Title
NEURAL NETWORKS AND THE HUMAN GENOME PROJECT

Authors
Smith, S.D.G.
Hutchinson, M.S.
Flies, M.A.

Publication Date
1989-02-01

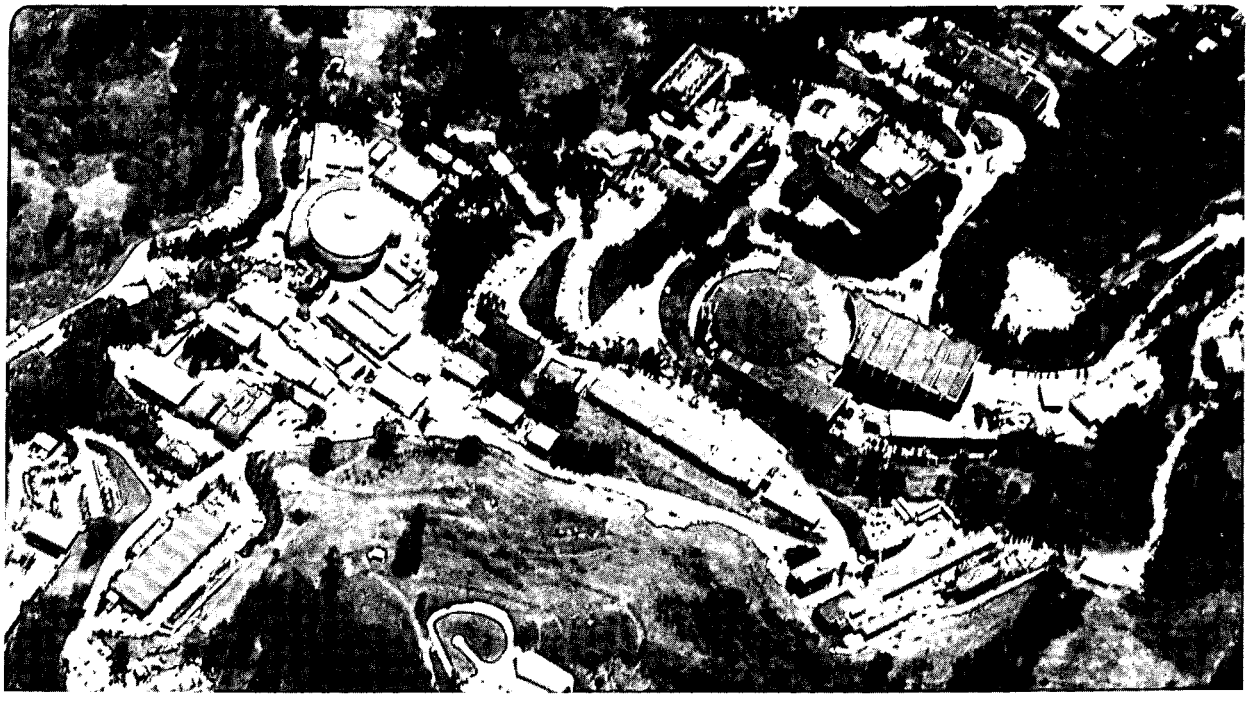
Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Information and Computing Sciences Division

Neural Networks and the Human Genome Project

S.D.G. Smith, M.S. Hutchinson, and M.A. Flies

February 1989



Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

DISCLAIMER

LBL-25981

Neural Networks and the Human Genome Project

Scott D.G. Smith, Marjorie S. Hutchinson, and Mary Ann Flies

Advanced Development Projects
Information & Computing Sciences Division
Lawrence Berkeley Laboratory
One Cyclotron Road
Berkeley, CA 94720

February 1989

This work is supported by the Office of Health and Environmental Research Program of the Office of Energy Research, U.S. Department of Energy under contract number DE-AC03-76SF00098.

Table of Contents

1. Introduction

1.1 Neural Networks

Artificial neural networks were originally developed as an attempt to model the workings of the brain. Although these models have no where near the complexity of the brain or its ability, they nevertheless exhibit some interesting features that make them interesting tools for studying some of the problems that have so far proved intractable with more traditional techniques. They are currently being investigated for use in those problems requiring recognition and classification of complex patterns such as image feature extraction, speech recognition, or natural language understanding.

A neural net consists of a number of independent processing nodes (analogous to neurons) interconnected with weighted links (analogous to strength of synaptic signal). Each node executes in parallel, summing its weighted inputs and "firing" (i.e. putting a signal on the output links) whenever the weighted sum exceeds a threshold. The network "learns" by modifying the weights on the links in ways that bring the network closer to the desired response. In the typical network, after many training iterations it will have learned the desired response to the training set of data, and may have formed generalizations of the patterns that allow it to classify new and different examples using the same generalizations - often in spite of missing data or noisy data. Good introductions to neural nets are the article by Lippmann [Lip87] and the book by Rumelhart and McClelland [RMt86].

1.2 Brief Objective of Project

Many of the computing problems involved in the human genome project fall into the category of problems that a neural network might be expected to solve, including: the prediction of splice junctions in DNA sequences, the prediction of secondary and tertiary structures of proteins, and the categorization of gels. This project was a summer undergraduate computer science project under the direction of the second author. The

objective of this 10 week project was to explore how neural networks operate, what things affect this operation, and where they can play a role in the human genome project. The neural network software was written or modified by the first author. The third author was responsible for the programs to access the Genbank DNA sequence databank.

2. The Splice Junction Problem

For the purpose of exploring the operation of neural networks and determining what factors influence their performance, we concentrated on the splice junction problem.

2.1 Description of the problem

The splice junction problem involves determining the boundaries between the exons, those parts of the DNA that encode for protein, and the introns, where protein is not encoded. We attempted to find these boundaries by exploiting the pattern matching capabilities of neural networks. To simplify the problem we concentrated only on 5' termini (left hand boundaries) which contained the 'GT' marker.

K. Nakata, M. Kanehisa, and C. Delisi worked on this problem using the perceptron [NKD85]. Their best result predicting human exon/intron boundaries using a human training set was 91.4 percent. This was obtained using 42 true sequences and 58 false sequences for training and a window of 25 bases to each side of the boundary. When they included periodicity information, in addition to the perceptron, the degree of prediction increased to 92.4 percent.

We began by exploring the use of backpropagation which is the most widely used type of neural network for pattern matching. For implementing this type of network, we used the backpropagation package provided in the Rochester Connectionist Simulator (RCS) [GLM88]. In hopes of overcoming the inherent slowness of the backpropagation learning algorithm and to explicitly capture any higher order

correlations which may exist in the problem we decided to also explore the use of higher order networks [MG87,MGY87].

2.2 Using Backpropagation

Backpropagation neural networks have an input layer, one or more middle (hidden) layers, and an output layer of nodes. Each node of a layer is connected to every node in the layer directly above. Training is accomplished by setting the input nodes, letting the values propagate forward to the output nodes, comparing the output to the expected output, and propagating the errors back and changing the weights of the links to hidden and output layers. The hidden layer(s) allows backpropagation networks to capture nonlinear correlations in the patterns.

The Rochester Connectionist Simulator is a neural network simulator written in C for Sun workstations. It provides routines which the user calls, generally from within a C program, to construct and operate a neural network. Links may be represented with either integer or floating point values. When integer values are used a graphical interface is available which is useful for investigating how the simulation proceeds or for debugging the network configuration. However, this interface decreases the speed of the network considerably. The Rochester software provides procedures for the backpropagation network to propagate the output errors back through the network layers. The error propagation function used by RCS is shown in figure 1.

Starting configuration

Four input nodes were used to encode each base, one each for Guanine, Adenine, Cytosine, Thymine. Therefore, a window of five bases to either side of the boundary would have an input layer of forty nodes. This is referred to as a "one of N" input scheme in which only one input is set out of each group of N (see figure 2). A number of researchers, including Robert Hecht-Nielsen, have claimed that this is more effective than a coded input representation because it makes the information available to the network

more explicit.

We used the following default conditions when testing the effects of different changes: integer values for links, training rate of 1, momentum of .5, temperature of 1, window of 3 to the left of the boundary and 6 to the right, 1 hidden layer with 7 nodes, and a training set of 200 of each true boundaries and spurious analogies. The meanings of all the network parameters are discussed below and in greater detail in the RCS manual [GLM88]. We trained the network on equal numbers of true and false boundaries under the assumption that this would produce the most unbiased results.

With no changes to these parameters the network's best performance classified 92.6 percent of the true boundaries as true and 91.6 percent of the false boundaries as false. However, generally it did not do this well. To test the effect of each of the different parameters we varied one while keeping the others the same.

Effect of initial weight values

We started with the link weights set to random initial values between 200 and 800 (.2 and .8 after scaling by 1000, which the integer version of the RCS does). This is what the example provided in the RCS backpropagation package uses. However, the network would not learn and always gave the maximum activation of the output nodes regardless of the input, even after training. Examining the link values before and after training revealed that they did not change. The reason for this could be that for a network with a window of 3 to the left and 6 to the right, each hidden node had 36 inputs with 9 of them set. With all of the weights being positive each hidden node had an output of one. So, if there were 7 hidden nodes the output layer would receive 7 inputs each with a positive weight between .2 and .8 and so would give the maximum output possible. The reason that the backpropagation example provided worked is that it had only one input set at a time.

Switching the initial link values to zero allowed the network to learn. We later tried setting the links to random values between -500 and 500. The change in training

speed and output performance as a result of this change was uncertain, however, most researchers suggest the use of random initial weights so that is what we used.

Effect of the window size

Window size refers to the number of bases to either side of the boundary which are presented as input to the network. Since window size controls the amount of information available to the network, it seems likely that it would have a large effect on the performance of the network.

A number of researchers have proposed the existence of a consensus sequences for the donor site (5' end of the intron) which occurs from 3 bases left to 6 bases right of the boundary. It was for this reason that we concentrated on a window of that size. However, using a window that small restricts the amount of information available to the network. In hope of increasing the performance of the network by expanding the amount of information available to the it, we tried using a window of 10 bases to each side of the boundary. However, the results using this window were poorer than before (see figure 3). This may have been due to an increase in the amount of irrelevant information which the network could not effectively deal with.

The main drawback to using small windows is that the bases which are a large distance from a 'GT' marker may affect whether or not it is a true exon/intron boundary. However, attempts to use the backpropagation package provided with the RCS on large windows of 50 bases to either side of the boundary were difficult due to the slowness of the program. Even when not using the graphical interface it took approximately five minutes to train for one iteration on a set with 200 true and 200 false boundaries. We were able to train a network for 100 iterations with a window of 50 to either side. As figure 3 shows, the results were still poorer than when using a window of 3 to the left and 6 to the right.

Effect of size of the training set

Increasing the size of the training set produced substantial improvements in the performance of the network. Figure 4 shows the plots comparing the results for various training sizes. The graphs clearly show that a network trained on a set of 200 of each true and false boundaries performs better than one trained on a set of 100 of each. It is a little more confusing with larger training sets. A network trained on a set of 400 appears to do better with catching false boundaries, but there is no clear improvement in recognizing the true boundaries. It is yet more confusing with a training set of 600 (not shown on the graph). The system stabilizes to a result that is no better than the smaller training sets. Yet, in the first 50 iterations of training the results appear to be considerably better. Therefore, it appears that training with larger sets improves the performance of the network. However, there must be a point where increasing the size of the training set will not help, but it is difficult to tell from our results where that point is or if it has been reached.

Effect of varying equation constants

There are three main constants in backpropagation training: temperature, momentum, and training rate. We tried varying these to see what effect they had on the ability of the network to learn.

The temperature constant determines the slope of the sigmoid curve. The weighted sum activation is divided by the temperature before the exponential is taken. The default value for it is 1. When we used a network trained with a temperature of 0.7 the percent false found false improved very slightly (see figure 5). However, the percent of the true splice junctions found true declined. Furthermore, a network trained with a temperature of 0.1 found all possible boundaries false even when they where actually true boundaries.

The effect of the momentum term is, as the name implies, to give a preference for the current weight change to be in the same direction as the last change. The purpose

of this is to prevent the network from becoming stuck in a local minimum. The default value for the momentum is 0.5. Reducing the momentum to 0.3 produced an improvement of around one to two percent in the percent false-found-false (see figure 6). The change in the percent true-found-true is not obviously better. However, further decreasing the momentum to 0.1 gave results equal to or poorer than the default value. Also, increasing the momentum to 0.7 produced little change in the percent of false boundaries found false but apparently decreased the performance on the true boundaries. The significance of the improvement when using the momentum of 0.3 is questionable because of the large variations in the performance during training.

The training rate constant (BPlearn for RCS) controls the speed with which the network converges. The error propagation signal is multiplied by the training rate before it is propagated. BPlearn has a default value of 1. Setting the training rate to 0.5 produced little change in the results (see figure 7). The best results at a training rate of 0.1, on the other hand, was a slight improvement in both the percent true found true and the percent false found false. However, the overall change is uncertain, with improvements occurring in the percentage of correctly identified false boundaries being accompanied by decreases in the the percentage of correctly identified true boundaries. When we tried using a training rate of 0.01, the network was very slow in training and after 550 iterations of the training set had made little progress.

Effect of the number of hidden nodes

The number of hidden nodes affects the ability of the network to generalize and the performance of the trained network. Unfortunately, there is no good way to predict what the optimal number of hidden nodes for a specific problem is going to be. It has been suggested that the log of the number of inputs is a good approximation. This was why we started with 7 hidden nodes. In order to test if this was the optimal number we kept all other conditions the same and varied the number of hidden nodes. As figures 8 and 9 indicate, there does not seem to be a number of hidden nodes which produces

superior results. In fact, using zero hidden nodes does not appear to hurt the performance. This indicates that there is no higher order information which this type of network can utilize.

Using floating point values

Robert Hecht-Nielson has claimed that backpropagation can not perform well unless floating point values are used in the simulation instead of integers. In order to test the effect using floating point values has on the performance, we kept all of the parameters constant but used fsim (the floating point version of RCS) instead of sim. Training when using floating point values was extremely slow, taking thousands of iterations before the results stopped changing (graph of results not included). The changes in performance were continuous and gradual, unlike the drastic variations which occur when using integer values. However, the performance with networks using floating point values was poor even after training for large numbers of iterations.

Combination of best parameters

Attempts to utilize a combination of the best parameters to improve the performance of the network produced mixed results. We first tried a network with the training rate changed to 0.1, the momentum to 0.3, and the temperature to 0.7. After training on a set with 200 of each true and false boundaries for 600 iterations it classified 93.5 percent of the true boundaries correctly and 92.3 percent of the false boundaries correctly. This is an improvement over the performance using the default values. However, we next tried a network with a training rate of 0.1, a momentum of 0.3, a temperature of 0.5, and a training set of 600 of each true and false boundaries. The performance with this network was worse than when using the default values.

Conclusion on backpropagation

The best result we obtained was 93.5 percent for the true boundaries and 92.3 percent for the false boundaries. The most important factors in the performance of the

networks seems to be the window size and the size of the training set, although a proper combination of the equation constants is also important. The number of hidden nodes, or even if any hidden nodes are used, does not seem to be of great importance. This probably indicates that the only information which could be extracted was first order.

As many of the graphs show, the results often start out good, but decline in performance after continued training. This could be due to the networks being overtrained and memorizing unique features of the training set.

Summary, theory, and initial configuration

The hidden nodes used in a backpropagation network allow it to capture higher order correlations. However, because assigning weights to these hidden nodes is inherently inefficient, learning with backpropagation is very slow. Higher order networks are built from nodes which explicitly capture higher order terms, thereby eliminating the need for hidden units. Because hidden units are no longer needed, more efficient learning algorithms can be used [MG87, MGY87]. The hope is that this would result in increased learning speed and better performance in testing. Unfortunately, the RCS could not store weights for higher order terms, so we wrote our own simulation software for higher order networks.

The equation we used for calculating the output is as follows:

$$y_i = \sum_{j=1}^{j=N} W_1(i,j)x(j) + \sum_{j=1}^{j=N}\sum_{k=j+1}^{k=N} W_2(i,j,k)x(j)x(k) + \cdots$$

N is the number of inputs, x(j) and x(k) are the values for inputs j and k, and y(i) is the output for the ith higher order node. W_1(i,j) is the first order weight for the ith output and jth input. W_2(i,j,k) is the second order weight for the ith output and the pair of inputs i and j. A node is taken to be set if its output is greater than the constant TestsTrueThresh, which is normally zero.

For testing the ability of higher order networks in discriminating exon/intron boundaries we used a single higher order neuron. If this neuron gave an output greater than TestsTrueThresh then the input was classified as a true boundary, otherwise it was classified as a spurious analogy. We restricted the network to using only first and second order terms in order to reduce the number of weights needed. Two different learning rules are used with for this type of network, which we call delta change and percent change.

Delta change learning rule

The first learning rule tried, an extension of the perceptron learning rule, involved changing the weights by a constant delta. During training, if the output for a node was less than a constant TrueThreshold, but was supposed to be true, then the weights are changed according to the following function:

$$W_1(i,j)' = W_1(i,j) + x(j)delta$$

$$W_2(i,j,k)' = W_2(i,j,k) + x(j)x(k)delta$$

If the output for a node was greater that a constant FalseThreshold, but was supposed to be false then the above function is used, but delta term is subtracted instead of added. The values initially assigned to the constants delta, TrueThreshold, and FalseThreshold were 100, 1000, and -1000 respectively. The link weights were initialized to zero. A window of 3 to the left of the boundary and 6 to the right was used for testing purposes.

The value used for the constant delta turned out to be very important. As figure 10 shows, the smaller the delta used, the better the percent false found false. However, on the percent true found true, training with a delta of 10 did better than with a delta of 1.

Similarly to the backpropagation experiments, using a windows of 10 and 50 nodes to either side of the boundary produced poorer results (see figure 11). Also, it appears that a larger training set size produces better results, but it is less certain than with backpropagation (see figure 12).

As the above equations show, the weights for this type of network are only changed if the output was less than TrueThreshold for a true training example or if the output was greater than FalseThreshold for a false training example. If these conditions are never met then the network is distinguishing between true and false training examples 100 percent of the time and the weights will not change if training is continued. The number of iterations for the network to converge in this way changes based on the

parameters of training. Smaller values of delta increase the number of iterations needed for a network to converge, while larger windows decrease it.

Using only first order terms (which corresponds to using a perceptron) actually improved the performance of the network. The best performance was with a training set of 600 and a window of 3 to the left and 6 to the right of the boundary. This network classified 95.3 percent of the true boundaries as true and 92.2 percent of the false boundaries as false. This is a large improvement over the back propagation results but the absence of second order terms indicates that there is probably no higher order information available in this problem which the network can distinguish.

Percent change learning function

The percent change learning rule involves moving a certain percent of the distance to the correct output each time. This percent is controlled by the constant LearningRate. If the output for a node was supposed to be true, but was less than TrueThreshold then the weights are changed according to the following function:

$$W_1(i,j)' = W_1(i,j) + x(j)*LearningRate*(TrueThreshold - W_1(i,j))$$

$$W_2(i,j,k)' = W_2(i,j,k) + x(j)*x(k)*LearningRate*(TrueThreshold - W_1(i,j,k))$$

The same function is used if the output is supposed to be false and was greater than FalseThreshold, except the constant FalseThreshold is used in place of TrueThreshold.

As figure 13 shows, networks using this learning rule produce testing results with very large variations. The testing percentages do not appear to converge, at least in the first 500 iterations. The graphs also show that smaller values for the LearningRate produce less variations in the output performance, with the exception of a LearningRate of 1 which produced a constant performance which was very poor. The variation is even worse when one considers that even after 500 iterations the performance can change by 15 to 20 percent in one iteration for a network with a LearningRate of 0.5.

Because of the drastic variations in the performance of this learning rule, comparing it with the performance of the delta change learning rule is difficult.

However, it does not appear to produce better predictive results.

Conclusion on higher order networks

The higher order network did not produce better results than using backpropagation as was hoped it would. This may be due to a lack of higher order correlations in the problem. This hypothesis is supported by the result that using only first order terms improves the predictive ability of the network. However, one advantage of higher order networks over backpropagation is that the weight values produced in a higher order network can readily be interpreted to determine the relative importance of different bases in distinguishing between true and false boundaries. Two output nodes, one for each true and false boundaries, should be used if this is done. Such interpretation of weights is extremely difficult with backpropagation.

2.4 Input Data Considerations

We have two main considerations about the validity of the sample sequences used for training and testing.

Use of duplicate sequences

Sometimes GenBank entries contain repeats of the same sequence for one reason or another. If a sequence was repeated for the same gene entry, with the same boundaries, we only included it once. However, other sequences were listed twice, under different genes, with different boundaries. We don't know whether these were the same gene listed twice, or just a similar sequence naturally occurring twice. In the training file used for a window of 50 to each side we removed these, and noticed that they accounted for about 10% of the sequences. In the training files used for the networks with a window of 10 to each side and the ones used for the networks with a window of 3 to the left and 6 to the right we did not remove them. So, these networks may have been doubly trained on as many as 10% of the sequences. Further study of GenBank entries on this

point is recommended.

The testing file contained six of the boundaries on which the network had been trained when the training set size was 400 or greater. To our knowledge, no other sequence used for testing a network was contained in the file on which that particular network was trained. Every effort was made to insure that overlapping sequences were not used. However, the files were not compared for such duplicates as described in the previous paragraph.

Statistical accuracy of results

While we quote exact percents for the results in this paper, it should be noted that there was large variations in the testing results while the network was training. In addition, the sequences chosen for inclusion in the test sets may not be random enough. They were selected from at least five different GenBank/primate files in each case, but they only contained a total of 170 true Exon/Intron boundaries. We don't know whether this represents a truly random sample representative of all sequences.

2.5 Future Work On the Exon/Intron Problem

Boundaries not marked by 'GT'

A number of researchers have noted that there are exon/intron boundaries which do not have the 'GT' marker. There are three reasons why identification of these "non GT" splice junctions probably should not be attempted until other improvement have been made.

The first is that the number of these boundaries is likely to be small. In the testing set which we used there were only three boundaries which did not have the 'GT' marker. An important thing to notice is that this is far smaller than the number of true boundaries which the network classified as false. So, searching for boundaries without the 'GT' marker makes little sense until the percentage of true found true approaches

100.

Another consideration is that if we test all sequences the time to test a sequence will be around sixteen times as great since the network must test every positions not just those marked by 'GT'. So, there needs to be increases in the speed of the network before this can be attempted.

Finally, the number of false boundaries found true is likely to be 16 times as great, while the number of true boundaries found will only increase by a few. So, the number of false boundaries found as true will likely greatly overwhelm the true boundaries. Therefore, this problem should probably not be attempted unless there are large reductions in the number of false boundaries found true.

Searching for right hand boundaries

Finding the right hand boundary (3' termini) should be very similar to the problem described above of finding the left hand boundary. However, instead of looking for the 'GT' marker after the boundary, one should look for an 'AG' marker before it. Unfortunately, there may be a larger number of left hand boundaries without 'AG' than there are right hand boundaries without 'GT'. One can either train a network which will recognize both left and right hand boundaries (set up a new output for the right hand boundaries) or train different networks for recognizing each type of boundary. Both methods should probably be tried to see which gives the best result.

Further Work with Backpropagation

The number of hidden nodes used had a large effect on the performance of the network. However, all we were able to do was selectively test certain numbers of hidden nodes and compare the results. This gives only a gross approximation of the optimal number of hidden nodes. However, J. Sietsma and R.J.F. Dow described a method of pruning a network to get the smallest number of hidden nodes required for the problem [SD88]. Using their method may result in a more efficient network which produces

better results.

Backpropagation networks require two hidden layers to capture arbitrarily complex functions. Because of this we attempted to use two hidden layers with 7 nodes each (see figure 14). However, we got no improvement in the results in spite of suggestions by some researchers that an improvement of a couple of percent could be obtained in some problems. In addition after training for 400-600 iterations, the networks we used with two hidden layers would suddenly start finding all positions with a 'GT' marker as true boundaries even when they were not. We advise exploring the use of two hidden layers further to try to explain these results. Also, it has been suggested that training of two hidden nodes could be accomplished by first training one layer while keeping the weights on the other constant, then training the second layer while keeping the first locked.

The results for backpropagation might also be improved by training with a better combination of backpropagation constants. While the combinations which we tried gave mixed results, those may not be the best possible values. Using the floating point version with different constant values should also be tried. In addition, the effect of the bias unit, which is another parameter in backpropagation, should be explored.

Further Work with Higher Order Networks

Position invariant networks give the same result regardless of the location of a pattern in the input space. The output function used by a higher order position invariant network is:

$$y_i = W_1(i)\sum_{j=1}^{j=N} x(j) + \sum_{dj=1}^{j=M} W_2(i,dj)\sum_{j=1}^{j=N} x(j)x(j+dj) + \cdots$$

The advantage of a position invariant network is that the feature of interest does not have to be centered in the window. This could be especially valuable in the exon/intron problem since some of the boundary indicators may not occur in a specific location with reference to the boundary, but instead may just be near it. Thus, position invariant networks may be very effective at testing if an exon/intron boundary existed in a certain

region of say 100 bases. If this gave very accurate results then it could also be used to cut down on the number of false boundaries found true when using the networks described above. Also, testing would be faster since the size of the window would control how often a sequence would have to be tested (i.e. every 100 bases for a window of 100). It would also be able to find those boundaries which do not have a 'GT' marker.

Terrence Sejnowski suggested in a conversation with one of us that, since three DNA bases encode for one codon, it may be that third order terms have significance in the determining if a exon/intron boundary actually occurs at a 'GT' marker. However, our results with the number of hidden nodes and the good performance when using only a first order perceptron suggest that there is no significance to higher order terms. In addition, the number of weights that including third order would require is very large in the general case. However, weights could be stored for only consecutive triples since codons are three consecutive bases. Thus, only 3N-2 weights would have to be added, where N is the size of the window. Similarly, it might improve the results to only store second order weights for consecutive pairs since this could reduce the amount of irrelevant information.

We have not demonstrated that either of the higher order learning algorithms which we used are mathematically equivalent to the one used by T. Maxwell, C. Giles, and Y. Lee. This should be attempted and if they are not equivalent then the one they used should be tried.

Miscellaneous things to try

Early results indicated that the networks trained on equal numbers of true and false boundaries had difficulty in classifying false boundaries as true. So, we decided to train networks on the entire sequence rather than equal numbers of true and false boundaries. That procedure worked by reading in the positions of the true boundaries, scanning the sequence and for each position with a 'GT' marker, checking if it is a true

or false boundary and training accordingly. Assuming random distribution the sequence 'GT' should occur every sixteen bases. Since donor sites do not occur with nearly this frequency, the network is trained on many more false boundaries than true ones. It was hoped that this would reduce the number of false boundaries which were found to be true by the network. The few results that we had on this came from using both small training and test sets. They seemed to indicate an improvement in the percent false found true. This should be explored further. However, it seems likely that the performance on true boundaries would decline.

As figure 4 shows, the size of the training set has a large effect on the performance of a backpropagation network. While the effect of increasing the size of the training set from 400 to 600 of each true and false boundaries is unclear it seems likely the a much larger training set, one of a few thousand boundaries, would improve performance.

One way in which the performance of the network may be improved is to set all those weights whose absolute value is less than a small threshold to zero. This is done because small weights can hurt the performance of a network. We wrote a procedure to do this for higher order networks and preliminary testing indicated that it can improve the performance slightly. Zeroing of the weights in the higher order and backpropagation networks should be tried further.

2.6 Conclusion and Summary of Results

The best results we obtained were using only first order terms with the higher order network. This corresponds to using a perceptron. The network was trained on a set of 600 of each true and false boundaries and used a window of 3 bases to the left and 6 bases to the right. It classified 95.3 percent of the true boundaries as true and 92.2 percent of the false boundaries as false. This compares with the 91.4 percent obtained by K. Nakata, M. Kanehisa, and C. Delisi [NKD85] using a perceptron with a window of 50 bases to each side. The difference between the their best result and ours is probably

due to the larger training set and smaller window size we used. The fact that the result obtained using only first order terms were at least as good as those using backpropagation indicates that there are no higher order correlations of use in this problem. This is further supported by the observation that the number of hidden nodes seemed to have little effect with backpropagation.

3. Other Applications in the Human Genome Project

3.1 Secondary Structure of Proteins

Determining the structure of proteins using traditional means, such as X-ray crystallography can be very time-consuming. Therefore, predicting secondary structure of proteins from their primary amino acid sequence is of great interest to structural biologists. Ning Qian and Terrence J. Sejnowski have demonstrated the applicability of neural networks to this problem [QS88]. They obtained an accuracy of 64.3% using a backpropagation network with a window of 13 bases and 40 hidden nodes. However, they also estimated that a maximum accuracy of 70% is possible using local information. This could be tested by seeing if a higher order network does any better than a perceptron.

One possibility for improving the accuracy is to have feedback from a network which determines the localized tertiary structure into the input of the network determining secondary structure. It makes sense that this would improve performance because the existence of a certain tertiary feature may affect the secondary structure at that point.

3.2 Tertiary Structure of Proteins

One of the most straightforward ways of predicting tertiary structure is to use a backpropagation network with the primary amino acid sequence as the input and the distances between each amino acid as the output. However, there are a number of

drawbacks to this idea. One problem is that this involves a very large number of nodes and links since the number of outputs is the square of the number of amino acids in the input. Also, we have doubts as to whether a backpropagation network can perform a mapping from an input space to a higher dimensional output space. Finally, it would be extremely difficult if not impossible to reconstruct the tertiary structure of the protein from the distances between the amino acids. The angles as well as the distances between amino acids could be included in the output to make reconstruction of the tertiary structure easier. However, this would increase the dimension of the output.

An alternate approach is to test proteins for the existence of particular active sites. One possible way to do this is to use a position invariant higher order network with the amino acid sequence of the entire protein as the input. Position invariance is helpful because the sequences making up the active site may be from entirely different parts of the protein and the locations of them may vary from protein to protein.

Another method is to look for certain localized tertiary or super-secondary structures. This could be done with a similar method to that used by Sejnowski when he worked on secondary structure with backpropagation networks. However, a larger window would probably be needed. Perceptrons and higher order networks could also be tried.

3.3 Gel Categorization

Gels are used by biologists to determine the lengths of protein fragments. The farther the fragment moves through the gel the smaller the fragment is. The fragments show up on the gels as bands in different columns. What is needed is a way to group together those gels in which the band patterns are similar. The problem is that the patterns may be shifted, compressed, or deformed in other ways. So, networks which use a pixel by pixel comparison to classify pattern, such as Hebbian, Hopfield, and Grossberg networks, will not work well for this problem (See [Lip87] or [RMt86] for information about these other types of networks). In that sense the problem is similar to

recognizing handwritten characters, so using the Neocognitron [Fuk88] may be a good approach. However, a faster simulation than RCS or hardware networks are needed before that complex a network can be used practically.

One gel recognition problem is to recognize a characteristic gel containing only 1 or 2 bands per column. It should be possible to attack this problem using traditional pattern recognition networks, such as backpropagation.

Appendix -- Using the Programs Written for the Project

Programs written for Rochester software

The Rochester software is executed by typing "sim" for the integer version, "fsim" for the floating point version, and "gsim" for the graphics simulator. The procedures we wrote are executed by typing "call" and the command line for the procedure. For example to build a network called "IE" type "call build IE". It is useful to turn off the message that prints the number of cycles executed by using the command "echo off". The files "build" and "build5" are used with "gsim" by using the command "read". These files build the network, display the nodes, and set the mouse keys.

Programs written for higher order networks

The program HigherOrder simulates higher order neural networks Running it is mostly self explanatory. The program must be edited and recompiled to change the size of the window, the number of outputs, or the order of the network. To turn on the printing of the diagnostic messages type "set PrintDiagnostics 1".

References

[Fuk88] K. Fukushima. Neocognitron: a hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988.

[GLM88] Nigel H. Goddard, Kenton J. Lynne, and Toby Mintz. *Rochester Connectionist Simulator*. Technical Report TR233, University of Rochester Computer Science Department, Rochester, New York, 1988.

[Lip87] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP*, 4–22, April 1987.

[MG87] T. Maxwell and C. Giles. Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978, December 1987.

[MGY87] T. Maxwell, C. Giles, and Y.Lee. Generalization in neural networks: the contiguity problem. In *Proceedings of the First Annual IEEE Neural Network Conference*, pages 41–46, San Diego, Ca., 1987.

[NKD85] K. Nakata, M. Kanehisa, and C. DeLisi. Prediction of splice junctions in mRNA sequences. *Nucleic Acids Research*, 13(14):5327–5341, 1985.

[QS88] N. Qian and J. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, 202:865–884, 1988.

[RMt86] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Volume 1, MIT Press, Cambridge, Massachusetts, 1986.

[SD88] J. Sietsma and R.J.F. Dow. Neural net pruning - why and how. In *Proceedings of the IEEE Second Annual Conference on Neural Networks*, pages I325–I333, July 1988.

Code for Rochester Backpropagation Error Function

```
FLINT
UFh_o(up)
Unit     *up;
{
     FLINT          delta,
                    deltaw;
     Link           *lp;

/**----------------**
 ** activation code **
 **----------------**/
     if(TestFlagP(up, FORWARD_FLAG))
     {    up->potential = up->output = up->sites->value;
          BPendfwd(up);
     } else
/**------------------------**
 ** error-propagation code **
 **------------------------**/
     {
          delta = (BPlearn * (up->sites->data) * (up->output) *
                   (BP_ONE - up->output))/(BP_ONE * BP_ONE);
          for(lp = up->sites->inputs; lp != NULL; lp = lp->next)
          {    deltaw = ((delta * *(lp->value))/BP_ONE)
                        + BPmomentum * lp->data;
               lp->weight += deltaw;
               lp->data = deltaw;
               *(FLINT *)(lp->link_f) += (delta * lp->weight)/BP_ONE;
          }
          BPendrev(up);
     }
}
```

Figure 1

Code used for the Rochester backpropagation error function. Shows the use of the constants BPlearn and BPmomentum.

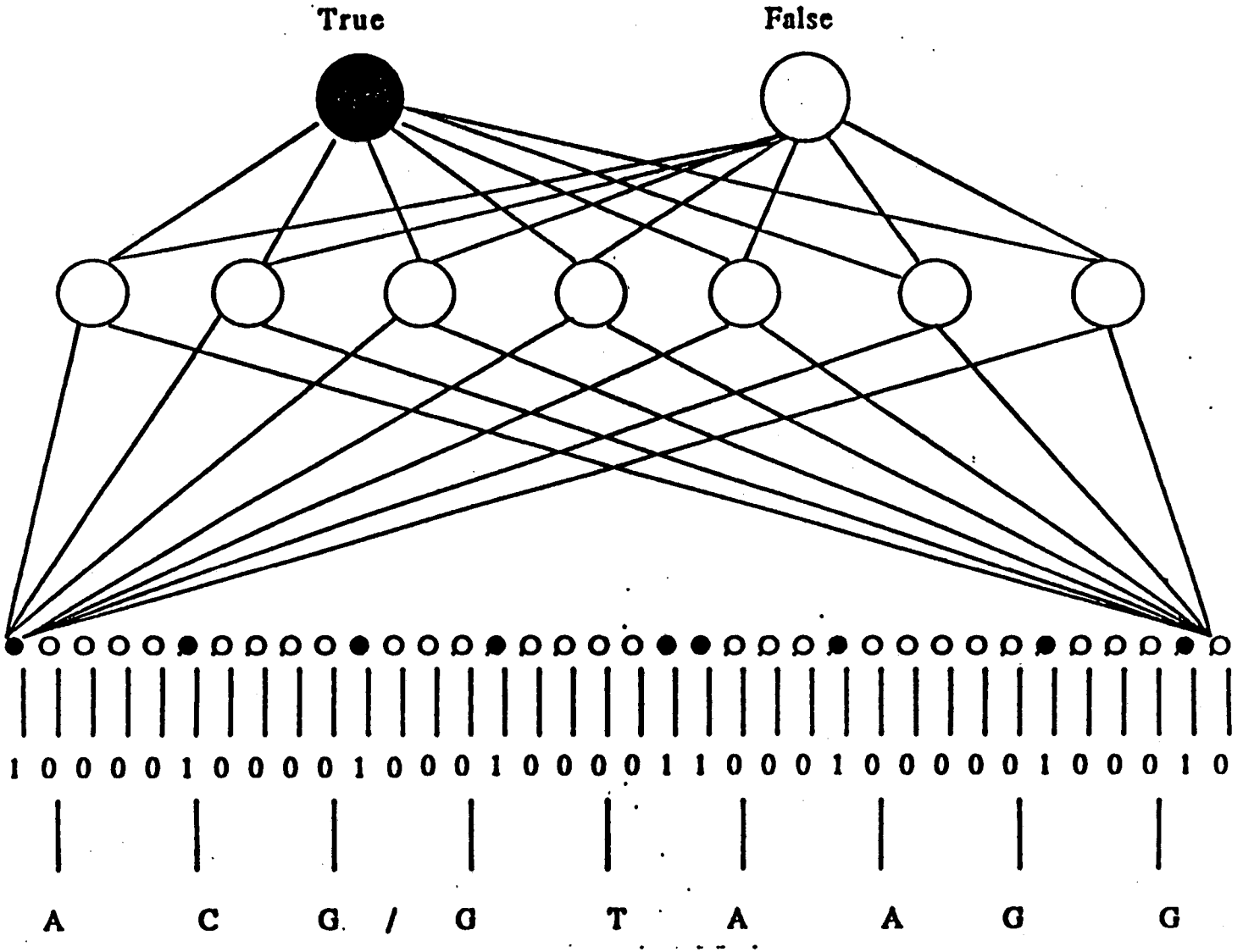


Figure 2

Schematic diagram of the backpropagation neural network used for the project. In this example, there are 27 input nodes, 7 hidden nodes and two output nodes. The input example shows a 9 base pair sequence to be tested and the coding of each base pair with 4 nodes. The splice junction boundary is indicated with the "/".

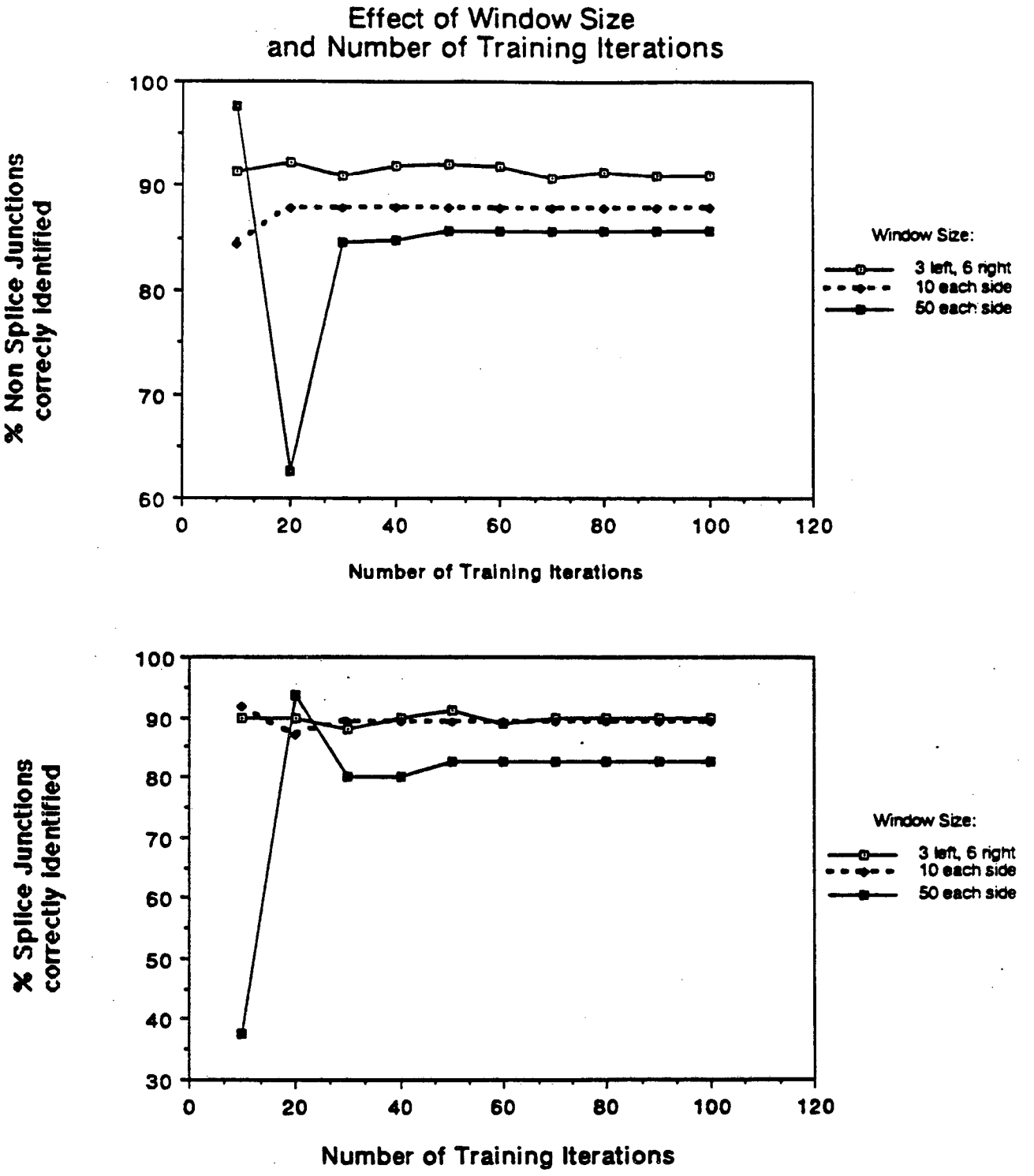


Figure 3

The effect of varying the size of the input window on the performance of the backpropagation network. The network was trained with a training set containing 200 true and 200 false examples. It was trained for each of the indicated number of iterations and then was tested with a separate test sequence set for which the correct results are also known. The percent correct responses was calculated and plotted against the number of training iterations. The best results were obtained with the 9 basepair window, 3 bases to the left of the 'GT' and 6 to the right. With each window size the network quickly reaches a level of performance after which further training is ineffective.

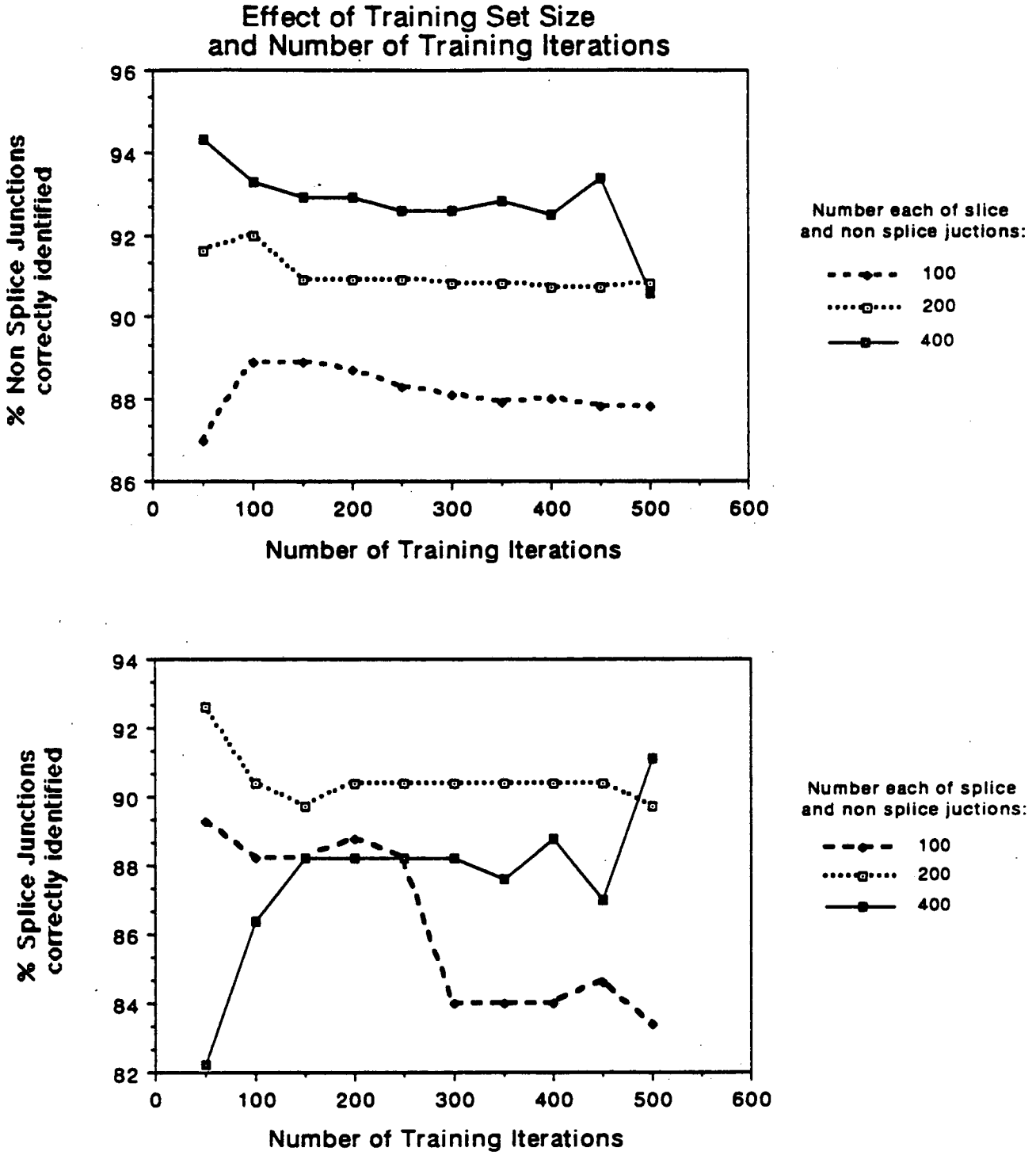


Figure 4

The effect of the size of the training set on the performance of the backpropagation network. The percent correct responses was calculated and plotted against the number of training iterations for each of three training set sizes. Poor results were obtained with the training set of 100 true and 100 false examples. Improved results were seen with 200. Further increases in the training set size were not obviously better.

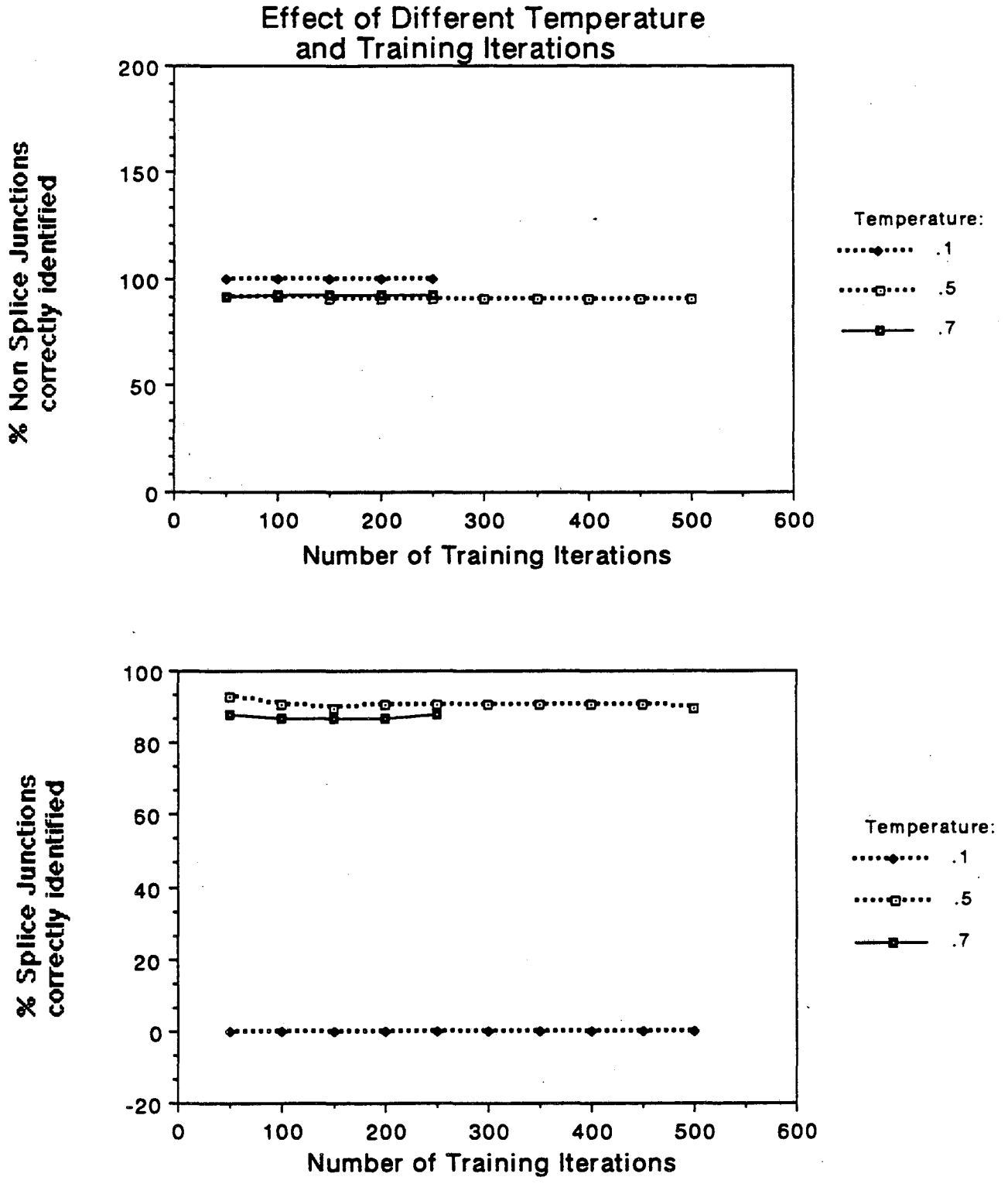
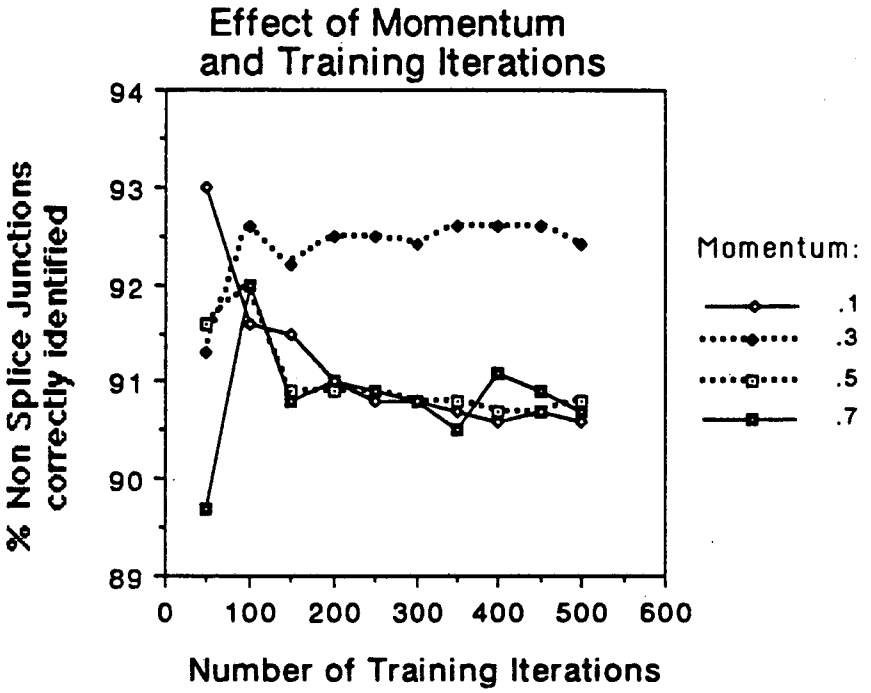


Figure 5

Effect of using different values for the temperature constant, a parameter in the Rochester backpropagation software. See text for an explanation.



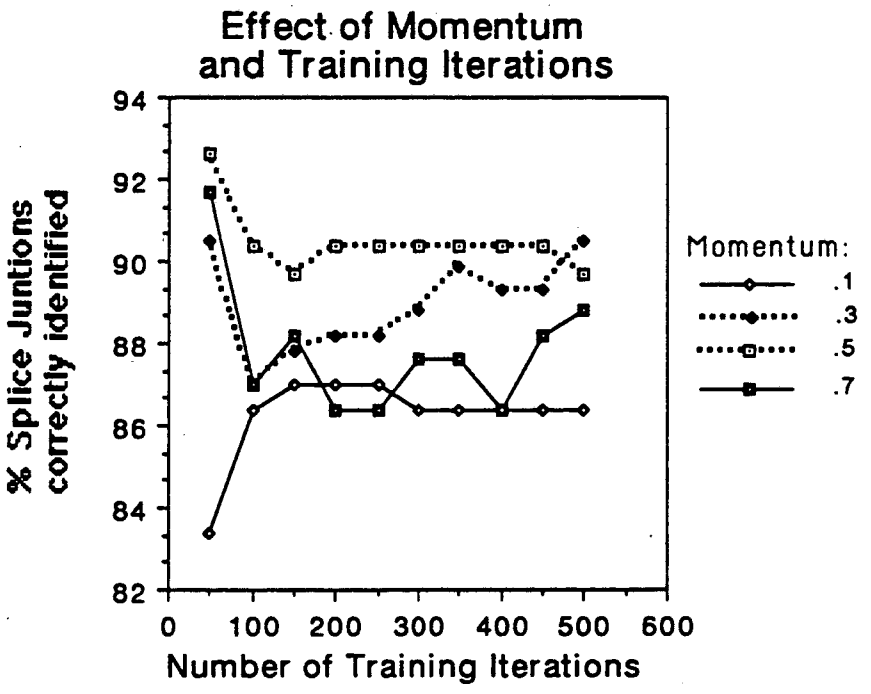


Figure 6

Effect of using different values for the momentum constant in the backpropagation error function. See text for an explanation.

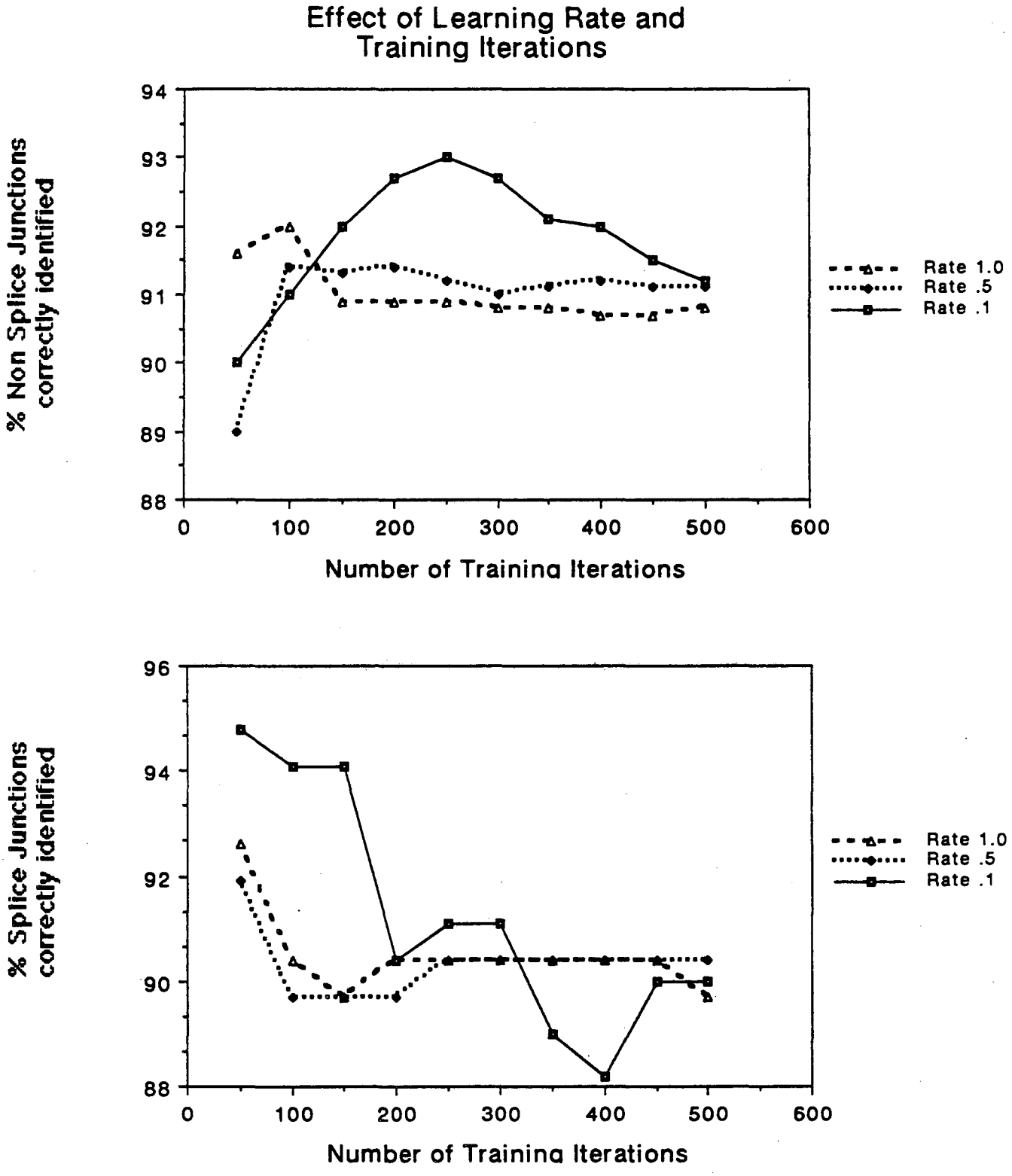


Figure 7

Effect of using different values for the training rate constant, BPlearn, in the backpropagation error function. See text for an explanation.

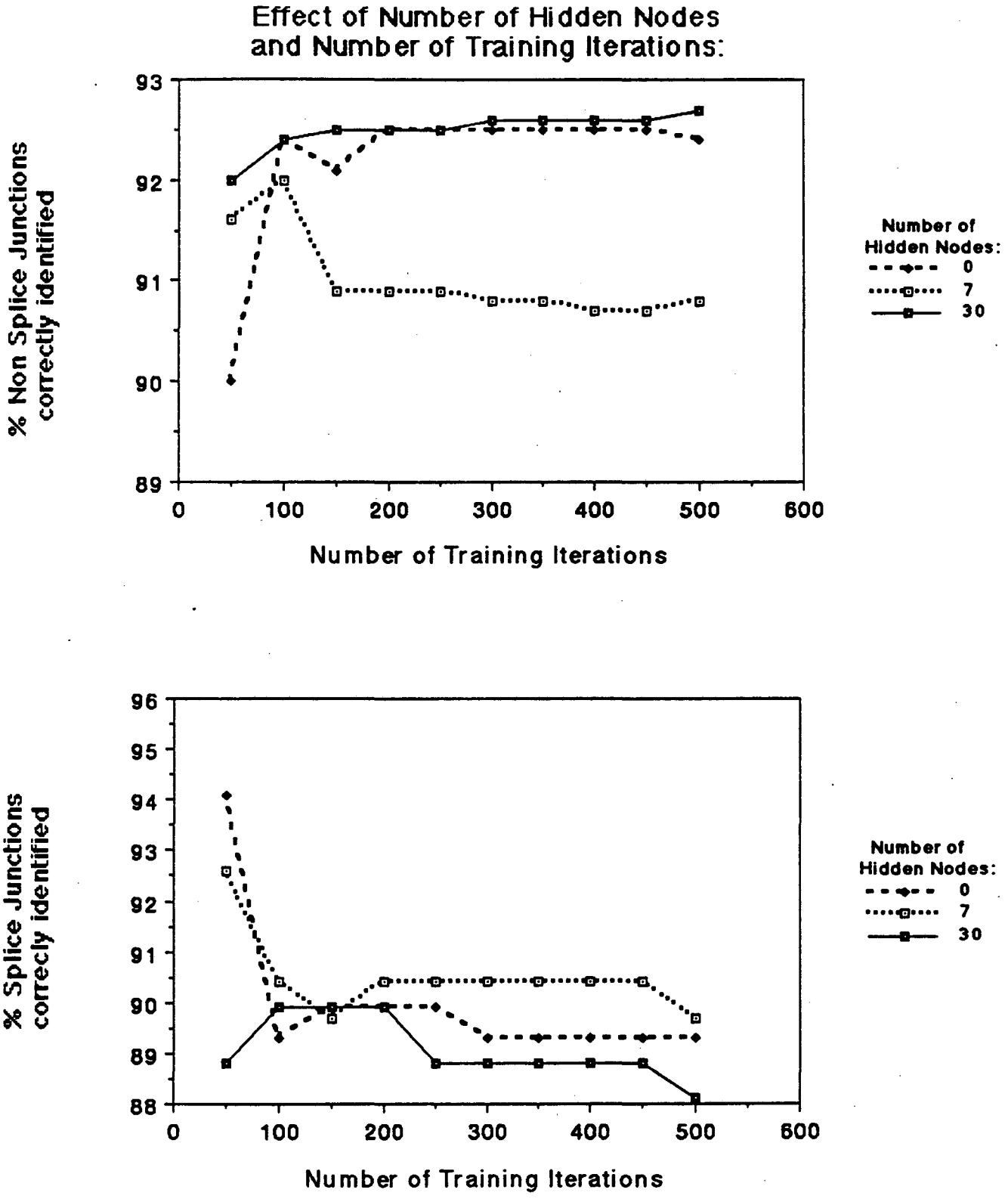


Figure 8

Effect of varying the number of hidden nodes. The network was tested after every 50 iterations for a total of 500 iterations.

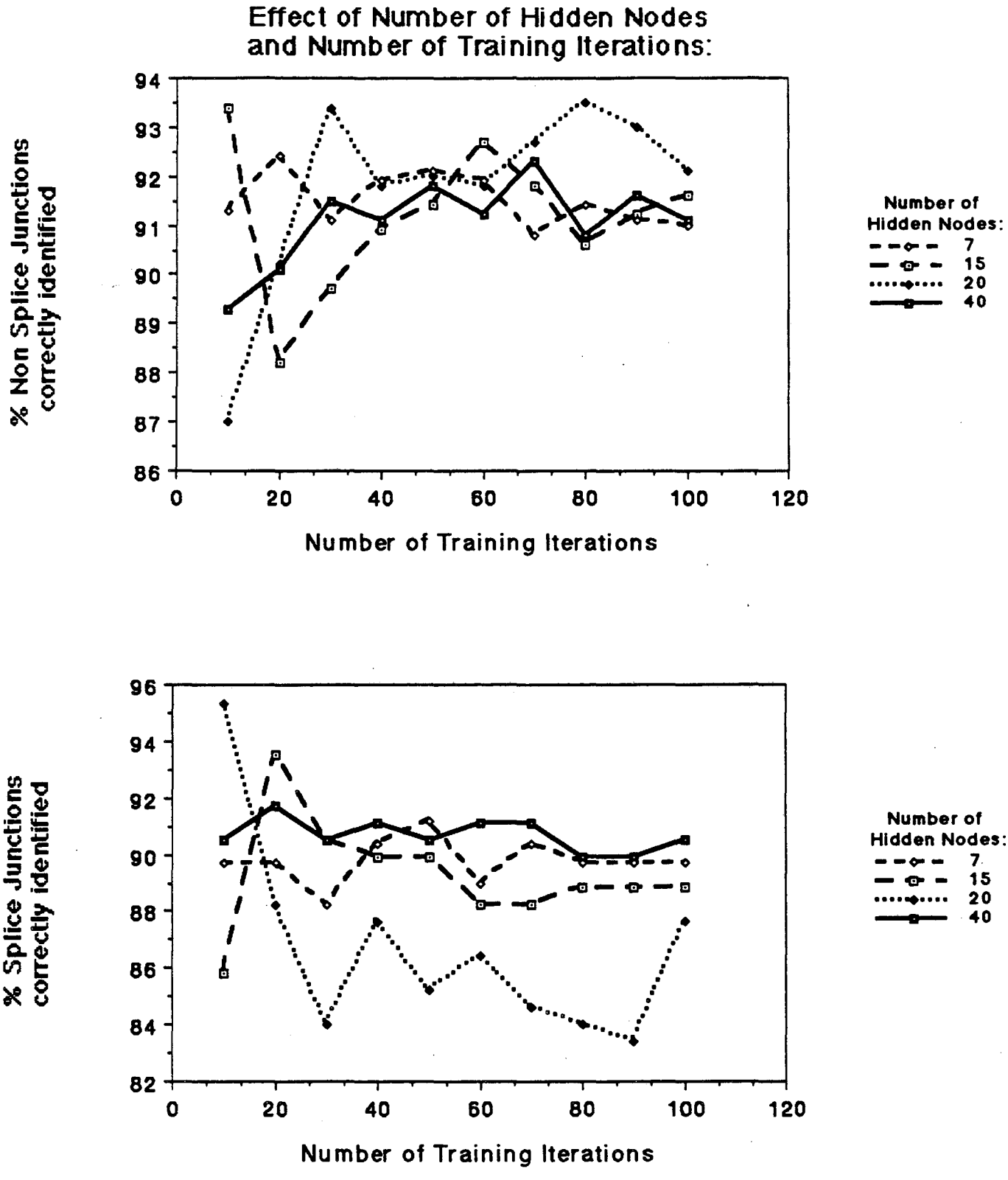


Figure 9

Another graph showing the effect of varying the number of hidden nodes. In these experiments the network was tested after every 10 iterations for a total of 100 iterations.

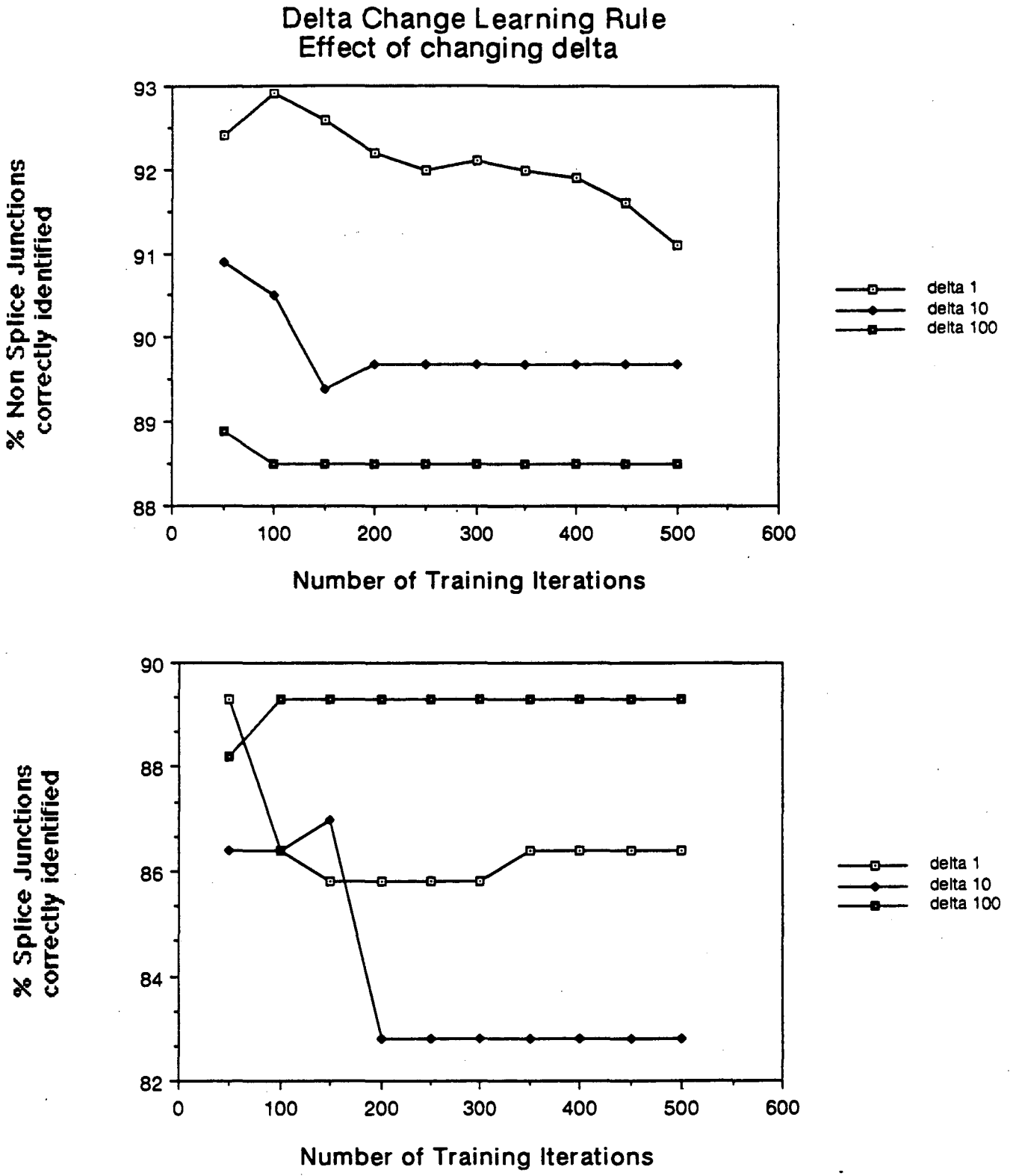


Figure 10

The effect of varying the constant delta on the performance of a higher order network using the delta change learning rule.

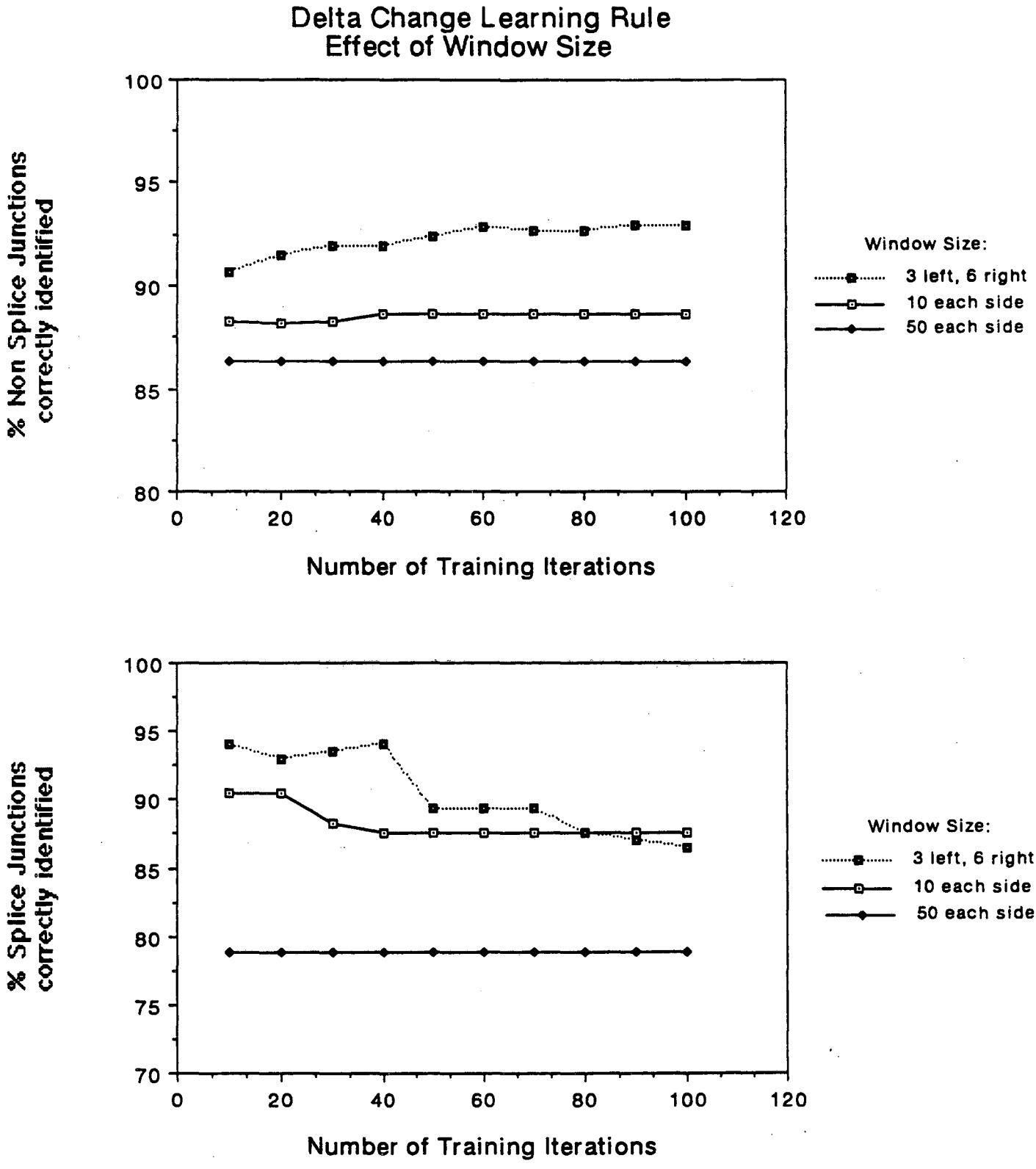


Figure 11

The effect of varying the window size on the performance of a higher order network using the delta change learning rule.

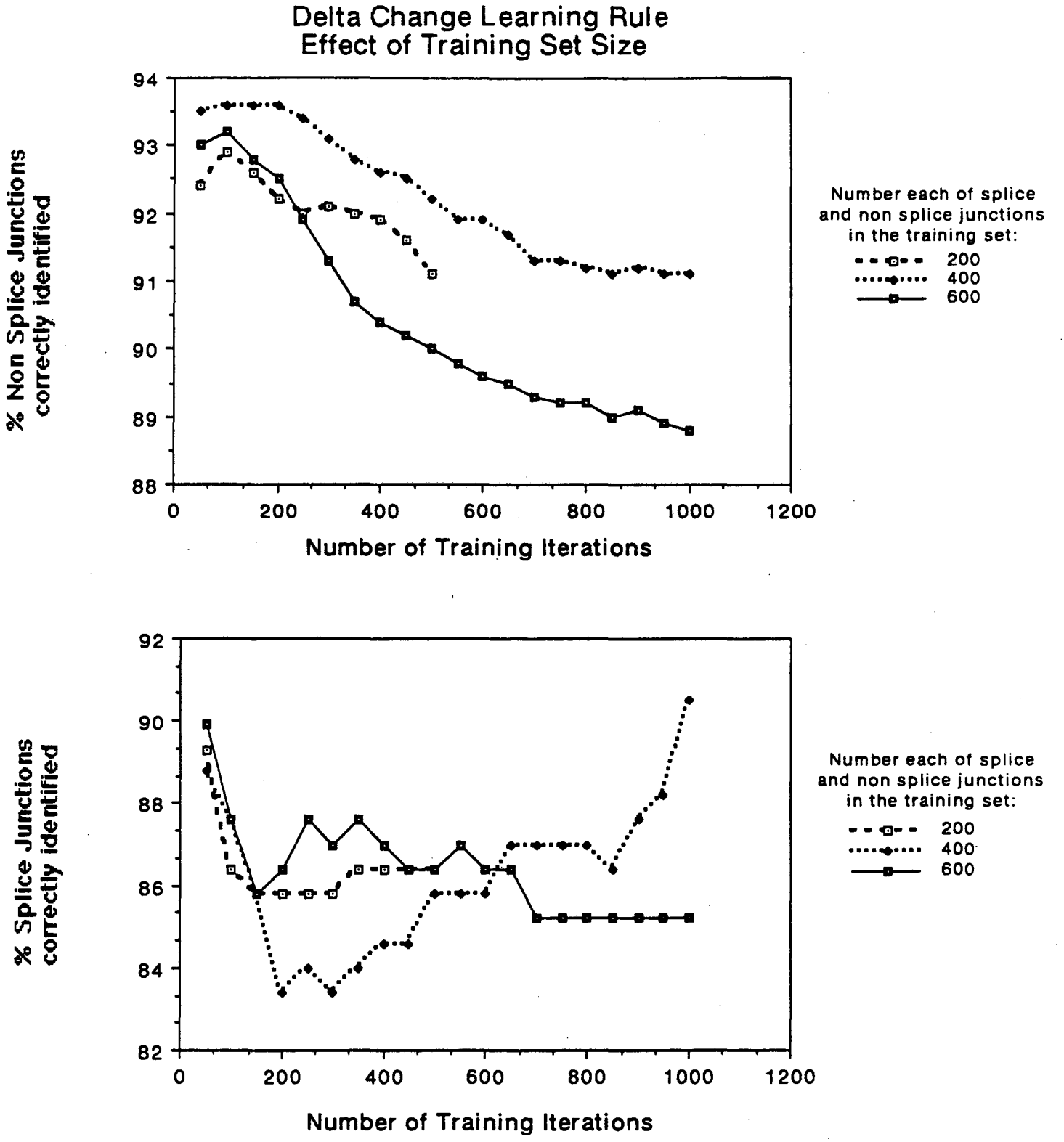


Figure 12

The effect of varying the size of the training set size on the performance of a higher order network using the delta change learning rule.

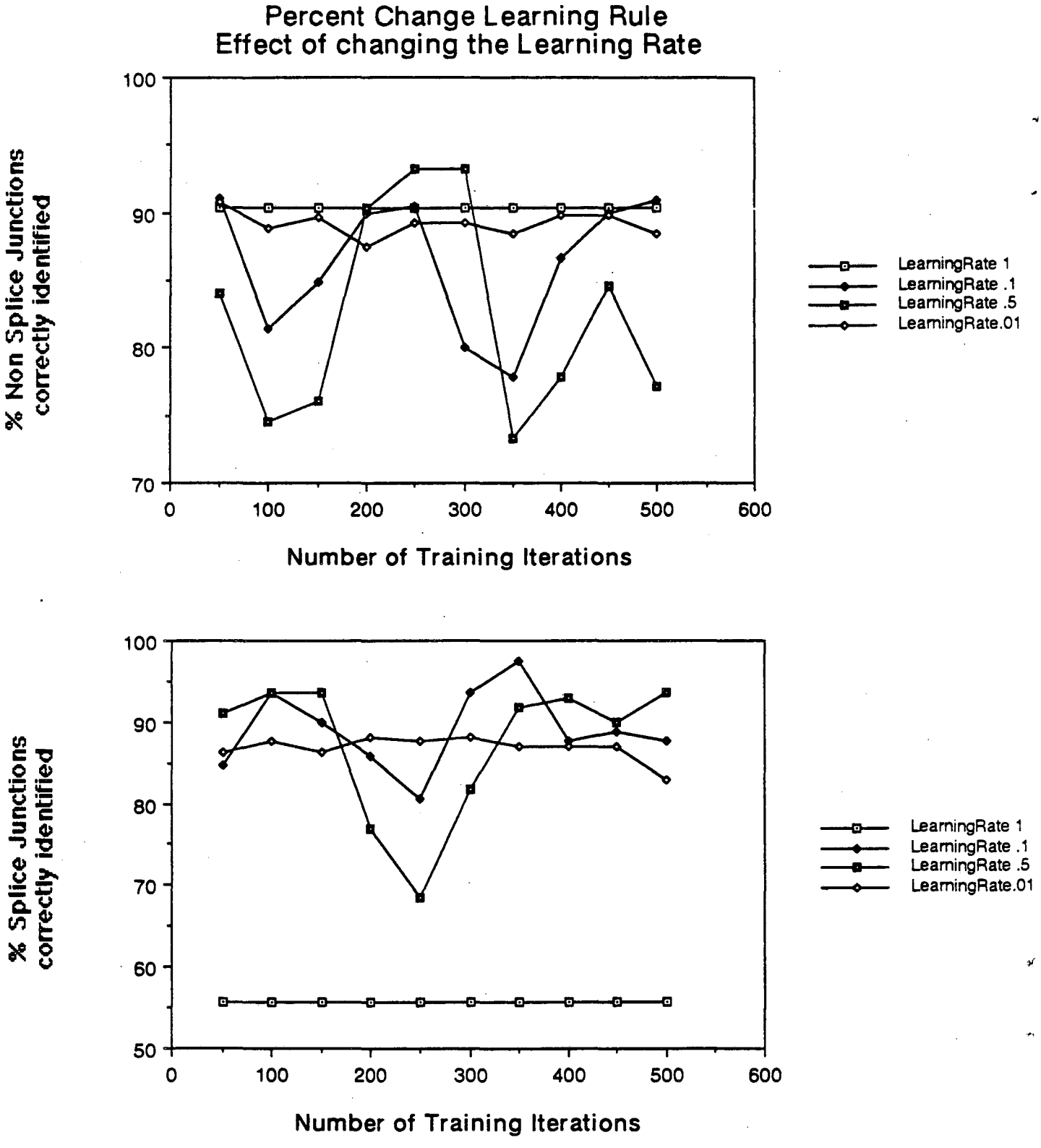


Figure 13

The effect of varying the learning rate constant on the performance of a higher order network using the percent change learning rule.

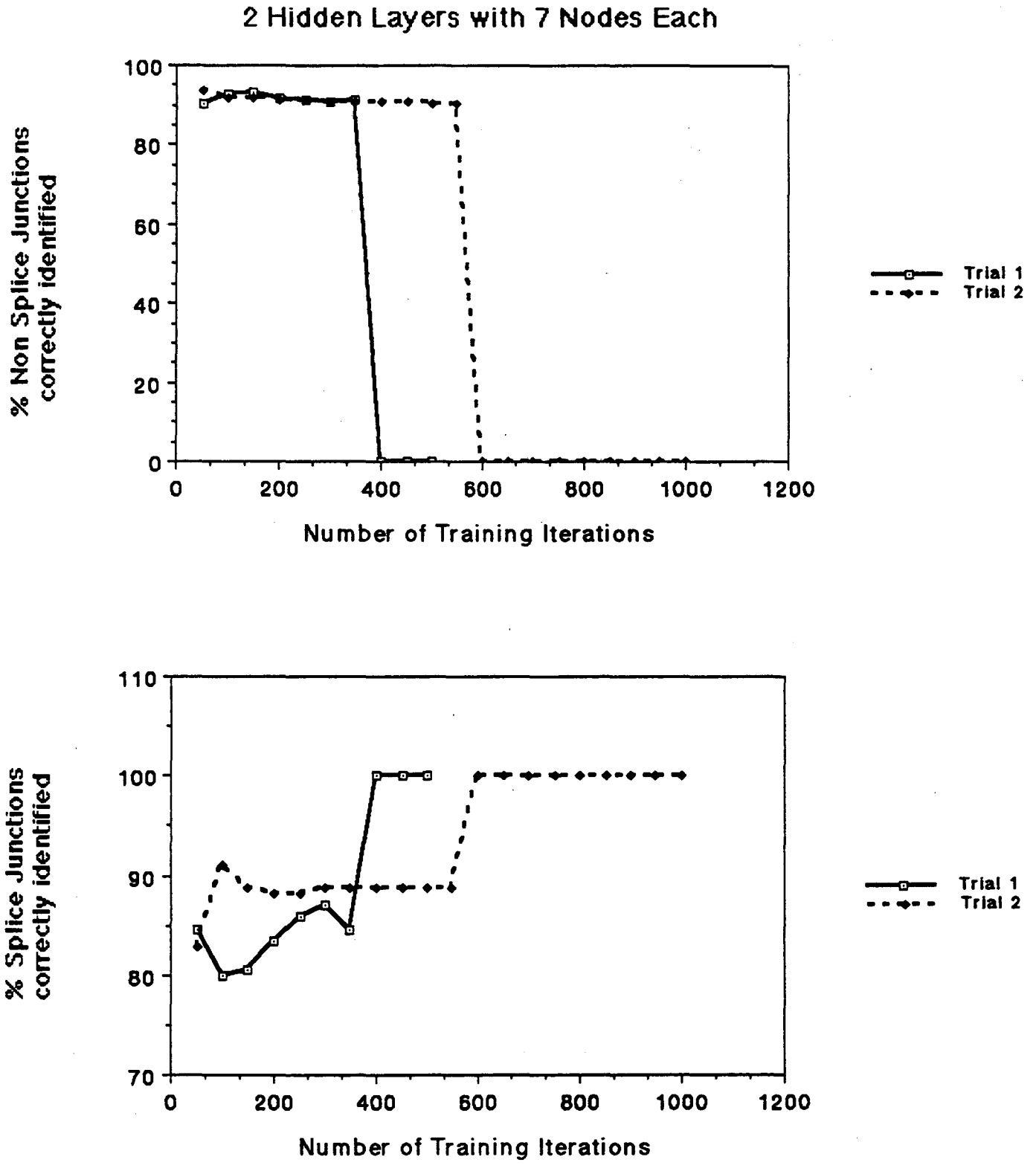


Figure 14

The results from using a backpropagation network with two hidden layers of seven nodes each.

LAWRENCE BERKELEY LABORATORY
TECHNICAL INFORMATION DEPARTMENT
1 CYCLOTRON ROAD
BERKELEY, CALIFORNIA 94720