Extraction of Rules from Artificial Neural Networks for Nonlinear Regression

Rudy Setiono
School of Computing
National University of Singapore
Kent Ridge, Singapore 119260

Wee Kheng Leow
School of Computing
National University of Singapore
Kent Ridge, Singapore 119260

Jacek M. Zurada*
Dept. of Electrical and Computer Engineering
University of Louisville
Louisville, KY 40208, USA

Abstract

Neural networks have been successfully applied to solve a variety of application problems including classification and function approximation. They are especially useful as function approximators because they do not require prior knowledge of the input data distribution and they have been shown to be universal approximators. In many applications, it is desirable to extract knowledge that can explain how the problems are solved by the networks. Most existing approaches have focused on extracting symbolic rules for classification. Few methods have been devised to extract rules from trained neural networks for regression. This article presents an approach for extracting rules from trained neural networks for regression. Each rule in the extracted rule set corresponds to a subregion of the input space and a linear function involving the relevant input attributes of the data approximates the network output for all data samples in this subregion. Extensive experimental results on 32 benchmark data sets demonstrate the effectiveness of the proposed approach in generating accurate regression rules.

Index terms: Regression, network pruning, rule extraction.

*This author's work was sponsored in part by the Department of the Navy, Office of Naval Research, Grant N00014-98-1-0568. The content of this information does not necessarily reflect the position of the Government.

1 Introduction

The last two decades have seen a growing number of researchers and practitioners applying neural networks to solve a variety of problems such as pattern classification and function approximation. For classification problems, the outputs, and often the inputs as well, are discrete. On the other hand, function approximation or regression problems have continuous inputs and outputs, and the function or regression may be nonlinear. In many applications, it is desirable to extract knowledge from trained neural networks for the users to gain a better understanding of how the networks solve the problems. Most existing research works have focused on extracting symbolic rules for solving classification problems [1]. Few methods have been devised to extract rules from trained neural networks for regression [2].

The rules generated from neural networks should be simple enough for human users to understand. Rules for function approximation normally take the form: *if (condition is satisfied), then predict* $y = f(x)$, where $f(x)$ is either a constant or a linear function of x, the attributes of the data. This type of rules is acceptable because of their similarity to the traditional statistical approach of parametric regression.

A single rule cannot approximate the nonlinear mapping performed by 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, a large number of rules, each applicable to only a small number of samples, do not provide meaningful or useful knowledge to the users. Hence, a balance must be achieved between rule accuracy and rule simplicity.

This paper describes a method called REFANN (Rule Extraction from Function Approximating Neural Networks) for extracting rules from trained neural networks for nonlinear function approximation or regression. It is shown that REFANN produces rules that are almost as accurate as the original 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, redundant hidden units and irrelevant input attributes are first removed by a pruning method called N2PFA (Neural Network Pruning for Function Approximation) before REFANN is applied. The continuous activation function of the hidden unit is then approximated by either a 3-piece or a 5-piece linear function. The various combinations of the domains of the piecewise linear functions divide the input space into subregions such that the function values for all inputs in the same subregion can be computed as a linear function of the inputs.

The paper is organized as follows: Section 2 presents related works for function approximation, pruning, and rule extraction. Sections 3 to 5 describe our algorithms, namely N2PFA, approximation of nonlinear activation function, and REFANN. Two illustrative examples that show how the algorithms work in detail are presented in Section 6. Extensive experiments have been performed which show the effectiveness of the proposed method for function approximation. The results from these experiments are presented in Section 7. Finally, Section 8 concludes the paper with a discussion on the interpretability of the extracted rules and a summary of our contributions.

2 Related Works and Motivation

Existing neural network methods for function approximation typically employ the Radial Basis Function (RBF) networks [3] or combinations of RBF networks and other methods [4]. A disadvantage of RBF networks is that they typically allocate a unit to cover a portion of the input space. As a result, many units are required to adequately cover the entire input space, especially for a high-dimensional input space with complex distribution patterns. Networks with too many hidden units are not suitable for rule extraction because many rules would be needed to express their outputs. In contrast, our method adopts the multilayer neural network with one hidden layer, which has been shown to be a universal function approximator.

The number of rules extracted from such a neural network increases with increasing number of hidden units in the network. To balance rule accuracy and rule simplicity (as discussed in Section 1), an appropriate number of hidden units must be determined, and two general approaches have been proposed in the literature. The constructive algorithms start with a few hidden units and add more units as needed to improve network accuracy [5, 6, 7]. The destructive algorithms, on the other hand, start with a large number of hidden units and remove those that are found to be redundant [8]. The number of useful input units corresponds to the number of relevant input attributes of the data. Typical algorithms usually start by assigning one input unit to each attribute, train the network with all input attributes and then remove network input units that correspond to irrelevant data attributes [9, 10]. Various measures of the contribution of an input attribute to the network's predictive accuracy have been proposed [11, 12, 13, 14, 15]. We have opted for the destructive approach since in addition to producing a network with the fewest hidden units, we also wish to remove as many redundant and irrelevant input units as possible.

Most existing published reports have focused on extracting symbolic rules for solving classification problems. For example, MofN algorithm [16] and GDS algorithm [17] extract MofN rules; BRAINNE [18], Bio-RE, Partial-RE, Full-RE [19], RX [20], NeuroRule [21] and GLARE [22] generate DNF (disjunctive normal form) rules; and FERNN [23] extracts either MofN or DNF rules depending on which kind of rules is more appropriate.

On the other hand, few methods have been devised to extract rules from trained neural networks for regression. ANN-DT [24] is one such algorithm which is capable of extracting rules from function approximating networks. The algorithm regards neural networks as "black boxes". It produces decision trees based on the networks' inputs and the corresponding outputs without analyzing the hidden units' activation values and the connection weights of the networks. In contrast, our REFANN method derives rules from minimum-size networks. It prunes the units from the original networks and extracts linear rules by approximating the hidden unit activation functions by piece-wise linear functions.

3 Network Training and Pruning Algorithm

In this section we describe the N2PFA training and pruning algorithm. The available data samples $(\mathbf{I}_p, y_p), p = 1, 2, \ldots, K$ where input $\mathbf{I}_p \in \mathbb{R}^N$ and target $y_p \in \mathbb{R}$, are first randomly divided into 3 subsets: the training, the cross-validation and the test sets. Using the training data set, a network with H hidden units is trained, so as to minimize the sum of squared errors $E(\mathbf{w}, \mathbf{v})$ augmented with a penalty term $\theta(\mathbf{w}, \mathbf{v})$:

$$E(\mathbf{w}, \mathbf{v}) = \sum_p (\tilde{y}_p - y_p)^2 + \theta(\mathbf{w}, \mathbf{v}) \tag{1}$$

$$\theta(\mathbf{w}, \mathbf{v}) = \epsilon_1 \left(\sum_{i=1}^{H} \sum_{j=1}^{N} \frac{\beta w_{ij}^2}{1 + \beta w_{ij}^2} + \sum_{i=1}^{H} \frac{\beta v_i^2}{1 + \beta v_i^2} \right) + \epsilon_2 \left(\sum_{i=1}^{H} \sum_{j=1}^{N} w_{ij}^2 + \sum_{i=1}^{H} v_i^2 \right) \tag{2}$$

where $\epsilon_1, \epsilon_2, \beta$ are positive penalty parameters, w_{ij} is the weight of the connections from input unit j to hidden unit i and v_i is the weight of the connection from hidden unit i to the output unit. The penalty term $\theta(\mathbf{w}, \mathbf{v})$ when minimized pushes the weight values towards the origin of the weight space, and in practice results in many final weights taking values near or at 0. Network connections with such weights may be removed from the network without sacrificing the network accuracy [25].

The hidden unit activation value A_{ip} for input $\mathbf{I}_p$ and its predicted function value $\tilde{y}_p$ are computed as follows:

$$A_{ip} = h(s_{ip}) = h\left(\sum_{j=1}^{N} w_{ij} I_{jp} \right) \tag{3}$$

$$\tilde{y}_p = \sum_{i=1}^{H} v_i A_{ip}, \tag{4}$$

I_{jp} is the value of input j for pattern p. The function $h(x)$ is the hidden unit activation function. This function is normally the sigmoid function or the hyperbolic tangent function. We have used the hyperbolic tangent function $\tanh(\xi) = (e^{\xi} - e^{-\xi})/(e^{\xi} + e^{-\xi})$.

A local minimum of the error function $E(\mathbf{w}, \mathbf{v})$ can be obtained by applying any nonlinear optimization methods such as the gradient descent method or the quasi-Newton method. In our implementation, we have used a variant of the quasi-Newton method, namely the BFGS method [26] due to its faster convergence rate than the gradient descent method. The BFGS method with line search also ensures that after every iteration of the method, the error function value will decrease. This is a property of the method that is not possessed by the standard backpropagation method with a fixed learning rate.

Once the network has been trained, its hidden and input units are inspected as candidates for possible removal by a network pruning algorithm. A pruning algorithm called N2PFA (Neural Network Pruning for Function Approximation) has been developed. This algorithm removes redundant and irrelevant units by computing the mean absolute error (MAE) of the network's prediction. In particular, ET and EX, respectively the MAE's on the training set $\mathcal{T}$ and the cross-validation set $\mathcal{X}$, are used to determine when pruning should be terminated:

$$ET = \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{I}_p, y_p) \in \mathcal{T}} |\tilde{y}_p - y_p| \qquad EX = \frac{1}{|\mathcal{X}|} \sum_{(\mathbf{I}_p, y_p) \in \mathcal{X}} |\tilde{y}_p - y_p| \; . \tag{5}$$

where $|\mathcal{T}|$ and $|\mathcal{X}|$ are the cardinality of the training and cross-validation sets, respectively.

Algorithm N2PFA

Given: Data set $(\mathbf{I}_p, y_p), p = 1, 2, \ldots, K$.

Objective: Find a neural network with reduced number of hidden and input units that fits the data and generalizes well.

Step 1. Split the data into 3 subsets: training, cross-validation, and test sets.

Step 2. Train a network with a sufficiently large number of hidden units to minimize the error function (1).

Step 3. Compute ET and EX, and set $ETbest = ET$, $EXbest = EX$, $Emax = \max\{ETbest, EXbest\}$.

Step 4. Remove redundant hidden units:

1. For each $i = 1, 2, \ldots, H$,
   set $v_i = 0$ and compute the prediction errors ET_i.
2. Retrain the network with $v_h = 0$ where $ET_h = \min_i ET_i$, and compute ET and EX of the retrained network.
3. If $ET \leq (1 + \alpha) Emax$ and $EX \leq (1 + \alpha) Emax$, then
   - Remove hidden unit h.
   - Set $ETbest = \min\{ET, ETbest\}$, $EXbest = \min\{EX, EXbest\}$ and $Emax = \max\{ETbest, EXbest\}$.
   - Set $H = H - 1$ and go to Step 4.1.

   Else use the previous setting of network weights.

Step 5. Remove irrelevant inputs:

1. For each $j = 1, 2, \ldots, N$,
   set $w_{ij} = 0$ for all i and compute the prediction errors ET_j.
2. Retrain the network with $w_{in} = 0$ for all i where $ET_n = \min_j ET_j$, and compute ET and EX of the retrained network.
3. If $ET \leq (1 + \alpha) Emax$ and $EX \leq (1 + \alpha) Emax$, then

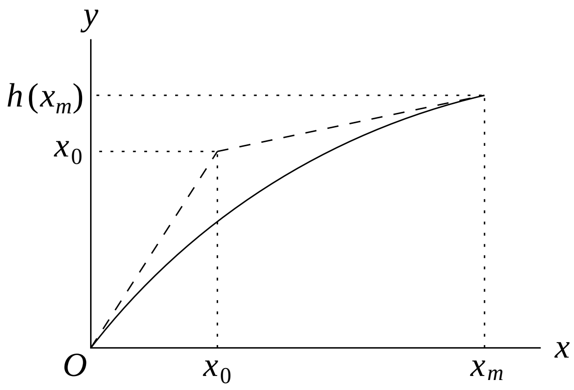

Figure 1: The $\tanh(x)$ function (solid curve) for $x \in [0, x_m]$ is approximated by a 2-piece linear function (dashed lines).

- Remove input unit n.
- Set $ETbest = \min\{ET, ETbest\}$, $EXbest = \min\{EX, EXbest\}$ and $Emax = \max\{ETbest, EXbest\}$.
- Set $N = N - 1$ and go to Step 5.1.

Else use the previous setting of network weights.

Step 6. Report the accuracy of the network on the test data set.

The value of $Emax$ is used to determine if a network unit can be removed. Typically, at the beginning of the algorithm when there are many hidden units in the network, the training mean absolute error ET will be much smaller than the cross-validation mean absolute error EX. The value of ET increases as more and more units are removed. As the network approaches its optimal structure, we expect EX to decrease. As a result, if only $ETbest$ is used to determine whether a unit can be removed, many redundant units can be expected to remain in the network when the algorithm terminates because $ETbest$ tends to be small initially. On the other hand, if only $EXbest$ is used, then the network would perform well on the cross-validation set but may not necessarily generalize well on the test set. This could be caused by the small number of samples available for cross-validation or an uneven distribution of the data in the training and cross-validation sets. Therefore, $Emax$ is assigned the larger of $ETbest$ and $EXbest$ so as to remove as many redundant units as possible without sacrificing the generalization accuracy. The parameter $\alpha > 0$ is introduced to control the chances that a unit will be removed. With a larger value of α, more units can be removed. However, the accuracy of the resulting network on the test data set may deteriorate. We have conducted extensive experiments to find a value for this parameter that works well for the majority of our test problems.

4 Approximating Hidden Unit Activation Function

Having produced the pruned networks, we can now proceed to extract rules that explains the network outputs as a collection of linear functions. The first step in our rule extraction method is to approximate the hidden unit activation function. We approximate the activation function $h(x) = \tanh(x)$ by either a 3-piece linear function or a 5-piece linear function.

4.1 3-piece Linear Approximation

Since $h(x)$ is antisymmetric, it is sufficient to illustrate the approximation just for the nonnegative values of x. 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 the piecewise linear function $L(x)$ as shown in Fig. 1. To ensure

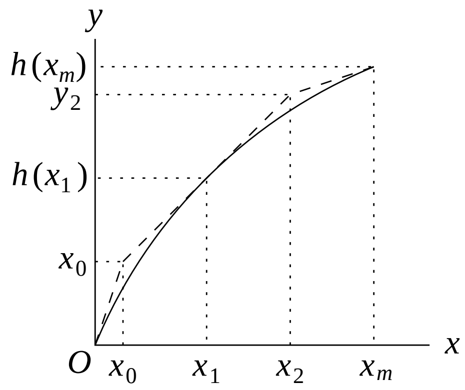

Figure 2: The $\tanh(x)$ function (solid curve) for $x \in [0, x_m]$ is approximated by a 3-piece linear function (dashed lines).

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

$$L(x) = \begin{cases} x & \text{if } 0 \le x \le x_0 \\ h'(x_m)(x - x_m) + h(x_m) & \text{if } x > x_0 \end{cases} . \tag{6}$$

The point of intersection x_0 of the two line segments is given by

$$x_0 = \frac{h(x_m) - x_m h'(x_m)}{h^2(x_m)} . \tag{7}$$

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

$$\begin{aligned} E_A &= \int_0^{x_m} (L(x) - h(x))\, dx \\ &= \frac{1}{2}[x_0^2 + (x_m - x_0)(x_0 + h(x_m))] - \ln\cosh x_m \\ &\to -\frac{1}{2} - \ln 0.5 \quad \text{as } x_m \to \infty . \end{aligned} \tag{8}$$

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

Another simple linearization method of approximating $h(x)$ is to under-estimate it by a 3-piece linear function. It can be shown that the total error of the under-estimation method is unbounded and is larger than that of the over-estimation method for $x_m > 2.96$.

4.2 5-piece Linear Approximation

By having more line segments, the function $h(x)$ can be approximated with better accuracy. Figure 2 shows how this function can be approximated by a 3-piece linear function for $x \in [0, x_m]$. The three dotted lines are given by the equations:

$$L(x) = \begin{cases} x & \text{if } 0 \le x \le x_0 \\ h'(x_1)(x - x_1) + h(x_1) & \text{if } x_0 < x < x_2 \\ h'(x_m)(x - x_m) + h(x_m) & \text{if } x \ge x_2 \end{cases} . \tag{9}$$

The underlying idea for this approximation is to find the point x_1 that minimizes the total area A of the triangle and the two trapezoids:

$$A = \frac{1}{2}\left[x_0^2 + (x_0 + y_2)(x_2 - x_0) + (y_2 + h(x_m))(x_m - x_2)\right] \tag{10}$$

x_0, x_2, and y_2 are expressed in terms of a constant x_m and the free parameter x_1:

$$x_0 = \frac{h(x_1) - x_1 h'(x_1)}{1 - h'(x_1)} \tag{11}$$

$$x_2 = \frac{x_1 h'(x_1) - x_m h'(x_m) - h(x_1) + h(x_m)}{h'(x_1) - h'(x_m)} \tag{12}$$

$$y_2 = \frac{h'(x_1)h'(x_m)(x_1 - x_m) + h'(x_1)h(x_m) - h'(x_m)h(x_1)}{h'(x_1) - h'(x_m)} \tag{13}$$

The bisection method [27] for one-dimensional optimization problems is employed to find the optimal value of x_1. The total error E_A of estimating $h(x)$ by this linear approximation is computed to be

$$\begin{aligned} E_A &= \int_0^{x_m} (L(x) - h(x))\, dx \\ &\to 0.071169 \quad \text{as } x_m \to \infty\ . \end{aligned} \tag{14}$$

5 Rule Generation

REFANN generates rules from a pruned neural network according to the following steps:

Algorithm REFANN

Given: Data set $(\mathbf{I}_p, y_p), p = 1, 2, \ldots, K$ and a pruned network with H hidden units.
Objective: Generate linear regression rules from the network.

Step 1. Train and prune a network with one hidden layer and one output unit.

Step 2. For each hidden unit $i = 1, 2, \ldots, H$:

1. Determine x_{im} from the training samples.
2. If the 3-piece linear approximation is used:
   - Compute x_{i0} (Eqn. 7).
   - Define the 3-piece approximating linear function $L_i(x)$ as:

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

   - Using the pair of points $-x_{i0}$ and x_{i0} of function $L_i(x)$, divide the input space into 3^H subregions.
3. Else if the 5-piece linear approximation is used:
   - Find x_{i1} using the bisection method and compute x_{i0} and x_{i2} according to Eqns. 11 and 12.
   - Define the 5-piece approximating linear function $L_i(x)$ as:

$$L_i(x) = \begin{cases} h'(x_{im})(x - x_{im}) + h(x_m) - h(x_{im}) & \text{if } x \le -x_{i2} \\ h'(x_{i1})(x - x_{i1}) + h(x_1) - h(x_{i1}) & \text{if } -x_{i2} < x < -x_{i0} \\ x & \text{if } -x_{i0} \le x \le x_{i0} \\ h'(x_{i1})(x - x_{i1}) + h(x_1) + h(x_{i1}) & \text{if } x_{i0} < x < x_{i2} \\ h'(x_{im})(x - x_{im}) + h(x_m) + h(x_{im}) & \text{if } x \ge x_{i2} \end{cases}$$

   - Using the points $-x_{i2}, -x_{i0}, x_{i0}, x_{i2}$ divide the input space into 5^H subregions.

Step 3. For each non-empty subregion, generate a rule as follows:

1. Define a linear equation that approximates the network's output $\hat{y}_p$ for input sample p in this subregion as the *consequence* of the extracted rule:

$$\hat{y}_p = \sum_{i=1}^{H} v_i L_i(s_{ip}), \text{ where} \tag{15}$$

$$s_{ip} = \sum_{j=1}^{N} w_{ij} I_{jp} \tag{16}$$

2. Generate the rule *condition*: ($\mathcal{C}_1$ and $\mathcal{C}_2$ and $\cdots$ $\mathcal{C}_H$), where $\mathcal{C}_i$ is either $s_{ip} < -x_{i0}$, $-x_{i0} \leq s_{ip} \leq x_{i0}$, or $s_{ip} > x_{i0}$ for the 3-piece approximation approach; or $\mathcal{C}_i$ is either $s_{ip} \leq -x_{i2}$, $-x_{i2} < s_{ip} < -x_{i0}$, $-x_{i0} \leq s_{ip} \leq x_{i0}$, $x_{i0} < s_{ip} < x_{i2}$, or $s_{ip} \geq x_{i2}$ for the 5-piece approximation approach.

Step 4. (Optional) Apply C4.5 [28] to simplify the rule conditions.

In general, a rule condition $\mathcal{C}_i$ is defined in terms of the weighted sum of the inputs s_{ip} (Eqn. 16) which corresponds to an oblique hyperplane in the input space. This type of rule condition can be difficult for the users to interpret. In some 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 the isolated inputs, and are easier for the users to understand. In some cases of real-life data, this enhanced interpretability would come at a possible cost of reduced accuracy. If the replacement of rule conditions is still desired, it can be achieved by employing a classification method such as C4.5 in the optional Step 4.

6 Illustrative Examples

The following examples of applying REFANN on two different data sets illustrate the working of the algorithm in more details[1]. The input attributes of the first data set are continuous, while those of the second data set are mixed. These two problems are selected because the pruned networks have few hidden units and the extracted rules have better accuracy than multiple linear regression. For both problems, we applied the 3-piece linear approximation.

Example 1. Pollution data set

The data set has 15 continuous attributes as listed in Table 1. The goal is to predict the total age-adjusted mortality rate per 100,000 (MORT). The values of all 15 input attributes were linearly scaled to the interval $[0, 1]$, while the target MORT was scaled so that it ranged in the interval $[0, 4]$.

One of the networks that had been trained for this data set was selected to illustrate in details how the rules were extracted by REFANN. This network originally had eight hidden units, but only one hidden unit remained after pruning. The number of training, cross-validation and test samples were 48, 6, and 6, respectively. Only the connections from inputs PRE, JAT, JUT, HOU, NOW, and SO2 were still present in the pruned network.

The weighted input value with the largest magnitude was taken as x_m and the value of x_0 was computed according to Eqn. 7 to be 0.6393. Therefore, the hyperbolic tangent function was approximated by

$$L_1(s_{1p}) = \begin{cases} 0.4155\, s_{1p} - 0.3501 & \text{if } s_{1p} < -0.6393 \\ s_{1p} & \text{if} - 0.6393 \leq s_{1p} \leq 0.6393 \\ 0.4155\, s_{1p} + 0.3501 & \text{if } s_{1p} > 0.6393 \end{cases}$$

The three subsets of the input space were defined by the following inequalities:

[1]The data sets have been downloaded from http://www.cs.waikato.ac.nz/~ml/weka/index.html.

Table 1: Attributes of the pollution data set.

Attribute	description
PRE	average annual precipitation in inches
JAT	average January temperature in degrees F
JUT	average July temperature in degrees F
OVR65	percentage of population aged 65 or older
POPN	average household size
EDUC	median school years completed by those over 22
HOU	percentage of housing units which are sound and with all facilities
DENS	population per square mile in urbanized areas in 1960
NOW	percentage of non-white population in urbanized areas in 1960
WWDRK	percentage of those employed in white collar occupations
POOR	percentage of families with income less than $3000
HC	relative hydrocarbon pollution potential
NOX	relative nitric oxides pollution potential
SO2	relative sulphur dioxide pollution potential
HUMID	annual average % relative humidity at 1pm

- Region 1: $s_{1p} < -0.6393 \Leftrightarrow -0.76$ PRE $+ 0.48$ JAT $+ 0.29$ JUT $+ 0.24$ HOU $- 0.98$ NOW $- 0.79$ SO2 $+ 0.24 < -0.6393$
- Region 2: $-0.6393 \leq s_{1p} \leq 0.6393 \Leftrightarrow -0.76$ PRE $+ 0.48$ JAT $+ 0.29$ JUT $+ 0.24$ HOU $- 0.98$ NOW $- 0.79$ SO2 $+ 0.24 \leq 0.6393$
- Region 3: $s_{1p} > 0.6393 \Leftrightarrow -0.76$ PRE $+ 0.48$ JAT $+ 0.29$ JUT $+ 0.24$ HOU $- 0.98$ NOW $- 0.79$ SO2 $+ 0.24 > 0.6393$

It should be noted that the coefficients of the two parallel hyperplanes that divide the input space into the three regions are equal to the weights w_{1j} from the j-th input unit to the hidden unit. Upon multiplying the coefficients of $L_1(s_{1p})$ by the connection weight value from the hidden unit to the output unit (Eqn. 15) and re-scaling the input and output data back into their original values, we obtain the following rules:

Rule Set 1:

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 $\hat{y} = Y_3$.

The predicted value $\hat{y}$ is given by one of the following 3 equations:

$$\begin{aligned} Y_1 &= 1030.51 + 0.95 \text{ PRE} - 0.59 \text{ JAT} - 0.70 \text{ JUT} - 0.55 \text{ HOU} + 1.39 \text{ NOW} + 0.16 \text{ SO2} \\ Y_2 &= 1075.68 + 2.28 \text{ PRE} - 1.43 \text{ JAT} - 1.68 \text{ JUT} - 1.31 \text{ HOU} + 3.34 \text{ NOW} + 0.37 \text{ SO2} \\ Y_3 &= 940.45 + 0.95 \text{ PRE} - 0.59 \text{ JAT} - 0.70 \text{ JUT} - 0.55 \text{ HOU} + 1.39 \text{ NOW} + 0.16 \text{ SO2} \end{aligned}$$

An optional final step of REFANN is to describe the 3 input subspaces by rule conditions generated by C4.5. All training samples p in Region 1 were given a target value of 1, in Region 2 a target value of 2, and in Region 3 a target value of 3. The following rules were generated by C4.5:

Rule Set 1a:

Rule 1: if SO2 > 146, then $\hat{y} = Y_1$.

Rule 2: if NOW > 27.1, then $\hat{y} = Y_1$.

Table 2: The errors of a pruned network, two rule sets extracted from it, and multiple linear regression for the pollution data on the test set. The error rates are computed in terms of the Root Mean Squared Error (RMSE), the Relative Root Mean Squared Error (RRMSE - Eqn. 17), the Mean Absolute Error (MAE), and the Relative Mean Absolute Error (RMAE - Eqn. 18).

	RMSE	RRMSE	MAE	RMAE
Pruned network	21.62	48.45	18.50	49.35
Rule set 1	20.63	46.22	17.12	45.67
Rule set 1a	20.64	46.25	17.17	45.81
Linear regression	69.61	155.96	53.42	142.51

Rule 3: if PRE > 27.1 and NOW ≤ 27.1, then $\hat{y} = Y_2$.

Rule 4: if PRE ≤ 13, then $\hat{y} = Y_3$.

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

The errors of the pruned neural network and the rule sets are shown in Table 2. We combined the training and cross-validation sets and obtained the coefficients of the linear regression that fit the samples. Using the backward regression option of SAS [29], none of the input attributes was found to be significant at the default significance level of 0.10. The errors from linear regression on the test set are included in Table 2 for comparison. In addition to the mean absolute error (MAE) and the root mean squared error (RMSE), we also show the relative root mean squared error (RRMSE) and the relative mean absolute error (RMAE):

$$\text{RRMSE} = 100 \times \sqrt{\sum_p (\tilde{y}_p - y_p)^2 / \sum_p (\overline{y} - y_p)^2} \tag{17}$$

$$\text{RMAE} = 100 \times \sum_p |\tilde{y}_p - y_p| / \sum_p |\overline{y} - y_p| \tag{18}$$

where the summation is computed over the samples in the test set and $\overline{y}$ is the average value of y_p in the test set. These relative errors are sometimes preferred over the usual sum of squared errors or the mean absolute error because they normalize the differences in the output ranges of different data sets. An RRMSE or an RMAE that is greater than 100 indicates that the method performs worse than the method that simply predicts the output using the average value of the samples.

The results from this example highlight the effectiveness of the REFANN algorithm in splitting the input space into subspaces, where in each subspace a linear equation is generated. By applying different equations depending on the values of the input data, the accuracy of the predictions is significantly improved. The error statistics also indicate that the prediction quality of the rules is very close to those of the pruned network.

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. 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 contained 318 samples, while the cross validation and test sets contained 40 samples each. The unary-coded data required the neural network to have 26 input units. One of the smallest pruned networks has only 1 hidden unit and 7 input units left. The relevant network inputs and their corresponding attributes in the original data set are the following: (1) $I_4 = 0.2$ iff cylinders [2] is greater than 3, (2) $I_9 = 0.2$ iff model is later than 78, (3) $I_{11} = 0.2$ iff model is later

[2] We use the 0/0.2 instead of 0/1 encoding scheme for discrete attributes (Section 7.1)

Table 3: The errors of a pruned network, two rule sets extracted from it, and multiple linear regression for the autoMpg data on the test set.

	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 set 2a	3.00	36.36	2.07	30.18
Linear regression	3.65	44.25	2.85	41.53

than 76, (4) $I_{14} = 0.2$ iff model is later than 73, (5) $I_{21} = 0.2$ iff origin is 1, (6) I_{23} is the continuous attribute horsepower, and (7) I_{24} is the continuous attribute weight.

Using the 3-piece linear approximation of the hidden unit activation function, a rule set consisting of just 2 rules was obtained:

Rule Set 2:

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: $s_{1p} < -0.8873 \Leftrightarrow 0.41\ I_4 + 1.07\ I_9 + 0.67\ I_{11} + 0.52\ I_{14} - 0.41\ I_{21} - 0.003\ I_{23} - 0.0004\ I_{24} < -1.7198$
- Region 2: $s_{1p} \geq -0.8873 \Leftrightarrow 0.41\ I_4 + 1.07\ I_9 + 0.67\ I_{11} + 0.52\ I_{14} - 0.41\ I_{21} - 0.003\ I_{23} - 0.0004\ I_{24} \geq -1.7198$

and the two corresponding linear equations are

$$\begin{aligned} Y_1 &= 16.26 + 0.54\ I_4 + 1.40\ I_9 + 0.88\ I_{11} + 0.69\ I_{14} - 0.53\ I_{21} - 0.0046\ I_{23} - 0.0005\ I_{24} \\ Y_2 &= 46.73 + 7.85\ I_4 + 20.30\ I_9 + 12.72\ I_{11} + 9.96\ I_{14} - 7.73\ I_{21} - 0.0661\ I_{23} - 0.0079\ I_{24} \end{aligned}$$

We obtained the following rule set from C4.5 after executing the optional Step 4 of the algorithm REFANN:

Rule Set 2a:

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

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

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

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

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

Rule 6: if $(I_{11} = 0.2)$ and $(I_{24} \leq 3432)$, then $\hat{y} = Y_2$.

Rule 7: if $(I_{11} = 0.2)$ and $(I_{23} \leq 115)$, then $\hat{y} = Y_2$.

Rule 8: if $(I_9 = 0.2)$, then $\hat{y} = Y_2$.

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

We compare the predictive accuracy of the extracted rules with the neural network and multiple linear regression in Table 3. 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 RMSE and MAE of this model. 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.

7 Experimental Results

Two sets of experiments were conducted to evaluate the performance of the proposed algorithms. The first set of experiments test N2PFA's accuracy in nonlinear function approximation on standard benchmark data (Section 7.1). The second set of experiments measure the complexity and accuracy of the rules extracted by REFANN (Section 7.2).

7.1 N2PFA Results

There have been a number of papers that propose algorithms for constructing and/or training neural network for regression [6, 30, 31, 32, 33]. A recent paper by Frank et al. [34] compares the results of naive Bayes methodology for regression to those from other regression methods including linear regression. Test results from 32 problems are reported in the paper. The data sets are available from their website (`http://www.cs.waikato.ac. nz/~ml/weka/index.html`) as part of the WEKA project. We tested our neural network pruning algorithm and rule extraction algorithm on these problems.

The data sets used in the experiment and the summary of their attribute features are listed in Table 4. They are shown in increasing order of the number of samples. Most of the data sets consist of both numeric and discrete attributes. The total number of attributes ranges from 2 to 25. Except for problem no. 19 pwLinear, all the other problems are from real world domains.

The following experimental setting were used to obtain the statistics from our network pruning algorithm:

- Ten-fold cross-validation scheme: We divided each data set randomly into ten subsets of equal size. Nine subsets were used as the training set $\mathcal{T}$. The samples in one of these subsets formed the cross-validation set $\mathcal{X}$. The predictive accuracy rates of the pruned networks and the extracted rules were computed on the one remaining subset of samples not used for training and cross-validation. This procedure was performed 10 times so that each subset was tested once. Test results were averaged over 20 ten-fold cross-validation runs.
- The same set of penalty parameter values in the penalty term (Eqn. 2) was used: $\epsilon_1 = 0.5, \epsilon_2 = 0.05$ and $\beta = 0.1$.
- The value of α was set to 0.02 when hidden units were being pruned. This value was reset to $\alpha = 0.10$ when input units were being pruned in Step 5 of N2PFA.
- The starting number of hidden units for all problems was 8. The numbers of input units are shown in Table 4. The number of input units includes one unit with a constant input value of one to implement hidden unit bias.
- The BFGS algorithm for network training was terminated (i.e. the network was assumed to have been trained) if the relative decrease in the error function value (Eqn. 1) after two consecutive iterations is less than 10^{-6}.
- One input unit was assigned to each continuous attribute in the data set. Discrete attributes were unary-coded. A discrete attribute with D possible values was assigned D network inputs.
- Continuous attribute values were scaled to range in the interval $[0, 1]$, while unary-encoded attribute values were either 0 or 0.2. We found that the 0/0.2 encoding produced better generalization than the usual 0/1 encoding for most of the data sets we experimented with.
- A missing continuous attribute value was replaced by the average of the non-missing values. A missing discrete attribute value was assigned the value "unknown" and the corresponding input $\mathbf{I}$ was set to the zero vector $\mathbf{0}$.
- The target output values were linearly scaled to range in the interval $[0, U]$, where U was either 4 or 16.

Table 4: Characteristics of the data sets used for experiments.

No.	Data set	Instances	Missing values (%)	Numeric attributes	Discrete attributes	Neural network inputs before pruning
1	schlvote	38	0.4	4	1	6
2	bolts	40	0.0	7	0	8
3	vineyard	52	0.0	3	0	4
4	elusage	55	0.0	1	1	14
5	pollution	60	0.0	15	0	16
6	mbagrade	61	0.0	1	1	3
7	sleep	62	2.4	7	0	8
8	auto93	93	0.7	16	6	62
9	baskball	96	0.0	4	0	5
10	cloud	108	0.0	4	2	10
11	fruitfly	125	0.0	2	2	9
12	echoMonths	130	7.5	6	3	11
13	veteran	137	0.0	3	4	11
14	fishcatch	158	6.9	5	2	15
15	autoPrice	159	0.0	15	0	16
16	servo	167	0.0	0	4	20
17	lowbwt	189	0.0	2	7	20
18	pharynx	195	1.1	1	10	37
19	pwLinear	200	0.0	10	0	11
20	autoHorse	205	1.1	17	8	69
21	cpu	209	0.0	6	1	37
22	bodyfat	252	0.0	14	0	15
23	breasttumor	286	0.3	1	8	39
24	hungarian	294	19.0	6	7	23
25	cholesterol	303	0.1	6	7	23
26	cleveland	303	0.1	6	7	23
27	autoMpg	398	0.2	4	3	26
28	pbc	418	15.6	10	8	30
29	housing	506	0.0	12	1	14
30	meta	528	4.3	19	2	66
31	sensory	576	0.0	0	11	33
32	strike	625	0.0	5	1	24

Our results are summarized in Tables 5 and 6. For comparison purpose, we also include the results from Frank et al. [34], who had applied four other regression methods on the data sets. Naive Bayes (NB) method [34] applies Bayes' theorem to estimate the probability density function of the target value y given an input sample $\mathbf{I}$. The method assumes that given the predicted value y, the attributes of $\mathbf{I}$ are independent of each other. LR is the standard linear regression method. Attribute selection was accomplished by backward elimination. The k-nearest-neighbor (kNN) is a distance-weighted k-nearest-neighbor method. The value of k varied from 1 to 20 and the optimal value of k was chosen using leave-one-out cross-validation on the training data. The model-tree predictor method M5$'$ generates binary decision trees with linear regression functions at the leaf nodes [35]. This method is an improved re-implementation of Quinlan's M5 [36].

To compare the pruned neural network accuracy on a test problem with that of another method, we computed the estimated standard error of the difference between the two means. The t statistic for testing the null hypothesis that the two means are equal was then obtained. We conducted a two-tailed test with significance level $\alpha = 0.01$. The means and standard deviations in columns NB, LR, kNN and K5$'$ shown in bold indicate that the error means are significantly higher or worse than the corresponding figures of the pruned neural networks. Those printed in italic are significantly

Table 5: The RRMSE ± the standard deviation from five regression methods. Pruned NN results are averages from 20 ten-fold cross-validation runs, while those of the other methods are from 10 ten-fold cross-validation runs. Figures in **bold** (*italic*) indicate that the RRMSE of the method is significantly higher (lower) than the average rate of the pruned network.

No.	Pruned NN	NB	LR	kNN	M5′
1	192.12 ± 37.87	**263.70 ± 87.20**	233.31 ± 93.30	**267.91 ± 97.10**	164.24 ± 70.80
2	39.26 ± 11.47	**57.61 ± 10.70**	**53.37 ± 10.60**	**77.95 ± 10.90**	32.28 ± 7.60
3	75.80 ± 9.74	82.90 ± 9.60	77.37 ± 9.70	75.52 ± 8.10	77.21 ± 9.90
4	55.39 ± 6.94	**65.51 ± 6.30**	53.21 ± 5.90	**70.94 ± 7.00**	49.36 ± 3.90
5	79.86 ± 9.25	**97.03 ± 11.10**	**98.86 ± 9.20**	85.99 ± 8.40	79.09 ± 8.50
6	100.37 ± 9.01	103.84 ± 6.80	*85.63 ± 4.00*	**118.54 ± 8.30**	*85.63* ± 4.00
7	102.23 ± 13.05	95.01 ± 10.50	*82.08 ± 7.60*	90.88 ± 8.50	92.48 ± 25.80
8	54.83 ± 5.38	**66.62 ± 6.40**	**67.25 ± 5.70**	**71.77 ± 4.20**	**61.79 ± 4.80**
9	89.84 ± 5.58	92.03 ± 4.00	*80.11 ± 2.10*	90.02 ± 3.60	*81.35 ± 2.30*
10	41.77 ± 3.51	**56.55 ± 2.90**	39.71 ± 2.20	**75.46 ± 2.80**	40.07 ± 2.00
11	107.88 ± 3.66	**123.54 ± 4.60**	*100.16 ± 0.50*	**120.30 ± 4.20**	*100.38 ± 0.60*
12	73.68 ± 3.14	**81.63 ± 2.20**	*68.93 ± 1.20*	72.61 ± 2.20	*68.50 ± 1.40*
13	102.70 ± 8.86	*93.66 ± 5.70*	97.39 ± 7.00	106.15 ± 8.50	*93.70 ± 6.00*
14	14.98 ± 0.88	**32.14 ± 2.20**	**30.76 ± 2.30**	**34.82 ± 4.20**	**16.61 ± 0.70**
15	40.58 ± 3.21	**43.63 ± 1.60**	**49.28 ± 3.40**	**46.05 ± 2.20**	37.95 ± **2.50**
16	27.56 ± 4.16	**75.07 ± 1.80**	**62.71 ± 9.60**	**57.21 ± 7.00**	**38.35 ± 2.20**
17	67.37 ± 5.04	64.32 ± 1.30	62.77 ± 2.00	70.02 ± 1.80	*62.27 ± 1.10*
18	67.01 ± 2.32	**87.38 ± 2.90**	**79.33 ± 2.60**	**80.23 ± 1.60**	**72.86 ± 2.10**
19	37.40 ± 0.97	**53.53 ± 1.10**	**51.08 ± 1.40**	**54.80 ± 1.60**	*33.19 ± 0.80*
20	26.18 ± 1.48	**39.11 ± 2.00**	**54.20 ± 5.40**	**44.81 ± 1.40**	**31.72 ± 2.50**
21	12.73 ± 1.79	**35.91 ± 4.40**	**52.17 ± 8.40**	**38.57 ± 5.90**	**18.99 ± 2.60**
22	12.93 ± 1.53	**26.73 ± 0.60**	12.50 ± 0.70	**36.72 ± 0.90**	*10.49 ± 0.90*
23	99.77 ± 1.19	**103.04 ± 1.50**	*97.23 ± 1.20*	**105.98 ± 1.20**	*97.03 ± 1.10*
24	73.41 ± 1.97	73.04 ± 2.30	74.16 ± 0.70	72.67 ± 1.70	**76.95 ± 2.10**
25	100.39 ± 1.14	**103.90 ± 1.50**	99.68 ± 1.70	**102.69** ± 1.10	**103.54 ± 1.90**
26	70.94 ± 1.85	**76.00 ± 2.00**	70.54 ± 1.00	**73.49** ± 1.40	**74.51 ± 2.10**
27	36.62 ± 0.88	**42.44 ± 0.70**	**38.43 ± 0.90**	**42.62 ± 1.10**	36.37 ± 0.60
28	83.10 ± 1.72	**87.33 ± 1.10**	*80.48 ± 0.70*	**87.73 ± 1.20**	**86.22 ± 1.40**
29	36.65 ± 1.45	**61.00 ± 1.60**	**52.78 ± 1.10**	**45.14 ± 1.50**	**39.20 ± 1.90**
30	119.98 ± 16.37	**238.24 ± 71.70**	**281.85 ± 51.10**	**240.99 ± 47.00**	**200.17 ± 75.30**
31	90.84 ± 1.31	**93.11 ± 0.90**	**94.58 ± 1.60**	90.07 ± 0.70	*86.10 ± 0.90*
32	87.89 ± 2.13	**162.37 ± 12.90**	*84.55 ± 1.70*	86.59 ± 2.30	*84.47 ± 1.20*

lower than the corresponding neural network error means.

Table 7 summarizes the results presented in Tables 5 and 6. We can see from the number of NN wins that the pruned neural networks perform quite well compared to the other methods for regression. The nearest competitor is M5$'$. Using the RRMSE as the performance measure, the number of NN wins is almost the same as that of losses. However, when the errors are measured in terms of RMAE, pruned NN outperforms M5$'$ in 13 data sets and produces significantly higher errors only on 4 data sets.

Table 6: The RMAE ± the standard deviation from five regression methods.

No.	Pruned NN	NB	LR	kNN	M5′
1	186.66 ± 44.07	249.71 ± 88.60	**347.09 ± 130.10**	225.52 ± 83.40	223.48 ± 100.30
2	33.17 ± 9.84	**53.63 ± 9.00**	**67.27 ± 16.70**	**67.80 ± 9.40**	36.19 ± 10.00
3	74.47 ± 8.78	80.87 ± 8.70	**89.15 ± 10.20**	74.09 ± 7.40	**86.47 ± 11.60**
4	53.93 ± 7.59	**63.57 ± 5.60**	58.61 ± 6.60	**66.00 ± 6.00**	53.67 ± 3.40
5	76.36 ± 6.83	**98.86 ± 14.00**	**110.73 ± 13.00**	85.76 ± 13.60	85.40 ± 12.40
6	98.12 ± 8.74	101.90 ± 7.90	93.51 ± 8.30	**112.10 ± 10.60**	93.51 ± 8.30
7	103.59 ± 14.73	96.65 ± 12.30	92.83 ± 11.10	95.17 ± 14.10	104.61 ± 24.10
8	49.73 ± 5.27	**61.15 ± 4.50**	**70.66 ± 5.80**	**65.08 ± 4.10**	**59.98 ± 4.20**
9	88.94 ± 5.35	89.96 ± 3.30	*83.27 ± 2.50*	88.83 ± 2.50	84.50 ± 2.40
10	37.97 ± 3.05	**49.60 ± 2.50**	39.18 ± 2.40	**68.10 ± 2.70**	38.84 ± 2.20
11	107.49 ± 4.33	**122.10 ± 4.40**	105.39 ± 3.40	**122.67 ± 4.10**	105.90 ± 3.20
12	69.66 ± 3.06	**75.04 ± 2.70**	72.27 ± 2.10	68.97 ± 2.50	*66.54 ± 2.30*
13	98.63 ± 11.90	*82.56 ± 3.70*	99.02 ± 6.40	101.56 ± 6.50	93.13 ± 6.10
14	11.46 ± 0.68	**23.28 ± 1.20**	**30.05 ± 2.60**	**21.31 ± 1.60**	**15.31 ± 0.60**
15	36.47 ± 2.69	**40.50 ± 2.40**	**46.04 ± 3.20**	38.56 ± 2.20	34.30 ± 2.50
16	22.82 ± 2.56	**55.77 ± 1.60**	**66.42 ± 8.20**	**53.40 ± 4.70**	**30.23 ± 1.60**
17	67.04 ± 5.71	63.40 ± 1.40	65.31 ± 2.30	65.61 ± 1.70	63.88 ± 1.40
18	64.64 ± 2.57	**80.05 ± 2.20**	**78.20 ± 2.60**	**74.76 ± 1.60**	**71.88 ± 2.20**
19	37.11 ± 1.28	**52.35 ± 0.90**	**52.38 ± 1.40**	**53.11 ± 1.40**	*34.03 ± 1.00*
20	21.30 ± 1.22	**29.37 ± 1.40**	**50.71 ± 6.20**	**29.53 ± 1.30**	**26.87 ± 1.60**
21	9.42 ± 0.96	**31.29 ± 3.00**	**56.63 ± 7.40**	**27.22 ± 2.50**	**17.34 ± 1.70**
22	7.11 ± 0.41	**21.92 ± 0.30**	**7.74 ± 0.10**	**34.95 ± 1.00**	*5.51 ± 0.10*
23	100.07 ± 1.42	**104.88 ± 1.20**	99.61 ± 1.80	**106.57 ± 1.60**	100.13 ± 1.80
24	53.46 ± 3.11	*39.81* ± 1.00	**93.72 ± 3.20**	*49.22 ± 2.90*	**57.72 ± 3.20**
25	100.02 ± 1.34	**101.91 ± 0.90**	**101.83 ± 2.20**	**102.37 ± 1.40**	**105.57 ± 2.30**
26	59.41 ± 1.51	59.37 ± 1.70	**66.21 ± 1.10**	**62.91 ± 1.60**	**64.89 ± 1.20**
27	32.43 ± 1.14	**37.55 ± 0.60**	**35.64 ± 0.90**	**36.82 ± 0.80**	32.07 ± 0.70
28	80.48 ± 1.48	81.55 ± 1.40	80.88 ± 1.10	**86.22 ± 1.50**	**83.66 ± 1.30**
29	34.86 ± 1.17	**56.74 ± 0.70**	**51.76 ± 0.50**	**40.76 ± 1.00**	**36.43 ± 1.20**
30	93.21 ± 10.03	**134.28 ± 23.00**	**236.82 ± 35.20**	**163.12 ± 21.40**	**119.84 ± 20.40**
31	90.97 ± 1.41	**94.02 ± 1.00**	**96.25 ± 1.50**	91.21 ± 0.80	*88.24 ± 1.10*
32	70.10 ± 2.39	**93.63 ± 2.90**	**75.46 ± 1.60**	68.42 ± 1.00	70.83 ± 1.50

Table 7: Summary of comparison between the results from pruned neural networks versus those from four other regression methods. The number of wins (losses) is the number of times that the errors from pruned neural networks are significantly lower (higher) than the corresponding errors of the other methods.

	Relative RMSE			Relative MAE		
Pruned NN versus	**Wins**	Ties	*Losses*	**Wins**	Ties	*Losses*
Naive Bayes	**25**	6	*1*	**22**	8	*2*
LR	**14**	10	*8*	**21**	10	*1*
kNN-RMSE/MAE	**22**	10	*0*	**20**	11	*1*
M5′	**12**	9	*11*	**13**	15	*4*

7.2 REFANN Results

We applied REFANN to each of the networks that had been pruned by the N2PFA algorithm. The results are summarized in Tables 8, 9, and 10. All the figures in these tables are averages and standard deviations from 20 replications of ten-fold cross-validation runs with different random data splitting for each ten-fold run. Table 8 shows the relative root mean squared errors (RRMSE), while Table 9 shows the relative mean absolute error (RMAE) from the 3-piece and 5-piece linear approximations. The results from pruned neural networks are also reproduced from Tables 5 and 6 for reference. To compare the accuracy of the approximations with that of M5$'$, we also conducted a two-tailed test with significance level $\alpha = 0.01$. Bold (italic) figures in Tables 8 and 9 bold indicate average error rates of the pruned networks or the extracted rules that are significantly lower (higher) than the corresponding rates of M5$'$.

Table 10 summarizes the comparison results between REFANN and M5$'$. Approximation of the hidden unit activation function by a 3-piece linear function deteriorates the predictive accuracy for most of the problems tested. On the other hand, the accuracy of the 5 piece-approximation are practically identical to the accuracy of the pruned networks.

Table 11 shows the average number of hidden units and input units in the pruned networks. For most of the problems, the number of remaining hidden units is significantly less than 8, the number of initial hidden units in the original networks. The number of input attributes left unpruned by the network pruning algorithm is shown in terms of the original attributes and the unary-encoded attributes. We have also included in this table the percentage reduction of the input attributes by pruning. The reduction in the original attributes of the data ranges from 0% for the servo data set to 90.62% for the cholesterol data set. In all of the 200 neural networks trained and pruned on samples from the servo data set, all four attributes were still present. The reduction in encoded input attributes by pruning ranges from 7.80% for the meta data set to 95.63% for the breasttumor data set. The average reductions in the number of original attributes and encoded attributes for the 32 test data sets are 42.27% and 54.60%, respectively.

The number of rules presented in Table 12 is the number of linear functions generated by the algorithm REFANN for each non-empty subregion of the input space. For problems such as servo and housing, REFANN generated a large number of rules which are not likely to aid our understanding about the relationship between the input and output of the data. For the remaining 30 data sets, REFANN has produced an average of 6.2 rules per set using the 3-piece approximation and 10.94 rules per set using the 5-piece approximation.

The fidelity of the rules was computed to measure how similar the predictions of the rules were compared to those of the pruned networks:

$$\text{Fidelity} = 100 \times \sum_p (\tilde{y}_p - \hat{y}_p)^2 / \sum_p (\overline{y} - y_p)^2$$

As expected, the fidelity of the rules from the 5-piece approximation method is higher compared to that of corresponding rules from 3-piece approximation method. As REFANN approximates the activation of the individual hidden units, it can be expected to generate rules with much higher fidelity than "pedagogical" approaches. Pedagogical approaches such as the ANN-DT method [24] are rule extraction methods that extract global relationship between the input and output of a network without directly investigating the hidden unit activations [1].

Finally, the last column of Table 12 shows the average CPU time needed to execute the ten-fold cross-validation once. The time reported is the total time to train and pruned ten neural networks plus the time to extract the rules from these networks using the 3-piece and 5-piece approximations. The codes for network training and pruning and for rule extraction were written in FORTRAN and run on a Sun Ultra Enterprise 450.

8 Comments and Conclusion

Essentially, producing rules understandable for humans in cases of approximation of nonlinear multivariable relationships is a challenging task. Rarely, if not only in trivial cases, are the outputs

Table 8: Comparison of the RRMSE of pruned neural networks, REFANN's 3-piece and 5-piece approximations and M5′. Figures in **bold** (*italic*) indicate that the RRMSE of the method is significantly lower (higher) than that of M5′.

No.	Data set	Pruned NN	3 piece-approx.	5 piece-approx.	M5′
1	schlvote	192.12 ± 37.87	192.56 ± 37.29	191.41 ± 37.66	164.24 ± 70.80
2	bolts	39.26 ± 11.47	40.08 ± 9.97	38.71 ± 10.53	32.28 ± 7.60
3	vineyard	75.80 ± 9.74	79.79 ± 9.32	74.29 ± 9.93	77.21 ± 9.90
4	elusage	55.39 ± 6.94	*57.01* ± *7.58*	55.14 ± 6.85	49.36 ± 3.90
5	pollution	79.86 ± 9.25	84.33 ± 9.63	81.24 ± 9.45	79.09 ± 8.50
6	mbagrade	*100.37* ± *9.01*	*99.65* ± *8.81*	*100.33* ± *8.96*	85.63 ± 4.00
7	sleep	102.23 ± 13.05	105.12 ± 13.31	103.43 ± 13.16	92.48 ± 25.80
8	auto93	**54.83** ± 5.38	57.19 ± 5.51	**55.44** ± **5.52**	61.79 ± 4.80
9	baskball	*89.84* ± *5.58*	*90.66* ± *6.00*	*90.06* ± *5.72*	81.35 ± 2.30
10	cloud	41.77 ± 3.51	42.57 ± 3.77	41.82 ± 3.62	40.07 ± 2.00
11	fruitfly	*107.88* ± *3.66*	*108.09* ± *3.77*	*107.93* ± *3.72*	100.38 ± 0.60
12	echoMonths	*73.68* ± *3.14*	*73.96* ± *3.35*	*73.63* ± *3.22*	68.50 ± 1.40
13	veteran	*102.70* ± *8.86*	*102.72* ± *8.89*	*102.71* ± *8.88*	93.70 ± 6.00
14	fishcatch	**14.98** ± **0.88**	*20.24* ± *1.33*	**15.05** ± **0.93**	16.61 ± 0.70
15	autoPrice	40.58 ± 3.21	*42.49* ± *3.42*	40.74 ± 3.41	37.95 ± 2.50
16	servo	**27.56** ± **4.16**	**30.66** ± **4.52**	**27.77** ± **4.27**	38.35 ± 2.20
17	lowbwt	*67.37* ± *5.04*	*67.30* ± *5.04*	*67.37* ± *5.04*	62.27 ± 1.10
18	pharynx	**67.01** ± **2.32**	**66.83** ± **2.30**	**66.85** ± **2.33**	72.86 ± 2.10
19	pwLinear	*37.40* ± *0.97*	*36.39* ± *1.04*	*37.32* ± *1.10*	33.19 ± 0.80
20	autoHorse	**26.18** ± **1.48**	30.08 ± 1.69	**26.63** ± **1.49**	31.72 ± 2.50
21	cpu	**12.73** ± **1.79**	17.66 ± 1.75	**15.72** ± **1.92**	18.99 ± 2.60
22	bodyfat	*12.93* ± *1.53*	*13.75* ± *1.44*	*13.25* ± *1.50*	10.49 ± 0.90
23	breasttumor	*99.77* ± *1.19*	*99.78* ± *1.19*	*99.78* ± *1.19*	97.03 ± 1.10
24	hungarian	**73.41** ± **1.97**	**74.35** ± **2.08**	**73.70** ± **1.96**	76.95 ± 2.10
25	cholesterol	**100.39** ± **1.14**	**100.40** ± **1.13**	**100.38** ± **1.14**	103.54 ± 1.90
26	cleveland	**70.94** ± **1.85**	**71.51** ± **1.99**	**70.84** ± **1.86**	74.51 ± 2.10
27	autoMpg	36.62 ± 0.88	*38.28* ± *0.90*	36.79 ± 0.89	36.37 ± 0.60
28	pbc	**83.10** ± **1.72**	**83.37** ± **1.78**	**83.20** ± **1.76**	86.22 ± 1.40
29	housing	**36.65** ± **1.45**	*42.03* ± *2.05*	**37.32** ± **1.46**	39.20 ± 1.90
30	meta	**119.98** ± **16.37**	**119.96** ± **16.36**	**119.95** ± **16.37**	200.17 ± 75.30
31	sensory	*90.84* ± *1.31*	*90.79* ± 1.44	*91.03* ± *1.38*	86.10 ± 0.90
32	strike	*87.89* ± *2.13*	*87.90* ± 2.13	*87.89* ± *2.13*	84.47 ± 1.20

Table 9: Comparison of the RMAE of pruned neural networks, REFANN's 3-piece and 5-piece approximations and M5′.

No.	Data set	Pruned NN	3 piece-approx.	5 piece-approx.	M5′
1	schlvote	186.66 ± 44.07	187.59 ± 43.22	186.12 ± 43.80	223.48 ± 100.30
2	bolts	33.17 ± 9.84	36.12 ± 9.53	33.80 ± 9.77	36.19 ± 10.00
3	vineyard	**74.47 ± 8.78**	78.59 ± 8.35	**73.71 ± 8.94**	86.47 ± 11.60
4	elusage	53.93 ± 7.59	54.69 ± 8.07	53.01 ± 7.41	53.67 ± 3.40
5	pollution	76.36 ± 6.83	81.39 ± 7.05	78.01 ± 6.83	85.40 ± 12.40
6	mbagrade	98.12 ± 8.74	96.55 ± 8.71	98.07 ± 8.77	93.51 ± 8.30
7	sleep	103.59 ± 14.73	106.35 ± 14.89	104.78 ± 14.80	104.61 ± 24.10
8	auto93	**49.73 ± 5.27**	**52.16 ± 5.24**	**50.17 ± 5.41**	59.98 ± 4.20
9	baskball	88.94 ± 5.35	89.68 ± 5.91	89.11 ± 5.49	84.50 ± 2.40
10	cloud	37.97 ± 3.05	37.87 ± 3.11	37.76 ± 3.12	38.84 ± 2.20
11	fruitfly	107.49 ± 4.33	107.61 ± 4.39	107.51 ± 4.37	105.90 ± 3.20
12	echoMonths	*69.66 ± 3.06*	69.17 ± 3.25	69.22 ± 3.13	66.54 ± 2.30
13	veteran	98.63 ± 11.90	98.65 ± 11.93	98.64 ± 11.92	93.13 ± 6.10
14	fishcatch	**11.46 ± 0.68**	*19.20 ± 1.25*	**12.31 ± 0.77**	15.31 ± 0.60
15	autoPrice	36.47 ± 2.69	*38.92 ± 2.97*	36.44 ± 2.91	34.30 ± 2.50
16	servo	**22.82 ± 2.56**	29.48 ± 3.31	**23.76 ± 2.73**	30.23 ± 1.60
17	lowbwt	67.04 ± 5.71	67.07 ± 5.72	67.04 ± 5.71	63.88 ± 1.40
18	pharynx	**64.64 ± 2.57**	**64.28 ± 2.54**	**64.45 ± 2.58**	71.88 ± 2.20
19	pwLinear	*37.11 ± 1.28*	*36.03 ± 1.29*	*37.00 ± 1.35*	34.03 ± 1.00
20	autoHorse	**21.30 ± 1.22**	26.84 ± 1.49	**22.00 ± 1.23**	26.87 ± 1.60
21	cpu	**9.42 ± 0.96**	**15.70 ± 1.33**	**14.49 ± 1.25**	17.34 ± 1.70
22	bodyfat	*7.11 ± 0.41*	*8.60 ± 0.52*	*7.83 ± 0.44*	5.51 ± 0.10
23	breasttumor	100.07 ± 1.42	100.08 ± 1.43	100.07 ± 1.42	100.13 ± 1.80
24	hungarian	**53.46 ± 3.11**	**51.56 ± 3.18**	**52.67 ± 3.10**	57.72 ± 3.20
25	cholesterol	**100.02 ± 1.34**	**100.04 ± 1.32**	**100.02 ± 1.34**	105.57 ± 2.30
26	cleveland	**59.41 ± 1.51**	**58.37 ± 1.48**	**58.80 ± 1.44**	64.89 ± 1.20
27	autoMpg	32.43 ± 1.14	*34.53 ± 1.12*	32.74 ± 1.17	32.07 ± 0.70
28	pbc	**80.48 ± 1.48**	**80.31 ± 1.57**	**80.28 ± 1.54**	83.66 ± 1.30
29	housing	**34.86 ± 1.17**	*42.32 ± 1.96*	35.80 ± 1.16	36.43 ± 1.20
30	meta	**93.21 ± 10.03**	**92.99 ± 9.98**	**93.08 ± 9.99**	119.84 ± 20.40
31	sensory	*90.97 ± 1.41*	*90.31 ± 1.60*	*90.87 ± 1.55*	88.24 ± 1.10
32	strike	70.10 ± 2.39	70.08 ± 2.38	70.08 ± 2.38	70.83 ± 1.50

Table 10: Summary of comparison between the results of M5′ versus those from pruned neural networks and REFANN's linear approximations. The number of wins (losses) is the number of times that the errors of pruned neural networks and the rules are significantly lower (higher) than the corresponding errors of M5′.

	Relative RMSE			Relative MAE		
M5′ versus	**Wins**	Ties	*Losses*	**Wins**	Ties	*Losses*
Pruned NN	**12**	9	*11*	**13**	15	*4*
3-piece approx.	**7**	9	*16*	**8**	17	*7*
5-piece approx.	**12**	9	*11*	**12**	17	*3*

Table 11: The average number of hidden units and the number of input attributes of the pruned networks.

No.	Data set	Hidden units	Original attributes		Encoded attributes	
			average	% Reduction	average	% Reduction
1	schlvote	1.03 ± 0.07	2.56 ± 0.38	48.80	2.56 ± 0.38	48.80
2	bolts	5.57 ± 0.33	4.65 ± 0.45	33.57	4.65 ± 0.45	33.57
3	vineyard	2.26 ± 0.34	1.22 ± 0.10	59.33	1.22 ± 0.10	59.33
4	elusage	2.32 ± 0.58	1.23 ± 0.12	38.50	1.62 ± 0.46	87.54
5	pollution	3.85 ± 0.24	10.46 ± 0.47	30.27	10.46 ± 0.47	30.27
6	mbagrade	1.34 ± 0.22	1.76 ± 0.13	12.00	1.76 ± 0.13	12.00
7	sleep	3.10 ± 0.40	3.34 ± 0.36	52.29	3.34 ± 0.36	52.29
8	auto93	3.35 ± 0.22	21.73 ± 0.18	1.23	32.30 ± 1.21	47.05
9	baskball	2.25 ± 0.21	2.15 ± 0.23	46.25	2.15 ± 0.23	46.25
10	cloud	3.15 ± 0.31	2.92 ± 0.27	51.33	2.97 ± 0.29	67.00
11	fruitfly	1.99 ± 0.52	1.14 ± 0.17	71.50	1.21 ± 0.25	84.88
12	echoMonths	1.58 ± 0.19	2.13 ± 0.23	76.33	2.40 ± 0.34	76.00
13	veteran	3.83 ± 0.42	2.94 ± 0.55	58.00	4.29 ± 0.78	57.10
14	fishcatch	4.95 ± 0.56	6.55 ± 0.16	6.43	7.52 ± 0.46	46.29
15	autoPrice	4.20 ± 0.27	9.70 ± 0.64	35.33	9.70 ± 0.64	35.33
16	servo	7.30 ± 0.17	4.00 ± 0.00	0.00	14.83 ± 0.23	21.95
17	lowbwt	1.10 ± 0.21	1.60 ± 0.36	82.22	1.63 ± 0.40	91.42
18	pharynx	1.00 ± 0.00	6.03 ± 0.48	45.18	9.04 ± 1.00	74.89
19	pwLinear	3.23 ± 0.22	5.83 ± 0.10	41.70	5.83 ± 0.10	41.70
20	autoHorse	3.26 ± 0.26	24.86 ± 0.16	0.56	41.60 ± 1.65	38.82
21	cpu	5.18 ± 0.26	5.17 ± 0.30	26.14	10.51 ± 2.34	70.81
22	bodyfat	6.93 ± 0.31	4.50 ± 0.96	67.86	4.50 ± 0.96	67.86
23	breasttumor	1.19 ± 0.19	1.44 ± 0.21	84.00	1.66 ± 0.42	95.63
24	hungarian	3.42 ± 0.44	9.81 ± 0.40	24.54	14.70 ± 0.85	33.18
25	cholesterol	1.35 ± 0.41	1.22 ± 0.19	90.62	1.24 ± 0.23	94.36
26	cleveland	3.23 ± 0.47	10.03 ± 0.73	22.85	14.98 ± 1.38	31.91
27	autoMpg	1.07 ± 0.07	3.68 ± 0.20	47.43	5.02 ± 0.39	79.92
28	pbc	1.94 ± 0.24	5.04 ± 0.32	72.00	5.05 ± 0.31	82.59
29	housing	6.61 ± 0.21	9.38 ± 0.41	27.85	9.38 ± 0.41	27.85
30	meta	3.75 ± 0.51	17.97 ± 0.54	14.43	59.93 ± 0.81	7.80
31	sensory	1.99 ± 0.02	5.08 ± 0.29	53.82	6.19 ± 0.41	80.66
32	strike	3.10 ± 0.71	4.19 ± 0.38	30.17	17.89 ± 1.26	22.22

Table 12: The average number of rules generated, their fidelity and the average CPU time for one ten-fold cross-validation run.

No.	Data set	3-piece approximation		5-piece approximation		CPU time
		# rules	Fidelity	# rules	Fidelity	(seconds)
1	schlvote	2.02 ± 0.05	96.71 ± 3.45	3.01 ± 0.03	99.74 ± 0.18	3.40
2	bolts	12.61 ± 2.46	94.50 ± 6.85	16.53 ± 3.00	98.63 ± 1.65	16.40
3	vineyard	2.14 ± 0.14	91.48 ± 2.85	3.37 ± 0.32	98.32 ± 0.57	5.50
4	elusage	2.42 ± 0.19	97.78 ± 1.22	3.68 ± 0.31	99.34 ± 0.45	17.75
5	pollution	13.79 ± 0.94	97.93 ± 0.59	28.27 ± 1.34	99.65 ± 0.14	43.25
6	mbagrade	2.43 ± 0.14	98.97 ± 0.39	3.85 ± 0.19	99.87 ± 0.08	4.15
7	sleep	8.40 ± 1.57	98.53 ± 0.52	15.98 ± 3.15	99.72 ± 0.12	10.85
8	auto93	10.84 ± 0.92	96.87 ± 0.70	24.19 ± 1.97	99.18 ± 0.20	641.15
9	baskball	5.77 ± 0.84	99.79 ± 0.11	10.86 ± 1.67	99.96 ± 0.02	6.65
10	cloud	5.40 ± 0.51	99.14 ± 0.89	10.00 ± 1.01	99.80 ± 0.06	25.85
11	fruitfly	2.56 ± 0.64	99.93 ± 0.08	3.50 ± 1.19	99.99 ± 0.02	15.05
12	echoMonths	2.82 ± 0.38	99.73 ± 0.06	4.28 ± 0.59	99.95 ± 0.01	23.20
13	veteran	6.17 ± 0.88	100.00 ± 0.00	7.97 ± 1.16	100.00 ± 0.00	18.55
14	fishcatch	9.35 ± 0.97	97.51 ± 0.65	15.03 ± 1.49	99.71 ± 0.05	54.95
15	autoPrice	16.17 ± 1.41	96.88 ± 0.81	38.20 ± 3.22	99.54 ± 0.14	78.85
16	servo	54.52 ± 2.26	96.77 ± 1.66	88.00 ± 3.30	99.56 ± 0.19	91.60
17	lowbwt	2.23 ± 0.14	99.99 ± 0.01	2.52 ± 0.28	100.00 ± 0.00	64.65
18	pharynx	2.02 ± 0.04	99.91 ± 0.02	3.21 ± 0.11	99.98 ± 0.00	176.55
19	pwLinear	10.22 ± 0.64	98.99 ± 0.08	21.29 ± 1.46	99.80 ± 0.03	55.35
20	autoHorse	10.97 ± 1.08	97.58 ± 0.28	24.32 ± 2.70	99.57 ± 0.05	1105.65
21	cpu	5.64 ± 0.43	97.77 ± 0.34	9.26 ± 0.81	98.65 ± 0.52	237.55
22	bodyfat	6.84 ± 0.64	99.73 ± 0.04	10.31 ± 0.93	99.94 ± 0.01	125.70
23	breasttumor	2.15 ± 0.13	100.00 ± 0.00	2.45 ± 0.29	100.00 ± 0.00	236.95
24	hungarian	12.25 ± 2.58	99.39 ± 0.11	23.96 ± 4.81	99.87 ± 0.02	169.10
25	cholesterol	2.15 ± 0.15	100.00 ± 0.00	3.44 ± 0.34	100.00 ± 0.00	124.70
26	cleveland	9.47 ± 1.52	99.64 ± 0.06	12.24 ± 1.59	99.91 ± 0.01	119.20
27	autoMpg	2.14 ± 0.15	98.52 ± 0.12	3.13 ± 0.23	99.71 ± 0.02	249.75
28	pbc	4.42 ± 0.40	99.75 ± 0.06	8.34 ± 0.60	99.93 ± 0.02	247.15
29	housing	50.35 ± 3.69	95.01 ± 0.69	139.37 ± 8.31	99.46 ± 0.06	285.20
30	meta	7.56 ± 1.79	99.99 ± 0.02	4.61 ± 0.70	100.00 ± 0.00	892.75
31	sensory	3.18 ± 0.12	98.78 ± 0.12	5.42 ± 0.19	99.69 ± 0.04	435.85
32	strike	8.65 ± 2.28	100.00 ± 0.00	5.10 ± 0.84	100.00 ± 0.00	248.55

constant in selected variables, and even this may only be true in selected subregions of the input space. Let us call such an approximation by a number of constant conditions a zero-order approximation. Such representation of the domain is perhaps the most naive, and it does not describe most of the problems sufficiently well. It usually also produces a large number of rules, each typically containing logic operators relating input variables. The main advantage of such rules is that they are both natural and simplest to understand. More typically in real-life problems, however, inputs are correlated, and output(s) needs to be expressed in terms of some form of algebraic expressions of input variables. This leads to the concept of a straightforward, linear functional link between variables which is explored in this paper.

Linearization of the relationships between input variables can be understood as a first-order approximation. As such, exploring an interpolation based on linear combinations of inputs introduces a more realistic approximation technique as compared with zero-order technique. Because this representation reflects interactions between input variables (which is disregarded in zero-order approach), once the rule has been produced, the increased burden of its interpretation is shifted on the user. To clearly understand the rule, or gain the knowledge about the domain, the user needs to grasp and reconcile all the existing interactions among variables. It should be noted that these interactions would differ in different subregions of input space, and they may involve multi-variable scenarios.

Although the proposed rules in the form of linear combinations are not as easy to interpret as the ones resulting from zero-order approximation, they still offer a fair degree of transparency. The coefficients of each variable present in the rule are indicative of the degree and sign of the correlation of that variable with the output. This important feature of the knowledge domain could not be extracted from rules based zero-order approximation. As to the interpretability of the rules presented in this paper, more complex problems lead naturally to more rules each with larger numbers of antecedents compared to simple problems. The proposed technique offers a trade-off between the number, accuracy and interpretability of the rules.

Comparing these three measures of quality is difficult for a number of reasons. Interpretability of the rules is highly subjective and renders itself to different verdicts. Even if it could be somehow quantified, we still need to consider the number and the accuracy of the rules. Given the lack of an existing standard and the trade-off between the three qualities of rule comparisons, we have evaluated the rules extracted by our method on the 32 benchmark data sets and have found our method to be comparable if not superior to other methods.

In this paper, we have described the algorithm N2PFA for pruning neural networks that have been trained for regression and the algorithm REFANN for extracting linear regression rules from the pruned networks. The algorithm N2PFA produces pruned networks that are no less accurate than other regression methods for many of the problems that we have tested. The algorithm REFANN attempts to provide an explanation for the network outputs by replacing the nonlinear mapping of a pruned network by a set of linear regression equations. Using the weights of a trained network, REFANN divides 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 or 5-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. For each subregion, a linear equation that approximates the network output is generated. Experiments performed on a wide range of real world problems show the effectiveness of the method in generating accurate rule sets with high fidelity.

Acknowledgment

This work was done while the first author was spending his sabbatical leave at the Computational Intelligence Lab, University of Louisville, Kentucky. He is grateful to the Department of Electrical and Computer Engineering, University of Louisville for providing office space and super-computer access.

References

[1] R. Andrews, J. Diederich and A. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge Based Systems*, vol. 8, no. 6, pp. 373–389, 1998.

[2] A.B. Tickle, R. Andrews, M. Golea and J. Diederich, "The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1057–1068, 1998.

[3] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281–294, 1988.

[4] M. Orr, J. Hallam, K. Takezawa, A. Murray, S. Ninomiya, M. Oide and T. Leonard, "Combining regression trees and radial basis function networks," *submitted to International Journal of Neural Systems*, 1999.

[5] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science*, vol. 1, no. 4, pp. 365-375, 1989.

[6] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. on Neural Networks*, vol. 8, no. 3, pp. 630-645, 1997.

[7] R. Setiono and L. C. K. Hui, "Use of a quasi-Newton method in a feedforward neural network construction algorithm," *IEEE Trans. on Neural Networks*, vol. 6, no. 1, pp. 273-277, 1995.

[8] M. C. Mozer and P. Smolensky, "Using relevance to reduce network size automatically," *Connection Science*, vol. 1, no. 1, pp. 3-16, 1989.

[9] R. Setiono and H. Liu, "Neural network feature selector," *IEEE Trans. on Neural Networks*, vol. 8, no. 3, pp. 654-662, 1997.

[10] J. M. Zurada, A. Malinowski and S. Usui, "Perturbation method for deleting redundant inputs of perceptron networks," *Neurocomputing*, vol. 14, no. 2, pp. 177-193, 1997.

[11] L. M. Belue and K. W. Bauer, "Determining input features for multilayer perceptrons," *Neurocomputing*, vol. 7, no. 2, pp. 111-121, 1995.

[12] B. Mak and R.W. Blanning, "An empirical measure of element contribution in neural networks," *IEEE Trans. on Systems, Man, and Cybernetics - Part C*, vol. 28, no. 4, pp. 561-564, 1998.

[13] J. M. Steppe and K. W. Bauer, "Improved feature screening in feedforward neural networks," *Neurocomputing*, vol. 13, no. 1, pp. 47-58, 1996.

[14] Steppe, J.M., K. W. Bauer and S.K. Rogers, "Integrated feature and architecture selection," *IEEE Trans. on Neural Networks*, vol. 7, no. 4, pp. 1007-1014, 1996.

[15] Y. Yoon, T. Guimaraes and G. Swales, "Integrating artificial neural networks with rule-based expert systems," *Decision Support Systems*, vol. 11, pp. 497-507, 1994.

[16] G.G. Towell and J.W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, no. 1, pp. 71–101, 1993.

[17] R. Blassig, "GDS: Gradient descent generation of symbolic rules," in Advances in Neural Information Processing Systems 6, Morgan Kaufmann, San Mateo, CA, pp. 1093–1100, 1994.

[18] S. Sestito and T. Dillon, *Automated Knowledge Acquisition*, Prentice Hall, 1994.

[19] I.A. Taha and J. Ghosh, "Symbolic interpretation of artificial neural networks," *IEEE Transactions of Knowledge and Data Engineering*, vol. 11, no. 3, pp. 448–462, 1998.

[20] R. Setiono, "Extracting rules from neural networks by pruning and hidden-unit splitting," *Neural Computation*, vol. 9, no. 1, pp. 205–225, 1997.

[21] R. Setiono and H. Liu, "Symbolic representation of neural networks," *IEEE Computer*, vol. 29, no. 3, pp. 71–77, 1996.

[22] A. Gupta, S. Park and S. M. Lam, "Generalized analytic rule extraction for feedforward neural networks," *IEEE Transactions of Knowledge and Data Engineering*, vol. 11, no. 6, pp. 985–991, 1998.

[23] R. Setiono and W.K. Leow, "FERNN: An algorithm for fast extraction of rules from neural networks," *Applied Intelligence*, vol. 12, no. 1/2, pp. 15–25, 2000.

[24] G. P. J. Schmitz, C. Aldrich and F.S. Gouws, "ANN-DT: An algorithm for extraction of decision trees from artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1392–1402, 1999.

[25] R. Setiono, "A penalty function approach for pruning feedforward neural networks," *Neural Computation*, vol. 9, no. 1, pp. 185–204, 1997.

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

[27] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization*, Academic Press, London and New York, 1981.

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

[29] SAS Technical Report A-102, SAS Regression Applications, SAS Institute Inc. Cary, NC, USA.

[30] E. Gelenbe, Z. H. Mao and Y. -D. Li, "Function approximation with spike random networks," *IEEE Trans. on Neural Networks*, vol. 10, no. 1, pp. 3-9, 1999.

[31] D. R. Hush and B. Horne, "Efficient algorithms for function approximation with piecewise linear sigmoidal networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1129–1141, 1998.

[32] T. Y. Kwok, and D. .Y. Yeung, "Objective functions for training new hidden units in constructive neural networks," *IEEE Trans. on Neural Networks*, vol. 8, no. 5, pp. 1131-1148, 1997.

[33] N. K. Treadgold and T. D. Gedeon, "Exploring constructive cascade networks," *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1335–1350, 1999.

[34] E. Frank, L. Trigg, G. Holmes and I. H. Witten, "Naive Bayes for regression," *Machine Learning*, vol. 41, no. 1, pp. 5–26, 2000.

[35] Y. Wang and I. H. Witten, "Induction of model trees for predicting continuous classes," in *Proc. of the Poster Papers of the European Conference on Machine Learning.* Prague: University of Economics, Faculty of Informatics and Statistics, 1997.

[36] R. Quinlan, "Learning with continuous classes," in *Proc. of the Australian Joint Conference on Artificial Intelligence*, pp. 343-348, Singapore, 1992.