RGANN: An Efficient Algorithm to Generate Rules from ANNs

S. M. Kamruzzaman

Abstract—This paper describes an efficient rule generation algorithm, called rule generation from artificial neural networks (RGANN) to generate symbolic rules from ANNs. Classification rules are sought in many areas from automatic knowledge acquisition to data mining and ANN rule extraction. This is because classification rules possess some attractive features. They are explicit, understandable and verifiable by domain experts, and can be modified, extended and passed on as modular knowledge. A standard three-layer feedforward ANN is the basis of the algorithm. A four-phase training algorithm is proposed for backpropagation learning. Comparing them to the symbolic rules generated by other methods supports explicitness of the generated rules. Generated rules are comparable with other methods in terms of number of rules, average number of conditions for a rule, and predictive accuracy. Extensive experimental studies on several benchmarks classification problems, including breast cancer, wine, season, golf-playing, and lenses classification demonstrate the effectiveness of the proposed approach with good generalization ability.

Index Terms—backpropagation artificial neural networks, clustering algorithm, continuous activation function, pruning algorithm, rule generation algorithm, symbolic rules.

I. INTRODUCTION

A RTIFICIAL neural networks (ANNs) have been successfully applied in a variety of problem domains [1]. In many applications, it is highly desirable to generate symbolic classification rules from these networks. Unlike a collection of weights, symbolic rules can be easily interpreted and verified by human experts. They can also provide new insights into the application problems and the corresponding data [2]. While the predictive accuracy obtained by ANNs is often higher than that of other methods or human experts, it is generally difficult to understand how ANNs arrive at a particular conclusion due to the complexity of the ANN architectures [4]. It is often said that an ANN is practically a “black box”. Even for an ANN with only single hidden layer, it is generally impossible to explain why a particular pattern is classified as a member of one class and another pattern as a member of another class, due to the complexity of the network [5].

This paper proposes an efficient algorithm RGANN, for generating symbolic rules from ANNs. A three-phase training algorithm is proposed for backpropagation learning.

In the first phase, appropriate network architecture is determined using weight freezing based constructive and pruning algorithm. In the second phase, the continuous activation values of the hidden nodes are discretized, by using an efficient heuristic clustering algorithm. And finally in the third phase, rules are generated through the frequently occurred pattern based rule generation algorithm RG by examining the discretized activation values of the hidden nodes.

II. RELATED WORK

Because of the strong research interest, as shown in the literature, a number of algorithms for extracting rules from trained ANNs have been developed in the last two decades [1]-[17]. In 1988 Saito and Nakano [8] proposed a medical diagnosis expert system based on a multilayer ANN. They treated the network as black box and used it only to observe the effects on the network output caused by change the inputs.

Two methods for extracting rules from ANN are described by Towell and Shavlik [9] in 1993. The first method is the subset algorithm [10], which searches for subsets of connections to a unit whose summed weight exceeds the bias of that node. The major problem with subset algorithms is that the cost of finding all subsets increases as the size of the power set of the links to each node. The second method, the MofN algorithm [11], is an improvement of the subset method that is designed to explicitly search for M-of-N rules from knowledge based ANNs. Instead of considering an ANN connection, groups of connections are checked for their contribution to a node’s activation. This is done by clustering the ANN connections.

In 1993, Craven and Shavlik [9] proposed a method that uses sampling and queries. Instead of searching for rules from the ANN, the problem of rule extraction is viewed as a learning task. The target concept is the function computed by the network and the ANN input features are the inputs for the learning task. Conjunctive rules are extracted from the ANN with the help of two oracles.

In 1995, H. Liu and S. T. Tan [10] proposed X2R, a simple and fast algorithm that can be applied to both numeric and discrete data, and generate rules from data sets. It can
generate perfect rules in the sense that the error rate of the rules is not worse than the inconsistency rate found in the original data. The rules generated by X2R, are order sensitive, i.e. the rules should be fired in sequence.

Also in 1995, R. Setiono and H. Liu [14] presented a novel way to understand an ANN by extracting rules with a three phase algorithm: firstly, a weight decay backpropagation network is built so that important connections are reflected by their bigger weights; secondly, the network is pruned such that insignificant connections are deleted while its predictive accuracy is still maintained; and finally, rules are extracted by recursively discretizing the hidden node activation values.

In 1996, R. Setiono [5] proposed a rule extraction algorithm for extracting rules from pruned ANNs for breast cancer diagnosis. It is described in detail how the activation values of a hidden node can be clustered such that only a finite and usually small number of discrete values need to be considered while at the same time maintaining the network accuracy. A small number of different discrete activation values and a small number of connections from the inputs to the hidden units will yield a set of compact rules for the problem.

In 1996, R. Setiono also proposed a rule extraction algorithm named NeuroRule [4] which can extracts symbolic classification rules from a pruned network with a single hidden layer in two steps. Firstly, the rules that explain the network outputs are generated in terms of the discretized activation values of the hidden units. Secondly, the rules that explain the discretized hidden unit activation values are generated in terms of the network inputs. When the two sets of rules are merged, a DNF representation of network classification can be obtained.

In 1997, R. Setiono [5] proposed a rule extraction (RX) algorithm to extract rules from a pruned ANN. The process of extracting rules from a trained ANN can be made much easier if the complexity of the ANN has firstly been removed. The pruning process attempts to eliminate as many connections as possible from the ANN, while at the same time tries to maintain the prespecified accuracy rate.

In 1998, Huan Liu [15] described a family of rule generators that can be used to extract classification rules in various applications. It involves versions that can handle noise in data, produce perfect rules, and can induce order independent or dependent rules. The basic idea of the proposed algorithm is simple: using first order information in the data to determine shortest sufficient conditions in a pattern that can differentiate the pattern from patterns belonging to other classes.

In 2000, R. Setiono and W. K. Leow [1] proposed a method, FERNN (fast extraction of rules from neural networks), for extracting symbolic rules from trained feedforward ANNs with a single hidden layer. The method does not require network pruning and hence no network retraining is necessary. Given a fully connected trained feedforward ANN with single hidden layer, FERNN first identifies the relevant hidden nodes by computing their information gains. For each relevant hidden node, its activation values is divided into two subintervals such that the information gain is maximized. FERNN finds the set of relevant ANN connections from the input nodes to the hidden nodes by checking the magnitudes of their weights. The connections with larger weights are identified as relevant. Finally, FERNN generates rules that distinguish the two subintervals of the hidden node activation values in terms of the network inputs.

Also in 2000, R. Setiono [16] presented M-of-N3, a new method for extracting M-of-N rules from ANNs. The topology of the ANN is the standard three-layered feedforward network. Nodes in the input layer are connected only to the nodes in the hidden layer, while nodes in the hidden layer are also connected to nodes in the output layer. Given a hidden node of a trained ANN with N incoming connections, show how the value of M can be easily computed. In order to facilitate the process of extracting M-of-N rules, the attributes of the dataset have binary values −1 or 1.

In 2002, R. Setiono, W. K. Leow and Jack M. Zurada [17] described a method, REFANN (rule extraction from function approximating neural networks), for extracting rules from trained ANNs for nonlinear regression. It is shown that REFAANN produces rules that are almost as accurate as the original networks from which the rules are extracted.

In 2006, S. M. Kamruzzaman and Md. Monirul Islam [2] proposed a new algorithm, REANN to extract rules from trained ANNs for medical diagnosis problems. This paper investigates the rule extraction process for only 3 data sets include breast cancer, diabetes and lenses.

III. OBJECTIVES OF THE RESEARCH

Multilayer feedforward ANNs trained by using the backpropagation-learning algorithm is limited to searching for a suitable set of weights in an apriori fixed network topology. Too small networks are unable to learn the problem well while overly large networks tend to over fit the training data, and consequently result in poor generalization performance. This paper proposes a hybrid approach with both constructive and pruning components for automatic determination of simplified ANN architectures. The objectives of the research are summarized as follows:

(i) To develop an efficient algorithm for generating symbolic rules from ANNs,
(ii) To find an efficient method for clustering the outputs of hidden nodes, and
(iii) To generate concise rules with high predictive accuracy.

IV. PROPOSED ALGORITHM

It is becoming increasingly apparent that without some form of explanation capability, the full potential of ANNs
may not be realized. The rapid and successful proliferation of applications incorporating ANNs in many fields, such as commerce, science, industry, medicine etc., offers a clear testament to the capability of ANN paradigm. Generating symbolic rules from trained ANN is one of the promising areas that are commonly used to explain the functionality of ANNs. The aim of this section is to introduce a new algorithm named RGANN (rule generation from ANNs) to generate symbolic rules from trained ANNs. Detailed descriptions of RGANN are presented below.

A. The RGANN Algorithm

A standard three-layer feedforward ANN is the basis of the proposed algorithm RGANN. The hyperbolic tangent function, which can take any value in the interval \([-1, 1]\), is used as the hidden node activation function. Rules are generated from near optimal ANN by using a new rule generation algorithm, RG. The aim of RGANN is to search for simple rules with high predictive accuracy. The major steps of RGANN are summarized in Fig. 1 and explained further as follows:

**Step 1**
Create an initial ANN architecture. The initial architecture has three layers, including an input, an output, and a hidden layer. The number of nodes in the input and output layers is the same as that of the problem. Initially, the hidden layer contains only one node. The number of nodes in the hidden layer is automatically determined by using a basic constructive algorithm. Randomly initialize all connection weights within a certain small range.

**Step 2**
Remove redundant input nodes, and connections between input nodes and hidden nodes and between hidden nodes and output nodes by using a basic pruning algorithm. When pruning is completed, the ANN architecture contains only important nodes and connections. This architecture is saved for the next step.

**Step 3**
Discretize the outputs of hidden nodes by using an efficient heuristic clustering algorithm. The reason for discretization is that the outputs of hidden nodes are continuous, and thus the rules can not be readily extractable from the ANN.

**Step 4**
Generate the rules that map the inputs and outputs relationships. The task of the rule generation is accomplished in three phases. In the first phase, rules are generated by using the rule generation algorithm, RG, to describe the outputs of ANN in terms of the discretized output values of the hidden nodes. In the second phase, rules are generated by RG, to describe the discretized output values of the hidden nodes in terms of the inputs. Finally in the third phase, rules are generated by combining the rules generated in the first and second phases.

**Step 5**
Prune redundant rules generated in Step 4 by replacing specific rules with more general ones.

**Step 6**
Check the classification accuracy of the network. If the accuracy falls below an acceptable level, i.e. rule pruning is not successful, then stop. Otherwise go to Step 5.

The rules generated by RGANN are compact and comprehensible, and do not involve any weight values. The accuracy of the rules from pruned networks is as high as the accuracy of the original networks. The important features of RGANN are the rule generated by RG is recursive in nature and is order insensitive, i.e., the rules need not be required to fire sequentially.

1) Constructive Algorithm

One drawback of the traditional backpropagation algorithm is the need to determine the number of nodes in the hidden layer prior to training. To overcome this difficulty, many algorithms that construct a network dynamically have been proposed [18]-[20]. The most well known constructive algorithms are dynamic node creation (DNC) [21], feedforward neural network construction (FNNC) algorithm and the cascade correlation (CC) algorithm [22].

The constructive algorithm used in RGANN is based on the feedforward neural network construction (FNNC) algorithm proposed by Rudy Setiono and Huan Liu [23]. In FNNCA the training process is stopped when the classification accuracy on the training set is 100% [24]. However, it is not possible to get 100% classification accuracy for most of the benchmark classification problems. In addition, higher classification accuracy on the training set does not guarantee the higher generalization ability i.e. classification accuracy on the testing set. Thus a validation set is used in this study to stop the training of the network. In this study, it has been proposed that the output of a hidden node can be frozen when its output does not change much in the successive training cycles. This weight freezing method can be considered as combination of the two extremes: for training all the weights of ANNs and

![Fig. 1 Flow chart of the RGANN algorithm.](image-url)
for training the weights of only the newly added hidden node of ANNs [25]. The major steps of constructive algorithm used in RGANN are summarized in Fig. 2 and explained further as follows:

**Step 1**
Create an initial ANN consisting of three layers, i.e., an input, an output, and a hidden layer. The number of nodes in the input and output layers is the same as the number of inputs and outputs of the problem. Initially the hidden layer contains only one node i.e. \( h = 1 \). Randomly initialize all connection weights within a certain range.

**Step 2**
Train the network on the training set by using backpropagation algorithm until the error is almost constant for a certain number of training epochs, \( \tau \), is specified by the user.

**Step 3**
Compute the ANN error \( E \) on validation set. If \( E \) is found unacceptable (i.e., too large), then assume that the ANN has inappropriate architecture, and go to the next step. Otherwise stop the training process. The ANN error \( E \) is calculated according to the following equations:

\[
E(w, v) = \frac{1}{2} \sum_{i=1}^{k} \sum_{p=1}^{C} (S_{pi} - t_{pi})^2
\]

where, \( k \) is the number of patterns, \( C \) is the number of output nodes, and \( t_{pi} \) is the target value for pattern \( x_i \) at output node \( p \). \( S_{pi} \) is the output of the network at output node \( p \).

**Step 4**
Compare each hidden node’s output \( X(n) \) at training epoch \( n \) with its previous value \( X(n - \tau) \). If \( X(n) \cong X(n - \tau) \), freeze the input weights of that node.

**Step 5**
Add one hidden node to hidden layer. Randomly initialize the weights of the arcs connecting this new hidden node with input nodes and output nodes. Set \( h = h + 1 \) and go to step 2.

**Step 6**
Test the generalization ability of the of the final network by the testing set.

2) Pruning Algorithm

Pruning offers an approach for dynamically determining an appropriate network topology. Pruning techniques begin by training a larger than necessary network and then eliminate weights and nodes that are deemed redundant [25], [26].

As the nodes of the hidden layer are determined automatically by constructive algorithm in RGANN, the aim of this pruning algorithm used here is to remove, as many unnecessary connections as possible. A node is pruned if all the connections to and from the node are pruned. Typically, methods for removing weights from the network involve adding a penalty term to the error function. It is hope that by adding a penalty term to the error function, unnecessary connections will have small weights, and therefore pruning can reduce the complexity of the network significantly. The simplest and most commonly used penalty term is the sum of the squared weights.

Given a set of input patterns \( x_i \in \mathbb{R}^n, i = 1, 2, \ldots k \), let \( w_m \) is a \( p \)-dimensional vector weights for the arcs connecting the input layer and the \( m \)-th hidden node, \( m = 1, 2, \ldots, h \). The weight of the connection from the \( l \)-th input node to the \( m \)-th hidden node is denoted by \( w_{ml} \), \( v_m \) is a \( C \)-dimensional vector of weights for the arcs connecting the \( m \)-th hidden node and the output layer. The weight of the connection from the \( m \)-th hidden node to the \( p \)-th output node is denoted by \( v_{pm} \). It has been suggested that faster convergence can be achieved by minimizing the cross entropy function instead of squared error function [27]. 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

\[
F(w, v) = -\sum_{i=1}^{k} \sum_{p=1}^{C} (t_{pi} \log S_{pi} + (1 - t_{pi}) \log(1 - S_{pi}))
\]

\( S_{pi} \) is the output of the network.
\[ S_{pi} = \sigma \left( \sum_{m=1}^{h} \delta((x_i)^T w_m) v_{pm} \right) \]  

(5)

h is the number of hidden nodes in the network, \( x_i \) is an n-dimensional input pattern, \( i=1, 2, \ldots, k \), where \((x_i)^T w_m\) denotes the scalar product of the vectors \( x_i \) and \( w_m \). \( \delta(.) \) is the hyperbolic tangent function and \( \sigma(.) \) is the logistic sigmoid function. \( P(w, v) \) is a penalty term used for weight decay.

\[
P(w, v) = \varepsilon_1 \left( \sum_{m=1}^{h} \beta (w_m)^2 + \sum_{m=1}^{h} V_m \right) + \varepsilon_2 \left( \sum_{m=1}^{h} (w_m)^2 + \sum_{m=1}^{h} V_m \right) \]  

(6)

The values for the weight decay parameters \( \varepsilon_1, \varepsilon_2 > 0 \) must be chosen to reflect the relative importance of the accuracy of the network versus its complexity. More weights may be removed from the network at the cost of a decrease in the network accuracy with larger values of these two parameters. They also determine the range of values where the penalty for each weight in the network is approximately equal to \( \varepsilon_1 \). The parameter \( \beta > 0 \) determines the steepness of the error function near the origin.

This pruning algorithm removes the connections of the ANN according to the magnitudes of their weights. As the eventual goal of RGANN is to get a set of simple rules that describe the classification process, it is important that all unnecessary connections and nodes must be removed. In order to remove as many connections as possible, the weights of the network must be prevented from taking values that are too large [28]. At the same time, weights of irrelevant connections should be encouraged to converge zero. The penalty function is found to be particularly suitable for these purposes.

3) Heuristic Clustering Algorithm

The process of grouping a set of physical or abstract objects into classes of similar objects is called clustering. A cluster is a collection of data objects that are similar within the same cluster and are dissimilar to the objects in other clusters. A cluster of data objects can be treated collectively as one group in many applications [29]. There exist a large number of clustering algorithms in the literature such as k-means, k-medoids [30] [31]. The choice of clustering algorithm depends both on the type of data available and on the particular purpose and application.

After applying pruning algorithm in RGANN, the ANN architecture produced by constructive algorithm contains only important connections and nodes. Nevertheless, rules are not readily extractable because the hidden node activation values are continuous. The discretization of these values paves the way for rule generation.

It is found that some hidden nodes of an ANN maintain almost constant output while other nodes change continuously during the whole training process [32]. Fig. 3 shows a hidden node maintains almost constant output after some training epochs. In RGANN, no clustering algorithm is used when hidden nodes maintain almost constant output. If the outputs of hidden nodes do not maintain constant value, a heuristic clustering algorithm is used.

The aim of the clustering algorithm is to discretize the output values of hidden nodes. The algorithm places candidates for discrete values such that the distance between them is at least a threshold value \( \varepsilon \). A very small \( \varepsilon \) will always guarantee that the network with discrete activation values will have the same accuracy as the original network with continuous activation values. The algorithm can then be run again with a larger value of \( \varepsilon \) to reduce the number of clusters. The steps of the heuristic clustering algorithm are summarized in Fig. 4 and explained further as follows:

**Fig. 3. Output of hidden nodes.**

**Fig. 4. Flow chart of the heuristic clustering algorithm.**

**Step 1**

Let \( \varepsilon \in (0, 1) \). \( D \) is the activation values in the hidden node. \( \delta_1 \) is the activation value for the first pattern. The first cluster, \( H(1) = \delta_1 \), count = 1, and sum \((1) = \delta_1 \), set \( D = 1 \).

**Step 2**

For each pattern \( p_i \ i = 1, 2, 3, \ldots, k \). Checks whether subsequent activation values can be clustered into one of the existing clusters. The distance between an activation value under consideration and its nearest cluster, \[ |\delta - H(j)| \] is computed. If this distance is less than \( \varepsilon \), then the activation value is clustered in cluster \( j \). Otherwise, this activation
value forms a new cluster. Let \( \delta \) be its activation value. If there exists an index \( j \) such that
\[
|\delta - H(j)| = \min_{j \in \{1, \ldots, \Omega\}} |\delta - H(j)| \quad \text{and} \quad |\delta - H(j)| \leq \varepsilon
\]
set \( \text{count}(H(j)) = \text{count}(H(j)) + 1 \),
\[
\text{sum}(H(j)) = \text{sum}(H(j)) + \delta, \quad \text{else } D = D + 1,
\]
\[
H(D) = \delta, \text{count}(D) = 1, \text{sum}(D) = \delta.
\]

**Step 3**
Replace \( H \) by the average of all activation values that have been clustered into this cluster: \( H(j) = \frac{\text{sum}(j)}{\text{count}(j)}, j = 1, 2, 3, \ldots, D \).

**Step 4**
Once the activation values of all hidden nodes have been obtained, the accuracy of the network is checked with the activation values at the hidden nodes replaced by their discretized values. An activation value \( \delta \) is replaced by \( H(j) \), where index \( j \) is chosen such that \( j = \arg\min_{j \in \{1, \ldots, \Omega\}} |\delta - H(j)| \). If the accuracy of the network falls below the required accuracy, then \( \varepsilon \) must be decreased and the clustering algorithm is run again, otherwise stop.

For a sufficiently small \( \varepsilon \), it is always possible to maintain the accuracy of the network with continuous activation values, although the resulting number of different discrete activations can be impractically large. The best \( \varepsilon \) value is one that gives a high accuracy rate after the clustering and at the same time generates as few clusters as possible. A simple way of obtaining an optimal value for \( \varepsilon \) is by searching in the interval \((0, 1)\). The number of clusters and the accuracy of the network can be checked for all values of \( \varepsilon = i \zeta, i = 1, 2, \ldots, \) where \( \zeta \) is a small positive scalar, e.g. 0.10. Note also that it is not necessary to fix the value of \( \varepsilon \) equal for all hidden nodes.

4) *Rule Generation Algorithm (RG)*

Classification rules are sought in many areas from automatic knowledge acquisition [33], [34] to data mining [35], [36] and ANN rule extraction because some of their attractive features. They are explicit, understandable and verifiable by domain experts, and can be modified, extended and passed on as modular knowledge. The RG is composed of three major functions:

(i) Rule Generation- This function chooses the most frequently occurred pattern as the base to generate rule, then the next frequently occurred, etc. In this way, RG can also handle noise in the data.

(ii) Rule Clustering- Rules are clustered in terms of their class levels; and

(iii) Rule Pruning- Redundant or more specific rules in each cluster are removed.

A default rule should be chosen to accommodate possible unclassifiable patterns. If rules are clustered, the choice of the default rule is based on clusters of rules. The steps of the Rule Generation (RG) algorithm are summarized in Fig. 5 and explained further as follows:

**Step 1** Generate Rule:

Sort-on-ferq(data-without-Duplicates);
\( i = 0 \);
while (data-without-Duplicates is NOT empty) {

**Step 2**
Cluster Rule:
Cluster rules according to their class levels. Rules generated in Step 1 are grouped in terms of their class levels. In each rule cluster, redundant rules are eliminated; specific rules are replaced by more general rules.

**Step 3**
Prune Rule:
replace specific rules with more general ones;
remove noise rules;
eliminate redundant rules;

**Step 4**
Check whether all patterns are covered by any rules. If yes then stop, otherwise continue.

**Step 5**
Determine a default rule.
A default rule is chosen when no rule can be applied to a pattern.

RG exploits the first order information in the data and finds shortest sufficient conditions for a rule of a class that can differentiate it from patterns of other classes. It can generate concise and perfect rules in the sense that the error rate of the rules is not worse than the inconsistency rate found in the original data. The novelty of RG is that the rule generated by it is order insensitive, i.e. the rules need not be required to fire sequentially.

**V. EXPERIMENTAL STUDIES**

This section evaluates the performance of RGANN on several well-known benchmark classification problems.
including breast cancer, wine, season, golf playing, and lenses which are widely used in machine learning and ANN research. The data sets representing all the problems were real world data and obtained from the UCI machine learning benchmark repository.

A. Data Set Description

The following subsections briefly describe the data set used in this study. The characteristics of the data sets are summarized in Table I. The detailed descriptions of the data sets are available at ics.uci.edu in directory /pub/machine-learning-databases [37].

B. Experimental Setup

In all experiments, one bias node with a fixed input 1 was used for the hidden and output layers. The learning rate was set between [0.1, 1.0] and the weights were initialized to random values between [-1.0, 1.0]. A hyperbolic tangent function \( \delta(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}} \) was used as the hidden node activation function and a logistic sigmoid function \( \sigma(y) = \frac{1}{1+e^{-y}} \) as the output node activation function.

In this study, all data sets representing the problems were divided into two sets: the training set and the testing set. The numbers of examples in the training set and testing set were chosen to be the same as those in other works, in order to make the comparison with those works possible. The sizes of the training and testing data sets used in this study are given as follows:

Breast cancer data set: The first 350 examples are used for the training set and the rest 349 for the testing set.
Wine data set: The first 89 examples are used for the training set and the rest 89 for the testing set.

C. Experimental Results

Tables II-VI show the ANN architectures produced by RGANN and training epochs over 10 independent runs on five benchmark classification problems. The initial architecture was selected before applying the constructive algorithm, which was used to determine the number of nodes in the hidden layer. The intermediate architecture was the outcome of the constructive algorithm, and the final architecture was the outcome of pruning algorithm used in RGANN.

| Data Sets   | No. of Examples | Input Attributes | Output Classes |
|-------------|-----------------|------------------|----------------|
| Breast Cancer | 699             | 9                | 2              |
| Wine        | 178             | 13               | 3              |
| Season      | 11              | 3                | 4              |
| Golf Playing | 14              | 4                | 2              |
| Lenses      | 24              | 4                | 3              |

The breast cancer data

The purpose of this problem is to diagnose a breast tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes were for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

The wine data

In a classification context, this is a well-posed problem with "well behaved" class structures. A good data set for first testing of a new classifier, but not very challenging. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. Number of instances 178, number of attributes 13. All attributes are continuous. This was a two-class problem.

The season data

The season data set contains discrete data only. There are 11 examples in the data set, each of which consisted of three elements. These are weather, tree and temperature. This was a four-class problem.

The golf playing data

The golf playing data set contains both numeric and discrete data. There are 14 examples in the data set, each of which consisted of four-elements. These are outlook, temperature, humidity and wind. This is a two-class problem.

The lenses data

The data set contains 24 examples and are complete and noise free. The examples highly simplified the problem. The attributes do not fully describe all the factors affecting the decision as to which type, if any, to fit. Number of Instances: 24. Number of Attributes: 4; age, spectacle prescription, astigmatic and tear production rate. All attributes are nominal. This was three-class problem: hard contact lenses, soft contact lenses and not contact lenses.
It is seen that RGANN can automatically determine compact ANN architectures. For example, for the breast cancer data, RGANN produces more compact architecture. The average number of nodes and connections were 6.8 and 5.8 respectively; in most of the 10 runs 5 to 6 input nodes were pruned.

Fig. 6 shows the smallest of the pruned networks over 10 runs for breast cancer problem. The pruned network has only 1 hidden node and 5 connections. The accuracy of this network on the training data and testing data were 96.275% and 93.429% respectively. In this example only three input attributes \( A_1, A_6 \) and \( A_9 \) were important and only three discrete values of hidden node activations were needed to maintain the accuracy of the network.

The discrete values found by the heuristic clustering algorithm were 0.987, -0.986 and 0.004. Of the 350 training data, 238 patterns have the first value, 106 patterns the second value, and rest 6 patterns the third value. The weight of the connection from the hidden node to the first output node was 3.0354 and to the second node was –3.0354. Fig. 7 shows the training time error for breast cancer problem. It was observed that the training error decreased and maintained almost constant value for a long time after some training epochs and then fluctuates. The fluctuation was made due to the pruning process. As the network was retrained after completing the pruning process thus the training error again maintained almost constant value.

### TABLE III
ANN ARCHITECTURES AND TRAINING EPOCHS FOR WINE DATA. THE RESULTS WERE AVERAGED OVER 10 INDEPENDENT RUNS

| Initial Architecture | Intermediate Architecture | Final Architecture | No. of Epoch |
|----------------------|---------------------------|-------------------|--------------|
| No. of Node | No. of Connection | No. of Node | No. of Connection | No. of Node | No. of Connection |
| Mean | 17 (13-1-3) | 16 | 18.5 | 40 | 18 | 26.5 | 213 |
| Min | 17 (13-1-3) | 16 | 17 | 16 | 17 | 20 | 193 |
| Max | 17 (13-1-3) | 16 | 20 | 64 | 19 | 43 | 237 |

### TABLE IV
ANN ARCHITECTURES AND TRAINING EPOCHS FOR SEASON DATA. THE RESULTS WERE AVERAGED OVER 10 INDEPENDENT RUNS

| Initial Architecture | Intermediate Architecture | Final Architecture | No. of Epoch |
|----------------------|---------------------------|-------------------|--------------|
| No. of Node | No. of Connection | No. of Node | No. of Connection | No. of Node | No. of Connection |
| Mean | 8 (3-1-4) | 7 | 8.9 | 13.3 | 8.7 | 11.2 | 88.2 |
| Min | 8 (3-1-4) | 7 | 8 | 7 | 8 | 9 | 73 |
| Max | 8 (3-1-4) | 7 | 10 | 14 | 10 | 16 | 101 |

### TABLE V
ANN ARCHITECTURES AND TRAINING EPOCHS FOR GOLF PLAYING DATA. THE RESULTS WERE AVERAGED OVER 10 INDEPENDENT RUNS

| Initial Architecture | Intermediate Architecture | Final Architecture | No. of Epoch |
|----------------------|---------------------------|-------------------|--------------|
| No. of Node | No. of Connection | No. of Node | No. of Connection | No. of Node | No. of Connection |
| Mean | 7 (4-1-2) | 6 | 8.2 | 13.2 | 7.9 | 10.5 | 94.5 |
| Min | 7 (4-1-2) | 6 | 7 | 6 | 7 | 6 | 86 |
| Max | 7 (4-1-2) | 6 | 9 | 18 | 9 | 14 | 103 |

### TABLE VI
ANN ARCHITECTURES AND TRAINING EPOCHS FOR LENSES DATA. THE RESULTS WERE AVERAGED OVER 10 INDEPENDENT RUNS

| Initial Architecture | Intermediate Architecture | Final Architecture | No. of Epoch |
|----------------------|---------------------------|-------------------|--------------|
| No. of Node | No. of Connection | No. of Node | No. of Connection | No. of Node | No. of Connection |
| Mean | 8 (4-1-3) | 7 | 9.1 | 14.7 | 8.9 | 12.1 | 109.2 |
| Min | 8 (4-1-3) | 7 | 8 | 7 | 8 | 7 | 97 |
| Max | 8 (4-1-3) | 7 | 10 | 21 | 10 | 17 | 128 |
TABLE VII

| Data Sets          | No. of Generated Rules | Training Set Accuracy | Testing Set Accuracy |
|--------------------|------------------------|-----------------------|----------------------|
| Breast Cancer      | 2                      | 96.28 %               | 93.43 %              |
| Wine               | 3                      | 91.01 %               | 83.15 %              |
| Season             | 4                      | 100 %                 | 100 %                |
| Golf Playing       | 3                      | 100 %                 | 100 %                |
| Lenses             | 8                      | 100 %                 | 100 %                |

1) Generated Rules

The number of rules generated by RGANN and the accuracy of the rules in training and testing data sets were described in Table VII, but the visualization of the rules in terms of the original attributes were not discussed. The following subsections discussed the rules generated by RGANN in terms of the original attributes. The number of conditions per rule and the number of rules generated were also visualized here.
Table VII shows the number of generated rules and the rules accuracy for the five-benchmark problems. In most of the cases RGANN produces fewer rules with better accuracy. It was observed that two to three rules were sufficient to solve the problems. The accuracies were 100% for three data sets including season, golf playing, and lenses classification. These data sets have a lower number of examples.

VI. COMPARISONS

This section compares experimental results of RGANN with the results of other works. The primary aim of this work is not to evaluate RGANN in order to gain a deeper understanding of rule generation without an exhaustive comparison between RGANN and all other works.

Table VIII compares the RGANN results of the breast cancer problem with those produced by REANN [2], NN RULES [4], DT RULES [4], C4.5 [33], NN-C4.5 [38], OC1 [38], and CART [39] algorithms. RGANN and REANN achieved best performance although NN RULES was the closest second, but number of rules generated by RGANN are 2 whereas these were 4 for NN RULES.

Table IX shows RGANN results of wine data. RGANN achieved 91.01% accuracy on wine data by generating 3 rules. No detailed previous work found for showing comparison of this data set.

Table X compares the RGANN results of the season data with those produced by RULES [40] and X2R [13]. All three algorithms achieved 100% accuracy. This is possible because the number of examples is low. RGANN generated 8 rules, whereas RULES did 7 and X2R did 6.

Table XI compares RGANN results of golf playing data with those produced by RULES, RULES-2 [41], and X2R. All four algorithms achieved 100% accuracy because the lower number of examples. Number of generated rules by RGANN are 3 whereas these were 8 for RULES and 14 for RULES-2.

Table XII compares RGANN results of lenses data with those produced by REANN, PRISM [42]. Both algorithms achieved 100% accuracy because the lower number of examples. Number of generated rules by RGANN are 8 whereas these were 9 for PRISM.

VII. CONCLUSION

This work is an attempted to open up these black boxes by generating symbolic rules from it through the proposed efficient rule generation algorithm RGANN. The algorithm can generate concise rules from standard feedforward ANN. An important feature of the rule generation algorithm, RG, is its recursive nature. The rules are concise, comprehensible, order insensitive and do not involve any weight values. The accuracy of the rules from a pruned network is as high as the accuracy of the fully connected network.

Extensive experiments have been carried out in this study to evaluate how well RGANN performed on five benchmark classification problems in ANNs including breast cancer, wine, season, golf playing, and lenses classification problems in comparison with other algorithms. In almost all cases, RGANN outperformed the others.
It is noted here that in our previous work REANN, the algorithm was tested for only 3 data sets of medical diagnosis problems but this RGANN algorithm is general one capable of generating concise rules from a wide variety of benchmark data sets.

REFERENCES

[1] R. Setiono and W. K. Leow, “FERNN: An algorithm for fast extraction of rules from neural networks,” Applied Intelligence, vol. 12, 2000, pp. 15-25.

[2] S. M. Kamruzzaman, and Md. Monirul Islam, “An Algorithm to Extract Rules from Artificial Neural Networks for Medical Diagnosis Problems”, International Journal of Information Technology (IJIT), ISSN: 0218-7957, Vol. 12, No. 8, 2006, pp. 41-59.

[3] S. M. Kamruzzaman, “Extracting Symbolic Rules for Medical Diagnosis Problem”, Proceedings of 8th International Conference on Computer and Information Technology (ICCIT-2005), ISBN-984-32-2873-1, Islamic University of Technology (IUT), Dhaka, Bangladesh, December 28-30, 2005, pp. 602-607.

[4] R. Setiono and H. Liu, “Symbolic presentation of neural networks,” IEEE Computer, March 1996, pp. 71-77.

[5] R. Setiono, “Extracting rules from neural networks by pruning and hidden-unit node splitting,” Neural Computation, vol. 9, 1997, pp. 205-225.

[6] R. Andrews, J. Diederich and A. B.,Tickle, “Survey and critique of techniques for extracting rules from trained artificial neural networks,” Knowledge Based System, vol. 8, 1995, pp. 373-389.

[7] Ashish Darbari, “Rule Extraction from Trained ANN: A Survey,” Technical Report, Department of Computer Science, Dresden University of Technology, Dresden, Germany, 2000.

[8] K. Saito and R. Nakano, “Medical diagnosis expert system based on PDP model,” Proceedings of IEEE International Conference on Neural Networks, IEEE Press, 1988, pp. 1255-1262.

[9] G. G., Towell and J. W., Shavlik, “Extracting refined rules from knowledge-based system neural networks,” Machine Learning, vol. 13, 1993, pp. 71-101.

[10] L. Fu, “Rule learning by searching on adapted nets,” Proceedings of the Ninth National Conference on Artificial Intelligence, The MIT Press, Menlo Park, CA, 1991, pp. 590-595.

[11] G. G., Towell and J. W., Shavlik, “Knowledge-based artificial neural networks,” Artificial Intelligence, vol. 70, 1994, pp. 119-165.

[12] M.W., Craven and J. W., Shavlik, “Using sampling and queries to extract rules from trained neural networks,” Proceedings of the Eleventh International Conference on Machine Learning, Morgan and Kaufmann, San Mateo, CA, 1994.

[13] H. Liu and S. T. Tan, “X2R: A fast rule generator,” Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vancouver, CA, 1995.

[14] R. Setiono and Huan Liu, “Understanding neural networks via rule extraction,” Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1995, pp. 480-485.

[15] Huan Liu, “A family of efficient rule generators,” Encyclopedia of Computer Science and Technology, vol. 39, Marcel Dekker Inc., New York, 1998, pp. 15-28.

[16] R. Setiono, “Extracting M-of-N rules from trained neural networks,” IEEE Transactions of Neural Networks, vol. 11, 2000, pp. 512-519.

[17] R. Setiono, W. K. Leow and Jack M. Zurada, “Extraction of Rules from Artificial Neural Networks for Nonlinear regression,” IEEE Trans. of Neural Networks, vol. 13, 2002, pp. 564-577.

[18] T. Y. Kwok and D. Y. Yeung, “Constructive algorithms for structured learning in feedforward neural networks for regression problems,” IEEE Transactions on Neural Networks, vol. 8, 1997, pp. 630-645.

[19] M. Monirul Islam, Xin Yao, and K. Murase, “A constructive algorithm for training cooperative neural network ensembles,” IEEE Transactions on Neural Networks, vol. 14, 2003, pp. 820-834.

[20] R. Parekh, J. Yang, and V. Honavar, “Constructive neural network learning algorithms for pattern classification,” IEEE Transactions on Neural Networks, vol. 11, 2000.

[21] T. Ash, “Dynamic node creation in backpropagation networks,” Connection Science, vol. 1, 1989, pp. 365–375.

[22] S. E. Fahlman and C. Lebiere, “The cascade-correlation learning architecture,” Advances in Neural Information Processing System 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524-532.

[23] R. Setiono and L.C.K. Hui, “Use of quasi-Newton method in a feedforward neural network construction algorithm”, IEEE Transactions on Neural Networks, vol. 6, 1995, pp. 273-277.

[24] S. M. Kamruzzaman, Ahmed Ryad Hasan, Abu Bakar Siddiquee, and Md. Ehsanul Hoque Mazumder, “Medical diagnosis using neural network,” Proceedings of the International Conference on Electrical and Computer Engineering (ICECE-2004), BUET, Dhaka, 2004, pp. 537-540.

[25] M. Monirul Islam, M. A. H. Akhand, M. Abdur Rahman and K. Murase, “Weight freezing to reduce training time in designing artificial neural networks”, Proceedings of 5th International Conference on Computer and Information Technology, EWU, Dhaka, 2002, pp. 132-136.

[26] J. Sietsema and R. J. F. Dow, “Neural net pruning-why and how?,” Proceedings of IEEE International Conference on Neural Networks, vol. 1 (San Diego), 1988, pp. 325-333.

[27] A. van Ooyen and B. Nienhuis, “Improving the convergence of backpropagation algorithm,” Neural Networks, vol. 5, 1992, pp. 465-471.

[28] R. Reed, “Pruning algorithms A survey,” IEEE Transactions on Neural Networks, vol. 4, 1993, pp. 740-747.

[29] Han Jiawei, Micheline Kamber, “Data Mining: Concepts and Techniques,” Morgan Kaufmann Publisher: CA, 2001.

[30] L. Kaufman, P. J. Rousseeuw, “Finding Groups in Data: An Introduction to Cluster Analysis,” John Wiley & Sons, 1990.

[31] T. Ng. Raymond, Jiawei Han, “Efficient and effective clustering methods for spatial data mining,” VLDB Conference, Santiago, Chile, 1994.

[32] M. Monirul Islam and K. Murase, “A new algorithm to design compact two hidden-layer artificial neural networks”, Neural Networks, vol. 4, 2001, pp. 1265–1278.

[33] J. R. Quinlan, “C4.5: Programs for Machine Learning,” Morgan Kaufmann, San Mateo, CA, 1993.

[34] S. Russel and P. Norvig, “Artificial Intelligence: A Modern Approach,” Prentice Hall, 1995.

[35] R. Agrawal, T. Imielinski, and A. Swami, “Database mining: A performance perspective,” IEEE Transactions on Knowledge and Data Engineering, vol. 5, 1993, pp. 914-925.

[36] J. S-Yen and A. L. P. Chen, “An efficient algorithm for deriving compact rules from databases,” Proceedings of the Fourth International Conference on Database Systems for Advanced Applications, 1995.

[37] Prechelt, “Proben1-A Set of Neural Network Benchmark Problems and Benchmarking Rules”, University of Karlsruhe, Germany, 1994.

[38] R. Setiono, “Techniques for extracting rules from artificial neural networks,” Plenary lecture presented at the 5th International Conference on Soft Computing and Information Systems, Iizuka, Japan, October 1998.
[39] L. Breiman, J. Friedman, R. Olshen, and C. Stone, “Classification and Regression Trees,” Wadsworth and Brooks, Monterey, CA, 1984.

[40] D. T. Pham and M. S. Aksoy, “Rules: A simple rule extraction system,” Expert Systems with Applications, vol. 8, 1995.

[41] D. T. Pham and M. S. Aksoy, “An algorithm for automatic rule induction,” Artificial Intelligence in Engineering, vol. 8, 1994.

[42] J. Cendrowska, “PRISM: An algorithm for inducting modular rules,” International Journal of Man-Machine Studies, vol. 27, 1987, pp. 349-370.

S. M. Kamruzzaman received the B. Sc. Engineering degree in Electrical and Electronic Engineering from Dhaka University of Engineering and Technology (formerly Bangladesh Institute of Technology, Dhaka), Bangladesh in 1997 and the M. Sc. Engineering degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 2005.

From March 1998 to December 2004, he was a Faculty of the Department of Computer Science and Engineering, International Islamic University Chittagong (IIUC), Chittagong, Bangladesh. From January 2005 to July 2006 he was an Assistant Professor with the Department of Computer Science and Engineering, Manarat International University, Dhaka, Bangladesh. In August 2006, he moved to the Department of Information and Communication Engineering as an Assistant Professor at the University of Rajshahi, Bangladesh. His research interests include signal processing, mobile and wireless communications, neural networks and their application, data mining, pattern recognition and image processing. Currently he is an author of 7 journals and 26 international conference papers. He attended 7 international conferences for presenting papers. He is a member of several national and international professional associations.