BACK PROPAGATION LEARNING EQUATIONS FROM THE MINIMIZATION OF RECURSIVE ERROR

Wayne E. Simon & Jeffrey R. Carter
Martin Marietta Astronautics Group
P. O. Box 179
Denver, CO 80201
(303) 977-3449

Abstract

The usual back propagation learning equations are equivalent to a gradient-descent solution to the problem of minimizing the output error of a neural network. As such, they exhibit a limited range of convergence and sometimes exhibit bizarre behavior if the starting point is far from a solution. The basic problem with a gradient-descent solution is that it has no information about the second derivatives of the error surface.

New learning equations are derived by minimizing recursive mean square error and formulated in the form of back propagation equations. Since even the new equations do not have an infinite radius of convergence, a learning technique is developed which insures that the network is never far from a solution.

Results for a simple exclusive-or exhibit robust learning at a rate about one hundred times faster than conventional back propagation learning. All problems tried were asymptotic in less than twenty-five epochs.

Introduction

Neural networks, or connectionist systems, have recently become very popular. Neural networks are intuitively attractive because they resemble important aspects of biological brains. The success of neural networks on such problems as speech recognition and handwritten character recognition has attracted the attention of such agencies as the Defense Advanced Research Projects Agency (DARPA) and the U. S. Postal Service, as well as companies such as AT&T.

A common approach to the use of neural networks is to train them using the standard back propagation learning equations. While this approach is responsible for many successes, it has a number of drawbacks. Back propagation shares all of the problems of simple gradient-descent solutions, including a limited convergence range and occasional bizarre behavior for certain initial configurations. It is very slow to converge, and may converge to a false minimum.

Standard Back Propagation Learning

The standard back propagation equations are sometimes called the "generalized Delta rule." The Delta rule was a learning technique which was used on perceptrons, and which mapped the errors of the output layer into adjustments of the weights on the connections between the input and output layers. This technique is not directly applicable to multi-layer networks. Rumelhart et al. [1] derived back propagation from the Delta rule by finding a way to propagate the output errors back to nodes in intermediate layers, thus allowing the Delta rule to be applied to the connections further back in the network. Once back propagation had been derived, it was shown to be an implementation of a simple gradient-descent solution to minimizing the output errors.

Back propagation suffers from the problems of all simple gradient-descent methods. These problems are largely due to the fact that such methods do not make use of information about the second derivatives of the error surface. Such information can be made available, and more powerful minimization techniques utilize it.

However, using the full second derivative information in a neural network would require a node to have information about other nodes to which it is not directly connected. The power of neural networks comes from their formulation as independent, concurrent nodes which exchange information only with nodes to which they are directly connected, so the use of all second derivatives would eliminate the independence of the nodes and the power of the neural network approach.

To overcome some of the limitations of simple gradient-descent methods, back propagation includes a learning rate constant, β, which is typically set to a value of about one tenth. This helps insure that the method will converge, but at the price of making such convergence very slow. Back propagation is a notoriously slow technique for realistically complex problems: A recent article reported convergence times of about 500,000 cycles, requiring overnight runs for a system with forty-four nodes [2]. For systems with thousands of nodes, back propagation will require weeks to converge, even with special hardware.

Back Propagation Learning by Minimizing Recursive Error

We first described the minimization of recursive mean square error in the context of mobile robot navigation [3]. That treatment used the full Hermitian in its solution and was not, therefore, directly applicable to neural networks. In this section we will show how the concept of minimizing recursive mean square error can be applied under the special restriction of a neural network: that changes to the value of a connection require only information about the two nodes which it connects.

The derivation of learning equations by minimization of recursive mean square error takes careful attention to detail. A distinction must be made between a function at the time of a particular

CH2767-2/89/0000-0155 $1.00 © 1989 IEEE

experience, and the argument of that function at the same time. Sometimes connections between specific layers of the network must be indicated, and sometimes connections in general must be indicated. In the end, an order-of-magnitude argument is used to discard most of the terms, and a careful definition of the meaning of error is used to make the learning equation for each connection identical and dependent only on the two nodes it connects.

Let the subscripts *i*, *j*, *k*, *l* indicate specific layers of a layered neural network: input, first layer, second layer, and output, respectively. Then w_{ij} is a connection from an input node to the first layer, w_{jk} is a connection from the first layer to the second layer, etc.

Let the subscripts *a*, *b*, *c*, *d* indicate unspecified layers of a network: w_{ab} is any connection in a network and w_{cd} is any other connection in a network.

Let the superscript *n* indicate the current experience of the neural network. Thus

Y_i^n = node state of an input node during the n^{th} experience.

Y_l^n = node state of an output node during the n^{th} experience.

D_l^n = desired state of an output node for the n^{th} experience.

The sigmoid node transfer function chosen is the hyperbolic tangent, so for

$\sum_k Y_k \cdot w_{kl}$ = input to the l^{th} node

$$Y_l = \tanh\left(\frac{1}{2}\sum_k Y_k \cdot w_{kl}\right) = \frac{1 - e^{-\sum_k Y_k \cdot w_{kl}}}{1 + e^{-\sum_k Y_k \cdot w_{kl}}} = \text{state of the } l^{th} \text{ node}$$

Note that the range of the hyperbolic tangent is from -1 to +1, rather than the usual sigmoid range from 0 to +1. This represents a scale factor and a zero shift and has no effect on the learning characteristics of the network. It is much more convenient for some later work which will add associative learning to the network and, for that reason, is defined in this manner for the present work.

Now a recursive mean square error can be defined

$$F_l^n\left(\vec{w}^n\right) = \left(\frac{P-1}{P}\right) \cdot F_l^{n-1}\left(\vec{w}^n\right) + \frac{1}{P}\frac{\left[Y_l^n\left(\vec{w}^{n-1}\right) - D_l^n\right]^2}{2}$$

P = characteristic length of recursive mean--memory of mean is reduced by the factor e (2.718...) by P following experiences

$\vec{w}^n$ = vector of all of the connections in a neural network after the nth experience

To minimize the recursive mean square error, take its first derivative (gradient) and solve for the changes in the connections to make the derivative equal to zero

$$\frac{\partial F_l^n(\vec{w})}{\partial w_{ab}} = \left(\frac{P-1}{P}\right)\frac{\partial F_l^{n-1}(\vec{w})}{\partial w_{ab}} + \frac{1}{P}\left[Y_l^n(\vec{w}) - D_l^n\right] \cdot \frac{\partial Y_l^n(\vec{w})}{\partial w_{ab}}$$

$$\frac{\partial^2 F_l^n(\vec{w})}{\partial w_{ab}\partial w_{cd}} = \left(\frac{P-1}{P}\right) \cdot \frac{\partial^2 F_l^{n-1}(\vec{w})}{\partial w_{ab}\partial w_{cd}} + \frac{1}{P} \cdot \frac{\partial Y_l^n(\vec{w})}{\partial w_{ab}} \cdot \frac{\partial Y_l^n(\vec{w})}{\partial w_{cd}}$$

$$\frac{\partial F_l^n\left(\vec{w}^n\right)}{\partial w_{ab}} = \frac{\partial F_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}} + \left(w_{cd}^n - w_{cd}^{n-1}\right) \cdot \frac{\partial^2 F_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}\partial w_{cd}} = 0$$

Now

$$\frac{\partial F_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}} = \left(\frac{P-1}{P}\right)\frac{\partial F_l^{n-1}\left(\vec{w}^{n-1}\right)}{\partial w_{ab}} + \frac{1}{P}\left[Y_l^n\left(\vec{w}^{n-1}\right) - D_l^n\right] \cdot \frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}}$$

but

$$\frac{\partial F_l^{n-1}\left(\vec{w}^{n-1}\right)}{\partial w_{ab}} = 0$$

is just the set of equations solved at the previous cycle, so

$$\frac{\partial F_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}} = \frac{1}{P}\left[Y_l^n\left(\vec{w}^{n-1}\right) - D_l^n\right] \cdot \frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}}$$

and

$$\left(w_{cd}^n - w_{cd}^{n-1}\right) \cdot \left[\left(\frac{P-1}{P}\right) \cdot \frac{\partial^2 F_l^{n-1}\left(\vec{w}^{n-1}\right)}{\partial w_{ab}\partial w_{cd}} + \frac{1}{P} \cdot \frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}} \cdot \frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{cd}}\right] =$$

$$-\frac{1}{P}\left[Y_l^n\left(\vec{w}^{n-1}\right) - D_l^n\right] \cdot \frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}}$$

Note that successive experiences are independent, so successive response states are also independent. The left term is just a recursive mean (over P experiences) of the product of first derivatives, second derivatives, and error of the output state with respect to different connections.

$$MEAN\left[\frac{\partial^2 F^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}\partial w_{cd}}\right] < MEAN\left[\frac{\partial^2 F^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}\partial w_{ab}}\right]$$

Thus the off-diagonal elements may be discarded and

$$\left(w_{ab}^n - w_{ab}^{n-1}\right) = \frac{-\frac{1}{P}\left[Y_l^n\left(\vec{w}^{n-1}\right) - D_l^n\right] \cdot \frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}}}{\left(\frac{P-1}{P}\right) \cdot \frac{\partial^2 F_l^{n-1}\left(\vec{w}^{n-1}\right)}{\partial w_{ab}^2} + \frac{1}{P}\left[\frac{\partial Y_l^n\left(\vec{w}^{n-1}\right)}{\partial w_{ab}}\right]^2}$$

Now let

$$E_l^n = D_l^n - Y_l^n\left(\vec{w}^{n-1}\right)$$

Simplifying the notation gives the learning equation

$$\Delta w_{ab} = \frac{\frac{1}{P} \cdot E_l \cdot \frac{\partial Y_l}{\partial w_{ab}}}{RM_P\left[\left(\frac{\partial Y_l}{\partial w_{ab}}\right)^2\right]}$$

where RM_P [...] = recursive mean of parameter in brackets with P as parametric memory

First, consider those connections which connect to an output node

$$\frac{\partial Y_l}{\partial w_{kl}} = \frac{\partial}{\partial w_{kl}}\left[\frac{1 - e^{-\sum_k Y_k \cdot w_{kl}}}{1 + e^{-\sum_k Y_k \cdot w_{kl}}}\right]$$

$$\frac{\partial Y_l}{\partial w_{kl}} = Y_k \cdot \left(\frac{1 - Y_l^2}{2}\right)$$

$$\Delta w_{kl} = \frac{1}{P} \cdot \frac{Y_k \cdot \left(\frac{1-Y_l^2}{2}\right) \cdot E_l}{RM_P\left[Y_k^2 \cdot \left(\frac{1-Y_l^2}{2}\right)^2\right]}$$

Defining a generalized error, E_l^*, and a generalized derivative, H_l^*, for the output nodes,

$$E_l^* = \left(\frac{1-Y_l^2}{2}\right) \cdot E_l, \quad H_l^* = \left(\frac{1-Y_l^2}{2}\right)$$

then the learning equation becomes

$$\Delta w_{kl} = \frac{1}{P} \cdot \frac{Y_k \cdot E_l^*}{RM_P\left[(Y_k \cdot H_l^*)^2\right]}$$

Now consider those connections which connect to the k-nodes, w_{jk}. Then

$$\frac{\partial Y_l}{\partial w_{jk}} = \frac{\partial Y_l}{\partial Y_k} \cdot \frac{\partial Y_k}{\partial w_{jk}}$$

$$\frac{\partial Y_l}{\partial w_{jk}} = Y_j \cdot \left(\frac{1-Y_k^2}{2}\right) \cdot \sum_l w_{kl}\left(\frac{1-Y_l^2}{2}\right)$$

The learning equation has the same form as that of the connections to the output if:

$$E_k^* = \left(\frac{1-Y_k^2}{2}\right) \cdot \sum_l w_{kl} \cdot E_l^*$$

$$H_k^* = \left(\frac{1-Y_k^2}{2}\right) \cdot \sum_l w_{kl} \cdot H_l^*$$

A notation for nodes in general can be written

$$E_c^* = \left(\frac{1-Y_c^2}{2}\right) \cdot \left[E_c + \sum_d w_{cd} \cdot E_d^*\right]$$

$$H_c^* = \left(\frac{1-Y_c^2}{2}\right) \cdot \left[H_c + \sum_d w_{cd} \cdot H_d^*\right]$$

This defines the propagation of error and derivative backward for all nodes. An output node has no forward connections, so the summations of propagated error and derivative are zero, while the local error is non-zero and the local derivative is one. Other nodes will have terms in the summation of propagated error and derivative, but, unless they also have a specified desired state, will have no local error or derivative.

Observe that the node error and derivative propagate away from the output and the node state propagates away from the input. The correction to a connection depends only on the state of the input-side node and the error and derivative of the output-side node:

$$\Delta w_{bc} = \frac{1}{P} \cdot \frac{Y_b \cdot E_c^*}{RM_P\left[(Y_b \cdot H_c^*)^2\right]}$$

Thus the state of a node depends only on the state of nodes connected to it, the error and derivative of a node depend only on its own values or the error and derivative of nodes to which it is connected (or both), and the learning rule changes connection values based only on the state of the input-side node and the error and derivative of the output-side node. Most learning equations use two parameters, α (momentum constant) and β (rate constant). This is equivalent to a recursive mean of the rate constant times the correction. Appropriate choices of constants are easier if the recursive mean is made explicit, or

$$\Delta w_{bc} = \frac{RM_Q\left[\frac{\beta}{P} \cdot Y_b \cdot E_c^*\right]}{RM_P\left[(Y_b \cdot H_c^*)^2\right]}$$

All of these derivations and calculations of the second derivatives of the error depend on the fact that the error is small; that is, that the particular values of the connections are very nearly the correct ones. This condition is satisfied by a simple trick. The initial response of the network to a particular experience is accepted as correct, and a sigmoid curve is used to smoothly change the observed response to the desired response as learning progresses. If the change in desired output is at least as slow as the learning itself, the network will always be near a solution, and, at the end of learning, will be near the desired solution.

Define:

$$G^n = G^{n-1} - \frac{1}{R}$$

= sigmoid variable for desired output transition

$D_{q,l}^{\infty}$ = desired output of the l^{th} output node for the q^{th} set of values of the input nodes

$D_{q,l}^{0}$ = initial observed output of the l^{th} output node for the q^{th} set of values of the input nodes

R = characteristic length of desired output sigmoid

Then

$$D_{q,l}^{n} = \frac{D_{q,l}^{-\infty}}{\left(1+e^{-G^n}\right)} + \frac{D_{q,l}^{\infty}}{\left(1+e^{G^n}\right)}$$

where:

$$D_{q,l}^{-\infty} = \left(1+e^{-G^0}\right) \cdot D_{q,l}^{0} - \frac{\left(1+e^{-G^0}\right) \cdot D_{q,l}^{\infty}}{\left(1+e^{G^0}\right)}$$

$Q' = S / R$ = distance to inflection point of sigmoid transition curve in terms of the characteristic length of sigmoid

S = distance to inflection point of sigmoid

Exclusive-Or Examples

Early work on training neural networks used the "exclusive-or" example because it is the simplest problem which is not linearly separable. The exclusive-or is an operator which takes two binary operands and produces a binary result. The operands and result may be considered to have values of zero and one. The result of the exclusive-or is one if exactly one of its two operands is one, and is zero otherwise. We will use the symbol ⊗ to represent the exclusive-or. Table I shows all of the possible combinations of operands and the result of applying the exclusive-or to them.

Table I
The Exclusive-Or Operator

A	B	A ⊗ B
0	0	0
0	1	1
1	0	1
1	1	0

Rumelhart et al. [1] describe a neural network with three input nodes (including the bias or threshold node, which always has an output of one), one intermediate node, and one output node. Figure 1 shows typical results of training such a system using the standard back propagation equations. Initial connection values were selected from a uniformly distributed random function with a range from -1 to +1. The top plot shows the error of the output node plotted against epochs, while the other four plots show the output of the system for each of the four input states. The system is fairly-well converged after 1,250 epochs, and output error is negligible after 2,500 epochs. The learning rate constant, β, had the unusually high value of two for this run. This high value was possible because of the simplicity of the exclusive-or problem. Figure 2 shows an input-output plot of the trained network. The abscissa is the value of the first operand and the ordinate is the value of the second operand. The crosshatching indicates areas where the output was less than 0.5. Since 0.5 is usually considered the dividing line between zero and one, the crosshatched areas can be considered areas where the output was zero, and the white area can be considered an area where the output was one. The training states for the network are located at the four corners of the figure. Note that this decision surface is not unique: Even with one intermediate node, other decision surfaces can be obtained which also correctly respond to the exclusive-or.

We made a number of runs of this problem to produce the results shown here. Although most had output error plots similar to that in Figure 1, some failed to converge in 2,500 epochs. For such a high value of β, at least, standard back propagation is sensitive to the initial configuration.

Figures 3 and 4 show the similar information for the same network configuration trained using the learning equations derived above. Since our equations allow node states from -1 to +1, we used input node states of -1 to represent operands of zero, and output states of -1 to represent a result of zero. Also, the crosshatching on Figure 4 indicates areas where the output was less than zero. The parameters used for the exclusive-or problem using our learning equations were $\beta = 2.0$, $P = 8$, $Q = 8$, $R = 8$, and $S = 0$. Initial connection values were specified with a uniformly distributed random function with a range from -0.1 to +0.1. The large value of β and the rapid transition to desired output specified by the values of R and S are possible because of the simplicity of the problem. Considering the extreme value of β used for the standard back propagation solution we felt that such values were necessary for a meaningful comparison. Output error is negligible after 25 epochs. Thus the network, using these equations, has learned the exclusive-or in about one-hundredth the number of epochs required for standard back propagation equations. The decision surface is similar to that obtained using standard back propagation equations.

As with the standard back propagation example, we made a number of runs of this problem. Unlike standard back propagation, all of these runs converged, despite the rapid transition of desired output. Our equations are absolutely convergent for a sufficiently slow transition of desired output.

Eight-State Example

The exclusive-or is an associative operator; that is,

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C = A \otimes B \otimes C$$

Table II shows all of the possible operands and results for A ⊗ B ⊗ C.

For a more difficult test of our equations, we taught a network to solve A ⊗ B ⊗ C. We used parameters of $\beta = 1.0$, $P = 16$, $Q = 4$, $R = 16$, and $S = 0$. The network had four intermediate nodes. The system's output error was negligible in less than 25 epochs, as shown in Figure 5.

This problem has one more operand than the simple exclusive-or, one more input node, and an additional dimension to its decision graph. Instead of having a decision surface, the results of this network have a decision volume. The x-axis is the first operand, the y-axis the second operand, and the z-axis is the third operand. Figure 6 shows the intersections of planes of constant z with this decision volume. As for Figure 4, the crosshatched areas have output values less than zero. Figure

6a, $z = -1$, is the same as a simple, two-operand exclusive-or. Figure 6j, $z = 1$, is the inverse of a two-operand exclusive-or.

Table II
Three-Operand Exclusive-Or

A	B	C	A ⊗ B ⊗ C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

For this example we used a more realistic number of intermediate nodes, half the number of input patterns, than for the simple exclusive-or example. This allows more rapid convergence and more complicated decision surfaces than when the minimum number of intermediate nodes are used.

Conclusion

These simple examples illustrate the power of the new learning equations. A true minimization of error is obtained, not just a gradient descent. Use of an order-of-magnitude argument to discard the off-diagonal elements of the second derivative matrix and careful definition of generalized error makes it possible for each node to be independent, requiring only knowledge of the node state of nodes connected to it and knowledge of the error and derivative of nodes to which it is connected. This, combined with the idea of a smooth transition from observed to desired behavior, should make practical the use of completely connected (feed-forward and feed-backward) networks in a fully parallel implementation.

References

1. Rumelhart, D., et al., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition; Volume 1: Foundations*, MIT Press, 1986.
2. Klimasauskas, C., "Neural Nets and Noise Filtering," *Dr. Dobb's Journal*, Volume 14, Issue 1, 1989 Jan.
3. Simon, W., and J. Carter, "Mobile Robot Navigation with Ambiguous Sparse Landmarks," W. Wolfe and W. Chun, Editors, Proc. SPIE 852, p. 264, 1988.

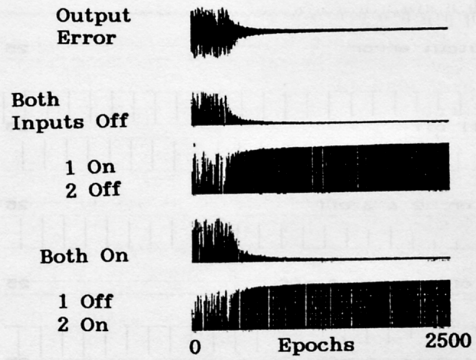


Figure 1. Learning Exclusive-Or with Usual Back Propagation

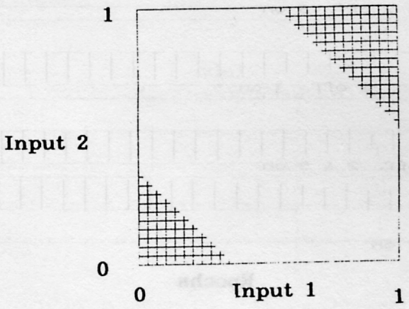


Figure 2. Decision Surface from Usual Back Propagation

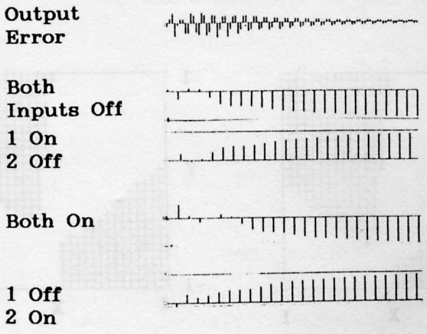


Figure 3. Learning Exclusive-Or with New Back Propagation

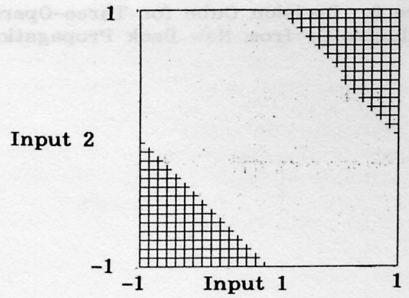


Figure 4. Decision Surface from New Back Propagation

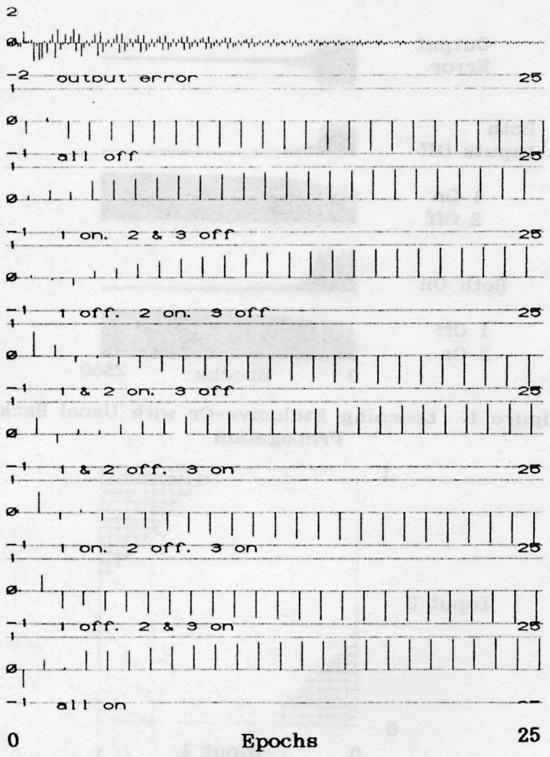
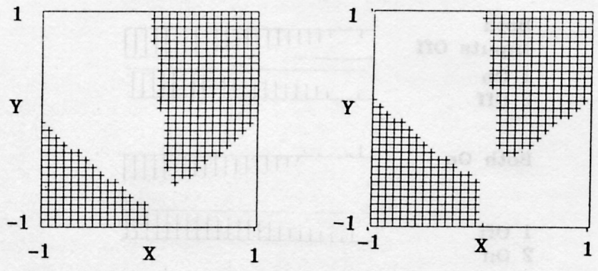
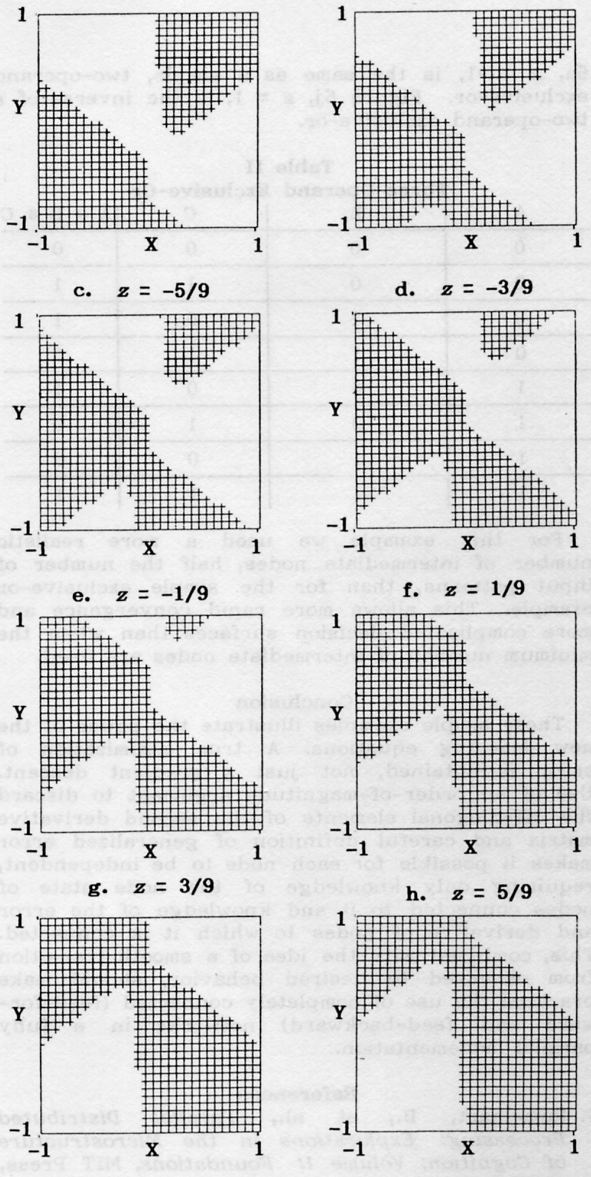


Figure 5. Learning Three-Operand Exclusive-Or with New Back Propagation



a. $z = -1$ b. $z = -7/9$

Figure 6. Decision Cube for Three-Operand Exclusive-Or from New Back Propagation



c. $z = -5/9$ d. $z = -3/9$

e. $z = -1/9$ f. $z = 1/9$

g. $z = 3/9$ h. $z = 5/9$

i. $z = 7/9$ j. $z = 1$

Figure 6. (continued)