GENERATING CONCISE SETS OF LINEAR REGRESSION RULES 
FROM ARTIFICIAL NEURAL NETWORKS

RUDY SETIONO and ARNULFO AZCARRAGA 
School of Computing 
National University of Singapore 
3 Science Drive 2 
Singapore 117543

Received (received date) 
Revised (revised date) 

Neural networks with a single hidden layer are known to be universal function approximators. However, due to the complexity of the network topology and the nonlinear transfer function used in computing the hidden unit activations, the predictions of a trained network are difficult to comprehend. On the other hand, predictions from a multiple linear regression equation are easy to understand but are not accurate when the underlying relationship between the input variables and the output variable is nonlinear. We have thus developed a method for multivariate function approximation which combines neural network learning, clustering and multiple regression. This method generates a set of multiple linear regression equations using neural networks, where the number of regression equations is determined by clustering the weighted input variables. The predictions for samples of the same cluster are computed by the same regression equation. Experimental results on a number of real-world data demonstrate that this new method generates relatively few regression equations from the training data samples. Yet, drawing from the universal function approximation capacity of neural networks, the predictive accuracy is high. The prediction errors are comparable to or lower than those achieved by existing function approximation methods.

Keywords: Machine Learning, Neural Networks, Regression, Clustering.

1. Introduction

Predicting a real-valued or continuous target feature is a research problem that has been attracting a great deal of interest in the machine learning community in recent years. Some of the methods proposed to solve this problem make use of Quinlan’s popular C4.5 ¹ algorithm that generates decision trees for classification. However, before applying this classification algorithm for functional prediction, one would normally need to discretize the continuous target feature, hence transforming the continuous regression task into one of categorical classification. Once the decision tree for classification has been generated, a predictor for the original continuous
feature is constructed at each leaf node. The target values of all samples that fall into the same leaf node would be approximated by the node’s predictor. The prediction is given either as a constant value, which is the average of the target values of all training samples that fall into the node, or as a linear regression function of the input attributes.

In the RECLA system\(^2\), a user can choose from one of the three methods for dividing the continuous target feature into subintervals: (1) equally probable intervals, where each subinterval contains the same number of data samples; (2) equal width interval; and (3) k-means clustering, where the sum of the distances of all data samples in an interval to its center-of-gravity (centroid) is minimized. The wrapper approach is employed for determining the number of subintervals that would give the best predictive accuracy\(^3\).

The Relative Unsupervised Discretization (RUDE) algorithm\(^4\) is a more recent method for discretizing the continuous attributes of a data set. RUDE discretizes the continuous target feature as well as all continuous input variables. A key component of this algorithm is a clustering algorithm which groups values of the target feature into subintervals that are characterized by similar values of some input attributes. Once the variables have been discretized, C4.5 is applied for solving the original regression problem. The experimental results on five benchmark problems show that when compared to the trees from the data sets discretized using the equal width interval and the k-means clustering methods, the decision trees generated using RUDE-discretized data sets have fewer nodes but lower predictive accuracy.

The main difficulty in casting the problem to predict continuous feature as a classification problem lies in the division of the interval of the continuous feature into a number of subintervals. Too many subintervals result in smaller deviations of the predictions within each subinterval. However, many subintervals translate into many classes in the resulting classification problem which would then degrade the prediction accuracy of the classification tree. Limiting the number of subintervals to generate a smaller classification tree may lead to a deterioration in the overall accuracy. This is due to the fact that the individual predictors now have to approximate the target feature values for samples that are quite “different” from one another but which still fall under the same leaf.

Several approaches have been developed which modify C4.5 to make it capable of handling continuous targets directly without the need for discretization. The “model tree” algorithm of Quinlan\(^5\) and its improved reimplementation M5\(^6\) are two such approaches. They combine the conventional C4.5 decision tree induction algorithm with the multiple linear regression method for prediction.

We propose an alternative approach using rules extracted from feedforward neural networks. A feedforward neural network with a single hidden layer can approximate any given continuous function on any compact subset with arbitrary precision provided that there is a sufficient number of units in the hidden layer\(^7\). In practice, such neural networks have been shown to yield good predictions for a wide range of real-world function approximation problems\(^8\). However, one major drawback often
associated with neural networks used for classification and regression is their lack of explanation power. There is no information on the structure of the data and nor on the underlying relationship between the input variables and the target feature. The method that we propose in this paper attempts to overcome this drawback.

We propose a method that extracts if-then rules from trained feedforward neural networks, where each rule corresponds to a linear regression formula for a subset of the input environment. The if-part of each rule is a condition statement representing the criteria for forming the input subset. The then-part is the corresponding linear equation that computes the target feature for those data points in the input subset defined by the if-part.

Our earlier work on neural network training and pruning for regression shows that for many problems, the number of remaining hidden units after pruning is relatively small. This has convinced us that it is feasible to extract from such networks useful, concise, and comprehensible explanation about the prediction. Our proposed method provides the explanation in terms of multiple linear regression rules that approximate the neural network output. It clusters the input data and generates a multiple linear regression equation for each data cluster. Before clustering is started, the input space of the data is transformed into one of a lower dimension. This transformation is achieved by making use the weights of the network connections between the input units and the hidden units. Our empirical results show that for many problems, it is possible to cluster the data into a small number of clusters, and to predict new samples using multiple linear regression equations while at the same time preserve the high predictive accuracy of the neural network.

The next section describes our neural network training and pruning method for function approximation. In Section 3 we present our method to generate multiple regression equations to predict the continuous target feature. In Section 4 we present our results and compare them with those from other methods for regression. Finally, in Section 5 we conclude the paper.

2. Network training and pruning

We divide the available data samples \((\mathbf{F}, y^i), i = 1, 2, \ldots\), where the input \(\mathbf{F} \in \mathbb{R}^N\) and the continuous target feature \(y^i \in \mathbb{R}\), randomly into the training set, the cross-validation set and the test set. A network with a single hidden layer consisting of \(H\) units (Figure 1) is trained so that the sum of squared errors of the training samples augmented with a penalty term is minimized. The augmented error function \(E(\mathbf{w}, \mathbf{v})\) is defined as follows

\[
E(\mathbf{w}, \mathbf{v}) = \sum_{i=1}^{K} (\hat{y}^i - y^i)^2 + P(\mathbf{w}, \mathbf{v})
\]

\[
P(\mathbf{w}, \mathbf{v}) = \epsilon_1 \left( \sum_{m=1}^{H} \frac{w_m^2}{1 + w_m^2} + \sum_{m=1}^{H} \frac{v_m^2}{1 + v_m^2} \right) +
\]
Generating Neural Network Regression Rules

\[ c_2 \left( \sum_{m=1}^{H} \sum_{e=1}^{N} u_{me}^2 + \sum_{m=1}^{H} v_m^2 \right) \] (2)

where \( K \) is the number of samples in the training data set, \( c_1 \) and \( c_2 \) are positive penalty parameters. The prediction of the neural network \( \hat{y} \) is computed as the linear combination of the hidden unit activations given \( \mathbf{I} \), the input attribute values of sample \( i \):

\[ \hat{y}_i = \sum_{m=1}^{H} h \left( (\mathbf{I})^T \mathbf{w}_m \right) v_m + \tau, \] (3)

\( \mathbf{w}_m \in \mathbb{R}^N \) is the vector of network weights from the input units to hidden unit \( m \), \( w_{me} \) is its \( e \)-th component, \( v_m \in \mathbb{R} \) is the network weight from hidden unit \( m \) to the output unit, \( h(\xi) \) is the hyperbolic tangent function \((e^\xi - e^{-\xi})/(e^\xi + e^{-\xi})\), and \((\mathbf{I})^T \mathbf{w}_m \) is the scalar product of \( \mathbf{I} \) and \( \mathbf{w}_m \), and \( \tau \) is the output unit's bias.

The penalty term \( P(\mathbf{w}, \mathbf{v}) \) is added to the sum of squared errors of the prediction to make it more likely for network units to be pruned. Extensive empirical experiments demonstrate that the addition of a penalty term to the error function improves the generalization ability of the resulting networks \(^{10} \). We have opted to incorporate the penalty term \( P(\mathbf{w}, \mathbf{v}) \) that we have used in conjunction with a neural network pruning algorithm \(^{11} \). Employing this penalty term enabled us to obtain pruned networks that are smaller than the networks obtained by other network pruning algorithms. In addition to better generalization, more concise symbolic classification rules can be extracted from smaller networks than more complex ones. For the function approximation problem we are addressing in this paper, we are able to generate fewer regression rules by pruning redundant hidden units from the network.

We minimize the error function \( E(\mathbf{w}, \mathbf{v}) \) by applying the BFGS method \(^{12,13} \) due to its fast convergence rate. After the network has been trained, irrelevant and
redundant hidden units and input units are removed from the network by applying
the algorithm N2PFA (Neural Network Pruning for Function Approximation)\(^9\). This is an iterative algorithm which 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 a significant deterioration in the network's
accuracy on this set of samples, then the unit will be removed.

3. Regression rules generation

The research question that we are addressing in this paper is the following:
Given a neural network that has been trained and pruned with good predictive
accuracy, is it possible to replace the network prediction by a set of multiple linear
regression equations without compromising the accuracy of the prediction?

Before we present our method for regression rule generation from neural net-
works, we re-emphasize our motivation in attempting to generate such rules. For
some applications, one may only be interested in getting accurate predictions. In
this situation, the pruned neural networks without the regression equations would
suffice. In other applications, however, one may want to know more about the rela-
tionship between the input and output variables. For such cases, we need to extract
rules from the network that are easily interpretable.

![Function approximation using linear and non-linear regression methods.](image)

Figure 2: Function approximation using linear and non-linear regression methods.

Figure 2 illustrates the approach we are taking in improving the transparency
of the neural network prediction systems. Suppose we are given a problem with a
target feature \(Y\) whose relationship to input \(X\) is governed by the non-linear curve
drawn as a solid, bold curve. For illustration purposes, we assume a single target
variable and a single input variable. A straightforward linear regression method
would try to fit a single linear equation, depicted by a straight dotted line, which
would suffer in terms of predictive accuracy. But since neural networks are universal
function approximators, they would be able to generate some nonlinear mapping, depicted in the figure as a dotted curve, that would deviate minimally from the actual (nonlinear) curve. Other non-linear regression methods would also generate similar nonlinear mappings that would minimize deviation from the actual curve. As we show in the experiments that follow, our method is comparable to these other function approximation methods in terms of predictive accuracy. In addition, our method provides explanation by subsequently converting the trained neural network into a set of easily interpretable rules.

![Diagram](image)

Figure 3: Converting neural network prediction to a set of multiple linear regression functions.

As shown in Figure 3, we break-up the input space into several subsets and generate single linear equations in each subset in such a way that the combination of these linear equations would approximate the neural network's nonlinear mapping. Once this is done, we have actually converted a neural network into a set of if-then rules. The if-part is a condition statement representing the criteria for forming the input subset. The then-part is the corresponding linear regression formula that computes the target feature for those data points in the input subset defined by the if-part. As a result, we obtain a prediction system that has very high accuracy rates, and are comprehensible and easy to interpret. If desired, parametric statistical tests can also be conducted to measure the significance of the linear models and the individual relationship between an input variable and the target feature.

Our neural network rule generation technique consists of two steps. The first step is to cluster the input attributes of the data. The sample data are grouped into clusters which are formed based on the proximity of the transformed data samples. The transformed data are in turn computed using the weights of the network connections from the input units to the hidden units. The second step is to generate a multiple regression equation for each data cluster.
3.1. Data clustering

![Diagram](image)

Figure 4: Approximation of the hyperbolic tangent function \( h(x) \) by a piecewise linear function.

As the mapping of the neural network is nonlinear, it is very unlikely that we can replace it by just one linear regression equation. Hence, we need to group the samples into clusters and fit the target feature of samples in different clusters using different regression equations.

The data clustering process itself consists of two steps. The first is to approximate the hyperbolic tangent activation function of the hidden unit by a 5-piece linear function. Since the function is antisymmetric, we show in Figure 4 the approximation only for the nonnegative value of \( x \in [0, x_m] \). The three dotted lines are given by the equations:

\[
L(x) = \begin{cases} 
  x & \text{if } 0 \leq x \leq x_0 \\
  h'(x_1)(x - x_1) + h(x_1) & \text{if } x_0 < x < x_2 \\
  h'(x_m)(x - x_m) + h(x_m) & \text{if } x \geq x_2 
\end{cases}
\]  

The underlying idea for this approximation is to find the point \( x_1 \) such that the total area of the triangle and the two trapezoids is minimized:

\[
\min_{x_1} A = \frac{1}{2} \left[ x_0^2 + (x_0 + y_2)(x_2 - x_0) + (y_2 + h(x_m))(x_m - x_2) \right] 
\]  

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

\[
x_0 = \frac{h(x_1) - x_1 h'(x_1)}{1 - h'(x_1)} 
\]  

\[
x_2 = \frac{x_1 h'(x_1) - x_m h'(x_m) - h(x_1) + h(x_m)}{h'(x_1) - h'(x_m)} 
\]
Generating Neural Network Regression Rules

\[ y_2 = \frac{T_1 + T_2 + T_3}{h'(x_1) - h'(x_m)} \]  
\[ T_1 = h'(x_1)h'(x_m)(x_1 - x_m) \]  
\[ T_2 = h'(x_1)h(x_m) \]  
\[ T_3 = h'(x_m)h(x_1) \]  

The bisection method\textsuperscript{14} for one-dimensional optimization problems is employed to find the optimal value of \( x_1 \). The total error \( E \) of estimating \( h(x) \) by this linear approximation is computed to be

\[ E = \int_0^{x_m} (L(x) - h(x)) \, dx \]  
\[ \quad \to 0.071169 \text{ as } x_m \to \infty. \]

Recall that the hidden activation of sample \( \mathbf{f} \) is computed as the hyperbolic tangent of the weighted inputs \( (\mathbf{f})^T \mathbf{w}_m \), where \( \mathbf{w}_m \) is the network connection weights from the input units to hidden unit \( m, m = 1, 2, \ldots, H \). We assign a label \( (L_1, L_2, \ldots, L_H) \) for each training sample according to the values of its weighted inputs:

\[
L_m = \begin{cases} 
1 & \text{iff } (\mathbf{f})^T \mathbf{w}_m \leq -x_2 \\
2 & \text{iff } -x_2 \leq (\mathbf{f})^T \mathbf{w}_m < -x_0 \\
3 & \text{iff } -x_0 \leq (\mathbf{f})^T \mathbf{w}_m < x_0 \\
4 & \text{iff } x_0 \leq (\mathbf{f})^T \mathbf{w}_m < x_2 \\
5 & \text{iff } (\mathbf{f})^T \mathbf{w}_m \geq x_2 
\end{cases}
\]  

for \( m = 1, 2, \ldots, H. \)

After all the samples have been labeled, we begin the actual clustering. Samples with the same label are collected into the same group and the first cluster is the group having the largest number of samples. Whether other groups can be joined into this cluster depends on the distance between the samples in the other groups and the cluster’s centroid (center-of-gravity). If the distance is larger than a pre-specified value, then a new cluster is formed. We implemented a variant of the centroid clustering method\textsuperscript{15}. The outline of this method is as follows.

**Program cluster**

**Given:** The weighted training inputs \( (\mathbf{f})^T \mathbf{w}_m, \ i = 1, 2, \ldots, K, m = 1, 2, \ldots, H \) and a positive threshold \( \delta_{Max} \).

**Objective:** Cluster the weighted inputs such that the maximum distance between any pair of clusters is less than \( \delta_{Max} \).

**Step 1.** Let \( x_m \) be the largest weighted input in magnitude.

**Step 2.** Compute \( x_0 \) and \( x_2 \) according to Equations (6) and (7).

**Step 3.** Label each input according to Equation (10) and group the samples according to their label. Let \( G_k \) be the group with the largest number of samples. Samples in \( G_k \) form the first cluster and set the number of clusters \( NC = 1. \)
Step 4. For cluster \(i = 1, 2, \ldots NC\) compute the distance between the cluster and all groups. Let \(G_j\) be the group with the maximum distance, \(\delta_j\) to its nearest cluster. If \(\delta_j > \delta_{\text{max}}\), then \(NC = NC + 1\), let samples in \(G_j\) form the new cluster, and repeat Step 4.

Step 5. For each group of samples, identify its nearest cluster and assign all samples in the group to this cluster.

In our implementation, we computed the distance between a cluster and a group of samples as the Euclidean distance between their centroids. The center of an \(H\) dimensional cluster or group is computed simply by averaging each of the \(H\) components of the weighted input vectors of all samples that belong to the same cluster/group. In Step 5, when a group of samples are assigned to a cluster, the cluster centroid is recomputed accordingly.

We attempt to reduce the number of groups of activation values, and therefore the number of multiple linear regression equations, by this clustering method. This is an extension of our previous work, where a regression equation is generated for each group of activation values. By further clustering these activation values, we are able to simplify the rule set.

3.2. Computing the regression equation

Prediction of the target feature for all samples in the same cluster is accomplished by the same multiple linear regression equation. Let \(N\) be the number of relevant input attributes of the data as determined by the pruned neural network*. Now let \((\mathbf{f}_i, y_i), i = 1, 2, \ldots K_j\) be the samples in cluster \(j\) and their corresponding target. The multiple linear regression model that predicts the target values is of the form

\[
\hat{y}_i = \beta_0 + \beta_1 f_{i1} + \ldots + \beta_N f_{iN}, \quad i = 1, 2, \ldots K_j
\]

or in matrix notation

\[
\hat{\mathbf{Y}} = \mathbf{X\beta}
\]

where \(\hat{\mathbf{Y}} \in \mathbb{R}^{K_j} \) and \(\mathbf{B} \in \mathbb{R}^{N+1}\). The matrix \(\mathbf{X} \in \mathbb{R}^{K_j \times (N+1)}\) comprises the values of the input attributes \(\mathbf{f}\) augmented by a column of ones. The least squares approximates of the regression coefficients are computed by finding the vector \(\mathbf{B}\) that satisfies the normal equation for the ridge regression problem

\[
(\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I})\mathbf{B} = \mathbf{X}^T\mathbf{Y}
\]

where \(\lambda > 0\) and \(\mathbf{I}\) is the identity matrix. Ridge regression is used instead of the normal least squares regression because for some clusters of the data, the matrix \(\mathbf{X}^T\mathbf{X}\) may not be full-ranked.

*After pruning, only a subset of the original input attributes of the data is usually found to be relevant for prediction. For simplicity of notation, we do not change our symbol for the dimensionality of the input variables before and after network pruning. The same applies for \(H\), which denotes the number of hidden units before and after pruning.
4. Experimental results

<table>
<thead>
<tr>
<th>Name</th>
<th>#samples</th>
<th>Attributes</th>
<th>NN inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Abalone</td>
<td>4177</td>
<td>1D, 7C</td>
<td>9</td>
</tr>
<tr>
<td>2. Auto-mpg</td>
<td>392</td>
<td>3D, 4C</td>
<td>25</td>
</tr>
<tr>
<td>3. Housing</td>
<td>506</td>
<td>1D, 12C</td>
<td>13</td>
</tr>
<tr>
<td>4. Machine</td>
<td>209</td>
<td>6C</td>
<td>6</td>
</tr>
<tr>
<td>5. Servo</td>
<td>167</td>
<td>4D</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 1: Test data sets. D = discrete attribute, C = continuous attribute

Experiments were conducted to assess the effectiveness of our proposed method relative to other methods for function approximation. They were performed on five publicly available data sets listed in Table 1.

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

The same experimental settings were used for all problems. The networks started with eight hidden units and the penalty parameters \( c_1 \) and \( c_2 \) were set to 0.5 and 0.05, respectively. Network pruning was terminated if removal of a hidden unit or an input unit caused the errors of the resulting network on the cross validation set to increase by more than 10% from its lowest value. The target values were normalized onto the interval \([0, 16]\). 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 input 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.

The errors of the prediction were computed as follows:

\[
\text{Root Mean Squared Errors (RMSE)} = \sqrt{\frac{1}{K} \sum_{i=1}^{K} (\hat{y}_i - y_i)^2} \tag{14}
\]

\[
\text{Mean Absolute Error (MAE)} = \frac{1}{K} \sum_{i=1}^{K} |\hat{y}_i - y_i| \tag{15}
\]

\(^{1}\)The data have been downloaded from http://www.ncc.up.pt/\~ltorgo/Research/.
Table 2: Summary of the results from neural network training and pruning.

where $\hat{K}$ is the number of samples in the test set. The number of attributes indicates the input variables that were still present after network pruning. For example, for the Servo data set, of the 19 neural network inputs for the encoded data, 14.72 were still present in the networks on average. In terms of the original input attributes of the data, all 4 attributes still have one or more of their encoded inputs left unpruned.

Table 3: The number of regression rules obtained by varying the maximum allowable distance between clusters.

Table 4: The error rates (MAE) obtained by varying the maximum allowable distance between clusters. The lowest errors are shown in **bold**.

The number of regression rules generated from a network depends on the maximum allowable distance between clusters. We experimented with three different maximum values: (1) $\delta_{Max} = \sqrt{0.5 \times H}$, (2) $\delta_{Max} = \sqrt{H}$, and (3) $\delta_{Max} = \sqrt{2 \times H}$, where $H$ is the number of hidden units in the pruned network. Note that $H$ is also the dimensionality of the weighted input attributes. The results are summarized in Table 3 and Table 4. The MAEs in this table indicate that the accuracy of
Generating Neural Network Regression Rules

the regression rules does not differ much from the accuracy of the pruned neural networks for problem Servo. For Machine, the MAE of the regression rules is twice as high as that of the pruned networks. The deterioration in predictive accuracy can be reduced by allowing more clusters. For example, when we set the maximum distance between clusters to $\delta_{MAX} = \sqrt{0.25 \times H}$, the MAE drops to $12.28 \pm 1.26$ but the average number of rules increases to $3.00 \pm 0.0$. 

<table>
<thead>
<tr>
<th>Data set</th>
<th>NN-regression</th>
<th>KRTrees</th>
<th>kNNTrees</th>
<th>LinearTrees</th>
<th>RUDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abalone</td>
<td>1.69 ± 0.01</td>
<td>1.7 ± 0.1</td>
<td>1.7 ± 0.1</td>
<td>1.8 ± 0.1</td>
<td>2.13 ± 0.09</td>
</tr>
<tr>
<td>Auto-mpg</td>
<td>2.07 ± 0.11</td>
<td>2.4 ± 0.4</td>
<td>2.3 ± 0.4</td>
<td>18.0 ± 5.6</td>
<td>3.96 ± 0.34</td>
</tr>
<tr>
<td>Housing</td>
<td>2.70 ± 0.16</td>
<td>2.8 ± 0.5</td>
<td>2.9 ± 0.4</td>
<td>3.9 ± 2.7</td>
<td>4.07 ± 0.34</td>
</tr>
<tr>
<td>Machine</td>
<td>15.04 ± 1.52</td>
<td>31.2 ± 15.1</td>
<td>31.5 ± 14.7</td>
<td>35.7 ± 11.7</td>
<td>51.49 ± 16.25</td>
</tr>
<tr>
<td>Servo</td>
<td>0.25 ± 0.02</td>
<td>0.4 ± 0.2</td>
<td>0.4 ± 0.2</td>
<td>0.9 ± 0.2</td>
<td>0.44 ± 0.16</td>
</tr>
</tbody>
</table>

Table 5: MAEs of NN rules and those of other regression methods.

The comparison of our method (NN-regression) with other regression methods is summarized in Table. The MAE of three variants of a regression tree generating method called HTL 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 KRT 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. The last column of the table shows the results from RUDE. 

The results of the experiments lead us to the following conclusions:

1. The prediction errors of the pruned neural networks are lower than the errors from the other regression methods on all five test data sets.

2. The complex nonlinear mapping between the input variables and the continuous target feature of the networks can be approximated by a relatively small set of multiple linear regression equations. The prediction errors increase as a result. However, the predictions are still as accurate as those from other methods. For one of the data sets studied (Machine), the average mean absolute error of the regression equations is less than half of its nearest competitor.

3. The total number of regression equations is small even when $\delta_{MAX} = \sqrt{0.25 \times H}$. For comparison, the average tree size generated by RUDE ranges from 21.5 for the Servo data set to 65.8 for Housing data set.
5. Discussion and conclusion

As prediction systems, neural networks have been known to be robust and they have outperformed other prediction methods in numerous benchmark data sets. However, one important drawback of neural networks is that they operate as "black boxes" since little is known about how they compute for the estimated value of the target feature. Thus, even if the predictive accuracy is very high on carefully constructed experiments, many are reluctant to use this powerful technology in real-world application.

In this paper, we attempt to remedy this problem by explaining the network predictions of a continuous target feature in terms of a collection of multiple linear regression equations. In our approach, each linear equation is a local regression formula for a subset of the input environment. As a result, the neural network is effectively converted into a set of if-then rules. The if-part is a condition statement representing the criteria for forming the input subset. The then-part is the corresponding linear regression formula that computes the target feature for those data points in the input subset defined by the if-part. Hence, we are able to obtain a prediction model that achieves very high accuracy rates, and yet is comprehensible and easy to interpret.

In our method, the sample data are grouped into clusters which are formed based on the proximity of the transformed data samples. The transformed data are computed using the weights of the network connections from the input units to the hidden units. For each data cluster, the method computes the coefficients of the linear regression equation by solving the normal equation formulation for a ridge regression problem.

By converting a neural network into a set of multiple linear regression equations, we show that the relationship between an input attribute and the target feature can be easily discovered by simply inspecting the relevant regression coefficient for samples that belong in the same cluster.

Using the mean absolute error as the performance measure, the rules generated from the neural networks achieve prediction accuracies that are as good as those from the other regression methods which first discretize the continuous target values and then build classification trees. Besides achieving high predictive accuracy, however, our proposed method generates fewer regression equations for the problems we tested and hence contributes towards better interpretability for the neural networks that we have trained.

References


Generating Neural Network Regression Rules