OPENING THE NEURAL NETWORK BLACKBOX: AN ALGORITHM FOR EXTRACTING RULES FROM FUNCTION APPROXIMATING ARTIFICIAL NEURAL NETWORKS

Rudy Setiono
Wee Kheng Leow
National University of Singapore
Singapore

James Y.L. Thong
Hong Kong University of Science and Technology
Hong Kong

Abstract

Artificial neural networks have been successfully applied to solve a variety of business applications involving classification and function approximation. In many such applications, it is desirable to extract knowledge from trained neural networks so that the users can gain a better understanding of the solution. Existing research works have focused primarily on extracting symbolic rules for classification problems with few methods devised for function approximation problems. In order to fill this gap, we propose an approach to extract rules from neural networks that have been trained to solve function approximation problems. The extracted rules divide the data samples into groups. For all samples within a group, a linear function of the relevant input attributes of the data approximates the network output. Experimental results show that the proposed approach generates rules that are more accurate than the existing methods based on decision trees and regression.

Keywords: Neural networks, knowledge acquisition, decision rules.

1. INTRODUCTION

Artificial neural networks are powerful tools for business decision making (Coakley and Brown, 1993; Dutta, Shekhar and Wong, 1994; Kim and McLeod, 1999; Salchenberger, Cinar and Lash, 1992; Tam and Kiang, 1992; Tana and Koh, 1992; Trippi and Turban, 1993, Walczak, 1999; Wilson and Sharda, 1994). They work particularly well for problems involving classification and data fitting/function approximation. Neural networks often predict with higher accuracy than other techniques because of the networks' capability of fitting any continuous function. The main drawback of applying neural networks to solve these problems is the lack of explanation power in the trained networks due to the complex structure of the networks. In many applications, it is desirable to extract knowledge from trained neural networks for the users to gain better understanding of the problems in hand. The extracted knowledge is usually expressed as symbolic rules of the form

if *condition*, then *consequence*.

In order to generate rules from neural networks that are easy for a human user to understand, the rules must be sufficiently simple yet accurate. The conditions of the rules describe a subregion of the input space, while the consequences of the rules for function approximation are of the form $Y = f(X)$, where $f(X)$ is either a constant or a linear function of X, the attributes of the data. This type of rules is easy to understand because of their similarity to the traditional statistical approach of parametric regression. Since a single rule will not approximate the nonlinear mapping of the network well, one possible solution is to

divide the input space of the data into subregions. Prediction for all samples in the same subregion will be performed by a single linear equation whose coefficients are determined by the weights of the network connections. With finer division of the input space, more rules are produced and each rule can approximate the network output more accurately. However, in general, too many rules – with each rule applying to only a small number of samples – do not provide meaningful or useful knowledge to the user. Hence, a balance must be achieved between rule accuracy and rule simplicity.

Most existing research works have focused on extracting symbolic rules for solving classification problems where the network outputs are discrete. A function approximation problem, on the other hand, has continuous output. The existing literature shows that only a few methods have been devised to extract rules from trained neural networks for function approximation (Tickle, Andrews, Golea and Diederich, 1998). To address this deficiency in research, this article proposes a new method called REFANN.

The REFANN (Rule Extraction from Function Approximating Neural Networks) method produces rules that are almost as accurate as the networks from which the rules are extracted. For some problems, there are sufficiently few rules that useful knowledge about the problem domain can be gained. REFANN works on a network with a single hidden layer and one linear output unit. To reduce the number of rules and to simplify the rule conditions, redundant network input units and hidden units are removed by pruning. The continuous activation function of each hidden unit is then approximated by a 3-piece linear function. The various combinations of the approximating linear functions divide the input space into subregions such that the function values for all inputs in the same subregion can be computed by a predicting linear function of the inputs. Extensive experiments have been performed and the results show that REFANN's accuracy in function approximation is better than those of existing methods based on decision trees and regression.

This article is organized as follows. Section 2 describes the network architecture and the training algorithm. Section 3 describes how the nonlinear activation function of a hidden unit is approximated by a piecewise linear function. Illustrative examples and results from our experiments are presented in Sections 4 and 5. Finally, in Section 6, we discuss future works and conclude the article.

2. NEURAL NETWORK ARCHITECTURE AND TRAINING

The neural network consists of one layer of N input units, a layer of H nonlinear hidden units and one output unit (Pendharkar and Rodger, 1999). Given an N-dimensional input pattern p, p = 1, 2,..., P, the network's hidden unit activation value A_{ip} and output unit value $\tilde{y}_p$ are computed as follows:

$$\tilde{y}_p = \sum_{i=1}^{H} v_i A_{ip} \tag{1}$$

$$A_{ip} = h\left(\sum_{j=1}^{N} w_{ij} I_{jp} \right) \tag{2}$$

where I_{jp} is the value of input unit j of pattern p, w_{ij} is the weight of the connection from input unit j to hidden unit i, v_i is the weight of the connection from hidden unit i to the output unit, and $h(x)$ is the hyperbolic tangent function $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. Let y_p be the target function value for input pattern p. We train a network such that the following augmented error function is minimized:

$$F(w,v) = \sum_{p=1}^{P} (\tilde{y}_p - y_p)^2 + \theta(w,v) \tag{3}$$

$$\theta(w,v) = \varepsilon \left(\sum_{i=1}^{H} \sum_{j=1}^{N} \frac{w_{ij}^2}{1+w_{ij}^2} + \sum_{i=1}^{H} \frac{v_i^2}{1+v_i^2} + \sum_{i=1}^{H} \sum_{j=1}^{N} w_{ij}^2 + \sum_{i=1}^{H} v_i^2 \right) \tag{4}$$

where ε is a small positive penalty parameter. The penalty term $\theta(w,v)$ is added to the usual sum of squared errors function so that unnecessary network connections will have small weights.

After training the network, a pruning algorithm (Hertz, Krogh and Palmer, 1991) is applied to remove redundant hidden units and irrelevant input attributes. The pruning algorithm removes an input or a hidden unit from the network if the removal does not deteriorate the network's prediction accuracy on the training samples. Removal of excessive units is crucial in obtaining a concise set of extracted rules. It should be noted that the rule extraction algorithm described in this article works on networks that have been pruned by any network pruning algorithm as well as unpruned networks.

Once a network that predicts the training samples with satisfactory accuracy has been obtained, its hidden unit activation function tanh(x) is approximated by a 3-piece linear function. The next section describes how this approximation is computed.

3. APPROXIMATING HIDDEN UNIT ACTIVATION FUNCTION

Since the hidden unit activation function $h(x)$ = tanh(x) is antisymmetric, it is sufficient to illustrate how the approximation is done just for the nonnegative values of x. The function $h(x)$ can be approximated by a piecewise linear function as follows. Suppose that the input x ranges from 0 to x_m. A simple and convenient approximation of $h(x)$ is to over-estimate it by a piecewise linear function $L(x)$ as shown in Figure 1.

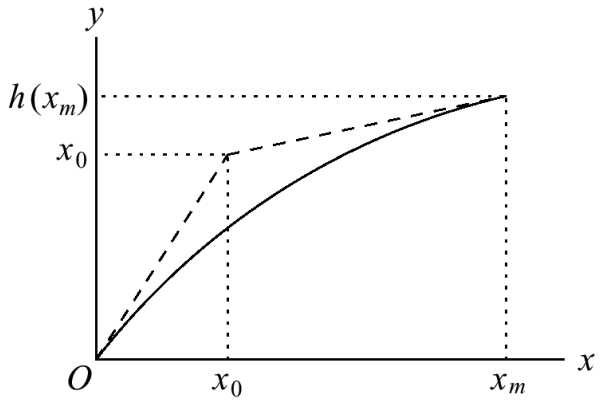

Figure 1. The function tanh(x) (solid curve) for $x \in [0, x_m]$ is approximated by the piecewise linear function $L(x)$ (dashed lines)

To ensure that $L(x)$ is larger than $h(x)$ everywhere between 0 to x_m, the line on the left should intersect the coordinate (0, 0) with a gradient of $h'(0) = 1$, and the line on the right should intersect the coordinate $(x_m, h(x_m))$ with a gradient of $h'(x_m) = 1 - h^2(x_m)$. Thus, $L(x)$ can be written as

$$L(x) = \begin{cases} x & \text{if } 0 \le x \le x_0 \\ h(x_m) - (x_m - x)h'(x_m) & \text{if } x > x_0 \end{cases} \quad (5)$$

The point of intersection x_0 of the two lines is given by

$$x_0 = \frac{h(x_m) - x_m h'(x_m)}{h^2(x_m)} \quad (6)$$

The total error E of estimating $h(x)$ by $L(x)$ is given by

$$E = \int_0^{x_m} (L(x) - h(x))dx$$

$$= \frac{1}{2}[x_0^2 + (x_m - x_0)(x_0 + h(x_m))] - \ln\cosh x_m \tag{7}$$

$$\rightarrow -\frac{1}{2} - \ln 0.5 \quad \text{as} \quad x_m \rightarrow \infty$$

That is, the total error is bounded by a constant value.

4. RULE GENERATION

REFANN generates rules from a neural network as follows:

1. Train and prune a network with one hidden layer and one output unit.
2. For each hidden unit $i = 1, 2, \ldots, H$:
   a. Determine x_{im} from the training samples and compute x_{i0} (Eqn. 6).
   b. Define the 3-piece approximating linear function $L_i(x)$ as:

$$L_i(x) = \begin{cases} -h(x_{im}) + (x_{im} + x)h'(x_{im}) & \text{if } x < -x_{i0} \\ x & \text{if } -x_{i0} \le x \le x_{i0} \\ h(x_{im}) - (x_{im} - x)h'(x_{im}) & \text{if } x > x_{i0} \end{cases}$$

3. Using the pair of points $-x_{i0}$ and x_{i0} of function $L_i(x)$, divide the input space into 3^H subregions.
4. For each non-empty subregion, generate a rule as follows:
   a. Define a linear equation that approximates the network's output for input sample p in this subregion as the *consequence* of the extracted rule:

$$\tilde{y}_p = \sum_{i=1}^{H} v_i L_i(s_{ip}) \tag{8}$$

$$s_{ip} = \sum_{j=1}^{N} w_{ij} I_{jp} \tag{9}$$

   b. Generate the rule *condition*: (C_1 and C_2 and $\cdots$ C_H), where C_i is either $s_{ip} < -x_{i0}$, $-x_{i0} \le s_{ip} \le x_{i0}$, or $s_{ip} > x_{i0}$.
5. (Optional step) Apply C4.5 (Quinlan, 1993) to simplify the rule conditions.

In general, a rule condition C_i is defined in terms of the weighted sum of the inputs s_{ip} (Eqn. 9) which corresponds to an oblique hyperplane in the input space. This kind of rule condition can be difficult for the users to understand. In many cases, the oblique hyperplanes can be replaced by hyperplanes that are parallel to the axes without affecting the prediction accuracy of the rules on the data set. Consequently, the hyperplanes can be defined in terms of only the inputs and are easier to understand. Such replacements of the rule conditions are achieved using C4.5. C4.5 is an induction algorithm which generates decision trees and decision rules for classification problems (Quinlan, 1993).

The following examples of applying REFANN on two different data sets illustrate the working of REFANN in more details. The input attributes of the first data set are continuous, while those of the second data set are mixed, discrete and continuous. Both data sets are publicly available from the University of California, Irvine repository (Blake and Merz, 1998). These two problems are selected because the pruned networks have few hidden units. The networks and extracted rules are also more accurate than other methods reported in the literature.

4.1 Example 1: CPU-Performance

The data set has 6 continuous attributes: (1) MYCT: machine cycle time, (2) MMIN: minimum main memory, (3) MMAX: maximum main memory, (4) CACH: cache memory, (5) CHMIN: minimum channels, and (6) CHMAX: maximum channels. The goal is to predict the CPU's relative performance based on the other computer characteristics (Ein-Dor and Feldmesser, 1987). There are 209 samples in the data set. The samples were randomly divided into a set consisting of 167 samples, a cross-validation set consisting of 21 samples, and a test set consisting of 21 samples. The input values are normalized so that they range in the interval [0,1]. A network with 8 hidden units is trained. After pruning, only one hidden unit remains. The connections from input MYCT and CHMIN are also removed, indicating that these input attributes are irrelevant. The weighted inputs $\sum_j w_{1j} I_{jp}$ for all samples p in the training data set are computed. The largest value among these weighted inputs is assigned as the value of x_m (Step 2(a)) and the value of x_0 is computed according to Eqn. 6. Approximation of the hidden unit activation function separates the samples into 2 groups, those with weighted inputs of less than $x_0 = -0.7354$ and those with weighted inputs greater than or equal to -0.7354. Hence, we approximate the activation function by a piecewise linear function:

$$L_1(s_{1p}) = \begin{cases} -0.5693 + 0.2256\, s_{1p} & \text{if } s_{1p} < -0.7354 \\ s_{1p} & \text{if } s_{1p} \geq -0.7354 \end{cases}$$

Since there is only one hidden unit, the predicted output for pattern p is simply set to $\tilde{y}_p = v_1 L_1(s_{1p})$ (Eqn. 8). After rescaling the inputs back to their original values, we obtain the following set of rules:

Rule set 1:
Rule 1: if Region 1, then $\tilde{y} = Y_1$.
Rule 2: if Region 2, then $\tilde{y} = Y_2$.
The division of the input data is as follows:

- Region 1: $s_{1p} < -0.7354 \Leftrightarrow$ 3.00 MMIN + 2.69 MMAX + 258.10 CACH + 281.53 CHMAX < 111189.41
- Region 2: $s_{1p} \geq -0.7354 \Leftrightarrow$ 3.00 MMIN + 2.69 MMAX + 258.10 CACH + 281.53 CHMAX ≥ 111189.41

and the rule consequences are linear equations Y_1 and Y_2:

$$Y_1 = 4.9616 + 0.0036 \text{ MMIN} + 0.0032 \text{ MMAX} + 0.3086 \text{ CACH} + 0.3366 \text{ CHMAX}$$
$$Y_2 = -453.0270 + 0.0159 \text{ MMIN} + 0.0143 \text{ MMAX} + 1.3662 \text{ CACH} + 1.4903 \text{ CHMAX}$$

The boundary between Region 1 and Region 2 can be approximated by rule conditions from C4.5 (Step 5) which do not involve any network weights. All training samples p with a weighted sum s_{1p} less than -0.7354 are labeled "Region 1", while all others are labeled "Region 2". C4.5 generates the following rules:

Rule set 1a:
Rule 1: if MMAX ≤ 24000 and CACH ≤ 142, then "Region 1"
Rule 2: if MMIN ≤ 2300 and CHMAX ≤ 38, then "Region 2"
Rule 3: if MMAX > 2300, then "Region 2"
Rule 4: if CACH > 142, then "Region 2"
Default Rule: "Region 1"

Table 1. Error Rates for CPU-Performance Data

	RMSE	RRMSE	MAE	RMAE
Pruned network	15.82	15.76	11.52	16.03
Rule set 1	21.52	21.44	13.02	18.11
Rule set 1a	21.52	21.44	13.02	18.11
Linear regression	42.54	42.39	35.44	49.29

Note: RMSE: Root Mean Squared Errors, RRMSE: Relative Root Mean Squared Errors, MAE: Mean Absolute Error, RMAE: Relative Mean Absolute Error.

The error rates of the network and the rule sets are shown in Table 1. In addition to the Root Mean Squared Errors (RMSE), the table also shows the errors of each model in terms of the Relative Root Mean Squared Errors (RRMSE), Mean Absolute Error (MAE), and Relative Mean Absolute Error (RMAE):

$$RMSE = \sqrt{\sum_{p=1}^{P} \frac{(\tilde{y}_p - y_p)^2}{P}}$$

$$RRMSE = 100 \times RMSE / \sqrt{\sum_{p=1}^{P} \frac{(\bar{y} - y_p)^2}{P}}$$

$$MAE = \frac{1}{P} \sum_{p=1}^{P} \left| \tilde{y}_p - y_p \right|$$

$$RMAE = 100 \times MAE / (\frac{1}{P} \sum_{p=1}^{P} \left| \bar{y} - y_p \right|)$$

where $\tilde{y}_p$ and $\bar{y}$ are the predicted value for sample p and the average value of all samples, respectively.

We also fit the data using multiple linear regression for comparison. Using the backward regression option of SAS, all attributes except CHMIN are found to contribute significantly to the regression model with the default confidence level of α = 0.10. This example clearly illustrates the effectiveness of the neural network approach in generating predicting linear equations. Compared to the traditional linear regression approach, the RMSE and MAE of the rules extracted by REFANN are 49% and 63% lower, respectively.

4.2 Example 2: AutoMpg Data Set

The target to be predicted in this problem is the city-cycle fuel consumption of different car models in miles per gallon (Kilpatrick and Cameron-Jones, 1998). The 3 discrete attributes of the data are (1) cylinders with possible values of 3, 4, 5, 6, and 8; (2) model with possible values of 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, and 82; and (3) origin with possible values of 1, 2, and 3. The 4 continuous attributes are (1) displacement, (2) horsepower, (3) weight, and (4) acceleration.

The training data set contains 318 samples, while the cross validation and test sets contain 40 samples each. The binary-coded data required the neural network to have 26 input units. One network input is needed for each possible value of the discrete attributes. The two ordinal discrete attributes cylinders and model are encoded using the thermometer scheme. Using this scheme, the first five network inputs I_1, I_2, I_3, I_4, I_5 are assigned the binary input values of (0,0,0,0,1), (0,0,0,1,1), (0,0,1,1,1), (0,1,1,1,1) and (1,1,1,1,1) if the number of cylinders is 3, 4, 5, 6, or 8, respectively. The attribute model requires 13 network inputs, I_6, ..., I_{18}. The network inputs I_{19}, I_{20}, and I_{21} are used for the nominal discrete attribute origin. Their input values are (0,0,1), (0,1,0) and (1,0,0), if origin of the car is 1, 2, or 3, respectively. The ranges of the 4 continuous attributes are normalized to [0,1] and 4 network input units I_{22}, I_{23}, I_{24}, I_{25} are assigned these normalized input values. Finally, input I_{26} is assigned a constant input value of one for all data samples for the hidden unit bias. With a hidden unit bias, the hyperplane defined by s_{ip} (Eqn. 9) does not necessarily pass through the origin.

One of the smallest pruned networks has only one hidden unit and 7 input units left. The relevant network inputs and their corresponding attributes in the original data set are the following: (1) I_4 = 1, iff cylinders is greater than 3, (2) I_9 = 1, iff model is later than 78, (3) I_{11} = 1, iff model is later than 76, (4) I_{14} = 1, iff model is later than 73, (5) I_{21} = 1, iff origin is 1, (6) I_{23} is the continuous attribute horsepower, and (7) I_{24} is the continuous attribute weight.

After approximating the hidden unit activation function by the piecewise linear function, a rule set consisting of just 2 rules is obtained:

Rule set 2:

Rule 1: if Region 1, then $\tilde{y} = Y_1$.

Rule 2: if Region 2, then $\tilde{y} = Y_2$.

The two subregions of the input space are defined as follows:

- Region 1: $s_{1p} < -0.8873 \Leftrightarrow 0.082\ I_4 + 0.214\ I_9 + 0.0134\ I_{11} + 0.0104\ I_{14} - 0.00164\ I_{21} - 0.003\ I_{23} - 0.0004\ I_{24} < -1.7198$
- Region 2: $s_{1p} \geq -0.8873 \Leftrightarrow 0.082\ I_4 + 0.214\ I_9 + 0.0134\ I_{11} + 0.0104\ I_{14} - 0.00164\ I_{21} - 0.003\ I_{23} - 0.0004\ I_{24} \geq -1.7198$

and the two corresponding linear equations are

$$Y_1 = 16.26 + 0.11\ I_4 + 0.28\ I_9 + 0.18\ I_{11} + 0.14\ I_{14} - 0.11\ I_{21} - 0.005\ I_{23} - 0.001\ I_{24}$$

$$Y_2 = 46.73 + 1.57\ I_4 + 4.06\ I_9 + 2.54\ I_{11} + 1.99\ I_{14} - 1.55\ I_{21} - 0.066\ I_{23} - 0.008\ I_{24}$$

We obtain the following rule set after running C4.5:

Rule set 2a:

Rule 1: if ($I_9 = 0$) and ($I_{23} > 115$) and ($I_{24} > 3432$), then $\tilde{y} = Y_1$.

Rule 2: if ($I_{11} = 0$) and ($I_{24} > 3574$), then $\tilde{y} = Y_1$.

Rule 3: if ($I_{11} = 0$) and ($I_{23} > 130$), then $\tilde{y} = Y_1$.

Rule 4: if ($I_{23} \leq 98$), then $\tilde{y} = Y_2$.

Rule 5: if ($I_{23} \leq 130$) and ($I_{24} \leq 3432$), then $\tilde{y} = Y_2$.

Rule 6: if ($I_{11} = 1$) and ($I_{24} \leq 3432$), then $\tilde{y} = Y_2$.

Rule 7: if ($I_{11} = 1$) and ($I_{23} \leq 115$), then $\tilde{y} = Y_2$.

Rule 8: if ($I_9 = 1$), then $\tilde{y} = Y_2$.

Default rule: $\tilde{y} = Y_2$.

We can rewrite the conditions of Rule set 2a in terms of the original attributes of the data and obtain the following equivalent set of rules:

Rule set 2b:

Rule 1: if (model is 78 or earlier) and (horsepower is greater than 115) and (weight is greater than 3432), then $\tilde{y} = Y_1$.

Rule 2: if (model is 76 or earlier) and (weight is greater than 3574), then $\tilde{y} = Y_1$.

Rule 3: if (model is 76 or earlier) and (horsepower is greater than 130), then $\tilde{y} = Y_1$.

Rule 4: if (horsepower is less than or equal to 98), then $\tilde{y} = Y_2$.

Rule 5: if (horsepower is less than or equal to 130) and (weight is less than or equal to 3432), then $\tilde{y} = Y_2$.

Rule 6: if (model is later than 76) and (weight is less than or equal to 3432), then $\tilde{y} = Y_2$.

Rule 7: if (model is later than 76) and (horsepower is less than or equal to 115), then $\tilde{y} = Y_2$.

Rule 8: if (model is later than 78), then $\tilde{y} = Y_2$.

Default rule: $\tilde{y} = Y_2$.

Table 2. Error Rates for the AutoMpg Data

	RMSE	RRMSE	MAE	RMAE
Pruned network	2.91	35.31	2.03	29.62
Rule set 2	3.00	36.33	2.07	30.10
Rule sets 2a/2b	3.00	36.36	2.07	30.18
Linear regression	3.65	44.25	2.85	41.53

Note. RMSE: Root Mean Squared Errors, RRMSE: Relative Root Mean Squared Errors, MAE: Mean Absolute Error, RMAE: Relative Mean Absolute Error.

Table 2 shows the error rates of the network and the rule sets. The errors of Rule set 2 are higher than those of the network due to the approximation of the hidden unit activation function. One of the samples in the test data set that falls in Region 1 is misclassified by Rule sets 2a/2b. This explains the small difference in the errors of Rule set 2 and those of Rule sets 2a/2b. The results from linear regression are also shown in Table 2 for comparison. The multiple linear regression model has 14 parameters that are significant at $\alpha = 0.10$. Fitting the data with more input attributes, however, does not give a better model as shown by the root mean squared error (RMSE) and mean absolute error of this model (MAE). By using a pruned neural network to divide the input space into two regions and having a linear equation in each of these regions for prediction, the RMSE and MAE are reduced by 18% and 27%, respectively.

5. EXPERIMENTAL RESULTS

The REFANN method has been tested on 10 function approximation problems from various domains. The data sets (see Table 3) are downloaded from http://www.ncc.up.pt/~ltorgo/Research/. These data sets are also available from the UCI repository (Blake and Merz, 1998).

Table 3. Data Sets Used in the Experiments

Name	No. of samples	Attributes	Prediction task
Abalone	4177	1 D, 7 C	age of abalone specimens
Artificial1	5000	10 C	artificially created data set
Artificial2	5000	10 C	artificially created data set
Auto-loss	164	11 D, 14 C	average loss payment per insured vehicle year
Auto-mpg	392	3 D, 4 C	car fuel consumption
Auto-price	159	14 C	car price
CPU-performance	209	6 C	relative CPU performance
Housing	506	1 D, 12 C	median value of homes in Boston suburbs
Servo	167	4 D	response time of a servomechanism
Wpbc	194	32 C	recurrence time of breast cancer

Note: D = discrete attribute, C = continuous attribute.

A ten-fold cross validation evaluation was conducted on each data set. The data were randomly divided into 10 subsets of equal size. Eight subsets were used for network training, one subset for deciding when network pruning should terminate, and one subset for measuring the predictive accuracy of the pruned network and the rules. This procedure was repeated 10 times so that each subset was tested once.

The same experimental settings were used for all the 10 problems. The networks started with 8 hidden units and the penalty parameter ε was set to 0.5. Network pruning was terminated if removal of a hidden unit or an input unit caused the accuracy of the resulting network on the cross-validation samples to drop by more than 5%. The coding scheme for the input data was as follows. One input unit was assigned to each continuous attribute in the data set. The values of the continuous attributes were normalized so that they ranged in the interval [0,1]. Discrete attributes were binary-coded. A discrete attribute with D possible values was assigned D network inputs, except when $D = 2$, where 1 input unit was sufficient.

The results are summarized in Table 4. The average MAEs of the unpruned neural networks (NN), pruned networks (PNN), and the extracted rules (Rules) are shown. The table also shows the average number of hidden units in the pruned networks. Since the hyperbolic tangent activation function is approximated by a 3-piece linear function, a pruned network with H hidden units may generate up to 3^H combinations of linearized activations. The last column of Table 4 shows the number of combinations that are actually represented by the data. This figure corresponds to the average number of linear equations generated to approximate the network output.

Table 4. Experimental Results of REFANN

Data Set	NN		PNN		Rules		Hidden units		No. of rules	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Abalone	1.4802	(0.0598)	1.5184	(0.0759)	1.6208	(0.0770)	5.60	(0.84)	22.40	(7.82)
Artificial1	0.8203	(0.0280)	0.8303	(0.0297)	0.8453	(0.0263)	2.00	(0.00)	2.50	(0.52)
Artificial2	1.0166	(0.0220)	1.0293	(0.0237)	1.2434	(0.0419)	5.00	(0.00)	34.80	(0.42)
Auto-loss	12.560	(3.55)	12.030	(3.50)	13.005	(3.72)	3.90	(0.99)	28.90	(14.07)
Auto-mpg	1.7893	(0.3550)	1.8026	(0.3673)	2.1146	(0.3855)	3.30	(0.67)	10.70	(4.40)
Auto-price	1369.04	(439.7)	1379.19	(395.23)	1581.02	(442.06)	3.40	(1.26)	12.30	(7.04)
CPU-performance	26.869	(9.87)	27.386	(10.98)	30.034	(11.17)	2.00	(1.15)	5.10	(1.73)
Housing	2.2596	(0.2446)	2.2699	(0.2583)	2.5619	(0.3120)	6.80	(1.13)	67.50	(26.09)
Servo	0.2383	(0.0647)	0.2234	(0.0542)	0.2502	(0.0607)	2.90	(1.10)	8.10	(4.58)
Wpbc	24.782	(3.93)	25.518	(4.329)	25.540	(4.284)	1.00	(0.00)	2.30	(0.48)

Note. NN: The average MAE's of unpruned neural networks, PNN: The average MAE's of pruned neural networks, Rules: The average MAE's of extracted rules, Hidden units: The average number of hidden units of the pruned networks, No. of rules: The average number of extracted rules.

The figures in Table 4 suggest that approximation of the hyperbolic tangent activation function by the 3-piece linear function is not appropriate for data sets with highly nonlinear relationship between the inputs and the outputs. The large number of rules and the relatively large difference between the accuracies of the pruned networks and the extracted rules (e.g., Artificial2) indicate such a problem. For these data sets, the networks should be employed when predicting the outputs of new samples.

Comparisons of REFANN with other methods designed to learn continuous target variables are presented in Table 5. The predictive accuracies of three variants of a regression tree generating algorithm called HTL are taken from a paper by Torgo (1997). These methods grow a binary tree by adding nodes to minimize the mean squared errors of the patterns in the leaf nodes. Prediction error is computed as the difference between the actual target value and the average target value of all training samples in the leaf node. Once the tree is fully-grown, predictions are made using different methods. The KR method employs kernel regression with a gaussian kernel function to compute the weights to be assigned to selected samples in a leaf node. The kNN prediction for a sample is computed as the average value of its k nearest neighbors. Each leaf node of the Linear Trees method is associated with a linear regression equation which is used for the prediction for all the samples in the node. Hence, among these three methods, the Linear Trees method is the most similar to REFANN in terms of how the predictions are computed.

Table 5. Comparison of REFANN Performance with Tree Regression Methods

Data Set	REFANN		KR Trees		kNN Trees		Linear Trees	
	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Abalone	1.6	(0.1)	1.7	(0.1)	1.7	(0.1)	1.8	(0.1)
Artificial1	0.8	(0.03)	0.9	(0.0)	0.9	(0.0)	0.8	(0.0)
Artificial2	1.2	(0.04)	1.1	(0.0)	1.1	(0.0)	1.3	(0.1)
Auto-loss	13.0	(3.7)	12.3	(4.5)	13.7	(4.4)	105.2	(34.9)
Auto-mpg	2.1	(0.4)	2.4	(0.4)	2.3	(0.4)	18.0	(5.6)
Auto-price	1581	(442)	1637	(570)	1662	(581)	2463	(518)
CPU-performance	30.0	(11.2)	31.2	(15.1)	31.5	(14.7)	35.7	(11.7)
Housing	2.6	(0.3)	2.8	(0.5)	2.9	(0.4)	3.9	(2.7)
Servo	0.25	(0.06)	0.4	(0.2)	0.4	(0.2)	0.9	(0.2)
Wpbc	25.5	(4.3)	28.5	(5.6)	28.5	(5.6)	28.2	(5.6)

Note: The figures shown are the average MAEs and their standard deviations in parentheses.

From Table 5, we can see that REFANN gives better predictions than the tree regression methods for most of the problems tested. REFANN is as good as or better than Linear Trees for all the problems tested. Large differences in the accuracies of these two methods are found for all problems except Abalone and the two artificial data sets. Compared to KR and kNN, REFANN's error rates are slightly higher on only two of the ten data sets, Artificial2 and Auto-loss.

6. CONCLUSION AND FUTURE WORKS

This article has presented an approach for extracting rules from function approximating neural networks. It is easier to explain the prediction of a data-fitting model in terms of linear equations than the nonlinear mapping of a neural network. Using the weights of a trained network, REFANN attempts to divide the input space of the data into a small number of subregions such that the prediction for the samples in the same subregion can be computed by a single linear equation. REFANN approximates the nonlinear hyperbolic tangent activation function of the hidden units using a simple 3-piece linear function. It then generates rules in the form of linear equations from the trained network. The conditions in these rules divide the input space into one or more subregions. In each subregion, a linear equation approximates the network output. Experiments performed on 10 function approximation problems show that REFANN has very encouraging performance.

By adopting a more sophisticated piecewise linear approximation of the hidden unit activation function, it should be possible to improve REFANN's prediction accuracy. For example, instead of over-estimating the hidden unit activation function, a more accurate approximation would minimize the total absolute error (or total squared errors) between the actual and the approximated activation values.

A second direction for improvement is to reduce the number of extracted rules. In the current implementation, REFANN approximates the activation of a hidden unit independently from those of the other units. A clustering algorithm may be employed to cluster the hidden unit activation values in the H-dimensional space. Experiments will be conducted to see if this approach generates fewer rules than does the current approach.

Finally, we note that neural networks have been known to predict with higher accuracy than other data fitting methods such as classical regression, nearest neighbor methods, and regression tree methods. The results from our experiment show that even with the hidden unit activation functions of the networks approximated by piecewise linear functions, the resulting networks still outperform other methods on most of the problems tested.

7. REFERENCES

Blake, C. L., and Merz, C. J. UCI Repository of Machine Learning Databases, Dept. of Information and Computer Science, University of California, Irvine, 1998, http://www.ics.uci.edu/ ~mlearn/MLRepository.html.

Coakley, J. R., and Brown, C. E. "Artificial Neural Networks Applied to Ratio Analysis in the Analytical Review Process," *Intelligent Systems in Accounting, Finance and Management* (2), 1993, pp. 19-39.

Dutta, S., Shekhar, S., and Wong, W. Y. "Decision Support in Non--conservative Domains: Generalization with Neural Networks," *Decision Support Systems* (11:5), 1994, pp. 527-544.

Ein-Dor, P., and Feldmesser, J. "Attributes of the Performance of Central Processing Units: A Relative Performance Prediction Model," *Communications of the ACM* (30:4), 1987, pp. 308-3177.

Kim, C. N., and McLeod, R. Jr. "Expert, Linear Models, and Nonlinear Models of Expert Decision Making in Bankruptcy Prediction: A Lens Model Analysis," *Journal of Management Information Systems* (16:1), Summer 1999, pp. 189-206.

Hertz, J., Krogh, A., and Palmer, R. G. *Introduction to the Theory of Neural Computation*, Lecture Notes Volume 1, Santa Fe Institute, Studies in the Sciences of Complexity, Addison Wesley Publishing Company, California, 1991.

Kilpatrick, D., and Cameron-Jones, M. "Numeric Prediction Using Instance-based Learning with Encoding Length Selection," *Progress in Connectionist-Based Information Systems,* Springer-Verlag, Singapore, 1998.

Pendharkar, P. C., and Rodger, J. A. "An Empirical Study of Non-binary Genetic Algorithm-based Neural Approaches for Classification," *Proceedings of the Twentieth International Conference on Information Systems*, 1999, pp. 155-165.

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

Salchenberger, L. M., Cinar, E. M., and Lash, N. A. "Neural Networks: A New Tool for Predicting Thrift Failures," *Decision Sciences* (23:4), July/August 1992, pp. 899-916.

Tam, K. Y., and Kiang, M. Y. "Managerial Applications of Neural Networks: The Case of Bank Failure Predictions," *Management Science* (38:7), 1992, pp. 926-948.

Tana, S. S., and Koh, H. C. "A Multi-layer Perceptron Model of Credit Scoring for Assessing Default Risk in Charge Card Applicants," *International Journal of Management* (14:2), 1997, pp. 250-255.

Tickle, A. B., Andrews, R., Golea, M., and Diederich, J. "The Truth Will Come to Light: Directions and Challenges in Extracting the Knowledge Embedded Within Trained Artificial Neural Networks," *IEEE Transactions on Neural Networks* (9:6), 1998, pp. 1057-1068.

Torgo, L. "Functional Models for Regression Tree Leaves," *Proceedings of International Machine Learning Conference*, D. Fisher (Ed), Morgan Kaufmann, San Mateo, CA, 1997.

Trippi, R. R., and Turban, E. *Neural Networks in Finance and Investing*, Probus Publishing Company, Chicago, 1993.

Walczak, S. "Gaining Competitive Advantage for Trading in Emerging Capital Markets with Neural Networks," *Journal of Management Information Systems* (16:2), Fall 1999, pp. 177-192.

Wilson, R. L., and Sharda, R. "Bankruptcy Prediction Using Neural Networks," *Decision Support Systems* (11:5), June 1994, pp. 545-557.