An Approach to Generate Rules from Neural Networks for Regression Problems

Rudy Setiono[1] and James Y.L. Thong[2]

[1] School of Computing

National University of Singapore

SINGAPORE 117543

[2] School of Business and Management

Hong Kong University of Science and Technology

Clear Water Bay, Kowloon

HONG KONG

Submitted to

European Journal of Operational Research

19 April 2001

Please direct all correspondence on this paper to Rudy Setiono.

[Email: rudys@comp.nus.edu.sg; Telephone: (65) 874-6297; Fax: (65) 779-4580]

An Approach to Generate Rules from Neural Networks for Regression Problems

Abstract

Artificial neural networks have been successfully applied to a variety of business application problems involving classification and regression. They are especially useful for regression problems as they do not require prior knowledge about the data distribution. In many 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 regression problems. In order to fill this gap, we propose an approach to extract rules from neural networks that have been trained to solve regression 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 linear regression. The approach is illustrated with three examples on various application problems.

Keywords: Neural networks, nonlinear regression, curve fitting, machine learning, knowledge-based systems.

1. Introduction

Artificial neural networks are powerful tools for business decision making [2, 5, 6, 15, 18, 19, 23, 25]. They work particularly well for problems involving classification and data fitting/regression. Neural networks often predict with higher accuracy than other techniques because of the networks' capability to fit any continuous functions [8]. One major drawback often associated with neural networks is their lack of explanation power. It is difficult to explain how the networks arrive at their solutions due to the complex nonlinear mapping of the input data by 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 at 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 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 normally 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's conditions satisfied by a handful 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 regression problem, on the other hand, has continuous output. Few methods have been devised to extract rules from trained neural networks for regression [20].

In this paper we present an approach to extract rules from neural networks for regression problems. For some applications where one is only interested in obtaining accurate predictions, the trained neural networks will suffice. In other applications, one may want to know more about the relationships between the input variables and the continuous output variable. One feasible approach is to replace the prediction of the trained neural network by a set of multiple linear regression equations without compromising the accuracy of the prediction. The proposed approach works on a network with a single hidden layer and one linear output unit. We impose the restriction on the number of hidden layers to reduce the errors in approximating the hidden unit activation function by a piece-wise linear function as these errors will propagate from the input layer to the output layer through the units in the hidden layers. Experimental evidence indicates that neural networks with just one hidden layer perform as well as those with more than one hidden layer and that the former are less prone to be trapped at a local minimum of the error function while being trained [24].

The hidden unit activation function for our neural networks is the hyperbolic tangent function. This function is used because it can be approximated easily and accurately by piece-wise linear functions. The approach presented in this paper can also be applied to other similarly shaped activation functions such as the sigmoid function. Some existing neural network methods for regression employ the radial basis function. As radial basis function networks allocate a unit to cover a portion of the input space, many units may be required to adequately cover the entire input space. Networks with many units are not good candidates for rule extraction as many rules are required to describe the output accurately. Our approach reduces the number of rules by pruning the redundant network units. The continuous activation function of each hidden unit is then approximated locally 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. Experiments using real-world data sets are performed and the results show that the predictive accuracy of the proposed approach is better than those of existing methods based on decision trees and multiple linear regression.

This paper is organized as follows. The next section describes our neural network training and pruning. In Section 3 we describe how the hidden unit activation function of the network is approximated locally by a 3-piece linear function such that the sum of the squared errors of the approximation is minimized. In Section 4 we present our approach which generates a set of regression rules from a neural network. In Section 5 we present our results and compare them with those from other methods for regression. Three examples illustrate how the decision rules

can be extracted from the pruned networks in Section 6. Finally, in Section 7 we conclude the paper.

2. Network training and pruning

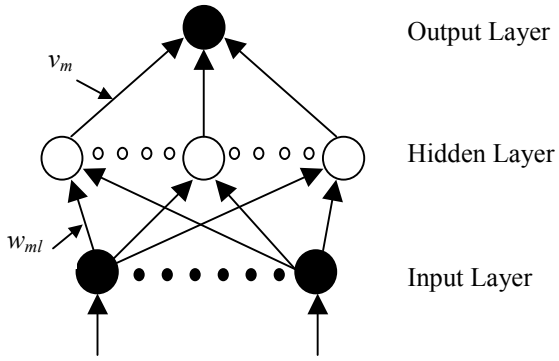

Fig. 1. Fully connected feedforward neural network with one hidden layer.

We divide the available data samples $(\mathbf{x}^i, y^i)$, $i = 1, 2, \ldots$, where $\mathbf{x}^i \in \mathrm{IR}^N$ and $y^i \in \mathrm{IR}$, randomly into a training set, a cross-validation set, and a test set. Using the training data set, a network with a single hidden layer consisting of H units (Figure 1) is trained so as to minimize the sum of squared errors $E(w,v)$ augmented with a penalty term $P(w,v)$:

$$E(w,v) = \sum_{i=1}^{K} (\tilde{y}^i - y^i)^2 + P(w,v) \tag{1}$$

$$P(w, v) = \varepsilon_1 \left(\sum_{m=1}^{H} \sum_{l=1}^{N} \frac{w_{ml}^2}{1 + w_{ml}^2} + \sum_{m=1}^{H} \frac{v_m^2}{1 + v_m^2}\right) + \varepsilon_2 \left(\sum_{m=1}^{H} \sum_{l=1}^{N} w_{ml}^2 + \sum_{m=1}^{H} v_m^2\right) \tag{2}$$

where K is the number of samples in the training data set, and ε_1 and ε_2 are positive penalty parameters. The weight vector $\mathbf{w}_m \in \mathrm{IR}^N$ is the vector of network weights from the input units to hidden unit m, w_{ml} is its l-th component. The weight $v_m \in \mathrm{IR}$ is the network weight from hidden unit m to the output unit. The penalty term $P(w,v)$ is augmented to the sum of squared errors to

provide a measure of complexity of the network [16]. Each nonzero weight of a network increases the penalty.

The predicted value for input sample $\mathbf{x}^i$ is $\tilde{y}^i$ and it is computed as the linear combination of the hidden unit activations:

$$\tilde{y}^i = \sum_{m=1}^{H} \tanh((\mathbf{x}^i)^T \mathbf{w}_m + \varphi_m) v_m + \tau \quad (3)$$

where $\tanh(\xi)$ is the hyperbolic tangent function $(e^{\xi} - e^{-\xi})/(e^{\xi} + e^{-\xi})$, $(\mathbf{x}^i)^T \mathbf{w}_m$ is the scalar product of $\mathbf{x}^i$ and $\mathbf{w}_m$, and φ_m and τ are the hidden units' and the output unit's bias, respectively.

The number of units in the hidden layer H is chosen to be sufficiently large such that the trained network can fit the training samples well. However, the resulting network may overfit the data and possess poor ability to generalize when given new samples. In order to overcome this problem, irrelevant and redundant hidden units and input units must be removed from the network. The algorithm that we employ to prune the network is the N2PFA (Neural Network Pruning for Function Approximation) algorithm [17]. In contrast to network pruning methods which remove one network connection at a time [5, 14], N2PFA iteratively checks the accuracy of the network on the cross-validation samples to determine if a unit can be removed. If the removal of an input or a hidden unit does not cause deterioration in the network's accuracy on these samples, then the unit will be removed and the network retrained. Note however, that the proposed approach works for networks that have been pruned using any pruning algorithm which removes redundant network connections and/or network units.

The benefit of network pruning is two-fold. First, pruned networks are less likely to overfit the training data samples, and hence they can be expected to generalize better than fully connected networks. Second, simplifying the network topology by pruning can be expected to result in a simpler set of extracted rules which are easier to analyze than a more complex one.

We obtain a local minimum of the error function $E(w,v)$ by applying the BFGS technique [4] due to its faster convergence rate than the traditional backpropagation technique. The same optimization technique is also used during the iterative pruning process.

3. Approximating the network activation function

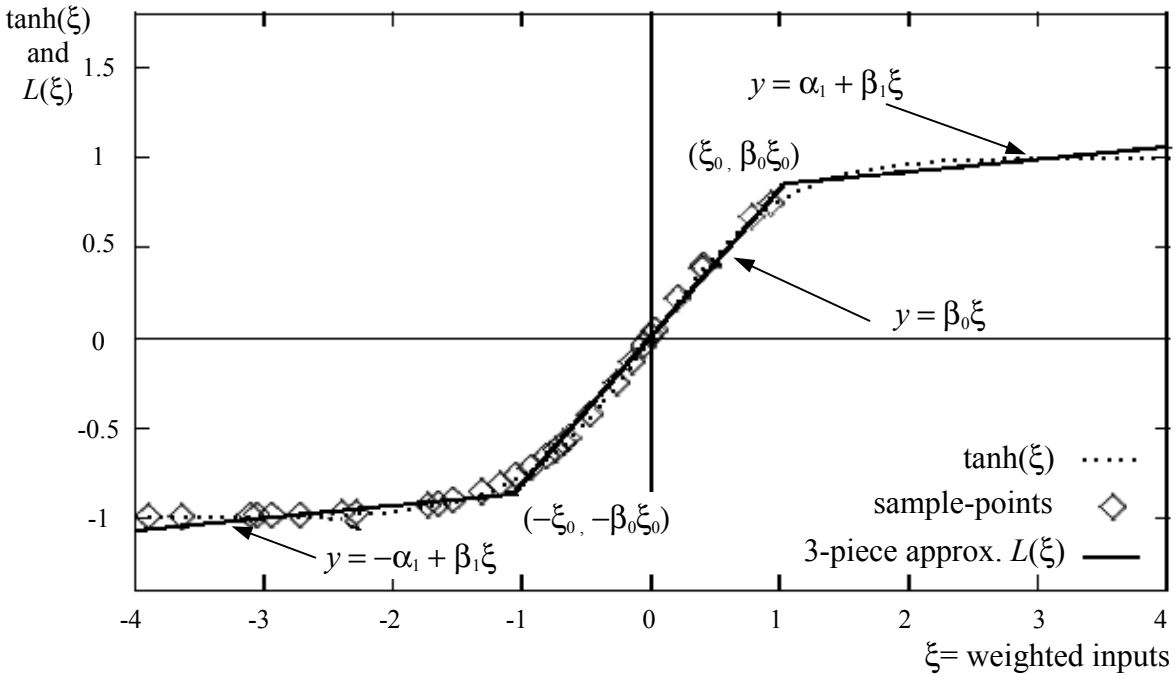

Fig. 2. The 3-piece linear approximation of tanh(ξ) given a set of sample points.

While a trained and pruned neural network can achieve higher accuracy in prediction compared to other regression methods, it is usually difficult to explain or understand how this prediction is reached because of the nonlinearity in the activation function involved in the computation of the network output (Equation 3). We attempt to explain the prediction of the network in terms of rules where the consequences of the rules are linear functions. The critical step in the rule extraction process is a local approximation of the hidden unit activation function by a 3-piece linear function.

The approach which we develop to approximate the hyperbolic tangent activation function entails finding the cut-off point ξ_0, the slope and the intersection of each of the three line segments of the piece-wise linear function $L(\xi)$ (see Figure 2). Since we would like to preserve the predictive accuracy of the network as much as possible when extracting the rules, it is crucial to approximate the activation function accurately by minimizing the sum of the squared errors of the line segments.

We minimize the sum of the squared deviations:

$$\min_{\xi_0, \beta_0, \beta_1} \sum_{i=1}^{K} (\tanh(\xi^i) - L(\xi^i))^2 \tag{4}$$

where $\xi^i = (\mathbf{x}^i)^T \mathbf{w}$, the weighted input of sample i and

$$L(\xi) = \begin{cases} -\alpha_1 + \beta_1 \xi & \text{if } \xi < -\xi_0 \\ \beta_0 \xi & \text{if } -\xi_0 \leq \xi \leq \xi_0 \\ \alpha_1 + \beta_1 \xi & \text{if } \xi > \xi_0 \end{cases} \tag{5}$$

Since tanh(0) = 0, we also require $L(0) = 0$, hence, for the middle line segment we need to determine only its slope β_0. For continuity and symmetry, we require the first line segment to pass through $(-\xi_0, -\beta_0\xi_0)$ and the third line segment $(\xi_0, \beta_0\xi_0)$. The slopes of the line segments that minimize the total sum of the squared deviations (Equation 4) are as follows:

$$\beta_0 = \frac{\sum_{|\xi^i| \le \xi_0} \xi^i \tanh(\xi^i)}{\sum_{|\xi^i| \le \xi_0} (\xi^i)^2} \tag{6}$$

$$\beta_1 = \frac{\sum_{|\xi^i| > \xi_0} (\xi^i - \xi_0)(\tanh(\xi^i) - \tanh(\xi_0))}{\sum_{|\xi^i| > \xi_0} (\xi^i - \xi_0)^2} \tag{7}$$

while the intercept α_1 is given by:

$$\alpha_1 = (\beta_0 - \beta_1)\xi_0 \tag{8}$$

The weighted inputs of all samples ξ^i are checked as a possible candidate for ξ_0 if they are positive and as a candidate for $-\xi_0$ if they are negative. The weighted inputs are first sorted in increasing order of their magnitude and then the search for optimal ξ_0 is conducted starting with the one which has the smallest magnitude.

4. Generating regression rules

A set of linear regression rules can be generated from a pruned network once the network hidden unit activation function $\tanh(\xi)$ has been approximated by the 3-piece linear function as described in the previous section. The steps to generate the decision rules from a pruned neural network are as follows:

1. For each hidden unit $m = 1, 2, ..., H$, generate the 3-piece linear approximation $L_m(\xi)$ (Equation 5).
2. Using the pair of points - ξ_{m0} and ξ_{m0} from function $L_m(\xi)$, divide the input space into 3^H subregions.
3. For each non-empty subregion, generate a rule as follows:
    a. Define a linear equation that approximates the network's output for input sample $\mathbf{x}^i$ in this subregion as the *consequence* of the rule:

$$\hat{y}^i = \sum_{m=1}^{H} v_m L_m(\xi_m^i) + \tau \tag{9}$$

$$\xi_m^i = (\mathbf{x}^i)^T \mathbf{w}_m \tag{10}$$

    b. Generate the rule *condition*: (C_1 and C_2 and $\cdots$ C_H), where C_m is either $\xi_m^i < -\xi_{m0}$, $-\xi_m^i \le \xi_{m0} \le \xi_m^i$, or $\xi_m^i > \xi_{m0}$.
4. (Optional step) Apply C4.5 [13] to simplify the rule conditions.

The rule condition (C_1 and C_2 and ... and C_H) defines intersections of half-spaces in the input space. As the boundaries of the region defined by such intersections involve the network weights, C4.5 [13] may be applied to remove these weights from the rule conditions. The result is a set of rules that can be much easier to understand and analyze as will be illustrated by the examples following the experimental results. C4.5 is a widely used computer package for solving classification problem. It generates decision trees where each of the non-leaf nodes splits the input space into two subspaces based on the values of a single attribute only. When the tree has been built, decision rules can be obtained by tracing all paths from the root to the leaf nodes.

5. Experimental results

The proposed approach has been tested on benchmark approximation problems from various domains. The data sets (see Table 1) are downloaded from Luis Torgo's home page http://www.ncc.up.pt/~ltorgo/Research/. They are also available from the UCI repository [1].

As commonly practiced in machine learning and to allow for a consistent comparison with other existing regression methods, we similarly performed a ten-fold cross validation evaluation 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 the last 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.

Table 1

Test data sets.

Data Set	No. of Samples	Attributes	Network Inputs	Prediction task
Abalone	4177	1D, 7C	9	age of abalone specimens
Auto-mpg	398	3D, 4C	25	car fuel consumption
Housing	506	1D, 12C	13	median value of homes in Boston suburbs
Machine	209	6C	6	relative CPU performance
Servo	167	4D	19	response time of a servo mechanism

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

To test the robustness of the proposed approach, the same experimental setting was used for all problems. The networks started with eight hidden units and the penalty parameters ε_1 and ε_2 were 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 set to drop by more than

10% from its best value. 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 range 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 one input unit was sufficient.

Table 2

Summary of the rules extracted from neural networks.

Data set	MAE	RMAE	No. of Rules	No. of Attributes
Abalone	1.57 ± 0.06	0.16 ± 0.01	4.10 ± 1.45	4.90 ± 2.64
Auto-mpg	1.96 ± 0.32	0.09 ± 0.01	7.50 ± 5.08	9.20 ± 6.07
Housing	2.53 ± 0.46	0.13 ± 0.04	25.30 ± 17.13	9.20 ± 2.70
Machine	20.99 ± 11.38	0.32 ± 0.13	3.00 ± 3.00	4.70 ± 0.67
Servo	0.34 ± 0.08	0.34 ± 0.08	4.70 ± 2.31	10.50 ± 4.45

Note. MAE: mean absolute error, RMAE: relative mean absolute error.

Table 2 summarizes the results of the experiment. The accuracy of the rules on a test data set is measured in terms of the Mean Absolute Error (MAE) and the Relative Mean Absolute Error (RMAE). They are computed as follows:

$$\text{MAE} = \frac{1}{P}\sum_{i=1}^{P} |(\hat{y}^i - y^i)| \tag{11}$$

$$\text{RMAE} = \frac{1}{P}\sum_{i=1}^{P} |(\hat{y}^i - y^i)/y^i| \tag{12}$$

The figures in the table represent the average and the standard deviation from the ten-fold cross validation run. In this table, we also show the average number of rules and the average number of selected attributes.

In Table 3 we compare the performance of our proposed approach to that of other methods for solving regression problems. The predictive accuracy of three variants of a regression tree generating method called HTL [21] are shown under the columns KRTrees, kNNTrees, and LinearTrees. HTL grows a binary regression tree by adding nodes to minimize the mean squared errors of the patterns in the leaf nodes. The prediction error for a training sample is computed as the difference between the actual target value and the average target value of all training samples in the same leaf node. Once the tree is generated, predictions for new data are made using different techniques. 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 is computed as the average values of its k nearest neighbors. Each leaf node of a Linear Tree is associated with a linear regression function which is used for prediction.

Table 3

MAEs of NN rules and other methods for regression problems.

Data Set	NN Rules	KRTrees	kNNTrees	LinearTrees	RUDE
Abalone	1.57 ± 0.06	1.7 ± 0.1	1.7 ± 0.1	1.8 ± 0.1	2.13 ± 0.09
Auto-mpg	1.96 ± 0.32	2.4 ± 0.4	2.3 ± 0.4	18.0 ± 5.6	3.96 ± 0.34
Housing	2.53 ± 0.46	2.8 ± 0.5	2.9 ± 0.4	3.9 ± 2.7	4.07 ± 0.34
Machine	20.99 ± 11.38	31.2 ± 15.1	31.5 ± 14.7	35.7 ± 11.7	51.49 ± 16.25
Servo	0.34 ± 0.08	0.4 ± 0.2	0.4 ± 0.2	0.9 ± 0.2	0.44 ± 0.16

The last column of Table 3 shows the results from RUDE [12]. RUDE (Relative Unsupervised DiscrEtization) method is a recently developed technique for discretization of continuous data. It discretizes not only the continuous target variable, but also all continuous input variables. A key component of this method is a clustering algorithm which groups values

of the target variables into subintervals that are characterized by more or less similar values of some input attributes. Once the variables have been discretized, C4.5 is applied to solve the original regression problem as a classification problem.

As noted by the developers of HTL and RUDE, the difficulty in casting a regression problem as a classification problem lies in the splitting of the continuous target variable. Splitting the interval into many fine subintervals leads to smaller deviations of the predictions within each subinterval. However, many subintervals translate into many classes in the resulting classification problem which would degrade the prediction accuracy of the classification tree. The wrapper approach [9] has been used to overcome this difficulty in HTL. It is an iterative approach that tries varying number of intervals to find one that gives the best-estimated accuracy. In contrast, our neural network approach to regression rule generation does not require the discretization of the continuous target, hence it avoids the difficulties associated with finding the suitable number of intervals.

It is clear that regression rules extracted from neural networks by our approach gives more accurate predictions than the other methods for all the five application problems tested. The decreases in average mean absolute error from the next lowest average error obtained by another method are 7.6% for Abalone, 18.3% for Auto-mpg, 9.6% for Housing, 32.7% for Machine, and 15.0% for Servo. In addition to higher accuracy, our approach also generates fewer rules. The mean size of RUDE trees ranges from 21.5 for Servo to 65.8 for Housing. For these two data sets, the numbers of rules from our neural networks are 4.70 and 25.30, respectively.

6. Illustrative examples

We show in this section how the decision rules can be extracted from pruned neural networks for three application problems, Machine, Auto-mpg, and Servo. The networks are selected because they only have one hidden unit and few input units left after pruning. The problems are chosen because their data sets have different combination of attributes. One data set has only continuous attributes, the second data set has mixed continuous and discrete attributes, while the third data set has only discrete attributes.

For comparison purpose, we also employ SAS [10] to obtain a multiple linear regression equation for the three problems. SAS stepwise regression procedure is used to find the attributes to be included in the regression equation and their coefficients.

Example 1. Machine

The data set has six 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 [7]. There are 209 samples in the data set. The samples were randomly divided into a training data set (167 samples), a cross validation set (21 samples) and a test set (21 samples). The input values are normalized so that they range in the interval [0,1]. After the network pruning has terminated, the cross validation samples are merged with the training samples to form the combined training data set of 188 samples, as the rule extraction algorithm does not require cross validation samples.

We show here how the rules are extracted from one of the ten pruned neural networks obtained during the ten-fold cross validation run. Of the eight hidden units in the original network, only one remains after pruning. The connections from inputs MYCT, CHMIN and CHMAX are also removed, indicating that these input attributes are not useful in predicting the target variable. The weighted inputs ξ^i of all samples i in the training data set are computed. Approximation of the hidden unit activation function separates the samples into three groups, those with weighted inputs of less than ξ_0 = -0.8596, those between -0.8596 and 0.8596 and those with weighted inputs greater than 0.8596. Using the computed optimal values of α_1, β_1 and β_0, the activation function is approximated by the 3-piece linear function:

$$L(\xi) = \begin{cases} -0.5909 + 0.1966\xi & \text{if } \xi < -0.8596 \\ 0.8841\xi & \text{if } -0.8596 \le \xi \le 0.8596 \\ 0.5909 + 0.1966\xi & \text{if } \xi > 0.8596 \end{cases}$$

Since there is only one hidden unit, the predicted output for pattern i is simply set to $\hat{y}^i = vL(\xi^i) + \tau$ (Equation 9), where v is the connection weight from the hidden unit to the output unit and τ is the output unit's bias. We obtain a set of rule consisting of three rules:

Rule set A:

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

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

Rule 3: if Region 3, then $\hat{y} = Y_3$.

In terms of the original (unscaled) attributes of the data, the hyperplanes that divide the input space into the three regions are as follows:

- Region 1: $L(\xi) < -0.8596 \Leftrightarrow 2.056$ MMIN + 3.770 MMAX + 0.003 CACH - 2.044 CHMAX < -0.8596
- Region 2: $-0.8596 \leq L(\xi) \leq 0.8596 \Leftrightarrow -0.8596 \leq 2.056$ MMIN + 3.770 MMAX + 0.003 CACH - 2.044 ≤ 0.8596
- Region 3: $L(\xi) > 0.8596 \Leftrightarrow 2.056$ MMIN + 3.770 MMAX + 0.003 CACH - 2.044 > 0.8596

The coefficients of the separating hyperplanes above are computed from the weights of the network connections from the input units to the hidden unit. The rule consequences are the linear equations Y_1, Y_2 and Y_3 that predict the target value of all samples that fall in the corresponding regions:

$$Y_1 = 14.23 + 0.002 \text{ MMIN} + 0.004 \text{ MMAX} + 0.331 \text{ CACH}$$
$$Y_2 = -388.49 + 0.009 \text{ MMIN} + 0.017 \text{ MMAX} + 1.486 \text{ CACH}$$
$$Y_3 = 598.75 + 0.002 \text{ MMIN} + 0.004 \text{ MMAX} + 0.331 \text{ CACH}$$

The boundaries that separate the three regions can be approximated by rule conditions from C4.5 (Step 4 of the approach) which do not involve any network weights. All training samples with a weighted sum $L(\xi)$ less than -0.8596 are labeled “Region 1”, those between -0.8596 and 0.8596 “Region 2”, while all others are labeled “Region 3”. C4.5 generates the following rules:

Rule set B:

Rule 1: if (MMAX ≤ 20970), then $\hat{y} = Y_1$

Rule 2: if (MMAX ≤ 24000) and (CACH ≤ 48), then $\hat{y} = Y_1$

Rule 3: if (24000 < MMAX ≤ 32000), then $\hat{y} = Y_2$

Rule 4: if (20970 < MMAX ≤ 32000) and (CACH > 48), then $\hat{y} = Y_2$

Rule 5: if (MMAX > 32000), then $\hat{y} = Y_3$

Default Rule: $\hat{y} = Y_1$

Note that the conditions of the rules cover disjoint subspaces of the data. Hence, the accuracy of the rules is not affected by the order in which these conditions are tested. The error rates of the pruned network and the rule sets are shown in Table 4. For the training samples, we also computed the coefficient of determination R^2 to measure the proportional reduction of the total variation by the neural network rules and the regression equation

$$R^2 = \frac{\sum_{i=1}^{P} (\hat{y} - y^i)^2}{\sum_{i=1}^{P} (\bar{y} - y^i)^2}$$

where $\bar{y}$ is the average target value of the samples.

Using the forward regression option of SAS, five of the attributes are found to be significant at the default confidence level. This example clearly illustrates the effectiveness of the neural network approach in generating linear equations for prediction. Compared to the traditional linear regression approach, the MAE and RMAE of the neural network rules on the test samples are 72% and 80% lower, respectively.

Table 4

The average prediction error rates for the Machine data.

	Training Data			Test Data	
Predictor	MAE	RMAE	$100 \times R^2$	MAE	RMAE
Pruned network	13.49	0.21	96.78	8.81	0.12
Rule set A	13.42	0.15	96.55	6.65	0.10
Rule set B	13.42	0.15	96.55	6.65	0.10
Linear regression	33.62	0.68	91.05	23.77	0.50

Example 2. Auto-mpg data set

The target to be predicted in this problem is the city-cycle fuel consumption of different car models in miles per gallon [11]. The three 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 four 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 requires the neural network to have 25 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 (1,0,0), (0,1,0), and (0,0,1), if origin of the car is 1, 2, or 3, respectively. The ranges of the four

continuous attributes are normalized to [0,1] and four network input units I_{22}, I_{23}, I_{24}, I_{25} are assigned these normalized input values.

One of the smallest pruned networks has only one hidden unit and three 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, (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 only two rules is obtained:

Rule set A:

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

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

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

- Region 1: $L(\xi) < 0.7639 \Leftrightarrow -0.357\, I_4 - 0.483\, I_9 + 0.0007\, I_{24} < 1.9079$
- Region 2: $L(\xi) \geq 0.7639 \Leftrightarrow -0.357\, I_4 - 0.483\, I_9 + 0.0007\, I_{24} \geq 1.9079$

and the two corresponding linear equations are:

$$Y_1 = 44.03 + 4.95\, I_4 + 6.70\, I_9 - 0.010\, I_{24}$$

$$Y_2 = 25.18 + 1.42\, I_4 + 1.93\, I_9 - 0.003\, I_{24}$$

We replace the decision boundary between the two regions by running C4.5 and obtain the following set of rules:

Rule set B:

Rule 1: if ($I_9 = 0$) and ($I_{24} > 3190$), then $\hat{y} = Y_2$.

Rule 2: if ($I_{24} > 3870$), then $\hat{y} = Y_2$.

Rule 3: if ($I_{24} \leq 3190$), then $\hat{y} = Y_1$.

Rule 4: if ($I_9 = 1$) and ($I_{24} \leq 3870$), then $\hat{y} = Y_1$.

Default rule: $\hat{y} = Y_1$.

Finally, we can rewrite the conditions of Rule set B in terms of the original attributes of the data and obtain the following equivalent set of rules:

Rule set C:

Rule 1: if (model is 78 or earlier) and (weight is greater than 3190), then $\hat{y} = Y_2$.

Rule 2: if (weight is greater than 3870), then $\hat{y} = Y_2$.

Rule 3: if (weight is less than or equal to 3190), then $\hat{y} = Y_1$.

Rule 4: if (model is later than 78) and (weight is less than or equal to 3870), then $\hat{y} = Y_1$.

Default rule: $\hat{y} = Y_1$.

Table 5 summarizes the error rates of the neural network rules and multiple linear regression for this example. The error rates of the rules on the training data are higher than those achieved by multiple linear regression, while their coefficients of determination R^2 are slightly lower on the training samples. However, the rules can predict samples in the test set with higher accuracy. This example clearly demonstrates that the model that better fits the training samples does not necessarily achieve better accuracy when predicting new samples. The multiple linear regression

model needs many more input attributes, 18, to fit the data compared to only three in the neural network and the extracted rules.

Table 5

The average prediction error rates for the Auto-mpg data.

	Training Data			Test Data	
Predictor	MAE	RMAE	$100 \times R^2$	MAE	RMAE
Pruned network	2.28	0.10	84.56	1.83	0.08
Rule set A	2.32	0.10	84.56	1.88	0.08
Rule sets B/C	2.32	0.10	84.55	1.88	0.08
Linear regression	2.13	0.09	87.16	2.15	0.09

Example 3. Servo

The four attributes of this data set and their possible values are (1) motor: A, B, C, D, E; (2) screw: A, B, C, D, E; (3) pgain: 3, 4, 5, 6; and (4) vgain: 1, 2, 3, 4, 5. The target values range between 0.13 and 7.10. The problem is to predict the response time of a servomechanism in terms of the two gain settings and the two choices of mechanical linkages.

We select one of the 10 networks obtained from the ten-fold cross-validation run to illustrate in details our proposed approach. This pruned network has one hidden unit. The training data set contain 135 samples, while the cross validation and test sets contain 16 samples each. Of the original 19 input units in the original neural network, only eight remain after pruning: (1) $I_1 = 1$ iff motor = A, (2) $I_2 = 1$ iff motor = B, (3) $I_4 = 1$ iff motor = D, (5) $I_6 = 1$ iff screw = A, (7) $I_{13} = 1$ iff pgain ≥ 4.

The value of ξ_0 (Equation 4) for this one hidden unit is 0.9252, while the values of β_0, α_1, and β_1 are 0.8695, 0.6914, and 0.1222, respectively. Hence, the hyperbolic tangent activation function is approximated by the 3-piece linear function:

$$L(\xi) = \begin{cases} -0.6914 + 0.1222\xi & \text{if } \xi < -0.9252 \\ 0.8695\xi & \text{if } -0.9252 \le \xi \le 0.9252 \\ 0.6914 + 0.1222\xi & \text{if } \xi > 0.9252 \end{cases}$$

The predicted output for sample i is simply set to $vL(\xi^i) + \tau$, where ξ^i is the weighted input:

$$\xi^i = (\mathbf{x}^i)^T \mathbf{w} = -0.609\ I_1 - 0.495\ I_2 + 0.952\ I_4 - 0.966\ I_6 + 2.385\ I_{13} - 2.466$$

and the output unit's bias $\tau = -2.466$. Upon substituting ξ^i into $L(\xi^i)$ and simplifying the resulting equation, we obtain the following set of rules:

Rule set A:

Rule 1: if $-0.609\ I_1 - 0.495\ I_2 + 0.952\ I_4 - 0.966\ I_6 + 2.385\ I_{13} < -0.9252$, then $\hat{y} = Y_1$

Rule 2: if $-0.9252 \le -0.609\ I_1 - 0.495\ I_2 + 0.952\ I_4 - 0.966\ I_6 + 2.385\ I_{13} \le 0.9252$, then $\hat{y} = Y_2$

Rule 3: if $-0.609\ I_1 - 0.495\ I_2 + 0.952\ I_4 - 0.966\ I_6 + 2.385\ I_{13} > 0.9252$, then $\hat{y} = Y_3$

where

$$Y_1 = 4.59 + 0.18\ I_1 + 0.15\ I_2 - 0.29\ I_4 + 0.29\ I_6 - 0.72\ I_{13}$$

$$Y_2 = 2.88 + 1.31\ I_1 + 1.06\ I_2 - 2.04\ I_4 + 2.07\ I_6 - 5.12\ I_{13}$$

$$Y_3 = 1.18 + 0.18\ I_1 + 0.15\ I_2 - 0.29\ I_4 + 0.29\ I_6 - 0.72\ I_{13}$$

We replace the conditions of the three rules in Rule set A by applying C4.5 and obtain:

Rule set B:

Rule 1: if ($I_4 = 0$) and ($I_6 = 0$) and ($I_{13} = 0$), then $\hat{y} = Y_1$

Rule 2: if ($I_2 = 1$) and ($I_6 = 1$) and ($I_{13} = 1$), then $\hat{y} = Y_1$

Rule 3: if ($I_1 = 1$) and ($I_6 = 1$) and ($I_{13} = 1$), then $\hat{y} = Y_1$

Rule 4: if ($I_4 = 0$) and ($I_6 = 1$) and ($I_{13} = 0$), then $\hat{y} = Y_2$

Rule 5: if ($I_6 = 0$) and ($I_{13} = 0$), then $\hat{y} = Y_3$

Rule 6: if ($I_1 = 0$) and ($I_2 = 0$) and ($I_{13} = 1$), then $\hat{y} = Y_3$

Rule 7: if ($I_4 = 1$) and ($I_6 = 0$), then $\hat{y} = Y_3$

Default Rule: $\hat{y} = Y_3$.

As the final step, we substitute the conditions of the above rules in terms of the original attributes of the data to obtain:

Rule set C:

Rule 1: if (motor is not D) and (screw is not A) and (pgain is 3), then $\hat{y} = Y_1$

Rule 2: if (motor is B) and (screw is A) and (pgain is greater than 3), then $\hat{y} = Y_1$

Rule 3: if (motor is A) and (screw is A) and (pgain is greater than 3), then $\hat{y} = Y_1$

Rule 4: if (motor is not D) and (screw is A) and (pgain is 3), then $\hat{y} = Y_2$

Rule 5: if (screw is not A) and (pgain is 3), then $\hat{y} = Y_3$

Rule 6: if (motor is not A) and (motor is not B) and (pgain is greater than 3), then $\hat{y} = Y_3$

Rule 7: if (motor is D) and (screw is not A), then $\hat{y} = Y_3$

Default Rule: $\hat{y} = Y_3$

Table 6

The average prediction error rates for the Servo data.

	Training Data			Test Data	
Predictor	MAE	RMAE	$100 \times R^2$	MAE	RMAE
Pruned network	0.37	0.39	85.27	0.33	0.35
Rule set A	0.40	0.43	84.40	0.30	0.32
Rule sets B/C	0.40	0.43	84.40	0.30	0.32
Linear regression	0.57	0.64	76.82	0.68	0.84

The average prediction error rates for the rule sets A, B, and C, as well as the pruned neural network from which these rules have been extracted are summarized in Table 6. For this particular example, the improvement in accuracy of the neural network and the rules extracted from this network over that of the traditional multiple regression method is very significant. In terms of the mean absolute error, the average error in prediction is reduced by 56%. In terms of the relative mean absolute error, the reduction in average error is 62%.

7. Conclusion

We have presented an approach that generates a set of linear equations from a neural network that has been trained and pruned for application problems involving regression. Linear equations that provide predictions of the continuous target values of data samples are obtained by locally approximating each hidden unit activation function by a 3-piece linear function. An approximating piece-wise linear function is computed for each hidden unit such that it minimizes the sum of squared deviations between the actual activation values of the training samples and their approximated values.

We have also evaluated the accuracy rates of the rules extracted by our approach by comparing them with other methods used for regression on five real world problems. Using the mean absolute error and the relative mean absolute error as the performance measures, we show that the rules generated from the neural networks achieve higher accuracy than those from the other methods for regression which first discretize the continuous target values and then build classification trees. Compared to the traditional statistical approach of multiple linear regression, our approach is also shown to be superior.

By converting the nonlinear mapping of a neural network into a set of linear regression equations, the approach derives symbolic rules to provide some explanations of how the predictions are obtained. As a result, better understanding about the application problem being solved can be expected.

References

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

[2] Coakley, J.R., and Brown, C.E. (1993). Artificial neural networks applied to ratio analysis in the analytical review process. *Intelligent Systems in Accounting, Finance and Management*, *2*, 19-39.

[3] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*, 303-314.

[4] Dennis, J.E.Jr., and Schnabel, R.E. (1983). *Numerical methods for unconstrained optimization and nonlinear equations*. Englewood Cliffs, New Jersey: Prentice Halls.

[5] Desai, V.S., and Bharati, R. (1998). The efficacy of neural networks in predicting returns on stock and bond indices. *Decision Sciences*, *29*(2), 527-544.

[6] Dutta, S., Shekhar, S., and Wong, W.Y. (1994). Decision support in non--conservative domains: Generalization with neural networks. *Decision Support Systems*, *11*(5), 527-544.

[7] Ein-Dor, P., and Feldmesser, J. (1987). Attributes of the performance of central processing units: A relative performance prediction model. *Communications of the ACM*, *30*(4), 308-317.

[8] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*, 251-257.

[9] John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proc. of the 11th International Conference on Machine Learning*, Morgan Kaufman, San Mateo, 121-129.

[10] Khattree, W., and Naik, D.N. (1999). *Applied multivariate statistics with SAS software*. Carey, NC: SAS Institute.

[11] Kilpatrick, D., and Cameron-Jones, M. (1998). Numeric prediction using instance-based learning with encoding length selection. *Progress in Connectionist-Based Information Systems,* Singapore: Springer-Verlag.

[12] Ludl, M-C., and Widmer, G. (2000). Relative unsupervised discretization for regression problems. In *Proc. of the 11th European Conference on Machine Learning, ECML 2000*, Lecture Notes in AI 1810, Springer, R.A. Mantaras and E. Plaza (Eds.), Barcelona, 246-253.

[13] Quinlan, R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufman.

[14] Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructures of cognition*, Vol. 1. D.E. Rumelhart and J.L. McCleland (Eds.), Cambridge, MA: The MIT Press, 318-362.

[15] Salchenberger, L.M., Cinar, E.M., and Lash, N.A. (1992). Neural networks: A new tool for predicting thrift failures. *Decision Sciences*, *23*(4), 899-916.

[16] Setiono, R. (1997). A penalty function approach for pruning feedforward neural networks, *Neural Computation*, *9*(1), 185-204.

[17] Setiono, R., and Leow, W.K. (2000). Pruned neural networks for regression. In *Proc. of the 6th Pacific Rim Conference on Artificial Intelligence, PRICAI 2000*, Lecture Notes in AI 1886, Springer, R. Mizoguchi and J. Slaney (Eds.), Melbourne, 500-509.

[18] Tam, K.Y., and Kiang, M.Y. (1992). Managerial applications of neural networks: The case of bank failure predictions. *Management Science*, *38*(7), 926-948.

[19] Tana, S.S., and Koh, H.C. (1992). A multi-layer perceptron model of credit scoring for assessing default risk in charge card applicants. *International Journal of Management*, *14*(2), 250-255.

[20] Tickle, A.B., Andrews, R., Golea, M., and Diederich, J. (1998). 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), 1057-1068.

[21] Torgo, L. (1997). Functional models for regression tree leaves. In *Proc. of the International Conference on Machine Learning, ICML-97*, Fisher, D. (Ed.), San Mateo, CA: Morgan Kaufman.

[22] Torgo, L., and Gama, J. (1997). Search-based class discretization. In *Proc. of the 9th European Conference on Machine Learning, ECML-97*, Lecture Notes in AI 1224, Springer, M. van Someren and G. Widmer (Eds.), Prague, 266-273.

[23] Trippi, R.R., and Turban, E., (1993). Eds. *Neural Networks in Finance and Investing*. Chicago: Probus Publishing Company.

[24] Villiers, J. and Barnard, E. (1993). Backpropagation neural nets with one and two hidden layers. *IEEE Transactions on Neural Networks*, *4*(1), 136-141.

[25] Wilson, R.L., and Sharda, R. (1994). Bankruptcy prediction using neural networks. *Decision Support Systems*, *11*(5), 545-557.