate output pattern can be generated from any input pattern.

structure of the patterns sufficiently to allow the solution to be learned. As illustrated in Figure 2, this can be done with a single hidden unit. The numbers on the arrows represent the strengths of the connections among the units. The numbers written in the circles represent the thresholds of the units. The value of  $+1.5$  for the threshold of the hidden unit insures that it will be turned on only when both input units are on. The value  $0.5$  for the output unit insures that it will turn on only when it receives a net positive input greater than  $0.5$ . The weight of  $-2$  from the hidden unit to the output unit insures that the output unit will not come on when both input units are on. Note that from the point of view of the output unit, the hidden unit is treated as simply another input unit. It is as if the input patterns consisted of three rather than two units.

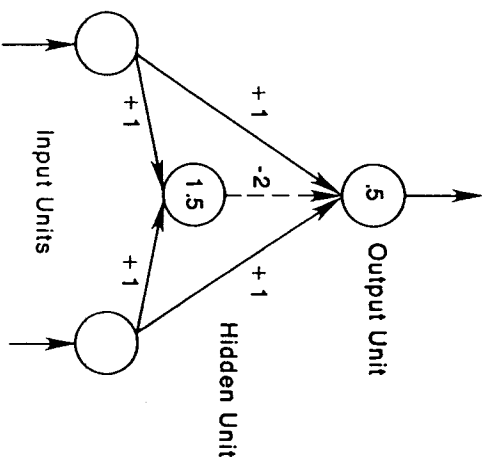


FIGURE 2. A simple XOR network with one hidden unit. See text for explanation.

The existence of networks such as this illustrates the potential power of hidden units and internal representations. The problem, as noted by Minsky and Papert, is that whereas there is a very simple guaranteed learning rule for all problems that can be solved without hidden units, namely, the perceptron convergence procedure (or the variation due originally to Widrow and Hoff, 1960, which we call the delta rule; see Chapter 11), there is no equally powerful rule for learning in networks with hidden units. There have been three basic responses to this lack. One response is represented by competitive learning (Chapter 5) in which simple *unsupervised* learning rules are employed so that useful hidden units develop. Although these approaches are promising, there is no external force to *insure* that hidden units appropriate for the required mapping are developed. The second response is to simply *assume* an internal representation that, on some a priori grounds, seems reasonable. This is the tack taken in the chapter on verb learning (Chapter 18) and in the interactive activation model of word perception (McClelland & Rumelhart, 1981; Rumelhart & McClelland, 1982). The third approach is to attempt to *develop* a learning procedure capable of learning an internal representation adequate for performing the task at hand. One such development is presented in the discussion of Boltzmann machines in Chapter 7. As we have seen, this procedure involves the use of stochastic units, requires the network to reach equilibrium in two different phases, and is limited to symmetric networks. Another recent approach, also employing stochastic units, has been developed by Barto (1985) and various of his colleagues (cf. Barto

modified. It should also be noted that there is no reason why some output units might not receive inputs from other output units in earlier layers. In this case, those units receive two different kinds of error: that from the direct comparison with the target and that passed through the other output units whose activation it affects. In this case, the correct procedure is to simply add the weight changes dictated by the direct comparison to that propagated back from the other output units.

## SIMULATION RESULTS

We now have a learning procedure which could, in principle, evolve a set of weights to produce an arbitrary mapping from input to output. However, the procedure we have produced is a gradient descent procedure and, as such, is bound by all of the problems of any hill climbing procedure—namely, the problem of local maxima or (in our case) minima. Moreover, there is a question of how long it might take a system to learn. Even if we could guarantee that it would eventually find a solution, there is the question of whether our procedure could learn in a reasonable period of time. It is interesting to ask what hidden units the system actually develops in the solution of particular problems. This is the question of what kinds of internal representations the system actually creates. We do not yet have definitive answers to these questions. However, we have carried out many simulations which lead us to be optimistic about the local minima and time questions and to be surprised by the kinds of representations our learning mechanism discovers. Before proceeding with our results, we must describe our simulation system in more detail. In particular, we must specify an activation function and show how the system can compute the derivative of this function.

*A useful activation function.* In our above derivations the derivative of the activation function of unit  $u_j$ ,  $f'_j(\text{net}_j)$ , always played a role. This implies that we need an activation function for which a derivative exists. It is interesting to note that the linear threshold function, on which the perception is based, is discontinuous and hence will not suffice for the generalized delta rule. Similarly, since a linear system achieves no advantage from hidden units, a linear activation function will not suffice either. Thus, we need a continuous, nonlinear activation function. In most of our experiments we have used the *logistic* activation function in which

$$o_{pj} = \frac{1}{1 + e^{-\sum_k w_{pk} o_{pk} + \theta_j}} \quad (15)$$

where  $\theta_j$  is a bias similar in function to a threshold.<sup>1</sup> In order to apply our learning rule, we need to know the derivative of this function with respect to its total input,  $\text{net}_{pj}$ , where  $\text{net}_{pj} = \sum_k w_{pk} o_{pk} + \theta_j$ . It is easy to show that this derivative is given by

$$\frac{\partial o_{pj}}{\partial \text{net}_{pj}} = o_{pj} (1 - o_{pj}).$$

Thus, for the logistic activation function, the error signal,  $\delta_{pj}$ , for an output unit is given by

$$\delta_{pj} = (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj}),$$

and the error for an arbitrary hidden  $u_j$  is given by

$$\delta_{uj} = o_{uj} (1 - o_{uj}) \sum_k \delta_{pk} w_{kj}.$$

It should be noted that the derivative,  $o_{pj} (1 - o_{pj})$ , reaches its maximum for  $o_{pj} = 0.5$  and, since  $0 \leq o_{pj} \leq 1$ , approaches its minimum as  $o_{pj}$  approaches zero or one. Since the amount of change in a given weight is proportional to this derivative, weights will be changed most for those units that are near their midrange and, in some sense, not yet committed to being either on or off. This feature, we believe, contributes to the stability of the learning of the system.

One other feature of this activation function should be noted. The system can not actually reach its extreme values of 1 or 0 without infinitely large weights. Therefore, in a practical learning situation in which the desired outputs are binary  $\{0,1\}$ , the system can never actually achieve these values. Therefore, we typically use the values of 0.1 and 0.9 as the targets, even though we will talk as if values of  $\{0,1\}$  are sought.

*The learning rate.* Our learning procedure requires only that the change in weight be proportional to  $\partial E_p / \partial w$ . True gradient descent requires that infinitesimal steps be taken. The constant of proportionality is the learning rate in our procedure. The larger this constant, the larger the changes in the weights. For practical purposes we choose a

<sup>1</sup> Note that the values of the bias,  $\theta_j$ , can be learned just like any other weights. We simply imagine that  $\theta_j$  is the weight from a unit that is always on.

learning rate that is as large as possible without leading to oscillation. This offers the most rapid learning. One way to increase the learning rate without leading to oscillation is to modify the generalized delta rule to include a *momentum* term. This can be accomplished by the following rule:

$$\Delta w_{ji}(n+1) = \eta (\delta_{pj} o_{pj}) + \alpha \Delta w_{ji}(n) \quad (16)$$

where the subscript  $n$  indexes the presentation number,  $\eta$  is the learning rate, and  $\alpha$  is a constant which determines the effect of past weight changes on the current direction of movement in weight space. This provides a kind of momentum in weight space that effectively filters out high-frequency variations of the error-surface in the weight space. This is useful in spaces containing long ravines that are characterized by sharp curvature across the ravine and a gently sloping floor. The sharp curvature tends to cause divergent oscillations across the ravine. To prevent these it is necessary to take very small steps, but this causes very slow progress along the ravine. The momentum filters out the high curvature and thus allows the effective weight steps to be bigger. In most of our simulations  $\alpha$  was about 0.9. Our experience has been that we get the same solutions by setting  $\alpha = 0$  and reducing the size of  $\eta$ , but the system learns much faster overall with larger values of  $\alpha$  and  $\eta$ .

*Symmetry breaking.* Our learning procedure has one more problem that can be readily overcome and this is the problem of symmetry breaking. If all weights start out with equal values and if the solution requires that unequal weights be developed, the system can never learn. This is because error is propagated back through the weights in proportion to the values of the weights. This means that all hidden units connected directly to the output inputs will get identical error signals, and, since the weight changes depend on the error signals, the weights from those units to the output units must always be the same. The system is starting out at a kind of *local maximum*, which keeps the weights equal, but it is a maximum of the error function, so once it escapes it will never return. We counteract this problem by starting the system with small random weights. Under these conditions symmetry problems of this kind do not arise.

## The XOR Problem

It is useful to begin with the exclusive-or problem since it is the classic problem requiring hidden units and since many other difficult

problems involve an XOR as a subproblem. We have run the XOR problem many times and with a couple of exceptions discussed below, the system has always solved the problem. Figure 3 shows one of the solutions to the problem. This solution was reached after 558 sweeps through the four stimulus patterns with a learning rate of  $\eta = 0.5$ . In this case, both the hidden unit and the output unit have *positive biases* so they are on unless turned off. The hidden unit turns on if neither input unit is on. When it is on, it turns off the output unit. The connections from input to output units arranged themselves so that they turn off the output unit whenever both inputs are on. In this case, the network has settled to a solution which is a sort of mirror image of the one illustrated in Figure 2.

We have taught the system to solve the XOR problem hundreds of times. Sometimes we have used a single hidden unit and direct connections to the output unit as illustrated here, and other times we have allowed two hidden units and set the connections from the input units to the outputs to be zero, as shown in Figure 4. In only two cases has the system encountered a *local minimum* and thus been unable to solve the problem. Both cases involved the two hidden units version of the

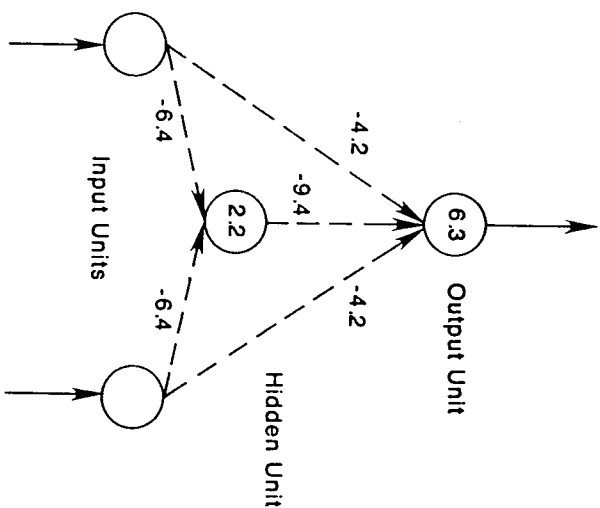


FIGURE 3. Observed XOR network. The connection weights are written on the arrows and the biases are written in the circles. Note a positive bias means that the unit is on unless turned off.

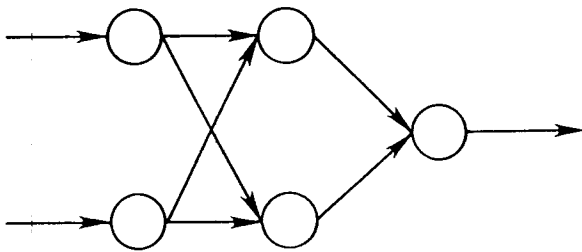


FIGURE 4. A simple architecture for solving XOR with two hidden units and no direct connections from input to output.

problem and both ended up in the same local minimum. Figure 5 shows the weights for the local minimum. In this case, the system correctly responds to two of the patterns—namely, the patterns 00 and 10. In the cases of the other two patterns 11 and 01, the output unit gets a net input of zero. This leads to an output value of 0.5 for both of these patterns. This state was reached after 6,587 presentations of each pattern with  $\eta=0.25$ .<sup>2</sup> Although many problems require more presentations for learning to occur, further trials on this problem merely increase the magnitude of the weights but do not lead to any improvement in performance. We do not know the frequency of such local minima, but our experience with this and other problems is that they are quite rare. We have found only one other situation in which a local minimum has occurred in many hundreds of problems of various sorts. We will discuss this case below.

The XOR problem has proved a useful test case for a number of other studies. Using the architecture illustrated in Figure 4, a student in our laboratory, Yves Chauvin, has studied the effect of varying the

<sup>2</sup> If we set  $\eta = 0.5$  or above, the system escapes this minimum. In general, however, the best way to avoid local minima is probably to use very small values of  $\eta$ .

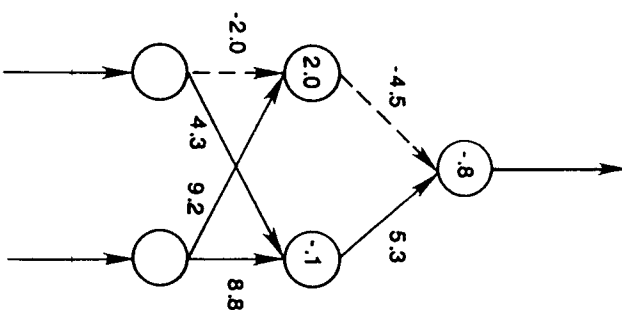


FIGURE 5. A network at a local minimum for the exclusive-or problem. The dotted lines indicate negative weights. Note that whenever the right most input unit is on it turns on *both* hidden units. The weights connecting the hidden units to the output are arranged so that when both hidden units are on, the output unit gets a net input of zero. This leads to an output value of 0.5. In the other cases the network provides the correct answer.

number of hidden units and varying the learning rate on time to solve the problem. Using as a learning criterion an error of 0.01 per pattern, Yves found that the average number of presentations to solve the problem with  $\eta = 0.25$  varied from about 245 for the case with two hidden units to about 120 presentations for 32 hidden units. The results can be summarized by  $P = 280 - 33 \log_2 H$ , where  $P$  is the required number of presentations and  $H$  is the number of hidden units employed. Thus, the time to solve XOR is reduced linearly with the logarithm of the number of hidden units. This result holds for values of  $H$  up to about 40 in the case of XOR. The general result that the time to solution is reduced by increasing the number of hidden units has been observed in virtually all of our simulations. Yves also studied the time to solution as a function of learning rate for the case of eight hidden units. He found an average of about 450 presentations with  $\eta = 0.1$  to about 68 presentations with  $\eta = 0.75$ . He also found that

learning rates larger than this led to unstable behavior. However, within this range larger learning rates speeded the learning substantially. In most of our problems we have employed learning rates of  $\eta = 0.25$  or smaller and have had no difficulty.

### Parity

One of the problems given a good deal of discussion by Minsky and Papert (1969) is the parity problem, in which the output required is 1 if the input pattern contains an odd number of 1s and 0 otherwise. This is a very difficult problem because the most similar patterns (those which differ by a single bit) require different answers. The XOR problem is a parity problem with input patterns of size two. We have tried a number of parity problems with input patterns ranging from size two to eight. Generally we have employed layered networks in which direct connections from the input to the output units are not allowed, but must be mediated through a set of hidden units. In this architecture, it requires at least  $N$  hidden units to solve parity with patterns of length  $N$ . Figure 6 illustrates the basic paradigm for the solutions discovered by the system. The solid lines in the figure indicate weights of +1 and the dotted lines indicate weights of -1. The numbers in the circles represent the biases of the units. Basically, the hidden units arranged

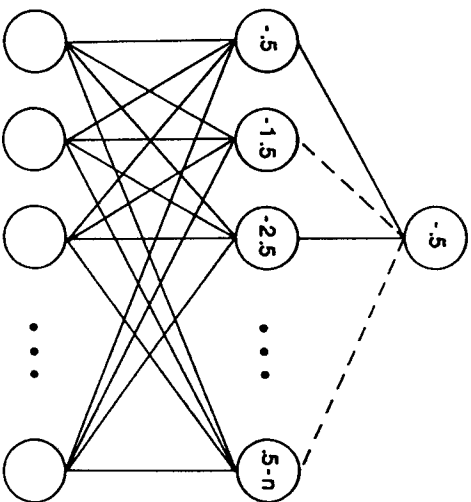


FIGURE 6. A paradigm for the solutions to the parity problem discovered by the learning system. See text for explanation.

themselves so that they count the number of inputs. In the diagram, the one at the far left comes on if one or more input units are on, the next comes on if two or more are on, etc. All of the hidden units come on if all of the input lines are on. The first  $m$  hidden units come on whenever  $m$  bits are on in the input pattern. The hidden units then connect with alternately positive and negative weights. In this way the net input from the hidden units is zero for even numbers and +1 for odd numbers. Table 3 shows the actual solution attained for one of our simulations with four input lines and four hidden units. This solution was reached after 2,825 presentations of each of the sixteen patterns with  $\eta = 0.5$ . Note that the solution is roughly a mirror image of that shown in Figure 6 in that the number of hidden units turned on is equal to the number of zero input values rather than the number of ones. Beyond that the principle is that shown above. It should be noted that the internal representation created by the learning rule is to arrange that the number of hidden units that come on is equal to the number of zeros in the input and that the particular hidden units that come on depend *only* on the number, not on which input units are on. This is exactly the sort of recoding required by parity. It is not the kind of representation readily discovered by unsupervised learning schemes such as competitive learning.

### The Encoding Problem

Ackley, Hinton, and Sejnowski (1985) have posed a problem in which a set of orthogonal input patterns are mapped to a set of orthogonal output patterns through a small set of hidden units. In such cases the internal representations of the patterns on the hidden units must be rather efficient. Suppose that we attempt to map  $N$  input patterns onto  $N$  output patterns. Suppose further that  $\log_2 N$  hidden units are provided. In this case, we expect that the system will learn to use the

TABLE 3

Number of $O_r$ Input Units	Hidden Unit Patterns	Output Value
0	1111	0
1	1011	1
2	1010	0
3	0010	1
4	0000	0

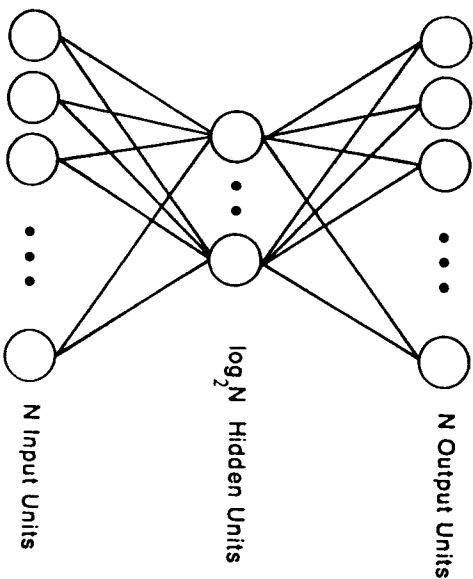


FIGURE 7. A network for solving the encoder problem. In this problem there are  $N$  orthogonal input patterns each paired with one of  $N$  orthogonal output patterns. There are only  $\log_2 N$  hidden units. Thus, if the hidden units take on binary values, the hidden units must form a binary number to encode each of the input patterns. This is exactly what the system learns to do.

hidden units to form a binary code with a distinct binary pattern for each of the  $N$  input patterns. Figure 7 illustrates the basic architecture for the encoder problem. Essentially, the problem is to learn an encoding of an  $N$  bit pattern into a  $\log_2 N$  bit pattern and then learn to decode this representation into the output pattern. We have presented the system with a number of these problems. Here we present a problem with eight input patterns, eight output patterns, and three hidden units. In this case the required mapping is the identity mapping illustrated in Table 4. The problem is simply to turn on the same bit in the

TABLE 4

Input Patterns	Output Patterns
10000000	10000000
01000000	01000000
00100000	00100000
00010000	00010000
00001000	00001000
00000100	00000100
00000010	00000010
00000001	00000001

output as in the input. Table 5 shows the mapping generated by our learning system on this example. It is of some interest that the system employed its ability to use intermediate values in solving this problem. It could, of course, have found a solution in which the hidden units took on only the values of zero and one. Often it does just that, but in this instance, and many others, there are solutions that use the intermediate values, and the learning system finds them even though it has a bias toward extreme values. It is possible to set up problems that require the system to make use of intermediate values in order to solve a problem. We now turn to such a case.

Table 6 shows a very simple problem in which we have to convert from a *distributed representation* over two units into a *local representation* over four units. The similarity structure of the distributed input patterns is simply not preserved in the local output representation.

We presented this problem to our learning system with a number of constraints which made it especially difficult. The two input units were only allowed to connect to a single hidden unit which, in turn, was allowed to connect to four more hidden units. Only these four hidden units were allowed to connect to the four output units. To solve this problem, then, the system must first convert the distributed

TABLE 5

Input Patterns	Hidden Unit Patterns	Output Patterns
10000000	.5 0 0 0	10000000
01000000	0 1 1 0	01000000
00100000	1 1 1 0	00100000
00010000	1 1 1 1	00010000
00001000	0 1 1 1	00001000
00000100	.5 0 1 1	00000100
00000010	1 0 1 .5	00000010
00000001	0 0 0 .5	00000001

TABLE 6

Input Patterns	Output Patterns
00	1000
01	0100
10	0010
11	0001

representation of the input patterns into various intermediate values of the singleton hidden unit in which different activation values correspond to the different input patterns. These continuous values must then be converted back through the next layer of hidden units—first to another distributed representation and then, finally, to a local representation. This problem was presented to the system and it reached a solution after 5,226 presentations with  $\eta = 0.05$ .<sup>3</sup> Table 7 shows the sequence of representations the system actually developed in order to transform the patterns and solve the problem. Note each of the four input patterns was mapped onto a particular activation value of the singleton hidden unit. These values were then mapped onto distributed patterns at the next layer of hidden units which were finally mapped into the required local representation at the output level. In principle, this trick of mapping patterns into activation values and then converting those activation values back into patterns could be done for any number of patterns, but it becomes increasingly difficult for the system to make the necessary distinctions as ever smaller differences among activation values must be distinguished. Figure 8 shows the network the system developed to do this job. The connection weights from the hidden units to the output units have been suppressed for clarity. (The sign of the connection, however, is indicated by the form of the connection—e.g., dashed lines mean inhibitory connections). The four different activation values were generated by having relatively large weights of opposite sign. One input line turns the hidden unit full on, one turns it full off. The two differ by a relatively small amount so that when both turn on, the unit attains a value intermediate between 0 and 0.5. When neither turns on, the near zero bias causes the unit to attain a value slightly over 0.5. The connections to the second layer of hidden units is likewise interesting. When the hidden unit is full on,

TABLE 7

Input Patterns	Singleton Hidden Unit	Remaining Hidden Units	Output Patterns
10	0	1 1 1 0	0010
11	.2	1 1 0 0	0001
00	.6	.5 0 0 .3	1000
01	1	0 0 0 1	0100

<sup>3</sup> Relatively small learning rates make units employing intermediate values easier to obtain.

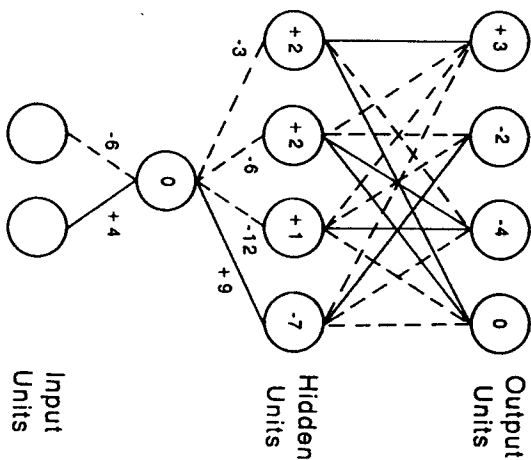


FIGURE 8. The network illustrating the use of intermediate values in solving a problem. See text for explanation.

the right-most of these hidden units is turned on and all others turned off. When the hidden unit is turned off, the other three of these hidden units are on and the left-most unit off. The other connections from the singleton hidden unit to the other hidden units are graded so that a distinct pattern is turned on for its other two values. Here we have an example of the flexibility of the learning system.

Our experience is that there is a propensity for the hidden units to take on extreme values, but, whenever the learning problem calls for it, they can learn to take on graded values. It is likely that the propensity to take on extreme values follows from the fact that the logistic is a sigmoid so that increasing magnitudes of its inputs push it toward zero or one. This means that in a problem in which intermediate values are required, the incoming weights must remain of moderate size. It is interesting that the derivation of the generalized delta rule does not depend on all of the units having identical activation functions. Thus, it would be possible for some units, those required to encode information in a graded fashion, to be linear while others might be logistic. The linear unit would have a much wider dynamic range and could encode more different values. This would be a useful role for a linear unit in a network with hidden units.