The Mathematical Programming and the Rule Extraction from Layered Feed-forward Neural Networks

Ray Tsaih
Department of Management Information Systems, National Chengchi University, Taipei, Taiwan, e-mail: tsaih@mis.nccu.edu.tw

Hsiou-Wei Lin
Department of International Business, National Taiwan University, Taipei, Taiwan, e-mail: plin@ccms.ntu.edu.tw

Wen-Chyan Ke
Department of International Business, National Taiwan University, Taipei, Taiwan, e-mail: wenchyan@mis.nccu.edu.tw
Abstract

The mathematical programming analysis, instead of a data analysis, is proposed for identifying the convex polyhedron associated with each rule. The area depicted in the rule premise covers a convex polyhedron in the input space, and the adopted approximation function for the output value is a multivariate polynomial function of $x$, the outside stimulus input. Moreover, the mathematical programming analysis is proposed for examining the extracted rules to explore features.

Subject Areas: Mathematical programming, Neural Networks, Rule extraction

1. Extracting Regression Rules from Neural Networks

Rumelhart and his colleagues in Rumelhart et al. (1986) proposed a learning algorithm, named the Back-Propagation learning algorithm, for training layered feed-forward neural networks. Since then, varieties of Artificial Neural Networks (ANN) have been widely used in many fields. However, in many applications, it is desirable to extract knowledge or rules from the trained ANN for the purpose of exhibiting a high degree of comprehensibility of ANN and gaining a better understanding of the problem domain. More precisely, it is desirable to have a solid way of extracting rules from a well-known ANN, like layered feed-forward neural networks, and then examining these extracted rules to gain more information about the problem domain.
In literature, there are some recent studies related to extracting rules from the trained ANN. For instance, Setiono & Liu (1997), Taha & Ghosh (1999) and Zhou et al. (2000) extract rules from a trained ANN for classification problems; Setiono et al. (2002) and Saito & Nakano (2002) from a trained ANN for regression problems. In their paper (2002, page 1279), Saito and Nakano state “one future direction for knowledge-extraction technique is enabling to deal with neural networks of real-valued outputs.” Thus, in this study we focus upon developing a solid way of extracting and analyzing regression rules that deal with the real-valued outputs.

From the literature, the rules extracted from the trained ANN for regression problem most typically take the following syntax:

\[
\text{If (premise), then (action).} \tag{1}
\]

The premise is a Boolean expression that must be evaluated as true for the rule to be applied. The action clause consists of a statement or a series of statements, and is executed only when the premise is true. Specifically, the regression rules take the following syntax:

\[
\text{If } (\mathbf{x} \in \text{the } i^{th} \textbf{Area}), \text{ then } (y' = g_i(\mathbf{x})), \tag{2}
\]

where \(y'\) is the approximation of the output value \(y\). The \textbf{Area} depicted in the premise covers a region in the input space and the approximation function \(g(\mathbf{x})\) can be a piece-wise linear function (cf. Setiono et al., 2002) or a multivariate polynomial function whose power values could not be restricted to integers (cf. Saito & Nakano, 2002). This type of rules is accept-
able because of its similarity to the traditional statistical approach of parametric regression.

The treatment of many topics using traditional statistical approach aims to strip away nonessential details and to reveal the fundamental assumptions and the structure of reasoning.

To identify the premise of a single rule, Setiono et al. (2002) and Saito & Nakano (2002) focus on a way of data analysis on the training sample set or the generated sample set. The generated sample set contains data instances yielded from the trained ANN. With either training or generated sample set for extracting rules, the amount of data instances is finite, and the premise of a resulted rule covers merely discrete points, instead of an area. Impractically, to really catch the rules embedded in the trained ANN, the size of the data set should reach infinity. Besides, Setiono et al. (2002) and Saito & Nakano (2002) have not provided a solid way of examining the extracted rules to gain useful information for the problem domain.

Here we propose the mathematical programming methodology for not only identifying regression rules extracted from layered feed-forward neural networks, but also examining the extracted rules to gain useful information. The Area depicted in the rule premise covers a convex polyhedron in the input space, and the adopted approximation function \( g(x) \) is a multivariate polynomial function of \( x \). The mathematical programming analysis, instead of a data analysis, is suggested for identifying the convex polyhedron associated with each rule. The mathematical programming analysis is also used to examine the extracted rules to ex-
2. The proposed mathematical programming analysis methodology

Without losing the generality, we take the network shown in Figure 1 as an illustration of the proposed methodology. The network is a three-layer feed-forward neural network with one output node. In Figure 1, \( y \) denotes the output value of the neural network, and \( \mathbf{x}' = (x_1, x_2, \ldots, x_m) \) whereas \( x_i \) denotes the \( i \)-th outside stimulus input, with \( i \) from 1 to \( m \), where \( m \) is the amount of stimulus input. \( \mathbf{w}_j' \equiv (z_{w_{j1}}, z_{w_{j2}}, \ldots, z_{w_{jm}}) \) stands for the weights between the \( j \)-th hidden node and the input layer, with \( j \) from 1 to \( p \), where \( p \) is the amount of used hidden nodes, and \( \mathbf{w}' \equiv (z_{w_{1}}, z_{w_{2}}, \ldots, z_{w_{p}}) \) stands for the weights between the output node and all hidden nodes. \( \theta_j \) is the bias of the \( j \)-th hidden node and \( \theta \) is the bias of the output node. The activation function \( \tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}} \) is used in all hidden nodes and the linear activation function is used in the output node. That is, for the \( c \)-th input \( x_c \), the activation value of the \( j \)-th hidden node \( h_j \) and the output value \( y \) are computed as in equations (3) and (4).

\[
    h_j = \tanh(\sum_{i=1}^{m} z_{w_{ji}} x_i + z_{\theta_j}) \tag{3}
\]

\[
    y = \sum_{j=1}^{p} z_{w_{j}} h_j + z_{\theta} \tag{4}
\]
Figure 1: The feed-forward neural network with one hidden layer and one output node.

To extract comprehensible multivariate polynomial rules from the layered feed-forward neural network with the tanh(t) activation function, an approximation of the tanh(t) function is necessary. The following way of approximation is proposed. Assume that we are interested in the first and second ordered differential information. Then, the following function \( g(t) \) is proposed to approximate tanh(t):

\[
g(t) = \begin{cases} 
1 & \text{if } t \geq \kappa \\
\beta_1 t + \beta_2 t^2 & \text{if } 0 \leq t \leq \kappa \\
\beta_1 t - \beta_2 t^2 & \text{if } -\kappa \leq t \leq 0 \\
-1 & \text{if } t \leq -\kappa
\end{cases}
\]

(5)

where \((\beta_1, \beta_2, \kappa) = \arg\left( \min_{\beta_1, \beta_2, \kappa} \int_{-\kappa}^{\kappa} (\tanh(t) - g(t))^2 dt, \text{ subject to } \beta_1 \kappa + \beta_2 \kappa^2 = 1 \right)\). \( g(t) \) is continuous at boundaries of four polyhedrons \((t = \kappa, t = 0, t = -\kappa)\), because \( \lim_{t \to \kappa^-} \beta_1 t + \beta_2 t^2 = 1, \lim_{t \to 0^+} \beta_1 t - \beta_2 t^2 = 0, \lim_{t \to 0^-} \beta_1 t + \beta_2 t^2 = 0, \lim_{t \to -\kappa^+} \beta_1 t - \beta_2 t^2 = -1 \).

With the numerical analysis of Sequential Quadratic Programming (cf. The MathWorks, Inc., 2002), we obtain \( \beta_1 \approx 1.0020101308531, \beta_2 \approx -0.251006075157012, \kappa \approx \ldots \)
separate polyhedrons in the space. For example, if \( x \in \{ x: -2 \theta_j \leq \omega^j_x \leq \kappa - 2 \theta_j \} \), then \( \tanh(\omega^j_x + 2 \theta_j) \) is approximated with \( \beta_j x_1 + \beta_j x_2 + (\beta_j + 2 \beta_j) t_j + \beta_j t_j^2 \). Thus, a comprehensible regression rule associated with a trained feed-forward neural network with \( p \) hidden nodes is like:

\[
\text{If } x \in \{ x: -2 \theta_j \leq \omega^j_x \leq \kappa - 2 \theta_j \text{ for all } j \},
\]

\[
\text{then } y' = \omega_1 + \sum_{j=1}^{p} \omega_j \beta_j x_1 + \beta_j x_2 + (\beta_j + 2 \beta_j) t_j + \beta_j t_j^2.
\]

To have a better representation of the area depicted in the rule premise, let us further introduce some notations. For the \( j \)-th hidden node, set \( \omega \) be 1 when the situation \( \kappa - 2 \theta_j \leq t_j \) holds; 2 when \( -2 \theta_j \leq t_j \leq \kappa - 2 \theta_j \) holds; 3 when \( -\kappa - 2 \theta_j \leq t_j \leq -2 \theta_j \) holds; or 4 when \( t_j \leq -\kappa - 2 \theta_j \) holds. Also, set \( \omega_1 = \omega^j_x, \omega_2 = \begin{bmatrix} 2 \omega^j_x \ \omega_3 = \begin{bmatrix} 2 \omega^j_x \\ -2 \omega^j_x \end{bmatrix}, \omega_4 = \begin{bmatrix} 2 \omega^j_x \\ -2 \omega^j_x \end{bmatrix}, \omega_5 = -2 \omega^j_x, \omega_6 = \kappa - 2 \theta_j, \omega_7 = (-2 \theta_j, \omega_j) - \kappa_j, \omega_8 = (-2 \theta_j - \kappa - 2 \theta_j), \omega_9 = \kappa + 2 \theta_j, g_0(t_j) = 1, g_1(t_j) = \beta_j + 2 \beta_j t_j + \beta_j t_j^2, g_2(t_j) = \beta_j x_1 + \beta_j x_2 + (\beta_j + 2 \beta_j) t_j + \beta_j t_j^2, \text{ and } g_3(t_j) = -1. \) Let \( i = (t_1, t_2, ..., t_j) \) with \( t \in \{1, ... , 4\} \).
2, 3, 4} for every $j$, $\mathbf{A}_i \equiv \begin{bmatrix} \mathbf{w}_{1i} \\ \mathbf{w}_{2i} \\ \vdots \\ \mathbf{w}_{pi} \end{bmatrix}$, and $\mathbf{b}_i \equiv (\mathbf{v}_{1i}, \mathbf{v}_{2i}, \ldots, \mathbf{v}_{pi})^t$. Thus, for example, the premise $\mathbf{x} \in \{x: -\theta \leq w_j^i x \leq \kappa - \theta \forall j = 1, 2, \ldots, p\}$ can be expressed as $\mathbf{x} \in \{x: \mathbf{A}_i \mathbf{x} \geq \mathbf{b}_i\}$ with $i = 2$ for every $j$.

In practice, the independent variables may have some constraints, and they are usually linear as shown in equation (7), where $a_{ij}$ and $b_{ij}$ are given constants. Let $\mathbf{A}_i \equiv \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$ and $\mathbf{b}_i \equiv (b_{1i}, b_{2i}, \ldots, b_{ni})^t$. Thus equation (7) can be expressed as $\mathbf{A}_i \mathbf{x} \geq \mathbf{b}_i$.

\[ a_{ij} x_1 + a_{2j} x_2 + \ldots + a_{mj} x_m \geq b_{ij}, i = 1, 2, \ldots, n \]  

(7)

In sum, there are $4^p$ polyhedrons in the input space, and the corresponding output value $y$ is approximated with a multivariate polynomial function in each polyhedron. The potential rule associated with the $i$-th polyhedron $\{x: \mathbf{A}_i \mathbf{x} \geq \mathbf{b}_i\}$ is similar to the one shown in equation (8), where $\mathbf{A}_i \equiv \begin{bmatrix} \mathbf{A}_i \\ \mathbf{b}_i \end{bmatrix}$.

\[ y' = -\theta + \sum_{j=1}^{p} w_j g_j(t_j) \]  

(8)

However, some of these $4^p$ potential rules are null, and the Simplex algorithm (cf. Hillier & Lieberman, 2001) can be applied to identify the null rules. If $\{x: \mathbf{A}_i \mathbf{x} \geq \mathbf{b}_i\}$ is empty, the rule associated with the $i$-th polyhedron is null. $\{x: \mathbf{A}_i \mathbf{x} \geq \mathbf{b}_i\}$ is a convex polyhedral set in
the input space because $A_i x \geq b_i$ consists of linear inequality constraints. Furthermore, $\{x: A_i x \geq b_i\}$ is non-empty if the linear programming (LP) problem (9) has an optimal solution.

In other words, if LP problem (9) has an optimal solution, then the rule associated with the $i$-th polyhedron $\{x: A_i x \geq b_i\}$ exists. Otherwise, that rule fails to exist. The process of extracting existing rules is summarized in Table 1.

\[
\text{Minimize: } \text{constant}
\]

Subject to: $A_i x \geq b_i$  \hspace{1cm} (9)

**Table 1: The process of rule extraction.**

<table>
<thead>
<tr>
<th>Step 1: Input all weights and biases of the trained feed-forward neural network to form $A_i$ and $b_i$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2: Input the constraints associated with the independent variables to form $A$ and $b$.</td>
</tr>
<tr>
<td>Step 3: For each of the $4^p$ potential rules, says</td>
</tr>
</tbody>
</table>

If $A_i x \geq b_i$, then $y' = \theta + \sum_{j=1}^{p} w_j g_j(t)$

where $A_i \equiv \begin{bmatrix} 1 A_i \\ 2 A_i \end{bmatrix}$ and $b_i \equiv \begin{bmatrix} 1 b_i \\ 2 b_i \end{bmatrix}$, examine whether the corresponding LP problem has an optimal solution. If the corresponding LP problem has an optimal solution, then the rule exists. Otherwise, that rule fails to exist.

Minimize: $\frac{\partial y'}{\partial x_k}$

Subject to: $A_i x \geq b_i$  \hspace{1cm} (10)

Maximize: $\frac{\partial y'}{\partial x_k}$

Subject to: $A_i x \geq b_i$  \hspace{1cm} (11)

Features embedded in the feed-forward neural network can be explored via further analyzing the existing rules. Take as an illustration the exploration of the relation between $y'$
and the $k$-th independent variable $x_k$. The null hypothesis $H_0$ states there is no relation between $y'$ and $x_k$, while an alternative hypothesis $H_1$ argues \( \frac{\partial y'}{\partial x_k} > 0 \), and another alternative hypothesis $H_2$ is that \( \frac{\partial y'}{\partial x_k} < 0 \). For the $i$-th polyhedron \( \{ x : A_i x \geq b_i \} \), \( \frac{\partial y'}{\partial x_k} \bigg|_{x \in [A_i x \geq b_i]} > 0 \) if the minimal solution to the optimization problem (10) is greater than zero, and \( \frac{\partial y'}{\partial x_k} \bigg|_{x \in [A_i x \geq b_i]} < 0 \) if the maximal solution to the optimization problem (11) is less than zero. Because of the approximation stated in equation (5), \( \frac{\partial y'}{\partial x_k} = \sum_{j=1}^{w \beta_j} \hat{g}_j (t_j) \) and

\[
\frac{\partial g_j (t_j)}{\partial x_k} = \begin{cases} 
0 & \text{if } t_j > \kappa - \theta_j \\
2 w_k (\beta_1 + 2 \beta_2, \theta_j) + 2 w_k \beta_2 t_j & \text{if } -\theta_j < t_j < \kappa - \theta_j \\
2 w_k (\beta_1 - 2 \beta_2, \theta_j) - 2 w_k \beta_2 t_j & \text{if } -\kappa - \theta_j < t_j < -\theta_j \\
0 & \text{if } t_j < -\kappa - \theta_j \end{cases}
\]

(12)

Thus, the optimization problems (10) and (11) are LP problems, and accordingly, they can be solved via the Simplex algorithm.

As for identifying features such as \( \frac{\partial^2 y'}{\partial x_k^2} < 0 \) if $x_i > x_j$, the proposed method is as follows. For instance, let \( \frac{\partial^2 y'}{\partial x_k^2} \) be a negative constant at the [3,3,3,3]-th polyhedron.

Thus \( \frac{\partial^2 y'}{\partial x_k^2} > 0 \) if $x_i < x_j$. Thus \( \frac{\partial^2 y'}{\partial x_k^2} < 0 \) if $x_i > x_j$.
3. The application to bond-pricing

We adopt a case of bond-pricing to examine the proposed methodology. The domain knowledge with respect to the bond pricing model has been well established and thus serves to help investigating the learning process. In equation (14), bond price at time $t$, denoted by $P_t$, is governed by four factors: (1) $r_t$, the market rate of interest at time $t$; (2) $F$, the face value of the bond, which generally equals 100; (3) $T_0$, term to maturity at time $t = 0$; and (4) $r_c$, the rate of periodic coupon payment.

\[
P_t = \sum_{k=1}^{n} \frac{Fr_k}{(1 + r_t)^{k-t}} + \frac{F}{(1 + r_t)^{n-t}} \tag{14}
\]

With the assumption of one coupon payment per year (that is, coupon payments are made every 12 months), there exist five well-known theorems with respect to bond prices. The theorems have been derived as follows: (Malkiel, 1962)

1. If a bond's market price increases, then its yield must decrease; conversely, if a bond's market price decreases, then its yield must increase. That is, $\frac{\partial P}{\partial r_t} < 0$.

2. If a bond's yield does not change over its life, then its discount or premium will decrease as its life gets shorter. That is, $\frac{\partial^2 P_t}{\partial T_t \partial r_t} < 0$, where $T_t \equiv T_0 - t$ is the term to maturity at time $t$.

3. If a bond's yield does not change over its life, then the magnitude of its discount or premium will decrease at an increasing rate as its life gets shorter. That is,
\[
\begin{align*}
\frac{\partial^2 P}{\partial T_t^2} &< 0 \quad \text{if } r_c > r_t \\
\frac{\partial^2 P}{\partial T_t^2} &> 0 \quad \text{if } r_c < r_t
\end{align*}
\]

4. A decrease in a bond's yield will raise the bond price by an amount which is greater in magnitude than the corresponding fall in the bond price, and the fall will occur if there is an equal-sized increase in the bond's yield. That is, \( \frac{\partial^2 P}{\partial r_t^2} > 0 \).

5. The magnitude of a bond's price change due to a change in its yield will be higher if its coupon rate is higher. That is, \( \frac{\partial^2 P}{\partial r_c \partial r_t} < 0 \). (Note: This theorem does not apply to bonds with a life of one year or to bonds that have no maturity date, known as consols, or perpetuities.)

We generate the training samples from a hypothetical period of 80 trading days, during which we derive \( r_t \) from a normal random number generator of \( N(\mu = 2\%, \, \sigma^2 = (0.1\%)^2) \).

Then we use six hypothetical combinations of terms to maturity and contractual interest rate as depicted in Table 2, and generate the data with eighty measures of \( t \), with \( t = 1/80, 2/80, \ldots, 80/80 \) via equation (14). Thus we have 480 training samples with input variables \( T_t, r_c \) and \( r_t \), and the desired output value of \( P_t \), where \( T_t \equiv T_0 - t \) is the term to maturity at time \( t \). The constraints of these input variables are listed in equation (15).

\[
1 \leq T_t \leq 4 \quad \text{AND} \quad 0 \leq r_c \leq 0.030 \quad \text{AND} \quad 0.016 \leq r_t \leq 0.023
\] (15)

Table 2: Six hypothetical short-term bonds. (Assume coupon payments are made annually.)
We adopt the Back Propagation learning algorithm to train 100 feed-forward neural networks, each of which has 4 hidden nodes and different initial weights and biases. The final weights and biases of the feed-forward neural network with the minimum sum of square error are as follows: $\theta_1 = 98.571$, $\theta_2 = -1.565$, $\theta_3 = 0.335$, $\theta_4 = -1.310$, $\theta_5 = -2.341$, $w_1^i = (-5.531, -1.995, 4.625, -0.871)$, $w_2^i = (0.393, -36.344, 15.955)$, $w_3^i = (0.145, -40.733, -36.784)$, $w_4^i = (0.409, 45.318, -62.477)$, and $w_5^i = (0.027, 50.463, 100.840)$. We take this neural network as an illustration. Thus

\[ t_1 = 0.393 T_t - 36.344 r_c + 15.955 r_t \]  
(16)

\[ t_2 = 0.145 T_t - 40.733 r_c - 36.784 r_t \]  
(17)

\[ t_3 = 0.409 T_t + 45.318 r_c - 62.477 r_t \]  
(18)

\[ t_4 = 0.027 T_t + 50.463 r_c + 100.840 r_t \]  
(19)

\[
\begin{align*}
    g_{11}(t_1) &= 1.000 & \text{if } t_1 \geq 3.561 \\
    g_{12}(t_1) &= -2.183 + 1.788t_1 - 0.251t_1^2 & \text{if } 1.561 \leq t_1 \leq 3.561 \\
    g_{13}(t_1) &= -0.953 + 0.216t_1 + 0.251t_1^2 & \text{if } -0.431 \leq t_1 \leq 1.561 \\
    g_{14}(t_1) &= -1.000 & \text{if } t_1 \leq -0.431
\end{align*}
\]  
(20)

\[
\begin{align*}
    g_{21}(t_2) &= 1.000 & \text{if } t_2 \geq 1.661 \\
    g_{22}(t_2) &= 0.307 + 0.834t_2 - 0.251t_2^2 & \text{if } -0.335 \leq t_2 \leq 1.661 \\
    g_{23}(t_2) &= 0.363 + 1.170t_2 + 0.251t_2^2 & \text{if } -2.331 \leq t_2 \leq -0.335 \\
    g_{24}(t_2) &= -1.000 & \text{if } t_2 \leq -2.331
\end{align*}
\]  
(21)
\[
g_{33}(t_3) = \begin{cases} 
1.000 & \text{if } t_3 \geq 3.306 \\
-1.744 + 1.660 t_3 - 0.25 t_3^2 & \text{if } 1.310 \leq t_3 \leq 3.306 \\
-0.882 + 0.344 t_3 + 0.25 t_3^2 & \text{if } -0.686 \leq t_3 \leq 1.310 \\
-1.000 & \text{if } t_3 \leq -0.686 
\end{cases}
\]
\[
g_{44}(t_4) = \begin{cases} 
1.000 & \text{if } t_4 \geq 4.337 \\
-3.721 + 2.177 t_4 - 0.25 t_4^2 & \text{if } 2.341 \leq t_4 \leq 4.337 \\
-0.970 - 0.173 t_4 + 0.25 t_4^2 & \text{if } 0.345 \leq t_4 \leq 2.341 \\
-1.000 & \text{if } t_4 \leq 0.345 
\end{cases}
\]  

(22)

\[y' = 98.571 - 5.531 g_{13}(t_1) - 1.995 g_{23}(t_2) + 4.625 g_{33}(t_3) - 0.871 g_{44}(t_4)\]  

(24)

Among the 256 \(4^4\) potential rules associated with this neural network, there are only eleven existing rules shown in Table 3. The remaining 245 potential rules are null. In Table 3, two polyhedrons are adjacent if they have adjacent values in one index and the same values in the other indexes. Table 4 displays the numbers of training samples contained in the corresponding polyhedron of each existing rule. Rules 3, 4, 5, 9, 10, and 11 provide the information of \(y\) in polyhedrons which contain no training samples.

### Table 3: The coefficients in each multivariate polynomial associated with each existing rule.

<table>
<thead>
<tr>
<th>(t_1)</th>
<th>(t_2)</th>
<th>(t_3)</th>
<th>(t_4)</th>
<th>(T_t)</th>
<th>(r_c)</th>
<th>(r_i)</th>
<th>(T_r)</th>
<th>(r_c)</th>
<th>(r_i)</th>
<th>(T_r^2)</th>
<th>(r_c^2)</th>
<th>(r_i^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>109.193</td>
<td>-3.526</td>
<td>403.598</td>
<td>-387.214</td>
<td>-1.955</td>
<td>-46.049</td>
<td>454.698</td>
<td>7785.223</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>106.798</td>
<td>-3.470</td>
<td>506.882</td>
<td>-180.822</td>
<td>-3.154</td>
<td>-48.446</td>
<td>3339.985</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>106.686</td>
<td>-3.568</td>
<td>534.183</td>
<td>-156.168</td>
<td>8.706</td>
<td>-37.735</td>
<td>11909.255</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>99.997</td>
<td>-0.057</td>
<td>191.007</td>
<td>-42.152</td>
<td>76.108</td>
<td>-83.242</td>
<td>5688.151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>98.294</td>
<td>2.277</td>
<td>390.772</td>
<td>-604.042</td>
<td>3.154</td>
<td>48.446</td>
<td>8908.414</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>102.280</td>
<td>-0.210</td>
<td>115.025</td>
<td>-223.889</td>
<td>89.168</td>
<td>-70.135</td>
<td>4239.977</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>99.885</td>
<td>-0.154</td>
<td>218.308</td>
<td>17.498</td>
<td>87.968</td>
<td>-72.532</td>
<td>8688.992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>101.734</td>
<td>-0.861</td>
<td>42.876</td>
<td>-124.422</td>
<td>46.161</td>
<td>-10.845</td>
<td>2334.218</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>99.339</td>
<td>-0.805</td>
<td>146.159</td>
<td>81.969</td>
<td>44.962</td>
<td>-13.241</td>
<td>2114.797</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>102.538</td>
<td>0.260</td>
<td>71.541</td>
<td>-204.800</td>
<td>49.537</td>
<td>-52.737</td>
<td>5850.108</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 14 -
Table 4: The number of training samples in the corresponding polyhedron of each rule.

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>R11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of training samples</td>
<td>1</td>
<td>79</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>240</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In practice, we may be interested in features of the first and second order differentiations of bond price; namely, the general principles regarding \( \frac{\partial y'}{\partial r_c} \), \( \frac{\partial y'}{\partial r_i} \), \( \frac{\partial^2 y'}{\partial T_i \partial r_c} \), \( \frac{\partial^2 y'}{\partial T_i \partial r_i} \), \( \frac{\partial^2 y'}{\partial r_c \partial r_i} \), \( \frac{\partial^2 y'}{\partial T_i^2} \), \( \frac{\partial^2 y'}{\partial r_i^2} \) and \( \frac{\partial^2 y'}{\partial r_c^2} \). Then, from the eleven existing rules, we can examine the corresponding features embedded in the feed-forward neural network. Table 5 reports the result of such examination, indicating that \( \frac{\partial y'}{\partial r_c} > 0 \) and \( \frac{\partial y'}{\partial r_i} < 0 \) are true in all polyhedrons. Namely, the result lends support to the two rules. The rules of \( \frac{\partial^2 y'}{\partial T_i \partial r_c} > 0 \), \( \frac{\partial^2 y'}{\partial T_i \partial r_i} < 0 \), \( \frac{\partial^2 y'}{\partial r_c \partial r_i} < 0 \) and \( \frac{\partial^2 y'}{\partial r_i^2} > 0 \) are true in almost all polyhedrons.

Table 5: The explored features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>( \frac{\partial y'}{\partial T_i} )</th>
<th>( \frac{\partial y'}{\partial r_c} )</th>
<th>( \frac{\partial y'}{\partial r_i} )</th>
<th>( \frac{\partial^2 y'}{\partial T_i \partial r_c} )</th>
<th>( \frac{\partial^2 y'}{\partial T_i \partial r_i} )</th>
<th>( \frac{\partial^2 y'}{\partial r_c \partial r_i} )</th>
<th>( \frac{\partial^2 y'}{\partial T_i^2} )</th>
<th>( \frac{\partial^2 y'}{\partial r_i^2} )</th>
<th>( \frac{\partial^2 y'}{\partial r_c^2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>R2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>R3</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>R4</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>R5</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>R6</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>R7</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>R8</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>R9</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>R10</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>R11</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The above learning process, nevertheless, is subject to a Type I error (Hogg & Tains,
1997) for \( \frac{\partial^2 y'}{\partial T_r \partial r_c} > 0 \). Specifically, \( \frac{\partial^2 y'}{\partial T_r \partial r_c} > 0 \) is not unequivocal in the bond-pricing field.

There are some relationships between \( T \) and \( r \) by intuition; for example, the price increases if \( T \) and \( r \) both increase and the price decreases if \( T \) and \( r \) both decrease. But we do not know exactly whether the price increases or decreases when \( T \) decrease while \( r \) increases, or vice versa. Thus \( \frac{\partial^2 y'}{\partial T_r \partial r_c} > 0 \) is not an unequivocal characteristic in the bond-pricing field.

The corresponding features extracted from the trained neural network can be compared with the ones derived from the well-known bond pricing theorems. Such a comparison can help us gain our knowledge regarding the bond-pricing, and also investigate whether the learning is effective.

Table 6 shows the result of comparing features embedded in the neural networks with the ones derived from the well-known theorems. This table demonstrates that these rules have on average 100.00% ((100.00+100.00)/2) in satisfaction of both features \( \frac{\partial y'}{\partial r_c} > 0 \) and \( \frac{\partial y'}{\partial r_r} < 0 \), and on average 72.24% ((90.91+81.82+66.67+40.00+81.82)/5) in satisfying the other four features. Moreover, each rule has an average satisfaction rate of 82.90% on these six features. The results of examining the extracted rules are proved positive.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10</th>
<th>R11</th>
<th>Ratio(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{\partial y'}{\partial r_c} &gt; 0 )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>100.00</td>
</tr>
<tr>
<td>( \frac{\partial y'}{\partial r_r} &lt; 0 )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>100.00</td>
</tr>
</tbody>
</table>

- 16 -
<table>
<thead>
<tr>
<th>( \frac{\partial^2 y'}{\partial T_i \partial r_i} &lt; 0 )</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>90.91</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{\partial^2 y'}{\partial r_i \partial r_i} &lt; 0 )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>81.82</td>
</tr>
<tr>
<td>( \frac{\partial^3 y'}{\partial r_i^3} &lt; 0, \text{if } r_i &gt; r_i )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>No</td>
<td>66.67</td>
</tr>
<tr>
<td>( \frac{\partial^3 y'}{\partial T_i^3} &gt; 0, \text{if } r_i &lt; r_i )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>-</td>
<td>40.00</td>
</tr>
<tr>
<td>( \frac{\partial^2 y'}{\partial r_i^2} &gt; 0 )</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>81.82</td>
</tr>
<tr>
<td>Ratio (^2)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>83.33</td>
<td>42.86</td>
<td>85.72</td>
<td>83.33</td>
<td>66.67</td>
<td>66.67</td>
<td>83.33</td>
</tr>
</tbody>
</table>

1. (The number of "Yes") / (The total number of "Yes" and "No") in one characteristic.
2. (The number of "Yes") / (The total number of "Yes" and "No") in one rule.
3. There are no areas.

4. Conclusions and Future Work

In this study, we propose a mathematical programming methodology for extracting and examining regression rules from layered feed-forward neural networks. The mathematical programming analysis, instead of a data analysis, is proposed for identifying the premises of multivariate polynomial rules. Also, the mathematical programming analysis is claimed with the aim to explore features from the extracted rules. The proposed methodology can provide regression rules and features not only in the polyhedrons with data instances, but also in the polyhedrons without data instances.

Furthermore, the proposed method can be applied to any non-linear rules and features, as long as the adopted approximating function holds the proper nonlinear information. The approximating function \( g(x) \) used in equation (5) here is designed as a piece-wise second order nonlinear function due to the assumption that we are interested in only the first and sec-
ond order differential information. With respect to the bond-pricing application, features with the first and second ordered characteristics can be explored from extracted rules. Generally, \( g(x) \) can be a piece-wise higher order nonlinear function, and the proposed method can be applied to the new \( g(x) \).

In contrast with Setiono et al. (2002), the approximating function used in equation (5) has a better total absolute error than the one associated with the approximating function proposed in Setiono et al. (2002). With the dataset used to extract rules approaches infinity, our total absolute error almost equals 0.124056, while theirs almost equals 0.142338 (Setiono, et al., 2002).

Issues worthy of future studies include the application of the proposed methodology to real world data, how to delete redundant constraints from the premise of a rule, and how to integrate extracted rules.

References


