NeuroRule: A Connectionist Approach to Data Mining

Hongjun Lu Rudy Setiono Huan Liu
Department of Information Systems and Computer Science
National University of Singapore
{luhj,rudys,liuh}@iscs.nus.sg

Abstract

Classification, which involves finding rules that partition a given data set into disjoint groups, is one class of data mining problems. Approaches proposed so far for mining classification rules for large databases are mainly decision tree based symbolic learning methods. The connectionist approach based on neural networks has been thought not well suited for data mining. One of the major reasons cited is that knowledge generated by neural networks is not explicitly represented in the form of rules suitable for verification or interpretation by humans. This paper examines this issue. With our newly developed algorithms, rules which are similar to, or more concise than those generated by the symbolic methods can be extracted from the neural networks. The data mining process using neural networks with the emphasis on rule extraction is described. Experimental results and comparison with previously published works are presented.

1 Introduction

With the wide use of advanced database technology developed during past decades, it is not difficult to efficiently store huge volume of data in computers and retrieve them whenever needed. Although the stored data are a valuable asset of an organization, most organizations may face the problem of *data rich but knowledge poor* sooner or later. This situation aroused the recent surge of research interests in the area of data mining [1, 9, 2].

One of the data mining problems is *classification*. Data items in databases, such as tuples in relational database systems usually represent real world entities. The values of the attributes of a tuple represent the properties of the entity. Classification is the process of finding the common properties among different entities and classifying them into *classes*. The results are often expressed in the form of rules – the *classification rules*. By applying the rules, entities represented by tuples can be easily classified into different classes they belong to. We can restate the problem formally defined by Agrawal *et al.* [1] as follows. Let A be a set of attributes $A_1, A_2, \ldots, A_n$ and $dom(A_i)$ refer to the set of possible values for attribute A_i. Let C be a set of classes $c_1, c_2, \ldots, c_m$. We are given a data set, *the training set* whose members are $(n+1)$-tuples of the form $(a_1, a_2, \ldots, a_n, c_k)$ where $a_i \in dom(A_i), (1 \leq i \leq n)$ and $c_k \in C (1 \leq k \leq m)$. Hence, the class to which each tuple in the training set belongs is known for supervised learning. We are also given a second large database of $(n+1)$-tuples, the testing set. The classification problem is to obtain a set of rules R using the given training data set. By applying these rules to the testing set, the rules can be checked whether they generalize well (measured by the predictive accuracy). The rules that generalize well can be safely applied to the application database with unknown classes to determine each tuple's class.

This problem has been widely studied by researchers in the AI field [28]. It is recently reexamined by database researchers in the context of large database systems [5, 7, 14, 15, 13]. Two basic approaches to the classification problems studied by AI researchers are the symbolic approach and the connectionist approach. The symbolic approach is based on decision trees and the connectionist approach mainly

Proceedings of the 21st VLDB Conference
Zurich, Swizerland, 1995

uses neural networks. In general, neural networks give a lower classification error rate than the decision trees but require longer learning time [17, 24, 18]. While both approaches have been well received by the AI community, the general impression among the database community is that the connectionist approach is not well suited for data mining. The major criticisms include the following:

1. Neural networks learn the classification rules by multiple passes over the training data set so that the learning time, or the training time needed for a neural network to obtain high classification accuracy is usually long.

2. A neural network is usually a layered graph with the output of one node feeding into one or many other nodes in the next layer. The classification rules are buried in both the structure of the graph and the weights assigned to the links between the nodes. Articulating the classification rules becomes a difficult problem.

3. For the same reason, available domain knowledge is rather difficult to be incorporated to a neural network.

Among the above three major disadvantages of the connectionist approach, the articulating problem is the most urgent one to be solved for applying the technique to data mining. Without explicit representation of classification rules, it is very difficult to verify or interpret them. More importantly, with explicit rules, tuples of a certain pattern can be easily retrieved using a database query language. Access methods such as indexing can be used or built for efficient retrieval as those rules usually involve only a small set of attributes. This is especially important for applications involving a large volume of data.

In this paper, we present the results of our study on applying the neural networks to mine classification rules for large databases with the focus on articulating the classification rules represented by neural networks. The contributions of our study include the following:

- Different from previous research work that excludes the connectionist approach entirely, we argue that the connectionist approach should have its position in data mining because of its merits such as low classification error rates and robustness to noise [17, 18].

- With our newly developed algorithms, explicit classification rules can be extracted from a neural network. The rules extracted usually have a lower classification error rate than those generated by the decision tree based methods. For a data set with a strong relationship among attributes, the rules extracted are generally more concise.

- A data mining system, NeuroRule, based on neural networks was developed. The system successfully solved a number of classification problems in the literature.

To better suit large database applications, we also developed algorithms for input data pre-processing and for fast neural network training to reduce the time needed to learn the classification rules [22, 19]. Limited by space, those algorithms are not presented in this paper.

The remainder of the paper is organized as follows. Section 2 gives a discussion on using the connectionist approach to learn classification rules. Section 3 describes our algorithms to extract classification rules from a neural network. Section 4 presents some experimental results obtained and a comparison with previously published results. Finally a conclusion is given in Section 5.

2 Mining classification rules using neural networks

Artificial neural networks are densely interconnected networks of simple computational elements, *neurons*. There exist many different network topologies [10]. Among them, the *multi-layer perceptron* is especially useful for implementing a classification function. Figure 1 shows a three layer feedforward network. It consists of an input layer, a hidden layer and an output layer. A node (neuron) in the network has a number of inputs and a single output. For example, a neuron H_j in the hidden layer has $x_1^i, x_2^i, \ldots, x_n^i$ as its input and α^j as its output. The input links of H_j has weights $w_1^j, w_2^j, \ldots, w_n^j$. A node computes its output, the *activation value* by summing up its weighted inputs, subtracting a threshold, and passing the result to a non-linear function f, the *activation function*. Outputs from neurons in one layer are fed as inputs to neurons in the next layer. In this manner, when an input tuple is applied to the input layer, an output tuple is obtained at the output layer. For a well trained network which represents the classification function, if tuple $(x_1, x_2, \ldots, x_n)$ is applied to the input layer of the network, the output tuple, $(c_1, c_2, \ldots, c_m)$ should be obtained where c_i has value 1 if the input tuple belongs to class c_i and 0 otherwise.

Our approach that uses neural networks to mine classification rules consists of three steps:

1. *Network training*
   A three layer neural network is trained in this step. The training phase aims to find the best set

of weights for the network which allow the network to classify input tuples with a satisfactory level of accuracy. An initial set of weights are chosen randomly in the interval [-1,1]. Updating these weights is normally done by using informations involving the gradient of an error function. This phase is terminated when the norm of the gradient of the error function falls below a prespecified value.

2. *Network pruning*
The network obtained from the training phase is fully connected and could have too many links and sometimes too many nodes as well. It is impossible to extract concise rules which are meaningful to users and can be used to form database queries from such a network. The pruning phase aims at removing redundant links and nodes without increasing the classification error rate of the network. A smaller number of nodes and links left in the network after pruning provide for extracting consise and comprehensible rules that describe the classification function.

3. *Rule extraction*
This phase extracts the classification rules from the pruned network. The rules generated are in the form of "if $(a_1 \theta v_1)$ *and* $(x_2 \theta v_2)$ *and* ... *and* $(x_n \theta v_n)$ then C_j" where a_i's are the attributes of an input tuple, v_i's are constants, θ's are relational operators $(=, \leq, \geq, <>)$, and C_j is one of the class labels. It is expected that the rules are concise enough for human verification and are easily applicable to large databases.

In this section, we will briefly discuss the first two phase. The third phase, rule extraction phase will be discussed in the next section.

2.1 Network training

Assume that input tuples in an n-dimensional space are to be classified into three disjoint classes $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$. We construct a network as shown in Figure 1 which consists of three layers. The number of nodes in the input layer corresponds to the dimensionality of the input tuples. The number of nodes in the output layer equals to the number of classes to be classified, which is three in this example. The network is trained with target values equal to $\{1,0,0\}$ for all patterns in set $\mathcal{A}$, $\{0,1,0\}$ for all patterns in $\mathcal{B}$, and $\{0,0,1\}$ for all tuples in $\mathcal{C}$. An input tuples will be classified as a member of the class $\mathcal{A}, \mathcal{B}$ or $\mathcal{C}$ if the largest activation value is obtained by the first, second or third output node, respectively.

There is still no clear cut rule to determine the number of hidden nodes to be included in the network. Too many hidden nodes may lead to overfitting of the data and poor generalization, while too few hidden nodes may not give rise to a network that learns the data. Two different approaches have been proposed to overcome the problem of determining the optimal number of hidden nodes required by a neural network to solve a given problem. The first approach begins with a minimal network and adds more hidden nodes only when they are needed to improve the learning capability of the network [3, 11, 19]. The second approach begins with an oversized network and then prunes redundant hidden nodes and connections between the layers of the network. We adopt the second approach since we are interested in finding a network with a small number of hidden nodes as well as the fewest number of input nodes. An input node with no connection to any of the hidden nodes after pruning plays no role in the outcome of classification process and hence can be removed from the network.

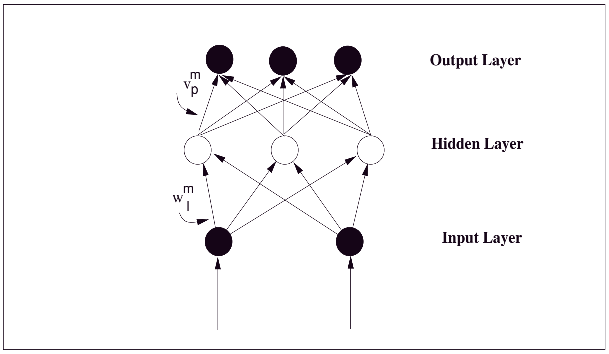

Figure 1: A three layer feedforward neural network.

The activation value of a node in the hidden layer is computed by passing the weighted sum of input values to a non-linear activation function. Let w_ℓ^m be the weights for the connections from input node ℓ to hidden node m. Given an input pattern $x^i, i \in \{1, 2, \ldots, k\}$, where k is the number of tuples in the data set, the activation value of the m-th hidden node is

$$\alpha^m = f\left(\sum_{\ell=1}^{n} \left(x_\ell^i w_\ell^m\right) - \tau^m\right),$$

where $f(.)$ is an activation function. In our study, we use the hyperbolic tangent function

$$f(x) := \delta(x) = (e^x - e^{-x})/(e^x + e^{-x})$$

as the activation function for the hidden nodes, which makes the range of activation values of the hidden nodes [-1, 1].

Once the activation values of all the hidden nodes have been computed, the p-th output of the network for input tuple x^i is computed as

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

where v_p^m is the weight of the connection between hidden node m and output node p and h is the number of hidden nodes in the network. The activation function used here is the sigmoid function,

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

which yields activation values of the output nodes in the range $[0, 1]$.

A tuple will be correctly classified if the following condition is satisfied

$$\max_p |e_p^i| = \max_p |S_p^i - t_p^i| \leq \eta_1, \tag{1}$$

where $t_p^i = 0$, except for $t_1^i = 1$ if $x^i \in \mathcal{A}$, $t_2^i = 1$ if $x^i \in \mathcal{B}$, and $t_3^i = 1$ if $x^i \in \mathcal{C}$, and η_1 is a small positive number less than 0.5. The ultimate objective of the training phase is to obtain a set of weights that make the network classify the input tuples correctly. To measure the classification error, an error function is needed so that the training process becomes a process to adjust the weights (w, v) to minimize this function. Furthermore, to facilitate the pruning phase, it is desired to have many weights with very small values so that they can be set to zero. This is achieved by adding a penalty term to the error function.

In our training algorithm, the cross entropy function

$$E(w,v) = -\sum_{i=1}^{k}\sum_{p=1}^{o}\left(t_p^i \log S_p^i + (1-t_p^i)\log(1-S_p^i)\right) \tag{2}$$

is used as the error function. In this example, o equals to 3 since we have 3 different classes. The cross entropy function is chosen because faster convergence can be achieved by minimizing this function instead of the widely used sum of squared error function [26].

The penalty term $P(w, v)$ we used is

$$\epsilon_1\left(\sum_{\mathrm{m}=1}^{\mathrm{h}}\sum_{\ell=1}^{\mathrm{n}}\frac{\beta(\mathrm{w}_\ell^\mathrm{m})^2}{1+\beta(\mathrm{w}_\ell^\mathrm{m})^2} + \sum_{\mathrm{m}=1}^{\mathrm{h}}\sum_{\mathrm{p}=1}^{\mathrm{o}}\frac{\beta(\mathrm{v}_\mathrm{p}^\mathrm{m})^2}{1+\beta(\mathrm{v}_\mathrm{p}^\mathrm{m})^2}\right) + \tag{3}$$
$$\epsilon_2\left(\sum_{\mathrm{m}=1}^{\mathrm{h}}\sum_{\ell=1}^{\mathrm{n}}(\mathrm{w}_\ell^\mathrm{m})^2 + \sum_{\mathrm{m}=1}^{\mathrm{h}}\sum_{\mathrm{p}=1}^{\mathrm{o}}\left(\mathrm{v}_\mathrm{p}^\mathrm{m}\right)^2\right),$$

where ϵ_1 and ϵ_2 are two positive weight decay parameters. Their values reflect the relative importance of the accuracy of the network versus its complexity. With larger values of these two parameters more weights may be removed later from the network at the cost of a decrease in its accuracy.

The training phase starts with an initial set of weights $(w, v)^{(0)}$ and iteratively updates the weights to minimize $E(w, v) + P(w, v)$. Any unconstrained minimization algorithm can be used for this purpose. In particular, the gradient descent method has been the most widely used in the training algorithm known as the backpropagation algorithm. A number of alternative algorithms for neural network training have been proposed [4]. To reduce the network training time, which is very important in the data mining as the data set is usually large, we employed a variant of the quasi-Newton algorithm [27], the BFGS method. This algorithm has a superlinear convergence rate, as opposed to the linear rate of the gradient descent method. Details of the BFGS algorithm can be found in [6, 23].

The network training is terminated when a local minimum of the function $E(w, v) + P(w, v)$ has been reached, that is when the gradient of the function is sufficiently small.

2.2 Network pruning

A fully connected network is obtained at the end of the training process. There are usually a large number of links in the network. With n input nodes, h hidden nodes, and m output nodes, there are $h(m + n)$ links. It is very difficult to articulate such a network. The network pruning phase aims at removing some of the links without affecting the classification accuracy of the network.

It can be shown that [20] if a network is fully trained to correctly classify an input tuple, x^i, with the condition (1) satisfied we can set w_ℓ^m to zero without deteriorating the overall accuracy of the network if the product $|v^m w_\ell^m|$ is sufficiently small. If $\max_p |v_p^m w_\ell^m| \leq 4\eta_2$ and the sum $(\eta_1 + \eta_2)$ is less than 0.5, then the network can still classify x^i correctly. Similarly, if $\max_p |v_p^m| \leq 4\eta_2$, then v_p^m can be removed from the network.

Our pruning algorithm based on this result is shown in Figure 2. The two conditions (4) and (5) for pruning depend on the magnitude of the weights for connections between input nodes and hidden nodes and between hidden nodes and output nodes. It is imperative that during training these weights be prevented from getting too large. At the same time, small weights should be encouraged to decay rapidly to zero. By using penalty function (3), we can achieve both.

2.3 An example

We have chosen to use a function described in [2] as an example to show how a neural network can be trained and pruned for solving a classification problem. The

Table 1: Attributes of the test data adapted from Agrawal et al.[2]

Attribute	Description	Value
salary	salary	uniformly distributed from 20,000 to 150,000
commission	commission	if salary $\geq 75000 \rightarrow$ commission $= 0$
		else uniformly distributed from 10000 to 75000.
age	age	uniformly distributed from 20 to 80.
elevel	education level	uniformly distributed from $[0, 1, \ldots, 4]$.
car	make of the car	uniformly distributed from $[1, 2, \ldots 20]$.
zipcode	zip code of the town	uniformly chosen from 9 available zipcodes.
hvalue	value of the house	uniformly distributed from $0.5k10000$ to $1.5k1000000$
		where $k \in \{0 \ldots 9\}$ depends on zipcode.
hyears	years house owned	uniformly distributed from $[1, 2, \ldots, 30]$.
loan	total amount of loan	uniformly distributed from 1 to 500000.

Neural network pruning algorithm (NP)

1. Let η_1 and η_2 be positive scalars such that $\eta_1 + \eta_2 < 0.5$.

2. Pick a fully connected network. Train this network until a predetermined accuracy rate is achieved and for each correctly classified pattern the condition (1) is satisfied. Let (w, v) be the weights of this network.

3. For each w_ℓ^m, if

$$\max_p |v_p^m \times w_\ell^m| \leq 4\eta_2, \qquad (4)$$

then remove w_ℓ^m from the network

4. For each v_p^m, if

$$|v_p^m| \leq 4\eta_2, \qquad (5)$$

then remove v_p^m from the network

5. If no weight satisfies condition (4) or condition (5), then remove w_ℓ^m with the smallest product $\max_p |v_p^m \times w_\ell^m|$.

6. Retrain the network. If accuracy of the network falls below an acceptable level, then stop. Otherwise, go to Step 3.

Figure 2: Neural network pruning algorithm

input tuple consists of nine attributes defined in Table 1. Ten classification problems are given in [2]. Limited by space, we will present and discuss a few functions and the experimental results.

Function 2 classifies a tuple in Group A if

$$((\textbf{age} < 40) \wedge (50000 \leq \textbf{salary} \leq 100000)) \vee$$

$$((40 \leq \textbf{age} < 60) \wedge (75000 \leq \textbf{salary} \leq 125000)) \vee$$

$$((\textbf{age} \geq 60) \wedge (25000 \leq \textbf{salary} \leq 75000)).$$

Otherwise, the tuple is classified in Group B.

The training data set consisted of 1000 tuples. The values of the attributes of each tuple were generated randomly according to the distributions given in Table 1. Following Agrawal *et al.* [2], we also included a perturbation factor as one of the parameters of the random data generator. This perturbation factor was set at 5 percent. For each tuple, a class label was determined according to the rules that define the function above.

To facilitate the rule extraction in the later phase, the values of the numeric attributes were discretized. Each of the six attributes with numeric values was discretized by dividing its range into subintervals. The attribute *salary* for example, which was uniformly distributed from 25000 to 150000 was divided into 6 subintervals: subinterval 1 contained all salary values that were strictly less than 25000, subinterval 2 contained those greater than or equal to 25000 and strictly less than 50000, etc. The *thermometer* coding scheme was then employed to get the binary representations of these intervals for inputs to the neural network. Hence, a salary value less that 25000 was coded as $\{000001\}$, a salary value in the interval $[25000, 50000)$ was coded as $\{000011\}$, etc. The second attribute *commission* was similarly coded. The interval from 10000 to 75000 was divided into 7 subintervals, each having a width of 10000 except for the last one, $[70000, 75000]$. Zero commission was coded by all zero values for the seven inputs. The coding scheme for the other attributes are given in Table 2.

Table 2: Binarization of the attribute values

Attribute	Input number	Interval width
salary	$\mathcal{I}_1$ - $\mathcal{I}_6$	25000
commission	$\mathcal{I}_7$ - $\mathcal{I}_{13}$	10000
age	$\mathcal{I}_{14}$ - $\mathcal{I}_{19}$	10
elevel	$\mathcal{I}_{20}$ - $\mathcal{I}_{23}$	-
car	$\mathcal{I}_{24}$ - $\mathcal{I}_{43}$	-
zipcode	$\mathcal{I}_{44}$ - $\mathcal{I}_{52}$	-
hvalue	$\mathcal{I}_{53}$ - $\mathcal{I}_{66}$	100000
hyears	$\mathcal{I}_{67}$ - $\mathcal{I}_{76}$	3
loan	$\mathcal{I}_{77}$ - $\mathcal{I}_{86}$	50000

With this coding scheme, we had a total of 86 binary inputs. The 87th input was added to the network to incorporate the bias or threshold in each of the hidden node. The input value to this input was set to one. Therefore the input layer of the initial network consisted of 87 input nodes. Two nodes were used at the output layer. The target output of the network was $\{1, 0\}$ if the tuple belonged to Group A, and $\{0, 1\}$ otherwise. The number of the hidden nodes was initially set as four.

There were a total of 386 links in the network. The weights for these links were given initial values that were randomly generated in the interval [-1,1]. The network was trained until a local minimum point of the error function had been reached.

The fully connected trained network was then pruned by the pruning algorithm described in Section 2.2. We continued removing connections from the neural network as long as the accuracy of the network was still higher than 90 %.

Figure 3 shows the pruned network. Of the 386 links in the original network, only 17 remained in the pruned network. One of the four hidden nodes was removed. A small number of links from the input nodes to the hidden nodes made it possible to extract compact rules with the same accuracy level as the neural network.

3 Extracting rules from a neural network

Network pruning results in a relatively simple network. In the example shown in the last section, the pruned network has only 7 input nodes, 3 hidden nodes, and 2 output nodes. The number of links is 17. However, it is still very difficult to articulate the network, i.e., find the explicit relationship between the input tuples and the output tuples. Research work in this area has been reported [25, 8]. However, to our best knowledge, there is no method available in the literature that can extract explicit and concise rules as the algorithm we will describe in this section.

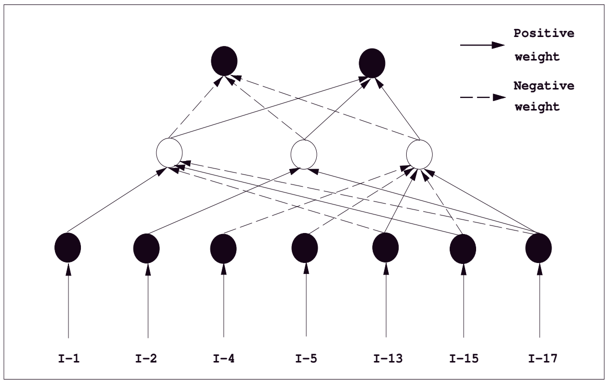

Figure 3: Pruned network for Function 2. Its accuracy rate on the 1000 training samples is 96.30 % and it contains only 17 connections.

3.1 Rule extracting algorithm

A number of reasons contribute to the difficulty of extracting rules from a pruned network. First, even with a pruned network, the links may be still too many to express the relationship between an input tuple and its class label in the form of *if ... then* $\cdots$ rules. If a node has n input links with binary values, there could be as many as 2^n distinct input patterns. The rules could be quite lengthy or complex even with a small n, say 7. Second, the activation values of a hidden node could be anywhere in the range [-1,1] depending on the input tuple. With a large number of testing data, the activation values are virtually continuous. It is rather difficult to derive the explicit relationship between the activation values of the hidden nodes and the output values of a node in the output layer.

Our rule extracting algorithm is outlined in Figure 4. The algorithm first discretizes the activation values of hidden nodes into a manageable number of discrete values without sacrificing the classification accuracy of the network. A small set of the discrete activation values make it possible to determine both the dependency among the output values and the hidden node values and the dependency among the hidden node activation values and the input values.

From the dependencies, rules can be generated [12]. Here we show the process of extracting rules from the pruned network in Figure 3 obtained for the classification problem Function 2.

The network has three hidden nodes. The activation values of 1000 tuples were discretized. The value of ϵ was set to 0.6. The results of discretization are

Rule extraction algorithm (RX)

1. Activation value discretization via clustering:

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

   (b) For all patterns $i = 2, 3, \ldots k$ in the training set:

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

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

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

   (c) Replace H by the average of all activation values that have been clustered into this cluster:

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

   (d) Check the accuracy of the network with the activation values δ^i at the hidden nodes replaced by δ_d, the activation value of the cluster to which the activation value belongs.

   (e) If the accuracy falls below the required level, decrease ϵ and repeat Step 1.

2. Enumerate the discretized activation values and compute the network output.

   Generate perfect rules that have a perfect cover of all the tuples from the hidden node activation values to the output values.

3. For the discretized hidden node activation values appeared in the rules found in the above step, enumerate the input values that lead to them, and generate perfect rules.

4. Generate rules that relate the input values and the output values by rule substitution based on the results of the above two steps.

Figure 4: Rule extraction algorithm (RX)

shown in the following table.

Node	No of clusters	Cluster activation values
1	3	(-1, 0, 1)
2	2	(0, 1)
3	3	(-1, 0.24, 1)

The classification accuracy of the network was checked by replacing the individual activation value with its discretized activation value. The value of $\epsilon = 0.6$ was sufficiently small to preserve the accuracy of the neural network and large enough to produce only a small number of clusters. For the three hidden nodes, the numbers of discrete activation values (clusters) are 3,2 and 3, or a total of 18 different outcomes at the two output nodes are possible. We tabulate the outputs $C_j (1 \leq j \leq 2)$ of the network according to the hidden node activation values $\alpha_m, (1 \leq m \leq 3)$ as follows.

α_1	α_2	α_3	C_1	C_2
-1	1	-1	0.92	0.08
-1	1	1	0.00	1.00
-1	1	0.24	0.01	0.99
-1	0	-1	1.00	0.00
-1	0	1	0.11	0.89
-1	0	0.24	0.93	0.07
1	1	-1	0.00	1.00
1	1	1	0.00	1.00
1	1	0.24	0.00	1.00
1	0	-1	0.89	0.11
1	0	1	0.00	1.00
1	0	0.24	0.00	1.00
0	1	-1	0.18	0.82
0	1	1	0.00	1.00
0	1	0.24	0.00	1.00
0	0	-1	1.00	0.00
0	0	1	0.00	1.00
0	0	0.24	0.18	0.82

Following Algorithm RX step 2, the predicted outputs of the network are taken to be $C_1 = 1$ and $C_2 = 0$ if the activation values α_m's satisfy one of the following conditions (since the table is small, the rules can be checked manually):

$$R_{11} : C_1 = 1, C_2 = 0 \Leftarrow \alpha_2 = 0, \alpha_3 = -1.$$
$$R_{12} : C_1 = 1, C_2 = 0 \Leftarrow \alpha_1 = -1, \alpha_2 = 1, \alpha_3 = -1.$$
$$R_{13} : C_1 = 1, C_2 = 0 \Leftarrow \alpha_1 = -1, \alpha_2 = 0, \alpha_3 = 0.24.$$

Otherwise, $C_1 = 0$ and $C_2 = 1$.

The activation values of a hidden node are determined by the inputs connected to it. In particular, the three activation values of hidden node 1 are determined by 4 inputs, $\mathcal{I}_1, \mathcal{I}_{13}, \mathcal{I}_{15}$, and $\mathcal{I}_{17}$. The activation values of hidden node 2 are determined by 2 inputs $\mathcal{I}_2$ and $\mathcal{I}_{17}$, and the activation values of hidden node 3 are determined by $\mathcal{I}_4, \mathcal{I}_5, \mathcal{I}_{13}, \mathcal{I}_{15}$ and $\mathcal{I}_{17}$. Note that only 5 different activation values appear in the above three rules. Following Algorithm RX step 3, we obtain rules that show how a hidden node is activated for the five different activation values at the three hidden nodes:

Hidden node 1:

$$
\begin{array}{llll}
R_{21}: & \alpha_1 = -1 & \Leftarrow & \mathcal{I}_{13} = 1 \\
R_{22}: & \alpha_1 = -1 & \Leftarrow & \mathcal{I}_1 = \mathcal{I}_{13} = \mathcal{I}_{15} = 0, \\
 & & & \mathcal{I}_{17} = 1
\end{array}
$$

Hidden node 2:

$$
\begin{array}{llll}
R_{23}: & \alpha_2 = 1 & \Leftarrow & \mathcal{I}_2 = 1 \\
R_{24}: & \alpha_2 = 1 & \Leftarrow & \mathcal{I}_{17} = 1 \\
R_{25}: & \alpha_2 = 0 & \Leftarrow & \mathcal{I}_2 = \mathcal{I}_{17} = 0
\end{array}
$$

Hidden node 3:

$$
\begin{array}{llll}
R_{26}: & \alpha_3 = -1 & \Leftarrow & \mathcal{I}_{13} = 0 \\
R_{27}: & \alpha_3 = -1 & \Leftarrow & \mathcal{I}_5 = \mathcal{I}_{15} = 1 \\
R_{28}: & \alpha_3 = 0.24 & \Leftarrow & \mathcal{I}_4 = \mathcal{I}_{13} = 1,\ \mathcal{I}_{17} = 0 \\
R_{29}: & \alpha_3 = 0.24 & \Leftarrow & \mathcal{I}_5 = 0,\ \mathcal{I}_{13} = \mathcal{I}_{15} = 1
\end{array}
$$

With all the intermediate rules obtained above, we can derive the classification rules as in Algorithm RX step 4. For example, substituting rule R_{11} with rules R_{25}, R_{26}, and R_{27}, we have the following two rules in terms of the original inputs:

$$
\begin{array}{lll}
R1: C_1 = 1, C_2 = 0 & \Leftarrow & \mathcal{I}_2 = \mathcal{I}_{17} = 0, \mathcal{I}_{13} = 0 \\
R_1': C_1 = 1, C_2 = 0 & \Leftarrow & \mathcal{I}_2 = \mathcal{I}_{17} = 0, \mathcal{I}_5 = \mathcal{I}_{15} = 1
\end{array}
$$

Recall that the input values of $\mathcal{I}_{14}$ to $\mathcal{I}_{19}$ represent coded age groups where $\mathcal{I}_{15} = 1$ if *age* is in [60, 80) and $\mathcal{I}_{17} = 1$ if *age* is in [20, 40). Therefore rule R_1' in fact can never be satisfied by any tuple, hence redundant.

Similarly, replacing rule R_{12} with $R_{21}, R_{22}, R_{23}, R_{24}, R_{26}$ and R_{27}, we have the following two rules:

$$
\begin{array}{lll}
R2: C_1 = 1, C_2 = 0 & \Leftarrow & \mathcal{I}_5 = \mathcal{I}_{13} = \mathcal{I}_{15} = 1. \\
R3: C_1 = 1, C_2 = 0 & \Leftarrow & \mathcal{I}_1 = \mathcal{I}_{13} = \mathcal{I}_{15} = 0, \mathcal{I}_{17} = 1.
\end{array}
$$

Substituting R_{13} with R_{21}, R_{22}, R_{25}, R_{28} and R_{29}, we have another rule:

$$
R4: C_1 = 1, C_2 = 0 \quad \Leftarrow \quad \mathcal{I}_2 = \mathcal{I}_{17} = 0, \mathcal{I}_4 = \mathcal{I}_{13} = 1.
$$

It is now trivial to obtain the rules in terms of the original attributes. Conditions of the rules after substitution can be rewritten in terms of the original attributes and classification problem as shown in Figure 5.

Given the fact that *salary* $\geq$ 75000 $\Leftrightarrow$ *commission* $= 0$, the above four rules obtained by the pruned network are identical to the classification Function 2.

3.2 Hidden node splitting and creation of a subnetwork

After network pruning and activation value discretization, rules can be extracted by examining the possible combinations in the network outputs as shown in the previous section. However, when there are still too many connections between a hidden node and input nodes, it is not trivial to extract rules, even if we can, the rules may not be easy to understand. To address the problem, a three layer feedforward subnetwork can be employed to simplify rule extraction for the hidden node. The number of output nodes of this subnetwork is the number of discrete values of the hidden node, while the input nodes are those connected to the hidden node in the original network. Tuples in the training set are grouped according to their discretized activation values. Given d discrete activation values $D_1, D_2, \ldots, D_d$, all training tuples with activation values equal to D_j are given a d-dimensional target value of all zeros expect for one 1 in position j. A new hidden layer is introduced for this subnetwork. This subnetwork is trained and pruned in the same ways as is the original network. The rule extracting process is applied for the subnetwork to obtain the rules describing the input and the discretized activation values.

This process is applied recursively to those hidden nodes with too many input links until the number of connection is small enough or the new subnetwork cannot simplify the connections between the inputs and the hidden node at the higher level. For most problems that we have solved, this step is not necessary. One problem where this step is required by the algorithm is for a genetic classification problem with 60 attributes. The details of the experiment can be found in [21].

4 Preliminary experimental results

Unlike the pattern classification research in the AI community where a set of classic problems have been studied by a large number of researchers, fewer well documented benchmark problems are available for data mining. In this section, we report the experimental results of applying the approach described in the previous sections to the data mining problem defined in [2]. As mentioned earlier, the database tuples consisted of nine attributes (See Table 1). Ten classification functions of Agrawal et al. [2] were used to generate classification problems with different complexities. The training set consisted of 1000 tuples and the testing data sets had 1000 tuples. Efforts were made

Rule 1.	If (**salary** $<$ 100000) $\wedge$ (**commission** $=$ 0) $\wedge$ (**age** $\leq$ 40), then Group A.
Rule 2.	If (**salary** $\geq$ 25000) $\wedge$ (**commission** $>$ 0) $\wedge$ (**age** $\geq$ 60), then Group A.
Rule 3.	If (**salary** $<$ 125000) $\wedge$ (**commission** $=$ 0) $\wedge$ (40 $\leq$ **age** $\leq$ 60), then Group A.
Rule 4.	If (50000 $\leq$ **salary** $<$ 100000) $\wedge$ (**age** $<$ 40), then Group A.
Default	Rule. Group B.

Figure 5: Rules generated by NeuroRule for Function 2.

to generate the data sets as described in the original functions. Among 10 functions described, we found that functions 8 and 10 produced highly skewed data that made classification not meaningful. We will only discuss functions other than these two. To assess our approach, we compare the results with that of C4.5, a decision tree-based classifier [16].

4.1 Classification accuracy

The following table reports the classification accuracy using both our system and C4.5 for eight functions. Here, classification accuracy is defined as

$$accuracy = \frac{no\ tuples\ correctly\ classified}{total\ number\ of\ tuples} \tag{6}$$

Func.	Pruned Networks		C4.5	
no	Training	Testing	Training	Testing
1	98.1	100.0	98.3	100.0
2	96.3	100.0	98.7	96.0
3	98.5	100.0	99.5	99.1
4	90.6	92.9	94.0	89.7
5	90.4	93.1	96.8	94.4
6	90.1	90.9	94.0	91.7
7	91.9	91.4	98.1	93.6
9	90.1	90.9	94.4	91.8

From the table we can see that the classification accuracy of the neural network based approach and C4.5 is comparable. In fact, the network obtained after the training phase has higher accuracy than what listed here, which is mainly determined by the threshold set for the network pruning phase. In our experiments, it is set to 90%. That is, a network will be pruned until further pruning will cause the accuracy to fall below this threshold. For applications where high classification accuracy is desired, the threshold can be set higher so that less nodes and links will be pruned. Of course, this may lead to more complex classification rules. Tradeoff between the accuracy and the complexity of the classification rule set is one of the design issues.

4.2 Rules extracted

Here we present some of the classification rules extracted from our experiments.

For simple classification functions, the rules extracted are exactly the same as the classification functions. These include functions 1, 2 and 3. One interesting example is Function 2. The detailed process of finding the classification rules is described as an example in Section 2 and 3. The resulting rules are the same as the original functions. As reported by Agrawal *et al.* [2], ID3 generated a relatively large number of strings for Function 2 when the decision tree is built. We observed similar results when C4.5rules was used (a member of ID3). C4.5rules generated 18 rules. Among the 18 rules, 8 rules define the conditions for Group A. Another 10 rules define Group B. Tuples that do not satisfy the conditions specified are classified as default class, Group B. Figure 6 shows the rules that define tuples to be a member of Group A.

By comparing the rules generated by C4.5rules (Figure 6) with the rules generated by NeuroRule in Figure 4, it is obvious that our approach generates better rules in the sense that they are more compact, which makes the verification and application of the rules much easier.

Functions 4 and 5 are another two functions for which ID3 generates a large number of strings. $\mathcal{CDP}$ [2] also generates a relatively large number of strings than for other functions. The original classification function 4, the rule sets that define Group A tuples extracted using NeuroRule and C4.5, respectively are shown in Figure 7.

The five rules extracted by NeuroRule are not exactly the same as the original function descriptions (Function 4). To test the rules extracted, the rules were applied to three test data sets of different sizes, shown in Table 3. The column *Total* is the total number of tuples that are classified as group A by each rule. The column *Correct* is the percentage of correctly classified tuples. E.g., rule $R1$ classifies all tuples correctly. On the other hand, among 165 tuples that were classified as Group A by rule $R2$, 6.1% of them belong to Group B, i.e. they were misclassified.

Rule 16:	$(\textbf{salary} > 45910) \wedge (\textbf{commission} > 0) \wedge (\textbf{age} > 59)$
Rule 10:	$(51638 < \textbf{salary} \leq 98469) \wedge (\textbf{age}\ \text{age} \leq 39)$
Rule 13:	$(\textbf{salary} \leq 98469) \wedge (\textbf{commission} \leq 0) \wedge (\textbf{age} \leq 60)$
Rule 6:	$(26812 < \textbf{salary} \leq 45910) \wedge (\textbf{age} > 61)$
Rule 20:	$(98469 < \textbf{salary} \leq 121461) \wedge (39 < \textbf{age} \leq 57)$
Rule 7:	$(45910 < \textbf{salary} \leq 98469) \wedge (\textbf{commission} \leq 51486) \wedge (\textbf{age} \leq 39) \wedge (\textbf{hval} \leq 705560)$
Rule 26:	$(125706 < \textbf{salary} \leq 127088) \wedge (\textbf{age} \leq 51)$
Rule 4:	$(23873\ \textbf{salary} \leq 26812) \wedge (\textbf{age} > 61) \wedge (\textbf{loan} > 237756)$

Figure 6: Group A rules generated by C4.5rules for Function 2.

(a) Original classification rules defining Group A tuples

Group A: $((\textbf{age} < 40) \wedge$
$(((\textbf{elevel} \in [0..1])?(25K \leq \textbf{salary} \leq 75K)) : (50K \leq \textbf{salary} \leq 100K)))) \vee$
$((40 \leq \textbf{age} < 60) \wedge$
$(((\textbf{elevel} \in [1..3])?(50K \leq \textbf{salary} \leq 100K)) : (75K \leq \textbf{salary} \leq 125K)))) \vee$
$((\textbf{age} \geq 60) \wedge$
$(((\textbf{elevel} \in [2..4])?(50K \leq \textbf{salary} \leq 100K)) : (25K \leq \textbf{salary} \leq 75K))))$

(b) Rules generated by NeuroRule

$R1$:	if $(40 \leq \textbf{age} < 60) \wedge (\textbf{elevel} \leq 1) \wedge (75\text{K} \leq \textbf{salary} < 100\text{K})$ then Group A
$R2$:	if $(\textbf{age} < 60) \wedge (\textbf{elevel} \geq 2) \wedge (50\text{K} \leq \textbf{salary} < 100\text{K})$ then Group A
$R3$:	if $(\textbf{age} < 60) \wedge (\textbf{elevel} \leq 1) \wedge (50\text{K} \leq \textbf{salary} < 75\text{K})$ then Group A
$R4$:	if $(\textbf{age} \geq 60) \wedge (\textbf{elevel} \leq 1) \wedge (\textbf{salary} < 75\text{K})$ then Group A
$R5$:	if $(\textbf{age} \geq 60) \wedge (\textbf{elevel} \geq 2) \wedge (50\text{K} \leq \textbf{salary} < 100\text{K})$ then Group A

(C) Rules generated by C4.5rules

Rule 30:	$(\textbf{elevel} = 2) \wedge (50762 < \textbf{salary} \leq 98490)$
Rule 25:	$(\textbf{elevel} = 3) \wedge (48632 < \textbf{salary} \leq 98490)$
Rule 23:	$(\textbf{elevel} = 4) \wedge (60357 < \textbf{salary} \leq 98490)$
Rule 32:	$(33 < \textbf{age} \leq 60) \wedge (48632 < \textbf{salary} \leq 98490) \wedge (\textbf{elevel} = 1)$
Rule 57:	$(\textbf{age} > 38) \wedge (102418 < \textbf{salary} \leq 124930 \wedge (\textbf{age} \leq 59) \wedge \textbf{elevel} = 4)$
Rule 37:	$(\textbf{salary} > 48632) \wedge (\textbf{commission} > 18543)$
Rule 14:	$(\textbf{age} \leq 39) \wedge (\textbf{elevel} = 0) \wedge (\textbf{salary} \leq 48632)$
Rule 16:	$(\textbf{age} > 59) \wedge (\textbf{elevel} = 0) \wedge (\textbf{salary} \leq 48632)$
Rule 12:	$(\textbf{age} > 65) \wedge (\textbf{elevel} = 1) \wedge (\textbf{salary} \leq 48632)$
Rule 48:	$(\textbf{car} = 4) \wedge (98490 < \textbf{salary} \leq 102418)$

Figure 7: Classification function 4 and rules extracted.

Table 3: Accuracy rates of the rules extracted for function 4

Rule	Test data size					
	1000		5000		10000	
	Total	Correct (%)	Total	Correct (%)	Total	Correct (%)
$R1$	22	100.0	111	100.0	239	100.0
$R2$	165	93.9	753	92.6	1463	92.3
$R3$	46	82.6	247	78.4	503	78.3
$R4$	51	82.4	305	87.9	597	89.4
$R5$	71	100.0	385	100.0	802	100.0

From Table 3 , we can see that two of the rules extracted classify the tuples correctly without errors. They are exactly the same as parts of the original function definition. Because the accuracy of the pruned network is not 100%, other rules extracted are not the same as the original ones. However, the rule extracting phase preserves the classification accuracy of the pruned network. It is expected that, with higher accuracy of the network, the accuracy of the extracted rules will be also improved.

When the same training data set was used as the input of C4.5rules, twenty rules were generated among which 10 rules define the conditions of Group A (Figure 7). Again, we can see that NeuroRule generates better rules than C4.5rules. Furthermore, rules generated by NeuroRule only reference those attributes appeared in the original classification functions. C4.5rules in fact picked some attributes, e.g. *car* , that does not appear in the original function.

5 Conclusion

In this paper we reported NeuroRule, a connectionist approach to mining classification rules from given databases. The approach consists of three phases: (1) training a neural network that correctly classifies tuples in the given training data set to a desired accuracy; (2) pruning the network while maintaining the classification accuracy; and (3) extracting explicit rules from the pruned network. The proposed approach was applied to a set of classification problems. The results of applying it to a data mining problem defined in [2] was discussed in detail. The results indicate that, using the proposed approach, high quality rules can be discovered from the given ten data sets. While considerable work on using neural networks for classification has been reported, none of them can generate rules with the quality comparable to those generated by NeuroRule.

The work reported here is our first attempt to apply the connectionist approach to data mining. A number of related issues are to be further studied. One of the issues is to reduce the training time of neural networks. Although we have been improving the speed of network training by developing fast algorithms, the time required for NeuroRule is still longer than the time needed by the symbolic approach, such as C4.5. As the long initial training time of a network may be tolerable, incremental training and rule extraction during the life time of an application database can be useful. With incremental training that requires less time, the accuracy of rules extracted can be improved along with the change of database contents.

References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining approaches. In *Proceedings of the 18th VLDB Conference*, 1992.

[2] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engineering*, 5(6), December 1993.

[3] T. Ash. Dynamic node creation in backpropgation networks. *Connection Science*, 1(4):365–375, 1989.

[4] R. Battiti. First- and second-order methods for learning: between steepest descent and newton's method. *Neural Computation*, 4:141–166, 1992.

[5] N. Cercone and M Tsuchiya. Guest editors, special issue on learning and discovery in databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(6), December 1993.

[6] J.E. Dennis Jr. and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentic Hall, Englewood Cliffs, NJ, 1983.

[7] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, Fall 1992.

[8] L. Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, 1994.

[9] J. Han, Y. Cai, and H. Cercone. Knowledge discovery in databases: An attribute oriented approach. In *Proceedings of the VLDB conference*, pages 547–559, 1992.

[10] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the theory of neural computation.* Addison-Wesley Pub. Company, 1991.

[11] Y. Hirose, K. Yamashita, and S. Hijiya. Backpropagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, 1991.

[12] H. Liu. X2R: A fast rule generator In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC'95)*, Vancourver, 1995.

[13] C.J. Matheus, P.K. Chan, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. *IEEE Trans. on Knowledge and Data Engineering*, 5(6), December 1993.

[14] G. Piatetsky-Shapiro. Editor, special isssue on knowledge discovery in databases. *International Journal of Intelligent Systems*, 7(7), September 1992.

[15] G. Piatetsky-Shapiro. Guest editor introduction: Knowledge discovery in databases - from research to applications. *International Journal of Intelligent Systems*, 5(1), January 1995.

[16] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[17] J.R. Quinlan. Comparing connectionist and symbolic learning methods. In S.J. Hanson, G.A. Drastall, and R.L. Rivest, editors, *Computational Learning Therory and Natural Learning Systems*, volume 1, pages 445–456. A Bradford Book, The MIT Press, 1994.

[18] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentic Hall, 1995.

[19] R. Setiono. A neural network construction algorithm which maximizes the likelihood function. *Connection Science to appear*, 1995.

[20] R. Setiono. A penalty function approach for pruning feedforward neural networks. *Submitted for publication*, 1994.

[21] R. Setiono. Extracting rules from neural network by pruning and hidden unit splitting. *Submitted for publication*, 1994.

[22] R. Setiono and H. Liu. Improving backpropgation learning with feature selection. *Applied Intelligence to appear*, 1995.

[23] D.F. Shanno and K.H. Phua. Algorithm 500: Minimization of unconstrained multivariate functions. *ACM Transaction on Mathematical Software*, 2(1):87–96, 1976.

[24] J.W. Shavlik, R.J. Mooney, and G.G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143, 1991.

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

[26] A. van Ooyen and Nienhuis B. Improving the convergence of the backpropagation algorithm. *Neural Networks*, 5:465–471, 1992.

[27] R.L. Watrous. Learning algorithms for connectionist networks: Applied gradient methods for nonlinear optimization. In *Proceedings of IEEE First International Conference on Neural Networks*, pages 619–627. IEEE Press, New York, 1987.

[28] S. M. Weiss and C. A. Kulikowski. *Computer Systems That Learn.* Morgan Kaufmann Publishers, San Mateo, California, 1991.