Association for Information Systems
AIS Electronic Library (AISeL)

ICEB 2003 Proceedings
International Conference on Electronic Business (ICEB)

Winter 12-9-2003

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

Rua-Huan Tsaih

Hsiou-Wei Lin

Wen-Chyan Ke

Cheng-Chang Lee

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

[1]Ray Tsaih, [2]Hsiou-Wei Lin, [2]Wen-Chyan Ke, [1]Cheng-Chang Lee
[1]Department of Management Information Systems, National Chengchi University, Taipei, Taiwan,
E-mail:tsaih@mis.nccu.edu.tw, czlee@mis.nccu.edu.tw
[2]Department of International Business, National Taiwan University, Taipei, Taiwan,
E-mail:plin@ccms.ntu.edu.tw, wenchyan@mis.nccu.edu.tw

Abstract

We propose a mathematical programming methodology for identifying and examining regression rules extracted from layered feed-forward neural networks. 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. The mathematical programming analysis, instead of a data analysis, is proposed for identifying the convex polyhedron associated with each rule. Moreover, the mathematical programming analysis is proposed for examining the extracted rules to explore features. An implementation test on bond pricing rule extraction lends support to the proposed methodology.

Keywords: Rule extraction, Neural Networks, Mathematical programming, Bond-pricing

1. Extracting Regression Rules from Neural Networks

Rumelhart and his colleagues in [3] 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, [7], [8] and [10] extract rules from a trained ANN for classification problems; [6] and [5] from a trained ANN for regression problems. In their paper [5, p.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:

If (premise), then (action). (1)

The premise is a Boolean expression that must be evaluated as true for the rule to be applied. For example, a rule premise is **x** ∈ {**x**: 0.2 £ $x_1 + 2x_2$ £ 10.2}, where **x** is the outside input vector. The action clause consists of a statement or a series of statements, and is executed only when the premise is true. For example, a rule action is that the output value y is approximated by $g(\mathbf{x})$. The approximation function $g(\mathbf{x})$ can be a piece-wise linear function (cf. [6].) or a multivariate polynomial function whose power values could not be restricted to integers (cf. [5].). Specifically, the regression rules take the following syntax:

If ($\mathbf{x} \in$ the i^{th} **Area**), then ($y' = g_i(\mathbf{x})$), (2)

where y' is the approximation of the output value y. The **Area** depicted in the premise covers a region in the input space. This type of rules is acceptable 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, [6] and [5] 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, [6] and [5] 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 $g(\mathbf{x})$ 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 explore features.

This paper is organized as follows. Section 2 gives details of the proposed method. Section 3 presents a study of applying the proposed method to the bond-pricing. Finally, Section 4 offers some conclusions and future work.

2. The Proposed Mathematical Programming methodology

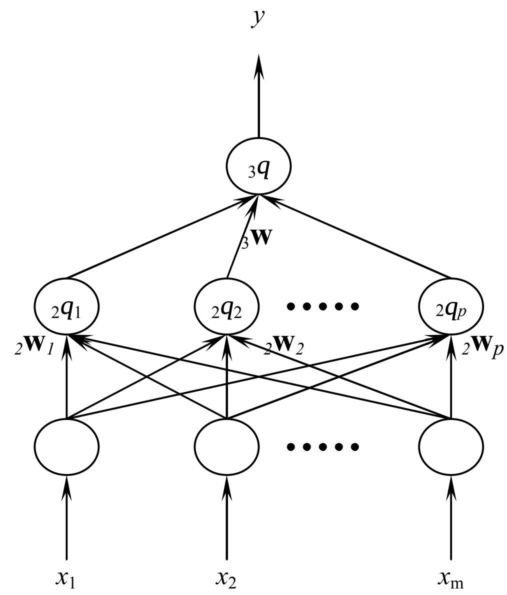

Figure 1: The feed-forward neural network with one hidden layer and one output node.

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 $\mathbf{x}^t \equiv (x_1, x_2,\ldots, x_m)$ whereas x_i denotes the i-th outside stimulus input, with i from 1 to m, where m is the amount of stimulus input. ${}_2\mathbf{w}_j^t \equiv ({}_2w_{j1}, {}_2w_{j2},\ldots, {}_2w_{jm})$ 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 ${}_3\mathbf{w}^t \equiv ({}_3w_1, {}_3w_2,\ldots, {}_3w_p)$ stands for the weights between the output node and all hidden nodes. ${}_2q_j$ is the bias of the j-th hidden node and ${}_3q$ is the bias of the output node. The activation function $\tanh(t) \equiv \dfrac{e^t - e^{-t}}{e^t + e^{-t}}$ is used in all hidden nodes and the linear activation function is used in the output node. That is, for the c-th input ${}_c\mathbf{x}$, the activation value of the j-th hidden node ${}_ch_j$ and the output value ${}_cy$ are computed as in equations (3) and (4).

$$ {}_ch_j = \tanh\left(\sum_{i=1}^{m} {}_2w_{ji}\, {}_cx_i + {}_2q_j\right) \tag{3} $$

$$ {}_cy = \sum_{j=1}^{p} {}_3w_j\, {}_ch_j + {}_3q \tag{4} $$

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 $\tanh(t)$:

$$ g(t) \equiv \begin{cases} 1 & \text{if } t \geq k \\ b_1 t + b_2 t^2 & \text{if } 0 \leq t \leq k \\ b_1 t - b_2 t^2 & \text{if } -k \leq t \leq 0 \\ -1 & \text{if } t \leq -k \end{cases} \tag{5} $$

where $(b_1, b_2, k) \equiv \arg(\min_{b_1,b_2,k} \int_{-\infty}^{\infty} (\tanh(t) - g(t))^2 \mathrm{d}t$, subject to $b_1k + b_2k^2 = 1$). $g(t)$ is continuous at boundaries of four polyhedrons ($t = k$, $t = 0$, $t = -k$), because $\lim_{t\to k^-} b_1 t + b_2 t^2 = 1$, $\lim_{t\to 0^+} b_1 t + b_2 t^2 = 0$, $\lim_{t\to 0^-} b_1 t - b_2 t^2 = 0$, and $\lim_{t\to -k^+} b_1 t - b_2 t^2 = -1$.

With the numerical analysis of Sequential Quadratic Programming (cf. [9].), we obtain $b_1 \cong 1.0020101308531$, $b_2 \cong -0.251006075157012$, $k \cong 1.99607103795966$, and $\min_{b_1,b_2,k} \int_{-\infty}^{\infty} (\tanh(t) - g(t))^2 \mathrm{d}t \cong 0.00329781871956464$.

$$ g(t_j + {}_2q_j) = \begin{cases} 1 & \text{if } t_j \geq k - {}_2q_j \\ (b_1\, {}_2q_j + b_2\, {}_2q_j^2) + (b_1 + 2b_2\, {}_2q_j)t_j + b_2 t_j^2 & \text{if } -{}_2q_j \leq t_j \leq k - {}_2q_j \\ (b_1\, {}_2q_j - b_2\, {}_2q_j^2) + (b_1 - 2b_2\, {}_2q_j)t_j - b_2 t_j^2 & \text{if } -k - {}_2q_j \leq t_j \leq -{}_2q_j \\ -1 & \text{if } t_j \leq -k - {}_2q_j \end{cases} \tag{6} $$

For the j-th hidden node, let $t_j \equiv {}_2\mathbf{w}_j^t\, \mathbf{x}$. Thus $\tanh({}_2\mathbf{w}_j^t\, \mathbf{x} + {}_2q_j)$ can be approximated with $g(t_j + {}_2q_j)$, which is defined in equation (6). In other words, for the j-th hidden node, the activation value is approximated with a polynomial form of single variable t_j in each of four separate polyhedrons in the $\mathbf{x}$ space. For example, if $\mathbf{x} \in \{\mathbf{x}$: $-{}_2q_j$ £ ${}_2\mathbf{w}_j^t\mathbf{x}$ £ $k - {}_2q_j\}$, then $\tanh({}_2\mathbf{w}_j^t\, \mathbf{x} + {}_2q_j)$ is approximated with $b_1\, {}_2q_j + b_2\, {}_2q_j^2 + (b_1 + 2\, b_2\, {}_2q_j)\, t_j + b_2\, t_j^2$. Thus, a comprehensible regression rule associated with a trained feed-forward neural network with p hidden nodes is like:

If $\mathbf{x} \in \{\mathbf{x}$: $-{}_2q_j$ £ ${}_2\mathbf{w}_j^t\mathbf{x}$ £ $k - {}_2q_j$ for all $j\}$, then $y' = {}_3q + \sum_{j=1}^{p} {}_3w_j\, (b_1\, {}_2q_j + b_2\, {}_2q_j^2 + (b_1 + 2\, b_2\, {}_2q_j)\, t_j + b_2\, t_j^2)$.

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 i_j be 1 when the situation $k - {}_2q_j$ £ t_j holds; 2 when $-{}_2q_j$ £ t_j £ $k - {}_2q_j$ holds; 3 when $-k - {}_2q_j$ £ t_j £ $-{}_2q_j$ holds; or 4 when t_j £ $-k - {}_2q_j$ holds. Also, set $\mathrm{w}_{j1} \equiv {}_2\mathbf{w}_j^t$, $\mathrm{w}_{j2} \equiv \begin{bmatrix} {}_2\mathbf{w}_j^t \\ -{}_2\mathbf{w}_j^t \end{bmatrix}$, $\mathrm{w}_{j3} \equiv \begin{bmatrix} {}_2\mathbf{w}_j^t \\ -{}_2\mathbf{w}_j^t \end{bmatrix}$, $\mathrm{w}_{j4} \equiv -{}_2\mathbf{w}_j^t$, $\mathrm{u}_{j1} \equiv k - {}_2q_j$, $\mathrm{u}_{j2} \equiv (-{}_2q_j, {}_2q_j - k)^t$, $\mathrm{u}_{j3} \equiv (-{}_2q_j - k, {}_2q_j)^t$, $\mathrm{u}_{j4} \equiv k$

+ ${}_2q_j$, $g_{j1}(t_j) \equiv 1$, $g_{j2}(t_j) \equiv b_1\,{}_2q_j + b_2\,{}_2q_j^2 + (b_1 + 2\,b_2\,{}_2q_j)\,t_j + b_2\,t_j^2$, $g_{j3}(t_j) \equiv b_1\,{}_2q_j - b_2\,{}_2q_j^2 + (b_1 - 2\,b_2\,{}_2q_j)\,t_j - b_2\,t_j^2$, and $g_{j4}(t_j) \equiv -1$. Let $i \equiv (i_1, i_2, ..., i_p)$ with $i_j \in \{1, 2, 3, 4\}$ for every j, ${}_1\mathbf{A}_i \equiv \begin{bmatrix} \mathrm{w}_{1i_1} \\ \mathrm{w}_{2i_2} \\ \vdots \\ \mathrm{w}_{pi_p} \end{bmatrix}$, and ${}_1\mathbf{b}_i \equiv (\mathrm{u}_{1i_1}, \mathrm{u}_{2i_2}, \ldots, \mathrm{u}_{pi_p})^t$. Thus, for example, the premise $\mathbf{x} \in \{\mathbf{x}: -{}_2q_j \leq {}_2\mathbf{w}_j^t\mathbf{x} \leq k - {}_2q_j\ \forall\, j = 1, 2, ..., p\}$ can be expressed as $\mathbf{x} \in \{\mathbf{x}: {}_1\mathbf{A}_i\,\mathbf{x} \geq {}_1\mathbf{b}_i$ with $i_j = 2$ 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 ${}_2b_i$ are given constants. Let

$${}_2\mathbf{A} \equiv \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n2} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \text{ and } {}_2\mathbf{b} \equiv ({}_2b_1, {}_2b_2, \ldots, {}_2b_n)^t.$$

Thus equation (7) can be expressed as ${}_2\mathbf{A}\,\mathbf{x} \geq {}_2\mathbf{b}$.

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{im}x_m \geq {}_2b_i,\ i = 1, 2, ..., n \qquad (7)$$

In sum, there are 4^p polyhedrons in the input space where the corresponding output value y is approximated with a multivariate polynomial function. The potential rule associated with the i-th polyhedron $\{\mathbf{x}: \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i\}$ in the input space is similar to the one shown in equation (8), where $\mathbf{A}_i \equiv \begin{bmatrix} {}_1\mathbf{A}_i \\ {}_2\mathbf{A} \end{bmatrix}$, $\mathbf{b}_i \equiv \begin{bmatrix} {}_1\mathbf{b}_i \\ {}_2\mathbf{b} \end{bmatrix}$.

$$\text{If } \mathbf{x} \in \{\mathbf{x}: \mathbf{A}_i\ \ \mathbf{x} \geq \mathbf{b}_i\} \text{ then } y' = {}_3q + \sum_{j=1}^{p} {}_3w_j\, g_{ji_j}(t_j) \qquad (8)$$

However, some of these 4^p potential rules are null. The Simplex algorithm (cf. [2].) can be applied to identify the null rules. If $\{\mathbf{x}: \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i\}$ is empty, the rule associated with the i-th polyhedron is null. $\{\mathbf{x}: \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i\}$ is a convex polyhedral set in the input space because $\mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i$ consists of linear inequality constraints. Furthermore, $\{\mathbf{x}: \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i\}$ 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 $\{\mathbf{x}: \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i\}$ exists. Otherwise, that rule fails to exist. The process of extracting existing rules is summarized in Table 1.

$$\begin{aligned} &\text{Minimize: } \mathit{constant} \\ &\text{Subject to: } \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i \end{aligned} \qquad (9)$$

Table 1: The process of rule extraction.

Step 1:	Input all weights and biases of the trained feed-forward neural network to form ${}_1\mathbf{A}_i$ and ${}_1\mathbf{b}_i$.
Step 2:	Input the constraints associated with the independent variables to form ${}_2\mathbf{A}$ and ${}_2\mathbf{b}$.
Step 3:	For each of the 4^p potential rules, says If $\mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i$, then $y' = {}_3q + \sum_{j=1}^{p} {}_3w_j\, g_{ji_j}(t_j)$ where $\mathbf{A}_i \equiv \begin{bmatrix} {}_1\mathbf{A}_i \\ {}_2\mathbf{A} \end{bmatrix}$ and $\mathbf{b}_i \equiv \begin{bmatrix} {}_1\mathbf{b}_i \\ {}_2\mathbf{b} \end{bmatrix}$, 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.

$$\begin{aligned} &\text{Minimize: } \frac{\partial y'}{\partial x_k} \\ &\text{Subject to: } \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i \end{aligned} \qquad (10)$$

$$\begin{aligned} &\text{Maximize: } \frac{\partial y'}{\partial x_k} \\ &\text{Subject to: } \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i \end{aligned} \qquad (11)$$

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 $\{\mathbf{x}: \mathbf{A}_i\,\mathbf{x} \geq \mathbf{b}_i\}$, $\left.\frac{\partial y'}{\partial x_k}\right|_{\mathbf{x}\in\{\mathbf{x}|\mathbf{A}_i\mathbf{x}\geq\mathbf{b}_i\}} > 0$ if the minimal solution to the optimization problem (10) is greater than zero, and $\left.\frac{\partial y'}{\partial x_k}\right|_{\mathbf{x}\in\{\mathbf{x}|\mathbf{A}_i\mathbf{x}\geq\mathbf{b}_i\}} < 0$ if the maximal solution to the optimization problem (11) is less than zero. Because of the approximation stated in equation (5), $\frac{\partial y'}{\partial x_k} = \sum_{j=1}^{p} {}_3w_j \frac{\partial \mathrm{g}_j(t_j)}{\partial x_k}$ and

$$\frac{\partial \mathrm{g}_j(t_j)}{\partial x_k} = \begin{cases} 0 & \mathit{if}\ t_j > k - {}_2q_j \\ {}_2w_{jk}(b_1 + 2b_2\,{}_2q_j) + 2\,{}_2w_{jk}b_2t_j & \mathit{if}\ -{}_2q_j < t_j < k - {}_2q_j \\ {}_2w_{jk}(b_1 - 2b_2\,{}_2q_j) - 2\,{}_2w_{jk}b_2t_j & \mathit{if}\ -k - {}_2q_j < t_j < -{}_2q_j \\ 0 & \mathit{if}\ t_j < -k - {}_2q_j \end{cases} . \qquad (12)$$

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 $\begin{cases} \frac{\partial^2 y'}{\partial x_k^2} < 0 & \mathit{if}\ x_i > x_j \\ \frac{\partial^2 y'}{\partial x_k^2} > 0 & \mathit{if}\ x_i < x_j \end{cases}$, 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 $x_i < x_j$" is not true at the [3,3,3,3]-th polyhedron, and "$\frac{\partial^2 y'}{\partial x_k^2} < 0$ if $x_i > x_j$" is true at the [3,3,3,3]-th polyhedron if and only if the LP problem (13) has an optimal solution.

$$\text{Minimize: } constant$$
$$\text{Subject to: } \mathbf{A}_{[3,3,3,3]}\,\mathbf{x}\ ^3\ \mathbf{b}_{[3,3,3,3]},\ x_i > x_j \tag{13}$$

3. The Rule Extraction in the Bond-Pricing Application

This section adopts 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) C, periodic coupon payment, which equals $F\,r_c$.

$$P_t \equiv \sum_{k=1}^{T_0} \frac{C}{(1+r_t)^{k-t}} + \frac{F}{(1+r_t)^{T_0-t}} \tag{14}$$

By assuming one coupon payment per year (that is, coupon payments are made every 12 months), there are five well-known theorems with respect to bond prices which have been derived as follows: [3]

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, $\frac{\partial P_t}{\partial r_t}$<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 size of its discount or premium will decrease at an increasing rate as its life gets shorter. That is, $\begin{cases} \frac{\partial^2 P_t}{\partial T_t^2} < 0 & if\ r_c > r_t \\ \frac{\partial^2 P_t}{\partial T_t^2} > 0 & if\ r_c < r_t \end{cases}$.
4. A decrease in a bond's yield will raise the bond's price by an amount which is greater in size than the corresponding fall in the bond's price, and the fall will occur if there is an equal-sized increase in the bond's yield. That is, $\frac{\partial^2 P_t}{\partial r_t^2}$>0.
5. The amount change in a bond's price due to a change in its yield will be higher if its coupon rate is higher. That is, $\frac{\partial^2 P_t}{\partial r_c \partial r_t}$<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 $N(2\%, (0.1\%)^2)$. 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, …, 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).

$$(1 \le T_t \le 4) \text{ AND } (0 \le r_c \le 0.030) \text{ AND } (0.016 \le r_t \le 0.023) \tag{15}$$

Table 2: Six hypothetical short-term bonds. Assume coupon payments are made annually.

Term to maturity (T_0)	Contractual interest rate (r_c)
2	0.0%
4	1.5%
2	3.0%
4	0.0%
2	1.5%
4	3.0%

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: ${}_3q$ = 98.571, ${}_2q_1$ = -1.565, ${}_2q_2$ = 0.335, ${}_2q_3$ = -1.310, ${}_2q_4$ = -2.341, ${}_3\mathbf{w}^t$ = (-5.531, -1.995, 4.625, -0.871), ${}_2\mathbf{w}_1^t$ = (0.393, -36.344, 15.955), ${}_2\mathbf{w}_2^t$ = (0.145, -40.733, -36.784), ${}_2\mathbf{w}_3^t$ = (0.409, 45.318, -62.477), and ${}_2\mathbf{w}_4^t$ = (0.027, 50.463, 100.840). We take this neural network as an illustration. Thus

$$t_1 = 0.393\ T_t - 36.344\ r_c + 15.955\ r_t \tag{16}$$
$$t_2 = 0.145\ T_t - 40.733\ r_c - 36.784\ r_t \tag{17}$$
$$t_3 = 0.409\ T_t + 45.318\ r_c - 62.477\ r_t \tag{18}$$
$$t_4 = 0.027\ T_t + 50.463\ r_c + 100.840\ r_t \tag{19}$$

$$\begin{cases} g_{11}(t_1) = 1.000 & if\ t_1 \ge 3.561 \\ g_{12}(t_1) = -2.183 + 1.788t_1 - 0.251t_1^2 & if\ 1.561 \le t_1 \le 3.561 \\ g_{13}(t_1) = -0.953 + 0.216t_1 + 0.251t_1^2 & if\ -0.431 \le t_1 \le 1.561 \\ g_{14}(t_1) = -1.000 & if\ t_1 \le -0.431 \end{cases} \tag{20}$$

$$\begin{cases} g_{21}(t_2) = 1.000 & if\ t_2 \ge 1.661 \\ g_{22}(t_2) = 0.307 + 0.834t_2 - 0.251t_2^2 & if\ \text{-}0.335 \le t_2 \le 1.661 \\ g_{23}(t_2) = 0.363 + 1.170t_2 + 0.251t_2^2 & if\ -2.331 \le t_2 \le -0.335 \\ g_{24}(t_2) = -1.000 & if\ t_2 \le -2.331 \end{cases} \tag{21}$$

$$\begin{cases} g_{31}(t_3) = 1.000 & if\ t_3 \geq 3.306 \\ g_{32}(t_3) = -1.744 + 1.660 t_3 - 0.251 t_3^2 & if\ 1.310 \leq t_3 \leq 3.306 \\ g_{33}(t_3) = -0.882 + 0.344 t_3 + 0.251 t_3^2 & if\ -0.686 \leq t_3 \leq 1.310 \\ g_{34}(t_3) = -1.000 & if\ t_3 \leq -0.686 \end{cases} \tag{22}$$

$$\begin{cases} g_{41}(t_4) = 1.000 & if\ t_4 \geq 4.337 \\ g_{42}(t_4) = -3.721 + 2.177 t_4 - 0.251 t_4^2 & if\ 2.341 \leq t_4 \leq 4.337 \\ g_{43}(t_4) = -0.970 - 0.173 t_4 + 0.251 t_4^2 & if\ 0.345 \leq t_4 \leq 2.341 \\ g_{44}(t_4) = -1.000 & if\ t_4 \leq 0.345 \end{cases} \tag{23}$$

$$y' = 98.571 - 5.531\ g_{1i_1}(t_1) - 1.995\ g_{2i_2}(t_2) + 4.625\ g_{3i_3}(t_3) - 0.871\ g_{4i_4}(t_4) \tag{24}$$

Among the 256 (4^4) potential rules associated with this neural network, there are only eleven existing rules shown in Table 3. Namely, the other 245 potential rules are null. In Table 3, two polyhedrons are adjacent if they have adjacent values in one index and same values in the other indexes. Table 4 displays the amount 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.

In practice, we may be interested in features of first and second order differentiations of bond price; namely, the general principles about $\frac{\partial y'}{\partial r_c}$, $\frac{\partial y'}{\partial r_t}$, $\frac{\partial y'}{\partial T_t}$, $\frac{\partial^2 y'}{\partial T_t \partial r_c}$, $\frac{\partial^2 y'}{\partial T_t \partial r_t}$, $\frac{\partial^2 y'}{\partial r_c \partial r_t}$, $\frac{\partial^2 y'}{\partial T_t^2}$, $\frac{\partial^2 y'}{\partial r_c^2}$ and $\frac{\partial^2 y'}{\partial r_t^2}$. 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 $\frac{\partial y'}{\partial r_t} < 0$ are true in all polyhedrons. Namely, the result lends support to the two rules. The rules of $\frac{\partial^2 y'}{\partial T_t \partial r_c} > 0$, $\frac{\partial^2 y'}{\partial T_t \partial r_t} < 0$, $\frac{\partial^2 y'}{\partial r_c \partial r_t} < 0$ and $\frac{\partial^2 y'}{\partial r_t^2} > 0$ are true in almost all polyhedrons.

The above learning process, nevertheless, is subject to a Type I error [1] for $\frac{\partial^2 y'}{\partial T_t \partial r_c} > 0$. Specifically, $\frac{\partial^2 y'}{\partial T_t \partial r_c} > 0$ is not unequivocal in the bond-pricing field. There are some relationships between T_t and r_c by intuition; for example, the price increases if T_t and r_c both increase and the price decreases if T_t and r_c both decrease. But we do not know exactly whether the price increases or decreases when T_t decrease while r_c increases, or vice versa. Thus $\frac{\partial^2 y'}{\partial T_t \partial r_c} > 0$ is not an unequivocal characteristic in the bond-pricing field.

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

Rule No.	i_1	i_2	i_3	i_4	Constant	T_t	r_c	r_t	$T_t r_c$	$T_t r_t$	$r_c r_t$	T_t^2	r_c^2	r_t^2
R1	2	2	3	2	109.193	-3.526	403.598	-387.214	-1.955	-46.049	-4459.398	0.419	5605.512	7785.223
R2	2	2	3	3	106.798	-3.470	506.882	-180.822	-3.154	-48.446	-8908.414	0.418	4492.314	3339.985
R3	2	3	3	2	109.081	-3.623	430.899	-362.559	9.905	-35.338	-7460.240	0.398	3944.009	6430.268
R4	2	3	3	3	106.686	-3.568	534.183	-156.168	8.706	-37.735	-11909.255	0.397	2830.810	1985.030
R5	3	2	3	3	99.997	-0.057	191.007	-42.152	76.108	-83.242	-5688.151	-0.010	824.626	2633.130
R6	3	3	2	2	98.294	2.277	390.772	-604.042	3.154	48.446	8908.414	-0.418	-4492.314	-3339.985
R7	3	3	3	2	102.280	-0.210	115.025	-223.889	89.168	-70.135	-4239.977	-0.031	276.322	5723.413
R8	3	3	3	3	99.885	-0.154	218.308	-17.498	87.968	-72.532	-8688.992	-0.031	-836.877	1278.175
R9	3	3	4	2	101.734	-0.861	42.876	-124.422	46.161	-10.845	2334.218	-0.225	-2107.996	1191.714
R10	3	3	4	3	99.339	-0.805	146.159	81.969	44.962	-13.241	-2114.797	-0.225	-3221.195	-3253.524
R11	4	3	3	2	102.538	0.260	71.541	-204.800	49.537	-52.737	-5850.108	0.184	2110.166	6076.841

Table 4: The amount of training samples in the corresponding polyhedron of each rule.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
Amount of training samples	1	79	0	0	0	80	240	80	0	0	0

Table 5: The explored features.

Char-acteristic	$\frac{\partial y'}{\partial T_t}$		$\frac{\partial y'}{\partial r_c}$		$\frac{\partial y'}{\partial r_t}$		$\frac{\partial^2 y'}{\partial T_t \partial r_c}$		$\frac{\partial^2 y'}{\partial T_t \partial r_t}$		$\frac{\partial^2 y'}{\partial r_c \partial r_t}$		$\frac{\partial^2 y'}{\partial T_t^2}$		$\frac{\partial^2 y'}{\partial r_c^2}$		$\frac{\partial^2 y'}{\partial r_t^2}$	
	>0	<0	>0	<0	>0	<0	>0	<0	>0	<0	>0	<0	>0	<0	>0	<0	>0	<0
R1	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No	Yes	No
R2	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No	Yes	No
R3	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes	No	Yes	No	Yes	No
R4	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes	No	Yes	No	Yes	No
R5	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No
R6	No	No	Yes	No	No	Yes	Yes	No	Yes	No	Yes	No	No	Yes	No	Yes	No	Yes
R7	No	No	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Yes	No
R8	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	Yes	No
R9	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes	No
R10	No	Yes	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
R11	Yes	No	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	Yes	No	Yes	No	Yes	No

Table 6: The results of the examining rules

Rule / Characteristic	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	Ratio(%)[1]
$\frac{\partial y'}{\partial r_c}>0$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**100.00**
$\frac{\partial y'}{\partial r_t}<0$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**100.00**
$\frac{\partial^2 y'}{\partial T_t \partial r_t}<0$	Yes	Yes	Yes	Yes	Yes	**No**	Yes	Yes	Yes	Yes	Yes	**90.91**
$\frac{\partial^2 y'}{\partial r_c \partial r_t}<0$	Yes	Yes	Yes	Yes	Yes	**No**	Yes	Yes	**No**	Yes	Yes	**81.82**
$\frac{\partial^2 y'}{\partial T_t^2}<0, if\ r_c>r_t$	-[3]	-	-	-	-	Yes	Yes	-	-	-	**No**	**66.67**
$\frac{\partial^2 y'}{\partial T_t^2}>0, if\ r_c<r_t$	Yes	Yes	Yes	Yes	**No**	**No**	**No**	**No**	**No**	**No**	-	**40.00**
$\frac{\partial^2 y'}{\partial r_t^2}>0$	Yes	Yes	Yes	Yes	Yes	**No**	Yes	Yes	Yes	**No**	Yes	**81.82**
Ratio[2]	**100**	**100**	**100**	**100**	**83.33**	**42.86**	**85.72**	**83.33**	**66.67**	**66.67**	**83.33**	-

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 is not a polyhedron that $r_c > r_t$ or $r_c < r_t$.

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 out knowledge about 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 these well-known theorems. This table demonstrates that these rules have a mean of 100.00% ((100.00+100.00)/2) in satisfaction of both features $\frac{\partial y'}{\partial r_c}$ > 0 and $\frac{\partial y'}{\partial r_t}$ < 0, and a mean of 72.24% ((90.91+81.82+66.67+40.00+81.82)/5) in satisfaction of other four features. Moreover, each rule has an average satisfaction of 82.90 % on these six features. The results of the extracted rules are proved positive.

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 $g(\mathbf{x})$ 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 second 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, $g(\mathbf{x})$ can be a piece-wise higher order nonlinear function, and the proposed method can be applied to the new $g(\mathbf{x})$.

In contrast with Setiono et al. [6] , 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. [6]. With the dataset used to extract rules approaches infinity, our total absolute error almost equals 0.124056, while theirs almost equals 0.142338 [6].

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

[1] Hogg, R. V., Tains, E. A. (1997). Probability and statistical inference-5th ed., New Jersey: Prentice Hall, pp. 394-455.

[2] Hillier, F. and Lieberman, G., (2001). Introduction to Operations Research, 7th ed., Singapore: McGraw Hill, pp. 109-189

[3] Malkiel, B. G., (1962). "Expectations, bond prices, and the term structure of interest rates." Quarterly Journal of Economics, Vol 76, No. 2, pp.197-218.

[4] Rumelhart, D.E., Hinton, G.E., and Williams, R. (1986). "Learning internal representation by error propagation." Parallel Distributed Processing. Cambridge, MA: MIT Press, Vol. 1, pp. 318-362.

[5] Saito, K., and Nakano R. (2002). "Extracting regression rules from neural networks." Neural Network, Vol. 15, No. 10, pp. 1297-1288.

[6] Setiono, R., Leow, W. K., and Zurada, J. M. (2002). "Extraction of rules from artificial neural networks for nonlinear regression," IEEE Transactions on Neural Networks, Vol. 13, No. 3, pp. 564-577.

[7] Setiono, R., and Liu. H. (1997). "NeuroLinear: From neural networks to oblique decision rules." Neurocomputing, Vol. 17, No. 1, pp. 1-24.

[8] Taha, I. A., and Ghosh, J. (1999). "Symbolic interpretation of artificial neural networks." IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 3, pp.448-463.

[9] The MathWorks, Inc. (2002). Optimization Toolbox User's Guide. [Online]. Available: http://www.mathworks.com/access

/helpdesk/help/pdf_doc/optim/optim_tb.pdf

[10] Zhou, R. R., Chen, S. F., and Chen, Z. Q. (2000). "A statistics based approach for extracting priority rules from trained neural networks." In: Proceedings of the IEEE-INNS-ENNS International Join Conference on Neural Network, Como, Italy, Vol. 3, pp. 401-406.