Neural Computation, Vol. 9, No. 1, pp. 205–225

Extracting rules from neural networks by pruning and hidden-unit splitting

Rudy Setiono

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 119260

Republic of Singapore

Email:rudys@iscs.nus.sg

Abstract

An algorithm for extracting rules from a standard three-layer feedforward neural network is proposed. The trained network is first pruned not only to remove redundant connections in the network, but more importantly, to detect the relevant inputs. The algorithm generates rules from the pruned network by considering only a small number of activation values at the hidden units. If the number of inputs connected to a hidden unit is sufficiently small, then rules that describe how each of its activation values is obtained can be readily generated. Otherwise, the hidden unit will be split and treated as output units, with each output unit corresponding to an activation value. A hidden layer is inserted and a new subnetwork is formed, trained, and pruned. This process is repeated until every hidden unit in the network has a relatively small number of input units connected to it. Examples on how the proposed algorithm works are shown using real-world data arising from molecular biology and signal processing. Our results show that for these complex problems, the algorithm can extract reasonably compact rule sets that have high predictive accuracy rates.

1 Introduction

One of the most popular applications of feedforward neural networks is distinguishing patterns in two or more disjoint sets. The neural network approach has been applied to solve pattern classification problems in diverse areas such as finance, engineering, and medicine. While the predictive accuracy obtained by neural networks is usually satisfactory, it is often said that a neural network is practically a "black box". Even for a network with only a single hidden layer, it is generally impossible to explain why a certain pattern is classified as a member of one set and another pattern as a member of another set, due to the complexity of the network.

In many applications, however, it is desirable to have rules that explicitly state under what conditions a pattern is classified as a member of one set or another. Several approaches have been developed for extracting rules from a trained neural network. Saito and Nakano (1988) proposed a medical diagnostic expert system based on a multi-layer neural network. They treated the network as a black box and used it only to observe the effects on the network output caused by changes in the inputs.

Two methods for extracting rules from neural network are described in Towell and Shavlik (1993). The first method is the subset algorithm (Fu 1991), which searches for subsets of connections to a unit whose summed weight exceeds the bias of that unit. The second method, the MofN algorithm, clusters the weights of a trained network into equivalence classes. The complexity of the network is reduced by eliminating unneeded clusters and by setting all weights in each remaining cluster to the average of the cluster's weights. Rules with weighted antecedents are obtained from the simplified network by translation of the hidden units and output units. These two methods are applied by Towell and Shavlik to knowledge-based neural networks that have been trained to recognize genes in DNA sequences. The topology and the initial weights of a knowledge-based network are determined using a problem-specific *apriori* information.

More recently, a method that uses sampling and queries was proposed (Craven and Shavlik 1994). Instead of searching for rules from the network, the problem of rule extraction is viewed as a learning task. The target concept is the function computed by the network and the network input features are the inputs for the learning task. Conjunctive rules are extracted from the neural network with the help of two oracles.

Thrun (1995) describes a rule-extraction algorithm which analyzes the input-output behavior of a network using Validity Interval Analysis. VI-Analysis divides the activation range of each network's unit into intervals, such that all network's activation values must lie within the intervals. The boundary of these intervals are obtained by solving linear programs. Two approaches of generating rule conjectures, *specific-to-general* and *general-to-specific*, are described. The validity of these conjectures are checked with VI-Analysis.

In this paper, we propose an algorithm to extract rules from a pruned network. We assume the network to be a standard feedforward backpropagation network with a single hidden layer that has been trained to meet a prespecified accuracy requirement. The assumption that the network has only a single hidden layer is not restrictive. Theoretical studies have shown that networks with a single hidden layer can approximate arbitrary decision boundaries (Hornik 1991). Experimental studies have also demonstrated the effectiveness of such networks (de Villiers and Barnard 1992).

The process of extracting rules from a trained network can be made much easier if the complexity of the network has first been reduced. The pruning process attempts to eliminate as many connections as possible from the network, while at the same time tries to maintain the prespecified accuracy rate. It is expected that less connections will result in more concise rules. No initial knowledge of the problem domain is required. Relevant and irrelevant attributes of the data are distinguished during the pruning process. Those that are relevant will be kept, others will be automatically discarded.

A distinguishing feature of our algorithm is that the activation values at the hidden units are clustered into discrete values. When only a small number of inputs are feeding a hidden unit, it is not difficult to extract rules that describe how each of the discrete activation values is obtained. When many inputs are connected to a hidden unit, then its discrete activation values will be used as the target outputs of a new subnetwork. A new hidden layer is inserted between the inputs and this new output layer. The new network is then trained and pruned by the same pruning technique as in the original network. The process of splitting of hidden units and creating new networks is repeated until each hidden unit in the network has only a small number of inputs connected to it, say no more than 5.

We describe our rule-extraction algorithm in Section 2. A network that has been trained and pruned to solve the splice-junction problem (Lapedes *et al.* 1989) is used to illustrate in detail how the algorithm works. Each pattern in the dataset for this problem is described by a 60-nucleotide-long DNA sequence, and the learning task is to recognize the type of boundary at the center of the sequence. This boundary can be an exon/intron boundary, intron/exon boundary or neither. We present the results of applying the proposed algorithm on a second problem, the sonar target classification problem (Gorman and Sejnowski 1988) in Section 3. The learning task is to distinguish between sonar returns from metal cylinders and those from cylindrically shaped rocks. Each pattern in this dataset is described by a set of 60 real numbers between 0 and 1. The datasets for both the splice-junction and the sonar target classification problems were obtained via ftp from the University of California-Irvine repository (Murphy and Aha 1992). Discussion on the proposed algorithm and comparison with related work are presented in Section 4. Finally, a brief conclusion is given in Section 5.

2 Extracting rules from a pruned network

The algorithm for rule-extraction from a pruned network basically consists of two main steps. The first step of the algorithm clusters the activation values of the hidden units into a small number of clusters. If the number of inputs connected to a hidden unit is relatively large, the second step of the algorithm splits the hidden unit into a number of units. The number of clusters found in the first step determines the number of new units. We form a new network by treating each new unit as an output unit and adding a new hidden layer. We train and prune this network, and repeat the process if necessary. The process of forming a new network by treating a hidden unit as a new set of output units and adding a new hidden layer terminates when, after pruning, each hidden unit in the network has a small number of input units connected to it. The outline of the algorithm is as follows.

Rule-extraction (RX) algorithm

1. Train and prune the neural network.

2. Discretize the activation values of the hidden units by clustering.

3. Using the discretized activation values, generate rules that describe the network outputs.

4. For each hidden unit:

   - If the number of input connections is less than an upper bound U_C, then extract rules to describe the activation values in terms of the inputs.

- Else form a subnetwork:

  (a) Set the number of output units equal to the number of discrete activation values. Treat each discrete activation value as a target output.

  (b) Set the number of input units equal to the inputs connected to the hidden units.

  (c) Introduce a new hidden layer.

  Apply RX to this subnetwork.

5. Generate rules that relate the inputs and the outputs by merging rules generated in Steps 3 and 4.

In Step 3 of algorithm RX, we assume that the numbers of hidden units and the number of clusters in each hidden unit are relatively small. If there are H hidden units in the pruned network, and Step 2 of RX finds C_i clusters in hidden unit $i = 1, 2, \ldots, H$, then there are $C_1 \times C_2 \ldots \times C_H$ possible combinations of clustered hidden-unit activation values. For each of these combinations, the network output is computed using the weights of the connections from the hidden units to the output units. Rules that describe the network outputs in terms of the clustered activation values are generated using X2R algorithm (Liu and Tan 1995). The algorithm is designed to generate rules from small datasets with discrete attribute values. It chooses the pattern with the highest frequency, generates the shortest rule for this pattern by checking the information provided by one data attribute at a time, and repeats the process until rules that cover all the patterns are generated. The rules are then grouped according to the class labels and redundant rules are eliminated by pruning. When there is no noise in the data, X2R generates rules with perfect accuracy on the training patterns.

If the number of inputs of a hidden unit is less than U_C, Step 4 of the algorithm also applies X2R to generate rules that describe the discrete activation values of that hidden unit in terms of the input units. Step 5 of RX merges the two sets of rules generated by X2R. This is done by substituting the conditions of the rules involving clustered activation values generated in Step 3 by the rules generated in Step 4.

We shall illustrate how algorithm RX works using the splice-junction domain in the next section.

2.1 The splice-junction problem

We trained a network to solve the splice-junction problem. This problem is a real-world problem arising in molecular biology that has been widely used as a test dataset for knowledge-based neural networks training and rule-extraction algorithms (Towell and Shavlik 1993).

The total number of patterns in the dataset is 3175 [1], each of the pattern consists of 60 attributes. The attribute corresponds to a DNA nucleotide and takes one of the following four values: G, T, C, or A. Each pattern is classified to be either IE (intron/exon boundary), EI (exon/intron boundary), or N (neither) according to the type of boundary at the center of the DNA sequence. Of the 3175 patterns, 1006 were used as training data. The four attribute values G,T,C and A were coded as {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}, respectively. Including the input for bias, a total of 241 input units were present in the original neural network. The target output for each pattern in class IE was $\{1,0,0\}$, in

[1] There are 15 more patterns in the dataset that we did not use because one or more attributes has a value that is not G, T, C, or A

class EI $\{0, 1, 0\}$, and in class N $\{0, 0, 1\}$. Five hidden units were sufficient to give the network more than 95 % accuracy rate on the training data.

We considered a pattern of type IE to be correctly classified as long as the largest activation value was found in the first output unit. Similarly, type EI and N patterns were considered to be correctly classified if the largest activation value was obtained in the second and third hidden unit, respectively. The transfer functions used were the hyperbolic tangent function at the hidden layer and the sigmoid function at the output layer. The cross entropy error measure was used as the objective function during training. To encourage "weight decay", a penalty term was added to this objective function. The details of this penalty term and the pruning algorithm are described in Setiono (1995). Aiming to achieve a comparable accuracy rate as those reported in Towell and Shavlik (1993), we terminated the pruning algorithm when the accuracy of the network dropped below 92 %.

The resulting pruned network is depicted in Figure 1. There was a large number of reduction in the number of network connections. Of the original 1220 weights in the network, only 16 remained. Only ten input units (including the bias input unit) remained out of the original 241 inputs. These inputs corresponded to seven out of the 60 attributes present in the original data. Two of the five original hidden units were also removed. As we shall see in the next subsection, the number of input units and hidden units plays a crucial role in determining the complexity of the rules extracted from the network. The accuracy of the pruned network is summarized in the Table 1.

2.2 Clustering hidden-unit activations

The range of the hyperbolic tangent function is the interval $[-1, 1]$. If this function is used as the transfer function at the hidden layer, a hidden-unit activation value may lie anywhere

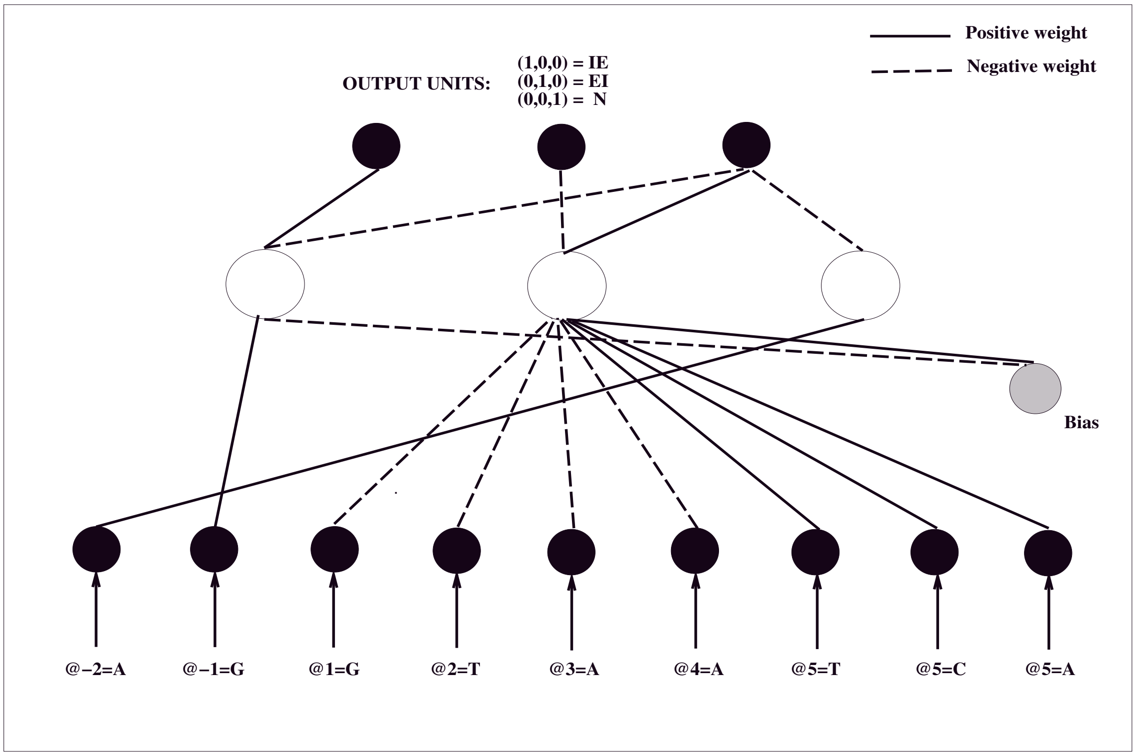

Figure 1: A pruned network for the splice-junction problem. Following convention, the original attributes have been numbered sequentially from -30 to -1 and 1 to 30. If the expression "@n=X" is true, then the corresponding input value is 1; otherwise it is 0.

Class	Training data		Testing data	
	Errors (total)	Accuracy (%)	Errors (total)	Accuracy (%)
IE	13 (243)	94.65	32 (765)	95.82
EI	11 (228)	95.18	42 (762)	94.49
N	53 (535)	90.09	150 (1648)	90.90
Total	77 (1006)	92.35	224 (3175)	92.94

Table 1: Accuracy of the pruned network on the training and testing datasets.

within this interval. However, it is possible to replace it by a discrete value without causing too much deterioration in the accuracy of the network. This is done by the clustering algorithm described below.

Hidden-unit activation-values clustering algorithm

1. Let $\epsilon \in (0,1)$. Let D be the number of discrete activation values in the hidden unit. Let δ_1 be the activation value in the hidden unit for the first pattern in the training set. Let $H(1) = \delta_1, count(1) = 1$, and $sum(1) = \delta_1$; set $D = 1$.

2. For each pattern $p_i, i = 2, 3, \ldots k$ in the training set:

   - Let δ be its activation value.
   - If there exists an index $\overline{j}$ such that

     $$\begin{aligned} |\delta - H(\overline{j})| &= \min_{j \in \{1,2,\ldots,D\}} |\delta - H(j)| \text{ and} \\ |\delta - H(\overline{j})| &\leq \epsilon, \end{aligned}$$

     then set $count(\overline{j}) := count(\overline{j}) + 1, sum(\overline{j}) := sum(\overline{j}) + \delta$

     else $D = D + 1, H(D) = \delta, count(D) = 1, sum(D) = \delta$.

3. Replace H by the average of all activation values that have been clustered into this cluster:

   $$H(j) := sum(j)/count(j),\ j = 1, 2 \ldots, D.$$

Step 1 initializes the algorithm. The first activation value forms the first cluster. Step 2 checks whether subsequent activation values can be clustered into one of the existing clusters. The distance between an activation value under consideration and its nearest cluster, $|\delta - H(\overline{j})|$, is computed. If this distance is less than ϵ, then the activation value is clustered in cluster $\overline{j}$. Otherwise, this activation value forms a new cluster.

Once the discrete values of all hidden units have been obtained, the accuracy of the network is checked again with the activation values at the hidden units replaced by their discretized values. An activation value δ is replaced by $H(\overline{j})$, where index $\overline{j}$ is chosen such that $\overline{j} = \mathrm{argmin}_j |\delta - H(j)|$. If the accuracy of the network falls below the required accuracy, then ϵ must be decreased and the clustering algorithm is run again. For a sufficiently small ϵ, it is always possible to maintain the accuracy of the network with continuous activation values, although the resulting number of different discrete activations can be impractically large.

The best ϵ value is one that gives a high accuracy rate after the clustering and at the same time generates as few clusters as possible. A simple way of obtaining an optimal value for ϵ is by searching in the interval $(0, 1)$. The number of clusters and the accuracy of the network can be checked for all values of $\epsilon = i\xi, i = 1, 2, \ldots$, where ξ is a small positive scalar, e.g. 0.10. Note also that it is not necessary to fix the value of ϵ equal for all hidden units.

For the pruned network depicted in Figure 1, we found the value of $\epsilon = 0.6$ worked well for all three hidden units. The results of the clustering algorithm were as follows:

1. Hidden unit 1: there were 2 discrete values: -0.04 and -0.99. Of the 1006 training data, 573 patterns had the first value and 433 patterns had the second value.

2. Hidden unit 2: there were 3 discrete values: 0.96, -0.10, and -0.88. The distribution of the training data was 760, 72, and 174, respectively.

Class	Training data		Testing data	
	Errors (total)	Accuracy (%)	Errors (total)	Accuracy (%)
IE	16 (243)	93.42	40 (765)	94.77
EI	8 (228)	96.49	30 (762)	96.06
N	52 (535)	90.28	142 (1648)	91.38
Total	76 (1006)	92.45	212 (3175)	93.32

Table 2: Accuracy of the pruned network on the training and testing datasets with discrete activation values at the hidden units.

3. Hidden unit 3: there were 2 discrete values: 0.96 and 0. Of the 1006 training data, 520 patterns had the first value and 486 patterns had the second value.

The accuracy of the network with these discrete activation values is summarized in Table 2. With $\epsilon = 0.6$, the accuracy was not the same as that achieved by the original network. In fact it was slightly higher on both the training data and the testing data. It appears that a much smaller number of possible values for the hidden-unit activations reduces overfitting of the data. With a sufficiently small value for ϵ, it is always possible to maintain the accuracy of a network after its activation values have been discretized. However, there is no guarantee that the accuracy can increase in general.

Two discrete values at hidden unit 1, three at hidden unit 2, and two at hidden unit 3 produced a total of 12 possible outputs for the network. These outputs are listed in Table 3.

Hidden unit activations			Predicted output			Classification
1	2	3	1	2	3	
-0.04	0.96	0.96	0.44	0.01	0.20	IE
-0.04	0.96	0.00	0.44	0.01	0.99	N
-0.04	-0.10	0.96	0.44	0.62	0.00	EI
-0.04	-0.10	0.00	0.44	0.62	0.42	EI
-0.04	-0.88	0.96	0.44	0.99	0.00	EI
-0.04	-0.88	0.00	0.44	0.99	0.02	EI
-0.99	0.96	0.96	0.00	0.01	0.91	N
-0.99	0.96	0.00	0.00	0.01	1.00	N
-0.99	-0.10	0.96	0.00	0.62	0.06	EI
-0.99	-0.10	0.00	0.00	0.62	0.97	N
-0.99	-0.88	0.96	0.00	0.99	0.00	EI
-0.99	-0.88	0.00	0.00	0.99	0.43	EI

Table 3: Output of the network with discrete activation values at the hidden units.

Let us define the notation $\alpha(i, j)$ to denote the hidden unit i taking its jth activation value. X2R generated the following rules that classify each of the predicted output classes:

- If $\alpha(1, 1)$ and $\alpha(2, 1)$ and $\alpha(3, 1)$ then output = IE.
- Else if $\alpha(2, 3)$ then output = EI.
- Else if $\alpha(1, 1)$ and $\alpha(2, 2)$ then output = EI.
- Else if $\alpha(1, 2)$ and $\alpha(2, 2)$ and $\alpha(3, 1)$ then output = EI.
- Default output: N.

Hidden unit 1 has only 2 inputs, one of which is the input for bias; while hidden unit 3 has only one input. Very simple rules in terms of the original attributes that describe the activation values of these hidden units were generated by X2R. The rules for these hidden units are

- If @-1=G, then $\alpha(1, 1)$, else $\alpha(1, 2)$.
- If @-2=A, then $\alpha(3, 1)$, else $\alpha(3, 2)$.

The expression @n=X is true implies that the corresponding input value is 1, otherwise the input value is 0.

The seven input units feeding the second hidden unit can generate a total of 64 possible instances. The three inputs @5 (= T, C, or A) can produce 4 different combinations ((0,0,0), (0,0,1), (0,1,0), or (1,0,0)), while the other four inputs can generate 16 different possible combinations. Rules that define how each of the three activation values of this hidden unit is obtained are not trivial to extract from the network. How this difficulty can be overcome is described in the next subsection.

2.3 Hidden unit splitting and creation of a subnetwork

If we set the value of U_C in algorithm RX to be less than 7, then in order to extract rules that describe how the three activation values of the second hidden unit are obtained, a subnetwork will be created.

Since seven inputs determined the three activation values, the new network also had the same set of inputs. The number of output units corresponded to the number of activation values; in this case, three output units were needed. Each of the 760 patterns whose activation values equal to 0.96 was assigned a target output of $\{1, 0, 0\}$. Patterns with activation values of -0.10 and -0.88 were assigned target outputs of $\{0, 1, 0\}$ and $\{0, 0, 1\}$, respectively. In order to extract rules with the same accuracy rate as the network's accuracy rate summarized in Table 2, the new network was trained to achieve 100 % accuracy rate. Five hidden units were sufficient to give the network this rate. The pruning process was terminated as soon as the accuracy dropped below 100 %.

When only seven of the original 240 inputs were selected, many patterns had duplicates. To reduce the time to train and prune the new network, all duplicates were removed and the 61 unique patterns left were used as training data. Since the network was trained to achieve 100 % rate, correct predictions of these 61 patterns guaranteed that the same rate of accuracy was obtained on all 1006 patterns.

The pruned network is depicted in Figure 2. Three of the five hidden units remained after pruning. Of the original 55 connections in the fully connected network, 16 were still present. The most interesting aspect of the pruned network was that the activation values of the first hidden unit was determined solely by four inputs, while those of the second hidden unit by the remaining three inputs. The activation values in the three hidden units were clustered. The value of $\epsilon = 0.2$ was found to be small enough for the network with

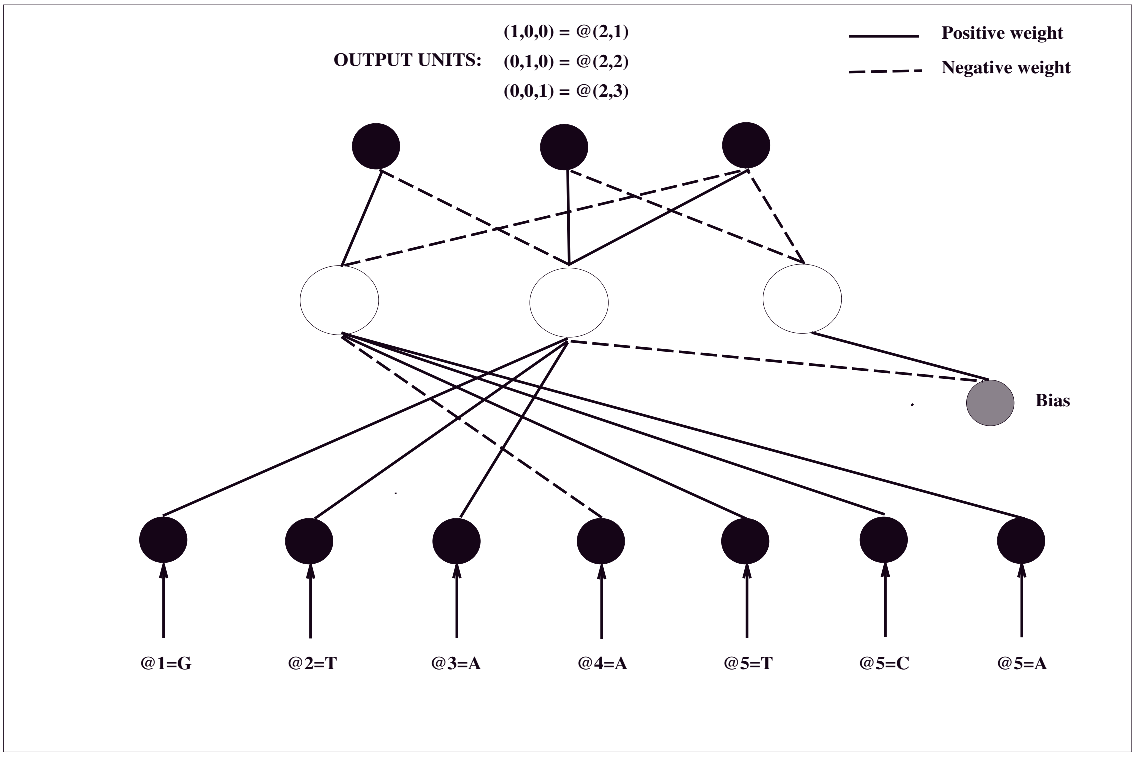

Figure 2: Pruned network for the second hidden unit of the original network depicted in Figure 1.

discrete activation values to still maintain a 100 % accuracy rate and large enough such that there were not too many different discrete values. The results were as follows:

1. Hidden unit 1: there were 3 discrete values: 0.98, 0.00, and -0.93. Of the original 1006 (61 training) patterns, 593 (39) patterns had the first value, 227 (15) the second value, and 186 (7) had the third value.

2. Hidden unit 2: there were 3 discrete values: 0.88, 0.33, and -0.99. The distribution of the training data was 122 (8), 174 (8), and 710 (45), respectively.

3. Hidden unit 3: there was only 1 discrete value: 0.98.

Since there were 3 different activation values at hidden units 1 and 2, and only 1 value at hidden unit 3, a total of 9 possible outputs for the network were possible. These possible

Hidden unit activations			Predicted output			Classification
1	2	3	1	2	3	
0.98	0.88	0.98	0.01	0.66	0.00	$\alpha(2,2)$
0.98	0.33	0.98	0.99	0.21	0.00	$\alpha(2,1)$
0.98	-0.99	0.98	1.00	0.00	0.00	$\alpha(2,1)$
0.00	0.88	0.98	0.00	0.66	0.98	$\alpha(2,3)$
0.00	0.33	0.98	0.00	0.21	0.02	$\alpha(2,2)$
0.00	-0.99	0.98	1.00	0.00	0.00	$\alpha(2,1)$
-0.93	0.88	0.98	0.00	0.66	1.00	$\alpha(2,3)$
-0.93	0.33	0.98	0.00	0.21	1.00	$\alpha(2,3)$
-0.93	-0.99	0.98	1.00	0.00	0.00	$\alpha(2,1)$

Table 4: Output of the network with discrete activation values at the hidden units for the pruned network in Figure 2.

outputs are summarized in Table 4.

In a similar fashion as before, let $\beta(i,j)$ denote the hidden unit i taking its jth activation value. The following rules that classify each of the predicted output classes were generated by X2R from the data in Table 4:

- If $\beta(2,3)$ then $\alpha(2,1)$.
- Else if $\beta(1,1)$ and $\beta(2,2)$ then $\alpha(2,1)$.
- Else if $\beta(1,1)$ and $\beta(2,1)$ then $\alpha(2,2)$.
- Else if $\beta(1,2)$ and $\beta(2,2)$ then $\alpha(2,2)$.
- Default output: $\alpha(2,3)$.

X2R extracted rules, in terms of the original attributes, describing how each of the six different activation values of the 2 hidden units was obtained. These rules are given in Appendix A as Level 3 rules. The complete rules generated from the splice-junction domain after the merging of rules in Step 5 of RX algorithm are also given in Appendix A. Since the subnetwork created for the second hidden unit of the original network had been trained to achieve 100 % accuracy rate and the value of ϵ used to cluster the activation values was also chosen to retain this level of accuracy, the accuracy rates of the rules were exactly the same as those of the network with discrete hidden-unit activation values listed in Table 2, i.e., 92.45 % on the training dataset and 93.32 % on the test dataset.

The results of our experiment suggest that the extracted rules are able to mimic the network from which they are extracted perfectly. In general, if there are sufficient number of discretized hidden-unit activation values and the subnetworks are trained and pruned to achieve 100 % accuracy rate, then the rules extracted will have the same accuracy rate as the network on the training data. It is possible to mimic the behavior of the pruned network with discrete activation values even on future patterns not in the training dataset. Let $\mathcal{S}$ be the set of all patterns that can be generated by the inputs connected to a hidden unit. When the discrete activation values in the hidden unit are determined only by a subset of $\mathcal{S}$, it is still possible to extract rules that will cover all possible instances that may be encountered in the future. For each pattern that is not represented in the training data, its continuous activation values are computed using the weights of the pruned network. Each of these activation values is assigned a discrete value in the hidden unit that is closest to it. Note that once the complete rules have been obtained, the network topology, the weights on the network connections, and the discrete hidden-unit activation values need not be retained. For our experiment on the splice-junction domain, we found that rules that cover all possible instances were generated without having to compute the activation values of patterns not already present in the training data. This was due to the relatively large number of patterns used during training and the small number of inputs found relevant for determining the hidden-unit activation values.

3 Experiment on dataset with continuous attributes

In the previous section, we have given a detailed description on how rules can be extracted from a dataset with only nominal attributes. In this section we shall illustrate how rules can also be extracted from a dataset having continuous attributes. The sonar returns classification problem (Gorman and Sejnowski 1988) was chosen for this purpose. The dataset consisted of 208 sonar returns, each of which was represented by 60 real numbers between 0.0 and 1.0. The task was to distinguish between returns from a metal cylinder and those from a cylindrically shaped rock. Returns from metal cylinder were obtained at aspect angles spanning 90^0, while those from rocks at aspect angle spanning 180^0.

Gorman and Sejnowski (1988) used three-layer feedforward neural networks with varying number of hidden units to solve this problem. Two series of experiments on this dataset were reported: an aspect-angle independent series and an aspect-angle dependent series. In the aspect-angle independent series, the training patterns and the testing patterns were selected randomly from the total set of 208 patterns. In the aspect-angle dependent series, the training and the testing sets were carefully selected to contain patterns from all available aspect-angles. It is not surprising that the performance of networks in the aspect-angle dependent series of experiments was better than those in the aspect-angle independent series.

We chose the aspect-angle dependent dataset to test our rule-extraction algorithm. The training set consisted of 49 sonar returns from metal cylinders and 55 sonar returns from rocks, while the testing set consisted of 62 sonar returns from metal cylinders and 42 sonar returns from rocks.

In order to facilitate rule extraction, the numeric attributes of the data were first converted into discrete ones. This was done by a modified version of the ChiMerge algorithm (Kerber 1992). The training patterns were first sorted according to the value of the attribute being discretized. Initial subintervals were formed by placing each unique value of the attribute in its own subinterval. The χ^2 value of adjacent intervals were computed and the pairs of adjacent subintervals with the lowest χ^2 value were merged. In Kerber's original algorithm, merging continues until all pairs of subintervals have χ^2 values exceeding a user defined parameter $\chi^2 - threshold$. Instead of setting a fixed threshold value as a stopping condition for merging, we continued merging the data as long as there was no inconsistency in the discretized data. By inconsistency, we mean two or more patterns from different classes are assigned identical discretized values. Using this strategy, we found that many attributes values were merged into just one subinterval. Attributes with values that had been merged into one interval could be removed without introducing any inconsistency in the discretized dataset. Let us denote the original 60 attributes by $\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_{60}$. After discretization, eleven of these attributes were discretized into two or more discrete values. These attributes and the subintervals found by the ChiMerge algorithm are listed in Table 5.

The thermometer coding scheme (Smith 1993) was used for the discretized attributes. Using this coding scheme, a total of 28 binary inputs were needed. Let us denote these inputs by $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{28}$. All patterns with attribute $\mathcal{A}_{35}$ less than 0.1300 were coded by $\mathcal{I}_1 = 0, \mathcal{I}_2 = 1$, while those with attribute value greater than or equal to 0.1300 by

Attribute	Subintervals
$\mathcal{A}_{35}$	$[0, 0.1300), [0.1300, 1]$
$\mathcal{A}_{36}$	$[0, 0.5070), [0.5070, 1]$
$\mathcal{A}_{44}$	$[0, 0.4280), [0.4280, 0.7760), [0.7760, 1]$
$\mathcal{A}_{45}$	$[0, 0.2810), [0.2810, 1]$
$\mathcal{A}_{47}$	$[0, 0.0610), [0.0610, 0.0910), [0.0910, 0.1530), [0.1530, 1]$
$\mathcal{A}_{48}$	$[0, 0.0762), [0.0762, 1]$
$\mathcal{A}_{49}$	$[0, 0.0453), [0.0453, 1]$
$\mathcal{A}_{51}$	$[0, 0.0215), [0.0215, 1]$
$\mathcal{A}_{52}$	$[0, 0.0054), [0.0054, 1]$
$\mathcal{A}_{54}$	$[0, 0.0226), [0.0226, 1]$
$\mathcal{A}_{55}$	$[0, 0.0047), [0.0047, 0.0057), [0.0057, 0.0064), [0.0064, 0.0127), [0.0127, 1]$

Table 5: Relevant attributes found by ChiMerge and their subintervals

$\mathcal{I}_1 = 1, \mathcal{I}_2 = 1$. The other ten attributes were coded in similar fashion.

A network with 6 hidden units and 2 output units was trained using the 104 binarized-feature training dataset. One addition input for hidden unit thresholds was added to the network giving a total of 29 inputs. After training was completed, network connections were removed by pruning. The pruning process was continued until the accuracy of the network dropped below 90 %. The smallest network with accuracy of more than 90 % was saved for rule extraction. Two hidden units and 15 connections were present in the pruned network. Ten inputs $\mathcal{I}_1, \mathcal{I}_3, \mathcal{I}_8, \mathcal{I}_{11}, \mathcal{I}_{12}, \mathcal{I}_{14}, \mathcal{I}_{20}, \mathcal{I}_{22}, \mathcal{I}_{25}$, and $\mathcal{I}_{26}$ were connected to the first hidden unit. Only one input, $\mathcal{I}_{17}$ was connected the second hidden unit. Two connections connected each of the 2 hidden units to the 2 output units.

The hidden unit activation values clustering algorithm found 3 discrete activation values, $-0.95, 0.94$ and -0.04 at the first hidden unit. The number of training patterns having these activation values were 58, 45, and 1, respectively. At the second hidden unit there was only one value, -1. The overall accuracy of the network was 91.35 %. All 49 metal cylinder returns in the training data were correctly classified and 9 rock returns were incorrectly classified as metal cylinder returns.

By computing the predicted outputs of the network with discretized hidden-unit activation values, we found that as long as the activation value of the first hidden unit equals -0.95, a pattern would be classified as a sonar return from metal cylinder. Since 10 inputs were connected to this hidden unit, a new network was formed to allow us to extract rules. It was sufficient to distinguish between activation values of -0.95 and those not equal to -0.95, ie. 0.94 and -0.04. Hence the number of output units in the new network was 2. Samples with activation values equal to -0.95 were assigned target values of $\{1, 0\}$, all other patterns were assigned target values of $\{0, 1\}$. The number of hidden units was 6.

The pruned network is depicted in Figure 3. Note that this network's accuracy rate was 100 %; it correctly distinguished all 58 training patterns having activation values equal to -0.95 from those having activation values 0.94 or -0.04. In order to preserve this accuracy rate, 6 and 3 discrete activation values were required at the two hidden units left in the pruned network. At the first hidden unit, the activations values found were $-1.00, -0.54, -0.12, 0.46, -0.76$ and 0.11. At the second hidden unit, the values were $-1.00, 1.00$ and 0.00. All 18 possible combinations of these values were given as input to X2R. The target for each combination was computed using the weights on the connections from the hidden units to the output units of the network.

Let $\beta(i, j)$ denote hidden unit i taking its jth activation values. The following rules that

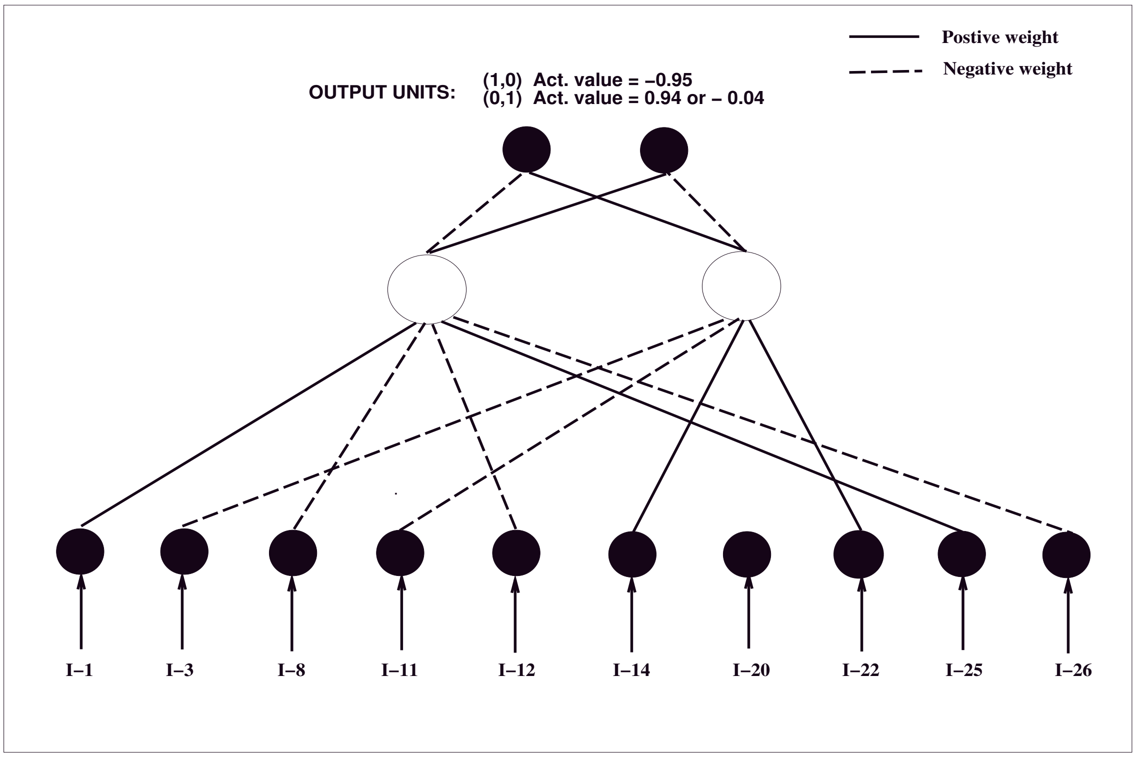

Figure 3: Pruned network trained to distinguish between patterns having discretized activation values equal to -0.95 and those not equal to -0.95.

describe the classification of the sonar returns were generated by X2R:

- If $\beta(1,1)$ then output = Metal.
- Else if $\beta(2,2)$ then output = Metal.
- Else if $\beta(1,2)$ and $\beta(2,3)$ then output = Metal.
- Else if $\beta(1,3)$ and $\beta(2,3)$ then output = Metal.
- Else if $\beta(1,5)$ then output = Metal.
- Default output: Rock.

The activation values of the first hidden units were determined by 5 inputs, $\mathcal{I}_1, \mathcal{I}_8, \mathcal{I}_{12}, \mathcal{I}_{25}$, and $\mathcal{I}_{26}$; while those of the second hidden units by only 4 inputs, $\mathcal{I}_3, \mathcal{I}_{11}, \mathcal{I}_{14}$, and $\mathcal{I}_{22}$. Four activation values at the first hidden unit (-1.00, -0.54, -0.12, and -0.76) and two values at the second hidden unit (1.00 and 0.00) were involved in the rules that classified a pattern as a return from metal cylinders. The rules that determined these activation values were as follows:

- Hidden unit 1: (initially, $\beta(1,1) = \beta(1,2) = \beta(1,3) = \beta(1,5) =$ false)
  - If $(\mathcal{I}_{25} = 0)$ and $(\mathcal{I}_{26} = 1)$ then $\beta(1,1)$.
  - If $(\mathcal{I}_8 = 1)$ and $(\mathcal{I}_{25} = 1)$ then $\beta(1,2)$.
  - If $(\mathcal{I}_8 = 0)$ and $(\mathcal{I}_{12} = 1)$ and $(\mathcal{I}_{26} = 0)$ then $\beta(1,2)$.
  - If $(\mathcal{I}_8 = 0)$ and $(\mathcal{I}_{12} = 1)$ and $(\mathcal{I}_{25} = 1)$ then $\beta(1,3)$.
  - If $(\mathcal{I}_1 = 0)$ and $(\mathcal{I}_{12} = 0)$ and $(\mathcal{I}_{26} = 0)$ then $\beta(1,3)$.
  - If $(\mathcal{I}_8 = 1)$ and $(\mathcal{I}_{26} = 0)$ then $\beta(1,5)$.
- Hidden unit 2: (initially, $\beta(2,2) = \beta(2,3) =$ false)

- If ($\mathcal{I}_3 = 0$) and ($\mathcal{I}_{14} = 1$) then $\beta(2,2)$.
- If ($\mathcal{I}_{22} = 1$) then $\beta(2,2)$.
- If ($\mathcal{I}_3 = 0$) and ($\mathcal{I}_{11} = 0$) and ($\mathcal{I}_{14} = 0$) and ($\mathcal{I}_{22} = 0$) then $\beta(2,3)$.

With the thermometer coding scheme used to encode the original continuous data, it is easy to obtain rules in term of the original attributes and their values. The complete set of rules generated for the sonar returns dataset are given in Appendix B. A total of 8 rules that classify a return as that from metal cylinder were generated. Two metal returns and three rock returns satisfied the conditions of one of these 8 rules. With this rule removed, the remaining 7 rules correctly classified 96 of the 104 training data (92.31 %) and 101 of the 104 testing data (97.12 %).

4 Discussion and comparison with related work

We have shown how rules can be extracted from a trained neural network without making any assumptions about the network's activations or having initial knowledge about the problem domain. If some knowledge is available, however, it can always be incorporated into the network. For example, connections in the network from inputs thought to be not relevant can be given large penalty parameters during training, while those thought to be relevant can be given zero or small penalty parameters (Setiono 1995). Our algorithm does not require thresholded activation function to force the activation values to be zero or one (Towell and Shavlik 1993; Fu 1991), nor does it require the weights of the connections to be restricted in a certain range (Blassig 1994).

Craven and Shavlik (1994) mentioned that one of the difficulties of the search based method such as Saito and Nakano (1988) and Fu (1991) is the complexity of the rules. Some rules are found deep in the search space and these search methods, having exponential complexity, may not be effective. Our algorithm, in contrast, finds intermediate rules embedded in the hidden units by training and pruning new subnetworks. The topology of any subnetwork is always the same, that is, a three-layer feedforward one regardless of how deep these rules are in the search space. Each hidden unit, if necessary, is split into several output units of a new subnetwork. When more than one new networks are created, each one of them can be trained independently. As the activation values of a hidden unit are normally determined by a subset of the original inputs, we can substantially speed up the training of the subnetwork by using only the relevant inputs and removing duplicate patterns. The use of a standard three-layer network is another advantage of our method; any off-the-shelf training algorithm can be used.

The accuracy and the number of rules generated by our algorithm are better than those obtained by C4.5 (Quinlan 1993), a popular machine learning algorithm that extracts rules from decision trees. For the splice-junction problem, C4.5's accuracy rate on the testing data was 92.13 % and the number of rules generated was 43. Our algorithm achieved one percent higher accuracy (93.32 %) with far fewer rules (10). The MofN algorithm achieved a 92.8 % average accuracy rate, the number of rules was 20, and the number of conditions per rule was more than 100. These figures were obtained from ten repetitions of tenfold cross-validation on a set of 1000 randomly selected training patterns (Towell and Shavlik 1993). Using 30 % of the 3190 samples for training, 10 % for cross-validation, and 60 % for testing, the Gradient Descent Symbolic Rule Generation was reported to achieve an accuracy of 93.25 %. There were 3 intermediate rules for the hidden units and 2 rules for the two output units defining the classes EI and IE (Blassig 1994).

For the sonar-return classification problem, the accuracy of the rules generated by our algorithm on the testing set was 2.88 % higher (97.12 % versus 94.24 %) than that of C4.5.

The number of rules generated by RX was only 7, three less than C4.5's rules.

5 Conclusion

Neural networks are often viewed as black boxes. While their predictive accuracy is high, one usually cannot understand why a particular outcome is predicted. In this paper, we have attempted to open up these black boxes. Two factors make this possible. The first is a robust pruning algorithm. By eliminating redundant weights, redundant input and hidden units are identified and removed from the network. Removal of these redundant units significantly simplifies the process of rule extraction and the extracted rules themselves. The second factor is the clustering of the hidden-unit activation values. The fact that the number of distinct activation values at the hidden units can be made small enough enables us to extract simple rules. An important feature of our rule-extraction algorithm is its recursive nature. When after pruning, a hidden unit is still connected to a relatively large number of inputs; in order to generate rules that describe its activation values, a new network is formed. The rule extraction algorithm is applied to new network. By merging the rules extracted from the new network and the rules extracted from the original network, hierarchical rules are generated. The viability of the proposed method has been shown on two sets of real-world data. Compact sets of rules that achieved a high accuracy rate were extracted for these datasets.

Acknowledgements

The author wishes to thank his colleague Huan Liu for providing the results of C4.5 and the two anonymous reviewers for their comments and suggestions which have greatly improved the presentation of this paper.

References

Blassig, R. 1994. GDS: Gradient descent generation of symbolic classification rules. In *Advances in Neural Information Processing Systems*, Vol. 6, 1093–1100. Morgan Kaufmann, San Mateo, CA.

Craven, M.W., and Shavlik, J.W. 1994. Using sampling and queries to extract rules from trained neural networks. In *Proc. of the Eleventh International Conference on Machine Learning*, Morgan Kaufmann, San Mateo, CA.

de Villiers, J., and Barnard, E. 1992. Backpropagation neural nets with one and two hidden layers. *IEEE Trans. on Neural Networks*, **4(1)**, 136–141.

Fu, L. 1991. Rule learning by searching on adapted nets. In *Proc. of the Ninth National Conference on Artificial Intelligence*, 590–595. AAAI Press/The MIT Press, Menlo Park, CA.

Gorman, R.P., and Sejnowski, T.J. 1988. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks* **1**, 75–89.

Hornik, K. 1991. Approximation capabilities of multilayer feedforward neural networks. *Neural Networks* **4**, 251–257.

Kerber, R. 1992. Chi-merge: discretization of numeric attributes. In *Proc. of the Ninth National Conference on Artificial Intelligence*, 123–128. AAAI Press/The MIT Press, Menlo Park, CA.

Lapedes, A., Barnes, C., Burks, C., Farber, R., and Sirotkin, K. 1989. Application of neural networks and other machine learning algorithms to DNA sequence analysis. In *Computers and DNA*, 157-182, Addison Wesley, Redwood City, CA.

H. Liu and S.T. Tan. X2R: A fast rule generator. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 1995.

Murphy, P.M., and Aha, D.W. 1992.

Quinlan, J.R. 1993. In *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Saito, K., and Nakano, R. 1988. Medical diagnosis expert system based on PDP model. *Proc. IEEE International Conference on Neural Networks*, IEEE Press, New York, I255–I262.

Setiono, R. 1995. A penalty-function approach for pruning feedforward neural networks. Submitted to *Neural Computation*.

Smith, M. 1993. *Neural networks for statistical modelling*. Van Nostrand Reinhold, New York, NY.

Thrun, S. 1995. Extracting rules from artificial neural networks with distributed representations. In *Advances in Neural Information Processing Systems*, Vol. 7, The MIT Press, Cambridge, MA.

Towell, G.G., and Shavlik, J.W. 1993. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, **13(1)**, 71–101.

Appendix A

Rules generated by the algorithm for the splice-junction domain (initial values for $\beta(i,j)$ and $\alpha(i,j)$ are false).

- Level 3 (rules that define the activation values of the subnetwork depicted in Fig. 2):
  1. If (@4=A $\wedge$ @5=C) $\vee$ (@4$\neq$A $\wedge$ @5=G), then $\beta(1,2)$,

     else if (@4=A $\wedge$ @5=G), then $\beta(1,3)$,

     else $\beta(1,1)$.
  2. If (@1=G $\wedge$ @2=T $\wedge$ @3=A), then $\beta(2,1)$,

     else if (@1=G $\wedge$ @2=T $\wedge$ @3$\neq$ A), then $\beta(2,2)$,

     else $\beta(2,3)$.
- Level 2 (rules that define the activation values of the second hidden unit of the network depicted in Fig. 1):

  If $\beta(2,3) \vee (\beta(1,1) \wedge \beta(2,2))$, then $\alpha(2,1)$,

  else if $(\beta(1,1) \wedge \beta(2,1)) \vee (\beta(1,2) \wedge \beta(2,2))$, then $\alpha(2,2)$,

  else $\alpha(2,3)$.
- Level 1 (rules extracted from the pruned network depicted in Fig. 1):

  If (@-1=G $\wedge$ $\alpha(2,1)$ $\wedge$ @-2=A), then IE,

  else if $\alpha(2,3)$ $\vee$ (@-1=G $\wedge$ $\alpha(2,2)$)$\vee$ (@-1$\neq$G $\wedge$ $\alpha(2,2)$ $\wedge$ @-2=A), then EI,

  else N.

Upon removal of the intermediate rules in Levels 2 and 3, we obtain the following equivalent set of rules:

```
IE :- @-2 'AGH----'.

IE :- @-2 'AG-V---'.
```

```
IE :- @-2 'AGGTBAT'.

IE :- @-2 'AGGTBAA'. (*)

IE :- @-2 'AGGTBBH'.

EI :- @-2 '--GT-AG'.

EI :- @-2 '--GTABG'.

EI :- @-2 '--GTAAC'.

EI :- @-2 '-GGTAAW'.

EI :- @-2 '-GGTABH'.

EI :- @-2 '-GGTBBG'.

EI :- @-2 '-GGTBAC'. (*)

EI :- @-2 'AHGTAAW'. (*)

EI :- @-2 'AHGTABH'. (*)

EI :- @-2 'AHGTBBG'. (*)

EI :- @-2 'AHGTBAC'. (*)

N.
```

1. An extended version of standard Prolog notation has been used to concisely express the rules. For example, the first rule indicates that an 'A' in position -2, a 'G' in position -1, and an 'H' in position 1, will result in the classification of the boundary type to be an intron/exon.

2. Following convention, the letter B means C or G or T, W means A or T, H means A or C or T, and V means A or C or G. The character '-' indicates any one of the four nucleotides A, C, G or T. Each rule marked by (*) classifies no more than three patterns out the 1006 patterns in the training set. Using the unmarked 10 rules for classification, the following accuracy rates are obtained.

Class	Training data		Testing data	
	Errors (total)	Accuracy (%)	Errors (total)	Accuracy (%)
IE	17 (243)	93.00	48 (765)	93.73
EI	10 (228)	95.61	40 (762)	94.75
N	49 (535)	90.84	134 (1648)	91.50
Total	76 (1006)	92.45	222 (3175)	93.01

Appendix B

Rules generated by the algorithm for the sonar-returns classification problem. (Original attributes are assumed to have been labeled A1,A2,..., A60.)

```
Metal :- A36 < 0.5070 and A48 >= 0.0762.

Metal :- A54 >= 0.0226.

Metal :- A45 >= 0.2810 and A55 < 0.0057.

Metal :- A55 < 0.0064 and A55 >= 0.0057.

Metal :- A36 < 0.5070 and A45 < 0.2810 and A47 < 0.0910 and
         A47 >= 0.0610 and A48 < 0.0762 and A54 < 0.0226 and
         A55 < 0.0057.

Metal :- A35 < 0.1300 and A36 < 0.5070 and A47 < 0.0610 and
         A48 < 0.0762 and A54 < 0.0226 and A55 < 0.0057.

Metal :- A36 < 0.5070 and A45 >= 0.2810 and A47 < 0.0910 and
         A48 < 0.0762 and A54 < 0.0226 and A55 >= 0.0064. (*)

Metal :- A36 < 0.5070 and A45 < 0.2810 and A47 < 0.0910 and
         A47 >= 0.0610 and A48 < 0.0762 and A54 < 0.0226 and
         A55 >= 0.0064. (**)

Rock.
```

The rule marked (*) does not classify any pattern in the training set. The conditions of the rule marked (**) are satisfied by 2 metal returns and 3 rock returns in the training set. Using the unmarked 6 rules, the following accuracy rates are obtained:

Class	Training data		Testing data	
	Errors (total)	Accuracy (%)	Errors (total)	Accuracy (%)
Metal	2 (49)	95.92	0 (62)	100.00
Rock	6 (55)	89.09	3 (42)	92.86
Total	8 (104)	92.31	3 (104)	97.12