

## **NeuroLinear: From neural networks to oblique decision rules**

**Rudy Setiono and Huan Liu**

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 119260

Republic of Singapore

Email:{rudys,liuh}@iscs.nus.sg

### **Abstract**

We present NeuroLinear, a system for extracting oblique decision rules from neural networks that have been trained for classification of patterns. Each condition of an oblique decision rule corresponds to a partition of the attribute space by a hyperplane that is not necessarily axis-parallel. Allowing a set of such hyperplanes to form the boundaries of the decision regions leads to a significant reduction in the number of rules generated while maintaining the accuracy rates of the networks. We describe the components of NeuroLinear in detail by way of two examples using artificial datasets. Our experimental results on real-world datasets show that the system is effective in extracting compact and comprehensible rules with high predictive accuracy from neural networks.

*Keywords:* Rule extraction, oblique-rule, pruning, discretization.

## **1 Introduction**

Neural networks have been widely applied to solve classification problems. Comparisons between neural networks and decision trees algorithms for these problems have shown that in general neural networks can produce better accuracy rates [3, 4, 16, 20]. Recent developments in algorithms that extract rules from neural networks have made neural network techniques even more attractive. The extracted rules allow one to explain the decision process of a neural network. It is not surprising that in the past few years great efforts

have been devoted to finding effective algorithms for extracting rules from a trained neural network.

Gallant's connectionist expert systems [6] and Saito and Nakano's RN method [17] are two early works that attempt to generate rules from neural networks. These systems, however, do not actually extract rules from the networks; instead, they try to generate an explanation for each particular outcome of the networks.

The KT algorithm developed by Fu [5] extracts rules from a trained network. It searches for subsets of connections to a network's unit with summed weight exceeding the bias of that unit. It is assumed that the unit's activation value is close to either 0 or 1. By searching for the proper subsets of the input connections, sets of rules are generated to describe under what conditions the unit's activation will take one of the two values.

The MofN algorithm of Towell and Shavlik [21] clusters the weights of the trained network into equivalence classes. Clusters that do not significantly affect the unit's activation are eliminated. The complexity of the network is further reduced by replacing the weights in all remaining clusters by the average weight of the individual cluster. The rules generation step is similar to Fu's KT algorithm.

A simple rule extraction algorithm is presented by Setiono and Liu [18]. The rules extracted from neural networks are comparable to those generated by decision trees [15] in terms of accuracy and comprehensibility. The basic idea behind the algorithm is the fact that it is generally possible to replace the continuous activations of the hidden units by a small number of discrete ones. Rule extraction is realized in two steps. First, rules that describe the network outputs in terms of the discretized activation values of the hidden units are generated. Second, rules that describe each discretized hidden unit activation values in terms of the network inputs are constructed. By merging the rules obtained in these two steps, a set of rules that relates the inputs and outputs of the network is obtained.

While the algorithm can generate symbolic rules that mimic the predicted outcome of the original network, it works only for data with binary inputs (the implicit assumption, that the various hidden unit activation values are determined by only a small number of

input values, excludes problems with continuous attributes where there can be infinitely many possible values taken by these attributes). Data with some continuous attributes need to be discretized before training the network.

An inherent problem introduced by the discretized data is that each condition of a rule involving a continuous attribute determines an axis-parallel decision boundary. For many classification problems, it is often more natural to allow oblique hyperplanes to form the boundaries of the decision regions. In other words, instead of imposing the axis-parallel constraint, being able to generate oblique decision hyperplanes makes it possible to let the learning algorithm determine what kind of hyperplanes is more suitable for the data in hand. Oblique hyperplanes are more general and they may substantially reduce the number of rule conditions needed to describe the decision region.

In this paper, we describe how a set of rules, where each rule condition is given in the form of the linear inequality

$$\sum_i c_i x_i < \eta \tag{1}$$

where  $c_i$  is a real coefficient,  $x_i$  the value of the attribute  $i$ , and  $\eta$  a threshold, can be extracted from a neural network. The neural network that we use is the standard feedforward neural network with a single hidden layer. In contrast to the tree growing algorithms which generate the rules in a level-by-level and top-down fashion, rules are extracted from a network in two steps: from the hidden layer to the output layer and from the input layer to the hidden layer. Classification rules are obtained by merging the rules from these two steps. Unlike the decision tree algorithms [15, 2] which consider smaller and smaller subsets of the data to improve the accuracy of the rules, the neural network approach for rule generation considers the entire training set as a whole. Our experimental results show that more compact sets of rules with high accuracy rates can be obtained by the network approach.

The organization of the paper is as follows. Section II describes NeuroLinear, a system that we have developed for extracting oblique decision rules from a neural network. In Section III, the steps of NeuroLinear are illustrated in detail by way of two examples

using artificial datasets. Section IV presents our experimental results on several real-world problems. For two of these problems, a heart disease diagnosis problem and a breast cancer diagnosis problem, we also describe how the rules are extracted by NeuroLinear. Section V analyzes the merits of generating a set of classification rules with NeuroLinear. It also highlights the differences between the rules generated from a network and those from the decision-tree method C4.5rules [15]. Section VI gives a brief conclusion of the paper.

## 2 Rule extraction with Neurolinear

The steps of extracting oblique decision rules from a neural network are as follows:

1. Select and train a network to meet a prespecified accuracy requirement.

Remove the redundant connections in the network by pruning while maintaining the accuracy.

2. Discretize the hidden unit activation values of the network.
3. Extract rules that describe the network outputs in terms of the discretized network activation values.

For each discretized hidden unit activation value, generate a rule in terms of the network's inputs.

Merge the two sets of rules obtained above.

The details of these steps are given below.

### *2.1. Neural Network Training and Pruning*

The basic structure of the neural network is a standard three-layer feedforward network, which consists of an input layer, a hidden layer, and an output layer. The number of input units corresponds to the dimensionality of the patterns in the problem. The number of output units is determined by the number of classes in the dataset. The number of hidden units depends on the problem in hand. Two approaches to determine a suitable number

of hidden units have been described in the literature. The first approach begins with a minimal number of hidden units, one or two, and more hidden units are added as they are needed to increase the accuracy of the network. The second approach begins with an oversized network and removes redundant connections in the network by pruning. In the process, hidden units that are not connected to any input units or/and output units can be removed as well. We adopt the second approach since we are also interested in removing input units that are irrelevant to the classification. During the pruning phase, irrelevant input units and hidden units can be identified and removed from the network.

Given an  $n$ -dimensional pattern  $x^i$ ,  $i \in \{1,2,\dots,k\}$  as input, let  $w_l^m$  be the weight for the connection from input unit  $l$ ,  $l \in \{1,2,\dots,n\}$  to hidden unit  $m$ ,  $m \in \{1,2,\dots,h\}$  and  $v_p^m$  be the weight from hidden unit  $m$  to output unit  $p$ ,  $p \in \{1,2,\dots,o\}$ . The  $p$ th output of the network for pattern  $x^i$  is obtained by computing

$$S_p^i = \sigma\left(\sum_{m=1}^h \alpha^m v_p^m\right), \quad (2)$$

where

$$\sigma(x) = \text{sigmoid}(x) = 1/(1 + e^{-x}), \quad (3)$$

$$\alpha^m = \delta\left(\sum_{l=1}^n x_l^i w_l^m\right), \quad (4)$$

$$\delta(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x}). \quad (5)$$

The target output for a pattern  $x^i$  that belongs to class  $C_j$  is an  $o$ -dimensional vector  $t^i$ , where  $t_p^i = 0$  if  $p \neq j$  and  $t_j^i = 1$ ,  $j, p = 1, 2, \dots, o$ . The backpropagation algorithm is applied to update the weights  $(w, v)$  and minimize the following function:

$$\theta(w, v) = F(w, v) + P(w, v),$$

where  $F(w, v)$  is the cross entropy function [22]:

$$F(w, v) = -\sum_{i=1}^k \sum_{p=1}^o \left( t_p^i \log S_p^i + (1 - t_p^i) \log(1 - S_p^i) \right).$$

and  $P(w, v)$  is a penalty term used for weight decay [8]:

$$P(w, v) = \epsilon_1 \left( \sum_{m=1}^h \sum_{\ell=1}^n \frac{\beta(w_\ell^m)^2}{1 + \beta(w_\ell^m)^2} + \sum_{m=1}^h \sum_{p=1}^o \frac{\beta(v_p^m)^2}{1 + \beta(v_p^m)^2} \right)$$

$$+ \epsilon_2 \left( \sum_{m=1}^h \sum_{\ell=1}^n (w_{\ell}^m)^2 + \sum_{m=1}^h \sum_{p=1}^o (v_p^m)^2 \right), \quad (6)$$

where  $\epsilon_1$ ,  $\epsilon_2$ , and  $\beta$  are positive decay parameters.

One of the advantages of having a neural network with only the relevant connections is that the behavior of the net can be explained by a simple set of rules [9]. Our pruning algorithm removes connections in the network based on their magnitude. The details of this algorithm and the experimental results on a number of well known classification problems are given in [19]. The effectiveness of the pruning algorithm is shown by the fact that for the many problems tested, the final pruned networks have only the relevant connections left regardless of the number of initial hidden units in the networks.

## 2.2. Chi2: Discretization of Hidden Unit Activation Values

The range of the activation values of the network's hidden units is the interval  $[-1, 1]$ , since they have been computed as the hyperbolic tangent of the weighted inputs (cf. Eqn. 4). In order to extract rules from the network, it is necessary that these values be grouped into a few clusters while preserving the accuracy of the network. Chi2 [11], an improved and automated version of ChiMerge [10], is the algorithm used for this purpose.

Given a dataset where each pattern is described by the values of the continuous attributes  $\mathcal{A}_1, \mathcal{A}_2, \dots$  and the class label of the pattern is known, Chi2 finds discrete representations of the dataset. Using the  $\chi^2$  statistic, Chi2 divides the range of the attributes into subintervals and assigns all values that fall in a subinterval a unique discrete value. The outline of the algorithm is as follows:

---

### The Chi2 algorithm

1. Let Chi-0 be an initial critical value.
2. For each attribute  $\mathcal{A}_j$ :
  - (a) Sort the data according to the the input values of attribute  $\mathcal{A}_j$ .
  - (b) Form an initial set of intervals such that each interval contains only one unique value.

3. Initialize all attributes as “unmarked”.

4. For each unmarked attribute  $\mathcal{A}_i$ :

(a) For each adjoining pairs of subintervals, compute their  $\chi^2$  values:

$$\chi^2 = \sum_{i=1}^2 \sum_{j=1}^k \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \quad (7)$$

where:

$k$ : the number of classes,

$A_{ij}$ : the number of samples in the  $i$ th interval,  $j$ th class,

$R_i$ : the number of samples in the  $i$ th interval =  $\sum_{j=1}^k A_{ij}$ .

$C_j$ : the number of samples in the  $j$ th class =  $\sum_{i=1}^2 A_{ij}$ .

$E_{ij}$  : expected frequency of  $A_{ij}$ ,  $E_{ij} = R_i \times C_j / N$ . If  $R_i$  or  $C_j = 0$ ,  $E_{ij}$  is set to 0.05,

$N$ : the total number of samples.

(b) Find two subintervals with the lowest  $\chi^2$  value. If this value is less than Chi-0 and if merging the subintervals does not introduce conflicting data<sup>1</sup>, then merge these subintervals, and repeat from Step 4(a). Else if merging the subintervals will introduce conflicting data, label attribute  $i$  as “marked”.

5. If there is still an unmarked attribute, then increase Chi-0 and repeat Step 4.

---

The Chi2 algorithm involves only 1 parameter, the initial critical value Chi-0. This critical value is used to determine whether the null hypothesis that the subintervals and the class labels are independent can be rejected. If the test statistic (Eqn. 7) exceeds the critical value Chi-0, the null hypothesis is rejected. Otherwise, the null hypothesis is not rejected and the subintervals are merged. The critical value Chi-0 is determined by the significance

---

<sup>1</sup>Conflicting data occur when there are two or more patterns from different classes with the same discretized attribute values.

level of the test,  $\alpha$ . For example, if  $\alpha = 0.5$  and the number of classes is 3, the critical value is 1.386. If no inconsistency in the data is introduced after merging of the subintervals with the initial value of Chi-0, the critical value Chi-0 is increased, i.e., the significance level is reduced to check the possibility of further merging of the subintervals.

### 2.3. Rule Generation

Rules are generated in two phases. First, rules that describe the classification are obtained in terms of the discretized hidden unit activation values. Second, rules that describe each discretized hidden unit activation values are obtained in terms of the original attributes of the dataset.

For the first phase, we have implemented an efficient rule generator called X2R [12]. It generates a set of rules which cover all the data with an error rate not exceeding the inconsistency rate present in the data. It is particularly suitable for moderate sized datasets with discrete attribute values. When there are few hidden units left in the pruned network where each of these hidden units has a relatively small number of different discrete activation values, it may be possible to find a set of classification rules without the help of any computer program. Examples in the next section illustrate this possibility.

The discretized activation values at all the remaining hidden units are not the only output of the Chi2 algorithm. Chi2 also provides the boundaries of the subintervals of the activation values after the merging process has terminated. Let  $N$  be the number of clusters or subintervals found by Chi2 for the activation values of hidden unit  $H$ . There are  $N + 1$  real numbers  $\mu_0, \mu_1, \mu_2, \dots, \mu_N$  such that  $-1 = \mu_0 < \mu_1 < \mu_2 < \dots < \mu_{N-1} < \mu_N = 1$ , which form these  $N$  clusters. An activation value  $\alpha$  of an input pattern will be discretized into the  $j$ -th cluster if  $\mu_{j-1} \leq \alpha < \mu_j$ . The activation value  $\alpha$  is obtained by applying the tangent hyperbolic function (5) to the weighted inputs. Hence, an activation value falls into subinterval  $[\mu_{j-1}, \mu_j)$  if its weighted inputs satisfy the condition

$$\tanh^{-1}(\mu_{j-1}) \leq \text{weighted inputs} < \tanh^{-1}(\mu_j), \quad (8)$$

where  $\tanh^{-1}(x)$  is the inverse of the tangent hyperbolic function

$$\tanh^{-1}(x) = \log((1+x)/(1-x))/2.$$

The condition (8) defines the halfspaces (in the case of  $j = 1$  or  $j = N$ ) or the intersection of two halfspaces (in the case of  $j = 2, 3, \dots, N - 1$ ) in the original input space of the dataset where a pattern will have a discretized hidden unit activation value located in the  $j$ -th subinterval.

Note that in order to obtain the rules that describe the relation between hidden unit activation values and the class labels, the weights of the connections between the hidden units and the output units are not needed. On the other hand, rule conditions that determine the subintervals of the discretized activation values are specified by the weights of the network connections from the input units to the hidden units. Combining the rules and conditions obtained from the two phases, we find decision boundaries in terms of the original attributes that classify the patterns of the dataset.

### 3 Illustrative examples

In this section, we describe in detail the steps of the NeuroLinear algorithm on two small synthetic problems. The first problem is designed to demonstrate how the algorithm discovers two linear equations that form the decision boundaries from a single network. Demonstrated in the second problem is how a piecewise-linear decision boundary can also be generated by the algorithm.

#### A. Example 1

The patterns for the first example were generated randomly. Each patterns has two inputs  $(x_1, x_2)$ , each of which is uniformly distributed in the interval  $[0, 1]$ . The target value  $t$  of each pattern is defined as follows (Fig. 1 (a)):

If  $((-x_2 + 1.5 x_1 \geq -0.25) \text{ AND } (x_2 \geq x_1))$ , then  $t = 1$ ,

Else  $t = 0$ .

Two thousand patterns were generated for the experiment as the training data. In addition, another 2000 patterns were similarly generated to form the test dataset. Ten

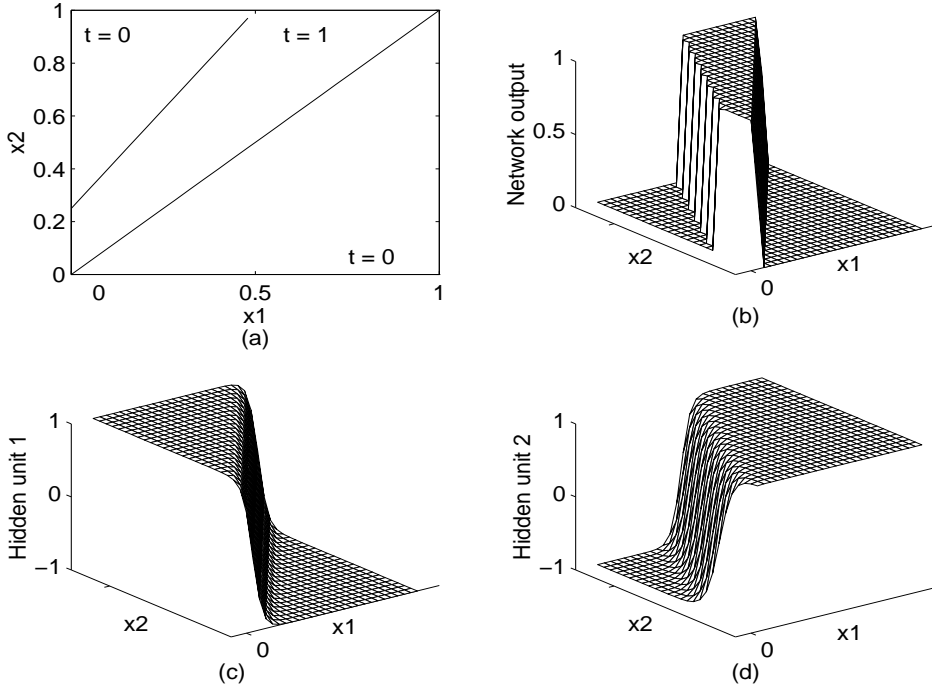


Figure 1: Example 1. (a) The lines  $x_2 = x_1$  and  $x_2 - 1.5 x_1 = 0.25$  form the decision boundaries of the problem. (b) Predicted output of a network with 3 hidden units. (c) The activation values at hidden unit 1. (d) The activation values at hidden unit 2. The activation values at hidden unit 3 are constant.

networks each having 3 input units, 8 hidden units, and 1 output unit were used as the starting networks. The 3rd input served as hidden unit threshold, its values for all patterns were 1. All initial weights of these networks were randomly and uniformly generated in the interval  $[-1, 1]$ . After the network training reached a local minimum solution, the pruning process was initiated. Pruning was terminated as soon as the accuracy of the network dropped below 98 %. The smallest network with at least 98% accuracy on the training data was saved for possible rule extraction. The summary of the 10 runs is given in Table 1.

The total number of epochs indicates that the cost of retraining the network after some connections had been removed was about three times the cost of training the fully connected network (all figures in parentheses are standard deviations). The effectiveness of

Table 1

Results of 10 runs of the network pruning algorithm for example 1. The figures in the second columns are the averages and standard deviations.

Total epochs	
Training	183.2 (67.53)
Pruning	590.8 (129.17)
Number of hidden units	
Before pruning	8 (0)
After pruning	3 (0)
Number of connections	
Before pruning	32 (0)
After pruning	8.10 (0.32)
Ave. accuracy on training set	
Before pruning	99.73 (0.10)
After pruning	98.99 (0.34)
Ave. accuracy on testing set	
Before pruning	99.91 (0.03)
After pruning	99.63 (0.10)

Table 2

Subintervals found by Chi2 for the activation values of the 3 hidden units of the network for example 1.

Hidden unit	Subintervals
1	$[-1, 0.0), [0.0, 1]$
2	$[-1, -0.09), [-0.09, 1]$
3	$[-1, 1]$

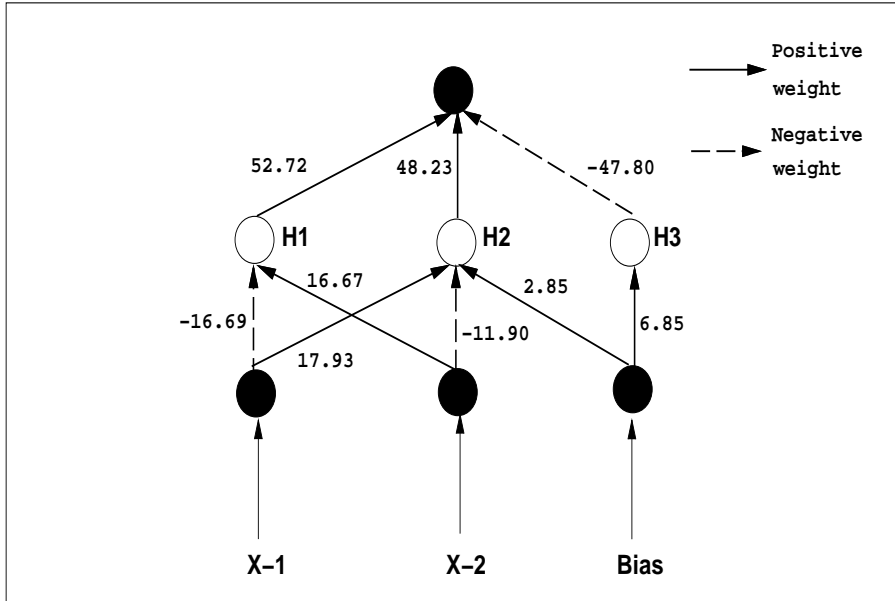


Figure 2: A pruned network for Example 1. Its accuracy rate on 2000 training patterns is 99.60 % and it has only 9 connections.

the pruning algorithm is shown by the small number of connections left in the networks. The small standard deviations for the average connections, the number of hidden units and the accuracy of the pruned networks suggest that all pruned networks had similar topology and accuracy. One of the pruned networks was selected for generation of piecewise-linear separators for the patterns. This network is shown in Fig. 2. The predicted network outputs, the activation of hidden units 1 and 2 for points in the square  $[0, 1] \times [0, 1]$  are shown in Fig. 1 (b-d).

The Chi2 algorithm was applied to the activation values of hidden units 1 and 2. Hidden unit 3 is only connected to the third input which is the bias, hence its activation values would be the same for all input patterns and were discretized to one value by Chi2. The results of Chi2 on hidden units 1 and 2 and the decision regions obtained were as follows:

1. Hidden unit 1. There were 2 clusters found. All activation values in the interval  $[-1, 0)$  formed the first cluster and all values in the interval  $[0, 1]$  formed the second cluster. Referring to Fig.2, the activation values at hidden unit 1 are determined by

the two inputs,  $x_1$  and  $x_2$ . The weights of the two connections from the input units are -16.69 and 16.67, respectively. Consequently, all patterns with inputs  $x_1$  and  $x_2$  such that  $-16.69x_1 + 16.67x_2 < 0$  will have activation values that fall in the first cluster. Otherwise, they will fall in the second cluster. Let us denote  $\alpha_1 = 1$  or 2 to indicate the two clusters of hidden unit 1. In summary, the following linear separator is obtained:

If  $-x_1 + x_2 < 0$ ,  $\alpha_1 = 1$ ,  
else  $\alpha_1 = 2$ .

2. Hidden unit 2. There were also 2 clusters found. All activation values in the interval  $[-1, -0.09)$  formed the first cluster and all values in the interval  $[-.09, 1]$  formed the second cluster. The common boundary of these two intervals is -0.09. Two inputs and a bias value determine the activation values of this hidden unit. The weights are 17.93, -11.90 and 2.85. Hence, if  $x_1$  and  $x_2$  are such that  $17.93x_1 - 11.90x_2 + 2.85 < \tanh^{-1}(-0.09)$ , then the pattern's activation values will be in the first interval, otherwise it will be in the second subinterval. Simplifying the inequality and using similar notation used earlier for the two clusters in hidden unit 1, we obtain:

If  $1.51x_1 - x_2 < -0.25$ ,  $\alpha_2 = 1$ ,  
else  $\alpha_2 = 2$ .

Activation values of each pattern were replaced by their cluster values (which are either 1 or 2). When this was done, only 3 distinct patterns remained, they were (2, 2), (1, 2) and (2, 1). They represent all points in the three regions in Fig. 1 (a). Only pattern (2, 2) has a target value equal to 1. Without much effort, we can conclude the following:

If  $-x_1 + x_2 \geq 0$  and  $1.51x_1 - x_2 \geq -0.25$ , target = 1.  
else target = 0.

Thus the original classification rules are rediscovered by NeuroLinear. Recovering these rules would be a difficult task for DT methods such as C4.5rules which generates decision

planes that are parallel to the coordinate axis. For this kind of problems, intuitively, DT methods generate trees of large size proportional to the sample size. Experimental evidence can be found in [7] (Fig. 6).

*B. Example 2*

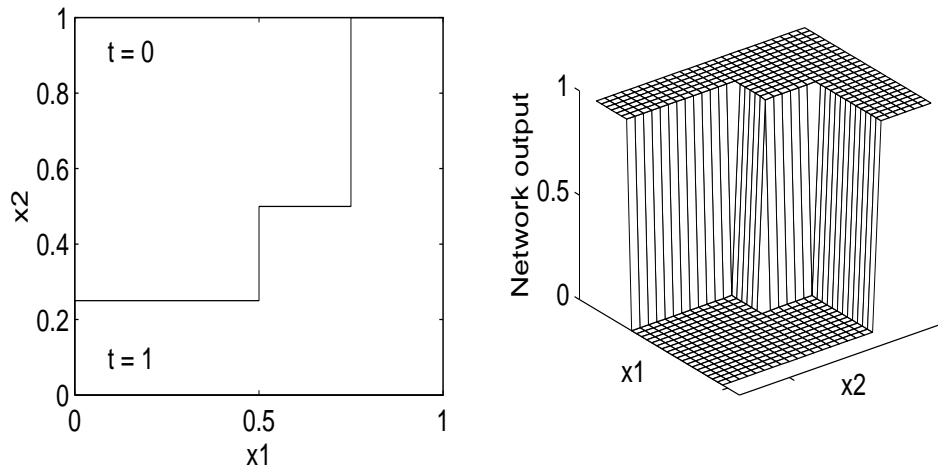


Figure 3: Example 2. A piecewise-linear function separates the two classes of the patterns (left) and the output of a neural network trained with 2000 patterns (right).

The patterns in this example also have 2 inputs. The values for these inputs were generated randomly and uniformly in the interval  $[0, 1]$ . The training and testing set each consisted of 2000 patterns. The class of each pattern was determined according to the following function:

- If  $x_2 \leq 0.25$ , then  $t = 1$ ,
- else if  $x_2 \leq 0.5$  and  $x_1 \geq 0.5$ , then  $t = 1$ ,
- else if  $x_1 \geq 0.75$ , then  $t = 1$ ,
- else  $t = 0$ .

The separating line between the two classes is piecewise-linear as shown in Fig. 3.

Table 3

Results of 10 runs of the network pruning algorithm for example 2.

Total epochs	
Training	263.7 (117.66)
Pruning	643.5 (277.71)
Number of hidden units	
Before pruning	8 (0)
After pruning	5 (0)
Number of connections	
Before pruning	32 (0)
After pruning	14.50 (1.72)
Ave. accuracy on training set	
Before pruning	99.39 (0.31)
After pruning	99.24 (0.29)
Ave. accuracy on testing set	
Before pruning	99.56 (0.12)
After pruning	99.63 (0.15)

As in Example 1, ten neural networks with 8 hidden units each were trained and pruned. The statistics of these runs are summarized in Table 3. One of the pruned network with an average number of connections is depicted in Fig. 4. This network was used to generate a separator between the two classes.

Chi2 discretized the hidden units activation values of the network. The subintervals found by the algorithm are summarized in Table 4. Let us denote  $\alpha_j = 1$  or 2 to indicate the two clusters of activation values of hidden unit  $j$ ,  $j = 1, 3, 4$  and 5, with  $\alpha_j = 1$  for the clusters in the first subintervals and  $\alpha_j = 2$  for the clusters in second subintervals. When all training patterns were given using their clustered activation values, nine distinct patterns were found. In effect, each of these nine patterns corresponds to a region determined by

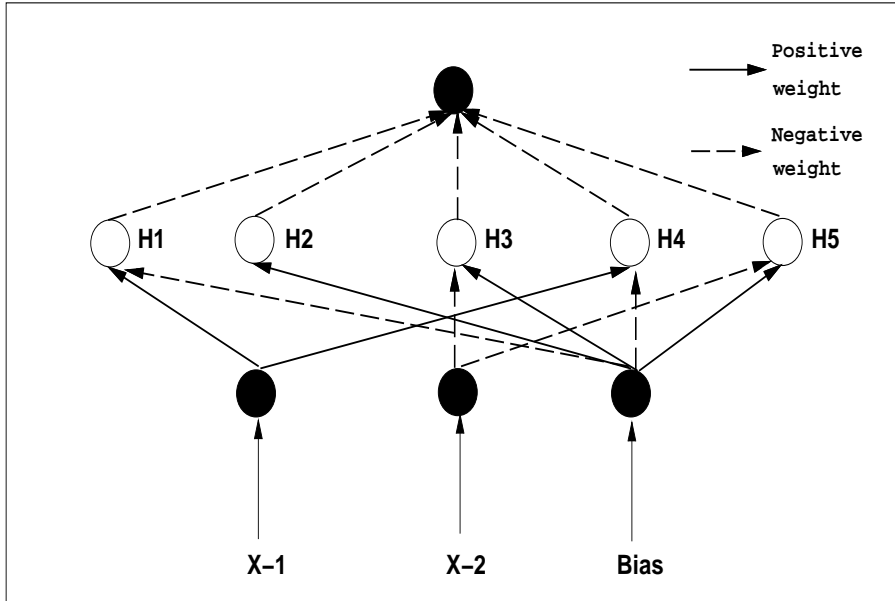


Figure 4: A pruned network with 15 connections for Example 2. Its accuracy rate on the 2000 training patterns is 99.55 %. Weights are omitted for clarity.

two or three line equations that are present in the problem.

Rules that determine the class labels were generated by X2R in terms of the clustered activation values at the 5 hidden units. They are as follows:

If  $\alpha_1 = \alpha_3 = 1$ , then  $t = 0$

Else if  $\alpha_1 = \alpha_4 = \alpha_5 = 1$ , then  $t = 0$

Default rule:  $t = 1$ .

The boundaries of the subintervals found by Chi2 and the weights of the network connections between input units and hidden units allow one to express the above rules in terms of the original attributes  $x_1$  and  $x_2$ :

If  $x_1 < 0.75$  and  $x_2 > 0.47$ , then  $t = 0$

Else if  $x_1 < 0.50$  and  $x_2 > 0.25$ , then  $t = 0$

Default rule:  $t = 1$ .

Table 4

Subintervals found by Chi2 for the activation values of the 5 hidden units of the network for example 2.

Hidden unit	Subintervals
1	$[-1, -0.09), [-0.09, 1]$
2	$[-1, 1]$
3	$[-1, 0.98), [0.98, 1]$
4	$[-1, 0.07), [0.07, 1]$
5	$[-1, -0.69), [-0.69, 1]$

## 4 Experimental results

Table 5

Datasets used in the experiments.

Dataset	Size	Attributes	
		Discrete	Continuous
Australian Credit Approval	690	8	6
Boston Housing Data	506	1	12
Cleveland Heart Disease	297	5	8
Wisconsin Breast Cancer	699	0	9
Sonar Target	208	0	60

We report our experiments with NeuroLinear in this section. Five datasets from the machine learning data repository at the University of California, Irvine [14] are used to compare the performance of NeuroLinear to that of C4.5rules [15]. C4.5rules is chosen because the code is widely available and it has been commonly used in the machine learning community. The datasets and the characteristics of their attributes are given in Table 5.

Following Towell and Shavlik [21], for each dataset, ten repetitions of ten-fold cross validation were performed using NeuroLinear. Each neural network was given a set of initial weights randomly generated in the interval  $[-1, 1]$ . For all networks, the following

Table 6

Accuracy rates (%) of C4.5rules and Neurolinear.

Dataset	C4.5rules	NeuroLinear	P-value
Australian Credit Approval	84.22 (2.93)	83.64 (5.74)	0.60
Boston Housing Data	83.81 (5.90)	80.60 (9.12)	0.28
Cleveland Heart Disease	75.45 (7.17)	78.15 (6.86)	0.24
Wisconsin Breast Cancer	95.28 (2.51)	95.73 (3.75)	0.71
Sonar Target	85.61 (8.64)	85.39 (12.77)	0.96

values were fixed: the number of hidden units was 4, the number of output unit was 1. The penalty parameters  $\epsilon_1$  and  $\epsilon_2$  were set at 0.1 and  $10^{-3}$ , respectively. The value of  $\beta$  was 10. C4.5rules was run to perform ten-fold cross validation with its default parameter values. The results of the experiments are summarized in Tables 6 and 7.

Table 7

Number of rules of C4.5rules and Neurolinear.

Dataset	C4.5rules	NeuroLinear	P-value
Australian Credit Approval	14.60 (2.88)	6.60 (4.40)	0.0001
Boston Housing Data	15.20 (3.01)	3.05 (3.23)	0.0001
Cleveland Heart Disease	12.90 (2.85)	5.69 (4.25)	0.0001
Wisconsin Breast Cancer	8.90 (1.20)	2.89 (2.52)	0.0001
Sonar Target	9.70 (1.57)	7.03 (3.73)	0.0003

In the two tables, the average accuracy rates and the average number of rules obtained by C4.5rules and NeuroLinear are given. The figures in parentheses are the standard deviations. The P-values are computed for testing the null hypothesis that the means of two groups of observations are equal. The P-values for the accuracy rates in Table 6 show that there is no significant difference in the mean accuracy rates of C4.5rules and NeuroLinear. On the other hand, the large differences in the numbers of rules of C4.5rules and NeuroLinear in Table 7 clearly demonstrate the effect of using oblique hyperplanes as the rule conditions.

Their corresponding small P-values verify the significance of these differences.

We describe next in detail how the rules are extracted by NeuroLinear on two datasets.

*A. Detailed analysis 1: The University of Wisconsin Breast Cancer Dataset.*

This data set has been used as the test data for several studies on pattern classification methods using linear programming techniques [1, 13] and statistical techniques [23]. Each pattern is described by nine attributes. The nine measurements taken from fine needle aspirates from human breast tissues correspond to cytological characteristics of a benign or of a malignant pattern. They are summarized in Table 8.

Table 8

The 9 attributes of the Wisconsin Breast Cancer dataset.

Attribute	Description
$\mathcal{A}_1$	clump thickness
$\mathcal{A}_2$	uniformity of cell size
$\mathcal{A}_3$	uniformity of cell shape
$\mathcal{A}_4$	marginal adhesion
$\mathcal{A}_5$	single epithelial cell size
$\mathcal{A}_6$	bare nuclei
$\mathcal{A}_7$	bland chromatin
$\mathcal{A}_8$	normal nucleoli
$\mathcal{A}_9$	mitosis

Each of these nine attributes of the fine needle aspirates was graded 1 to 10 at the time of sample collection, with 1 being the closest to benign and 10 the most anaplastic. Due to their ordinal nature, we treated the attribute values as continuous. The values were normalized such that they ranged in the interval  $[0, 1]$ . The training set consisted of 350 randomly selected patterns, 121 of which are malignant patterns and the remaining 229 benign patterns. The testing set consisted of 120 malignant patterns and 229 benign patterns. Benign patterns were given a target value of 0, while malignant ones 1.

A fully connected network was trained and then pruned. The number of input units was 10 and the number of hidden units was 4. Let us label the inputs  $\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_{10}$ . The input value of  $\mathcal{I}_{10}$  was set to 1 for all patterns. The weight of a connection between this input and a hidden unit is the threshold or bias at that hidden unit. The connection weights of the smallest network with an accuracy rate on the training data of at least 98% were saved. The pruning process was resumed and terminated when the accuracy of the network fell below 95%. The weights of the smallest network with accuracy rate higher than 95% were also saved for rules generation.

A network with only 10 connections and 98 % accuracy rate on the training set was found. Only 2 of the original 4 hidden units remained. Hidden unit 1 was connected to inputs  $\mathcal{I}_3, \mathcal{I}_4, \mathcal{I}_9$  and  $\mathcal{I}_{10}$ . Hidden unit 2 was also connected to 4 inputs:  $\mathcal{I}_3, \mathcal{I}_6, \mathcal{I}_8$  and  $\mathcal{I}_{10}$ .

The activation values of all 343 patterns that were correctly classified by the pruned network were discretized by Chi2. It found 4 clusters in the first hidden unit and 3 clusters in the second hidden unit. The subintervals that define the 4 clusters of the first hidden unit are  $[-1, 0.804), [0.804, 0.988), [0.988, 0.998), [0.998, 1]$ . The subintervals at the second hidden unit are  $[-1, 0.376), [0.376, 0.602), [0.602, 1]$ . A new data set with 2 columns of discrete values was generated. All patterns with discretized activation values at hidden unit  $i$  located in the  $j$ -th subinterval were given a value equal to  $j$  in their  $i$ -th column. The target values for these patterns were their original class label, which was either 0 (benign) or 1 (malignant). After removing all duplicates, only 10 unique patterns remained. Four had target value equal to 0, and the rest had target value equal to 1. Denoting column  $i$  having value  $j$  as  $\alpha_i = j$ , simple rules to distinguish the two sets were generated:

If  $\alpha_1 = 1$  or  $\alpha_2 = 3$ , then target = 1.

Default rule: target = 0.

Given the weights of the pruned network, it is easy to rewrite the rule in terms of the original attributes of the dataset. The condition  $\alpha_1 = 1$  is satisfied by a pattern if and only if its first hidden unit activation value is located in the interval  $[-1, 0.804)$ . An activation value of a pattern will be in this interval if and only if the weighted sum of its input is

less than  $\tanh^{-1}(0.804) = 1.110$ . Similarly, the second hidden unit activation values of a pattern will fall in the third subinterval  $[0.602, 1]$  if and only if its weighted sum is greater than or equal to  $\tanh^{-1}(0.602) = 0.696$ . We obtain the following rules:

If  $-5.62\mathcal{I}_3 - 3.16\mathcal{I}_4 - 2.93\mathcal{I}_9 + 4.94 < 1.110$  or  $2.77\mathcal{I}_3 + 4.05\mathcal{I}_6 + 2.10\mathcal{I}_8 - 1.46 \geq 0.696$ ,  
then target = 1 (or malignant).

Default rule: target = 0 (or benign).

Let  $\mathcal{A}_{j,\min}$  and  $\mathcal{A}_{j,\max}$  be the minimum and the maximum values of attribute  $\mathcal{A}_j$  in the data for all  $j = 1, 2, \dots, 9$ . Then for each pattern  $i$  the following equation relates the normalized input  $\mathcal{I}_j^i$  to its original value  $\mathcal{A}_j^i$ :

$$\mathcal{I}_j^i = \left( \mathcal{A}_j^i - \mathcal{A}_{j,\min} \right) / \left( \mathcal{A}_{j,\max} - \mathcal{A}_{j,\min} \right).$$

Since,  $\mathcal{A}_{j,\min} = 1$  and  $\mathcal{A}_{j,\max} = 10$  for all  $j$ , it is easy to find an equivalent set of rules in terms of the original attributes  $\mathcal{A}_j$ . We obtain

If  $5.62\mathcal{A}_3 + 3.16\mathcal{A}_4 + 2.93\mathcal{A}_9 > 46.18$  or  $2.77\mathcal{A}_3 + 4.05\mathcal{A}_6 + 2.10\mathcal{A}_8 \geq 28.324$ ,  
then target = 1 (or malignant).

Default rule: target = 0 (or benign).

The accuracy of the rules on the training set is 98.57 %. This figure is actually higher than the pruned network accuracy. Two patterns that were misclassified by the network and not included in the discretization process were correctly classified by the rules. The accuracy on the testing set is also slightly higher, 95.42%.

From the second and smaller network, accuracy rates of 96.86 % and 94.27% were obtained on the training and testing sets. There were 2 hidden units and only 5 connections in the network. Inputs  $\mathcal{I}_3$  and  $\mathcal{I}_{10}$  were connected to the first hidden unit, their weights were -1.33 and 0.72, respectively. The only input connected to the second hidden unit was input  $\mathcal{I}_6$  with weight equal to 2.22. The results of the Chi2 algorithm applied to the activation values at the hidden units are summarized in Table 9.

Table 9

Subintervals found by Chi2 for the Wisconsin Breast Cancer dataset. The network achieves 96.86 % accuracy rate on the training set and has only 5 connections.

Hidden unit	Subintervals
1	$[-1, 0.273), [0.273, 0.404), [0.404, 0.520), [0.520, 1]$
2	$[-1, 0.241), [0.241, 0.456), [0.456, 0.628), [0.628, 0.843), [0.843, 1]$

Rules that classify a pattern to be either benign or malignant are first generated in terms of its discretized hidden unit activation values. The rules are as follows:

If  $\alpha_1 = 1$ , then target = 1,

else if  $\alpha_1 = 2$  and  $\alpha_2 \geq 3$ , then target = 1,

else if  $\alpha_1 = 3$  and  $\alpha_2 \geq 4$ , then target = 1,

else if  $\alpha_2 = 5$ , then target = 1.

Default rule: target = 0.

In terms of the original attributes, we obtain the following piecewise-linear separator for the dataset:

If  $\mathcal{A}_3 > 4$ , then target = 1 (or malignant),

else if  $\mathcal{A}_3 > 3$  and  $\mathcal{A}_6 \geq 3$ , then target = 1 (or malignant),

else if  $\mathcal{A}_3 > 2$  and  $\mathcal{A}_6 \geq 4$ , then target = 1 (or malignant),

else if  $\mathcal{A}_6 \geq 6$ , then target = 1 (or malignant).

Default rule: target = 0 (or benign).

The accuracy rates of this set of rules are 97.14 % on the training data and 94.27 % on the testing data. For comparison, the corresponding accuracy rates of C4.5rules are 98.0 % and 92.6 %, respectively. There are 6 rules generated by C4.5rules. One rule has 1 condition, 3 have 2 conditions and 2 have 3 conditions. Attributes 1 to 7 appear in these rules. In

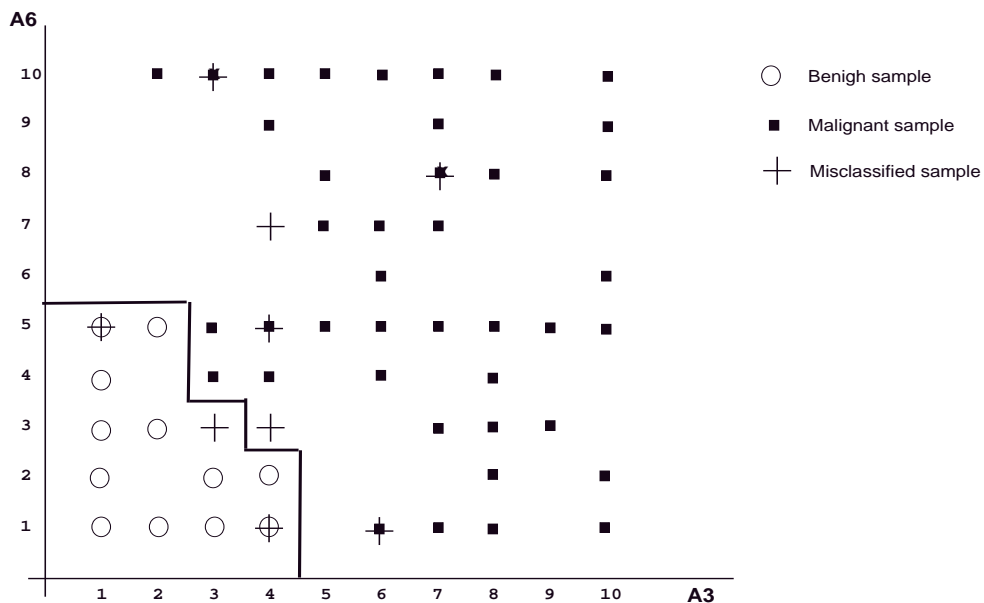


Figure 5: A piecewise-linear separator for the Wisconsin breast cancer data involving only 2 of the original 9 attributes. Ten out of 350 training patterns are incorrectly classified giving an accuracy rate of 97.14%.

fact, it is possible to generate a piecewise-linear separator involving only 2 attributes and achieve higher accuracy rate on the testing data. The separator generated from the pruned network is depicted in Fig. 5.

### B. Detailed analysis 2: Cleveland Heart Disease Dataset.

The dataset consists of 303 patterns. We discarded patterns with missing attribute values and used only the remaining 297 patterns. The patterns were divided randomly into training and testing set. Two third of the patterns formed the training set and the remaining one third the testing set. The training set consisted of 107 negative patterns and 91 positive patterns, while the testing set consisted of 53 negative patterns and 46 positive patterns. Positive patterns were given target values equal to 0, while negative patterns 1.

Each pattern is described by 13 attributes, 6 of which are continuous and the remaining 7 discrete. The attributes are summarized in Table 10. As can be seen from the table, one network input unit is assigned to each continuous attribute. Values of the continuous

attributes are normalized such that their range is  $[0, 1]$ . Each value of a discrete attribute with  $n$  possible values is represented by an  $n$ -bit binary coding, except when  $n = 2$  where only 1 bit is used. With an addition of 1 input unit to represent hidden unit bias values, the number of input units was 26. A network with 4 hidden units was trained and pruned with a minimum accuracy on the training set of 85 %. This figure for minimum accuracy was based on reported experimental results on this dataset [1]. The pruned network had only 2 hidden units and 13 connections left. Seven inputs  $\mathcal{I}_1, \mathcal{I}_5, \mathcal{I}_7, \mathcal{I}_9, \mathcal{I}_{13}, \mathcal{I}_{14}$  and  $\mathcal{I}_{16}$  were connected to the first hidden unit. Inputs  $\mathcal{I}_3, \mathcal{I}_{15}, \mathcal{I}_{18}$  and  $\mathcal{I}_{22}$  were connected to the second hidden unit. The accuracy rates of this network are 88.38% on the training data and 84.85% on the testing data.

Table 10

The attributes of the Cleveland heart disease dataset and the network input units assigned to them.

Attribute	Type	Possible values/Range	Network inputs
$\mathcal{A}_1$	Continuous	[29, 77]	$\mathcal{I}_1$
$\mathcal{A}_2$	Discrete	0 or 1	$\mathcal{I}_2$
$\mathcal{A}_3$	Discrete	1,2,3 or 4	$\mathcal{I}_3, \mathcal{I}_4, \mathcal{I}_5, \mathcal{I}_6$
$\mathcal{A}_4$	Continuous	[94, 200]	$\mathcal{I}_7$
$\mathcal{A}_5$	Continuous	[126, 564]	$\mathcal{I}_8$
$\mathcal{A}_6$	Discrete	0 or 1	$\mathcal{I}_9$
$\mathcal{A}_7$	Discrete	0,1 or 2	$\mathcal{I}_{10}, \mathcal{I}_{11}, \mathcal{I}_{12}$
$\mathcal{A}_8$	Continuous	[71, 202]	$\mathcal{I}_{13}$
$\mathcal{A}_9$	Discrete	0 or 1	$\mathcal{I}_{14}$
$\mathcal{A}_{10}$	Continuous	[0, 6.2]	$\mathcal{I}_{15}$
$\mathcal{A}_{11}$	Discrete	1,2 or 3	$\mathcal{I}_{16}, \mathcal{I}_{17}, \mathcal{I}_{18}$
$\mathcal{A}_{12}$	Discrete	0,1,2 or 3	$\mathcal{I}_{19}, \mathcal{I}_{20}, \mathcal{I}_{21}, \mathcal{I}_{22}$
$\mathcal{A}_{13}$	Discrete	3,6 or 7	$\mathcal{I}_{23}, \mathcal{I}_{24}, \mathcal{I}_{25}$

The Chi2 algorithm was then applied to discretize the activation values of 175 training

Table 11

All possible combinations of the discretized activation values at the two remaining hidden units in the pruned network trained for the Cleveland heart disease dataset.

Hidden unit 1	Hidden unit 2	Class label	Frequency
$\alpha_1 = 1$	$\alpha_2 = 1$	0	46
$\alpha_1 = 1$	$\alpha_2 = 2$	1	22
$\alpha_1 = 1$	$\alpha_2 = 3$	1	47
$\alpha_1 = 2$	$\alpha_2 = 1$	0	9
$\alpha_1 = 2$	$\alpha_2 = 2$	0	3
$\alpha_1 = 2$	$\alpha_2 = 3$	1	5
$\alpha_1 = 3$	$\alpha_2 = 1$	0	31
$\alpha_1 = 3$	$\alpha_2 = 2$	0	11
$\alpha_1 = 3$	$\alpha_2 = 3$	1	1

patterns that had been correctly classified by the pruned network. The results are as follows:

1. Hidden unit 1. There were 3 subintervals,  $[-1, -0.58)$ ,  $[-0.58, 0.86)$  and  $[0.86, 1]$ .
2. Hidden unit 2. There were also 3 subintervals,  $[-1, -0.54)$ ,  $[-0.54, 0.24)$  and  $[0.24, 1]$ .

Of the 9 possible combinations of the discretized activation values, 4 have target equal to 1, while the remaining 5 have target value equal to 0. They are summarized in Table 11. A set of simple rules can distinguish the two sets of patterns in terms of the clustered activation values:

If  $\alpha_1 = 1$  and  $\alpha_2 = 2$ , then target = 1,

else if  $\alpha_2 = 3$ , then target = 1.

Default rule: target = 0.

(where  $\alpha_i = j$  if the hidden unit  $i$  activation value of a pattern falls in the  $j$ -th subinterval,  $i = 1, 2, j = 1, 2, 3$ .)

With the weights of network connections and the subinterval boundaries found by Chi2, it is easy to find the regions defined by the network inputs where the discretized activation

values are unique. For hidden unit 1, it is sufficient to find the two halfspaces where  $\alpha_1 = 1$  and  $\alpha_1 > 1$ . We have that

$$\begin{aligned} &\text{If } -3.68\mathcal{I}_2 - 6.70\mathcal{I}_5 - 0.07\mathcal{I}_7 + 2.39\mathcal{I}_9 + 3.66\mathcal{I}_{13} \\ &\quad - 5.53\mathcal{I}_{14} + 2.24\mathcal{I}_{16} < \tanh^{-1}(-0.58), \text{ then } \alpha_1 = 1, \\ &\text{else } \alpha_1 > 1. \end{aligned}$$

Similarly, for hidden unit 2, from the network connections we find that

$$\begin{aligned} &\text{If } 1.09\mathcal{I}_3 + 1.99\mathcal{I}_{15} - 1.20\mathcal{I}_{18} - 1.70\mathcal{I}_{22} < \tanh^{-1}(-0.54), \text{ then } \alpha_2 = 1, \\ &\text{else if } 1.09\mathcal{I}_3 + 1.99\mathcal{I}_{15} - 1.20\mathcal{I}_{18} - 1.70\mathcal{I}_{22} < \tanh^{-1}(0.24), \text{ then } \alpha_2 = 2, \\ &\text{else } \alpha_2 = 3. \end{aligned}$$

Of the 26 network inputs, only 11 remained after pruning. Three are continuous and eight are binary. Inputs  $\mathcal{I}_3$  and  $\mathcal{I}_5$  are part of the binary representation of attribute  $\mathcal{A}_3$ . They can take one of the 3 possible combinations of (0, 0), (0, 1) or (1, 0). Inputs  $\mathcal{I}_{16}$  and  $\mathcal{I}_{18}$  as part of the binary representation of attribute  $\mathcal{A}_{11}$  can also have one of these three combinations. Each of the other 4 binary inputs  $\mathcal{I}_2, \mathcal{I}_9, \mathcal{I}_{14}$  and  $\mathcal{I}_{22}$  can take a value of either 0 or 1. Hence, assigning a value of 0 or 1 to each of the binary-valued inputs will result in 144 sets of rules involving only the continuous attributes. For example, one of these sets of rules in terms of the original attributes of the data are obtained by setting all the binary inputs to zero and substituting  $\tanh^{-1}(-0.58) = -0.66$ ,  $\tanh^{-1}(-0.54) = -0.60$ ,  $\tanh^{-1}(0.24) = 0.25$ :

$$\begin{aligned} &\text{If } -0.07\mathcal{A}_4 + 3.66\mathcal{A}_8 < -0.66 \text{ and } 1.99\mathcal{A}_{10} \geq -0.60, \text{ then target} = 1, \\ &\text{else if } 1.99\mathcal{A}_{10} > 0.25, \text{ then target} = 1. \end{aligned}$$

Default rule: target = 0.

The above set of rules is applicable to all patterns with  $\mathcal{A}_2 = 0, \mathcal{A}_3 = 2$  or  $= 4, \mathcal{A}_6 = 0, \mathcal{A}_9 = 0, \mathcal{A}_{11} = 2$ , and  $\mathcal{A}_{12} \neq 3$ . Rules for patterns with one or more nonzero discrete inputs are the same as the above except for the difference in the right-hand side of the inequalities in the rule conditions. Hence sets of rules applicable to different combinations

of discrete attribute values are obtained from a single network. Each of the rule set is different from the 'base case' where all the discrete values are zero only in the the right-hand side of the inequalities involving the continuous attributes. This attractive feature of the rule sets generated by our algorithm is generally not obtainable from decision tree classification methods such as C4.5rules.

The coefficients of the continuous attributes in the rules provide valuable informations. An answer to a query such as "what is the maximum decrease/increase that can be made to attribute  $\mathcal{A}_s$  of a pattern, such that the classification of the pattern remains the same?" is easy to obtain from the rules generated from the network. This kind of sensitivity analysis is not possible to be performed on decision trees.

The results of C4.5rules on this dataset are as follows. A set of 11 rules were generated. The number of conditions in a rule ranges from 1 to 5. The accuracy of these rules on the training set is 92.9 %, while the accuracy on the testing set is 74.7 %. The accuracy rates of our rules extracted from the pruned neural network are the same as those of the network, that is, 88.38 % on the training set and 84.85 % on the testing set.

## 5 Discussion

The rule extraction algorithm presented in this paper offers an alternative to traditional rule generation methods, such as decision tree induction, linear discriminant analysis and linear programming methods. Our system, NeuroLinear extracts oblique decision rules from a trained network. The experimental results on real world datasets show that NeuroLinear generates compact sets of rules with accuracy rates that are as good as those of C4.5rules. For a domain where a few oblique hyperplanes form the decision boundaries, it can be expected that NeuroLinear will perform well, while C4.5 will generate many axis-parallel planes.

Compared to other methods for rule extraction/generation from neural networks, NeuroLinear offers several advantages. It makes use of a standard three layer network, where no assumption is imposed on the magnitude of the connections' weights. There is also no

assumption made on the activation values of the hidden units or the hidden units' bias. NeuroLinear works well for problem domains with mixed discrete/continuous attributes without requiring discretization of the continuous data.

In comparing NeuroLinear to traditional methods, besides predictive accuracy, additional factors should be examined. These factors include computational efficiency, comprehensibility of the extracted rules, and consistency of the rules. We elaborate on these points by highlighting the differences between NeuroLinear and other methods for rule generation.

*Computational efficiency* There is no doubt that NeuroLinear, having a backpropagation network as its basic component, is computationally expensive compared to other approaches such as linear discriminant analysis, decision tree methods, or linear programming algorithms. To reduce the computation cost, we implemented a quasi-Newton method to minimize the error function of the network. Speedup of up to 2 order of magnitude over the traditional backpropagation method can be achieved by the quasi-Newton method.

*Rule comprehensibility* The rules generated by NeuroLinear are similar to those of linear discriminant or linear programming methods in the sense that they involve a set of coefficients that define the boundary of the decision regions. NeuroLinear rules which approximate the complex decision boundary of a neural network are more general than linear discriminant functions. Although simple linear discriminant functions may be easier to comprehend, their accuracy may not be satisfactory. Linear programming methods can generate complex piecewise linear decision boundary. NeuroLinear rules have an advantage over linear programming rules since the former involve only those attributes that are useful while the latter normally involve all input attributes. Network pruning removes all the irrelevant and redundant network connections, and in the process it also removes those attributes that are not useful. By removing unnecessary attributes, overfitting can be reduced and the generalization of the rules can be improved. Rules that involve only the relevant attributes are more compact, they are easier to analyze and understand.

Rules generated by a decision tree method such as C4.5rules do not involve any weight coefficient. However, it is not correct to conclude that such rules are always more compre-

hensible than rules with weights. When the decision boundaries are oblique such as those in Example 1, simple rules with weights are certainly more meaningful than a large set of rules that divide the decision regions into many small rectangular regions. The weights of a rule may also tell us more about the data in hand, for example, which attribute is the most influential factor in determining a certain class label.

When the problem domain is described by both nominal and continuous attributes, NeuroLinear rules exhibit an interesting feature not possessed by rules from C4.5. We may refer to patterns having all their relevant discrete attribute values equal to zero as the base level. Hence, a set of base-level rules that involves only the continuous attributes can be obtained. Similar sets of rules for other patterns with one or more of nonzero discrete attributes are generated by NeuroLinear. These rules differ from those of the base-level rules only in the threshold values. The different thresholds for the various combinations of the discrete attribute values provide potentially valuable information regarding the patterns in the dataset.

*Rule consistency* There are two aspects of measuring the consistency of the rules generated from neural networks. First, the consistency between the generated rules and the network outputs. Second, the consistency among the rules generated from different networks trained on the same problem domain.

NeuroLinear rules preserve the training accuracy of the network from which they are generated. On the testing dataset, our experiments show that when the network has been trained using a relatively large number of samples, the deviations in the predictions of the network and of the extracted rules are small.

An often mentioned drawback of the neural network approach for pattern classification is its nondeterministic nature. When the network is initialized with different sets of random weights, it is very likely that the training process terminates at different local minima. Obviously, different sets of rules will be generated by NeuroLinear. However, instead of taking this as a drawback, it can be viewed as an advantage of NeuroLinear over the deterministic approach such as decision tree method or linear discriminant method. When the number

of samples available for training is small and the number of attributes is relatively large (as often the case in real applications), there are many equally good sets of rules that achieve similar accuracy rates. NeuroLinear allows us to obtain these different sets of rules. Some of the rules may reveal more useful information about the data or may give more insight into the problem than others.

## 6 CONCLUSION

We have presented NeuroLinear, a system for generating classification rules using neural networks. The three components of NeuroLinear make it possible to generate oblique decision rules, these are:

1. an efficient neural network training and pruning algorithm,
2. the Chi2 algorithm for discretization of hidden unit activation values, and
3. the X2R algorithm for generating perfect rules from a small dataset with discrete attributes values.

Comparisons of experimental results using real-world datasets demonstrate that NeuroLinear can achieve similar accuracy rates as C4.5rules with far fewer rules. Different application domains may require different types of rules in order to ease the task of understanding these domains. Due to its salient features, NeuroLinear thus provides a viable alternative to methods such as decision tree induction, linear discriminant and linear programming methods. The application of neural networks can be significantly expanded by NeuroLinear.

### Acknowledgements

We would like to thank two anonymous reviewers for their valuable comments and suggestions.

## References

- [1] K.P. Bennett and O.L. Mangasarian, Neural network training via linear programming, in: P.M. Pardalos, ed., *Advances in Optimization and Parallel Computing*, (Elsevier Science Publishers B.V., Amsterdam, 1992) 56–67.
- [2] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees* (Wadsworth & Brooks/Cole Advanced Books & Software, Belmont, CA, 1984).
- [3] T.G. Dietterich, H. Hild and G. Bakiri, A comparative study of ID3 and backpropagation for english text-to-speech mapping, in *Machine Learning: Proceedings of the Seventh International Conference*, Austin, Texas (1990)
- [4] D.H. Fisher and K.B. McKusick. “An empirical comparison of ID3 and back-propagation,” in *Proceedings of 11th Int. Joint Conf. on AI*, (1989) 788–793.
- [5] L. Fu, Rule learning by searching on adapted nets, in *Proc. of the Ninth National Conference on Artificial Intelligence*, (1991) 590–595.
- [6] S. Gallant, Connectionist expert systems, *Comm. of the ACM*, 31 (2) (1988) 152–169.
- [7] H. Guo and S.B. Gelfand, Classification trees with neural network feature extraction, *IEEE Trans. on Neural Networks*, 3 (6) (1992) 923–933.
- [8] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the theory of neural computation*, (Addison Wesley, Redwood City, CA, 1991).
- [9] E. D. Karnin, A simple procedure for pruning back-propagation trained neural networks, *IEEE Trans. on Neural Networks* 1 (2) (1990) 239–242.
- [10] R. Kerber, ChiMerge: Discretization of numeric attributes, in *The Proc. of the Ninth National Conference on AI*, (AAAI Press/The MIT Press, 1992) 123–128.
- [11] H. Liu and R. Setiono, Chi2: Feature selection and discretization of numeric attributes, in *Proceedings of the 7th IEEE International Conference on Tools with Artificial Intelligence* (1995) 388–391.
- [12] H. Liu and S.T. Tan, X2R: A fast rule generator, in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* (IEEE Press, 1995).

- [13] O.L. Mangasarian, R. Setiono and W.H. Wolberg, Pattern recognition via linear programming: theory and application to medical diagnosis, in: T.F. Coleman and Y. Li, eds., *Large-scale Numerical Optimization*, (SIAM, Philadelphia, PA, 1990) 22–31.
- [14] P.M. Murphy and D.W. Aha, *UCI repository of machine learning databases [machine-readable data repository]*, Department of Information and Computer Science, University of California, Irvine, 1992.
- [15] J.R. Quinlan. *C4.5: Programs for Machine Learning*, (Morgan Kaufmann, San Mateo, CA, 1993).
- [16] J.R. Quinlan. Comparing connectionist and symbolic learning methods, in: S.J. Hanson, G.A. Drastall, and R.L. Rivest, eds., *Computational Learning Theory and Natural Learning Systems* (1) (A Bradford Book, The MIT Press, 1994) 445–456.
- [17] K. Saito and R. Nakano, Medical diagnosis expert system based on PDP model, *Proc. IEEE Intl. Conf. on Neural Networks* (IEEE Press, New York) (1988) I255-I266.
- [18] R. Setiono and H. Liu, Symbolic representation of neural networks, *IEEE Computer* (March 1996) 71–77.
- [19] R. Setiono. A penalty function approach for pruning feedforward neural networks, *Neural Computation*, 9 (1) (1997) 185–204.
- [20] J.W. Shavlik, R.J. Mooney and G.G. Towell, Symbolic and neural learning algorithms: An experimental comparison, *Machine Learning*, 6 (2) (1991) 111–143.
- [21] G.G. Towell and J.W. Shavlik, Extracting refined rules from knowledge-based neural networks, *Machine Learning* 13 (1) (1993) 71–101.
- [22] A. van Ooyen and B. Nienhuis, Improving the convergence of the backpropagation algorithm, *Neural Networks*, 5 (1992) 465–471.
- [23] W.H. Wolberg, M.A. Tanner and W.Y. Loh, Diagnostic schemes for fine needle aspirates of breast masses, *Analytical and Quantitative Cytology and Histology*, 10 (1988) 225–228.