EXTRACTING RULES FROM NEURAL NETWORKS TO PREDICT HIGH RETURN STOCKS

Monica Lam
Management Information Science Department
California State University, Sacramento
6000 J Street, Sacramento, CA 95819-6088, USA
Phone: 916-278-7037 FAX: 916-278-6757 Email: lamsm@csus.edu

ABSTRACT

Neural networks have been shown to be a powerful classification tool in financial applications. However, neural networks are basically black boxes that do not explain the classification procedure. The training results from neural networks, which are sets of connection weights expressed in numeric terms, hardly shed light on the importance of input attributes and their relationship for classification problems. To address this issue, researchers have developed different algorithms to extract classification rules from trained neural networks.

The purpose of this paper is to validate the prediction power of extracted rules from one algorithm GLARE (Generalized Analytic Rule Extraction). The input to the GLARE algorithm is a set of connection weights from a trained neural network, and the output is classification rules that can be used to predict new cases as well as to explain the classification procedure. We apply the conventional backpropagation and GLARE to a data set from the CompuStat database. The input to the prediction problem is a vector of financial statement variables, and the output is the rate of return on common shareholders' equity. To test the effect of the number of training epochs on rule extraction, we train the networks for 5 and 1000 epochs before rule extraction. To test the statistical significance of performance differences between conventional backpropagation and rules from neural networks, we perform paired t test for each pair of the average returns. The experimental results support the superiority of extracted rules to conventional backpropagation on selecting high return stocks.

INTRODUCTION

Since knowledge acquisition from human experts is a tedious and time consuming process, many research efforts have diverted to generating knowledge from past documents, cases, and solutions. Learning from examples, a major topic in machine learning, is to acquire classification knowledge from existing examples. Given a set of examples, each of which has an input attribute vector **x** and a corresponding class y, the learning algorithm induces the mapping function $f(\mathbf{x}) = y$. The mapping function is considered as the knowledge acquired from the machine learning process. In order to facilitate the storing and processing of knowledge in rule-based systems, it is preferable to have knowledge represented in symbolic rules like *IF (attribute x = p, y = q, z = r, ...) THEN (class = a).* Among various algorithms for learning from examples, backpropagation for neural networks is found to be robust and accurate [2, 4, 6, 23, 24]. However, backpropagation has the severe handicap of being unable to explain the training result. It is difficult, if not impossible, to interpret the set of connection weights from a trained network. It is well known that knowledge of relative importance of input attributes and their relationship can provide valuable information for data collection as well as for remedial actions in experimental research. There have been various efforts [7, 8, 19, 26] in designing rule extraction algorithms to extract classification rules from neural networks.

This paper attempts to evaluate a rule extraction algorithm named as GLARE [9] for financial applications. In order to verify the validity of extracted rules from GLARE, the performance of GLARE is compared with neural networks per se. The remainder of this paper is organized as follows. Section 1 briefly reviews and summarizes rule extraction algorithms in the literature. Section 2 summarizes the GLARE algorithm. Section 3 describes the experimental design. Section 4 describes and discusses experimental results. The last section concludes the paper and proposes some directions for rule extraction research in neural networks.

RULE EXTRACTION FOR NEURAL NETWORKS

This section briefly reviews some rule extraction algorithms in the literature. Gallant [8] suggests a hybrid system called connectionist expert system that uses a feedforward neural network to acquire knowledge and perform inference. Human experts provide dependence information about attributes to configure the network before training. Connection weights are represented using only positive or negative integers. The output of a node is clamped into 0, 1, or -1 corresponding to the logical meaning of unknown, true, or false respectively. The network is trained using the pocket algorithm which is a modification of the perceptron learning method [18]. The rule extraction procedure is to identify contributing nodes which can determine the value of an output node, and then use the contributing nodes to form the premise in an if-then rule. This method is limited to networks with output values 0, 1, or -1.

The subset method for rule extraction [19, 7] differs from Gallant's method in its utilization of threshold units in neural networks. For each hidden and output node in a neural network, the subset method carries out an exhaustive search to identify all subsets of contributing nodes which have a summation value greater than the threshold unit of the hidden or output node. Then, each subset of

contributing nodes is used as the premise in a rule for the hidden or output node. Since the search procedure has to be carried out iteratively to reach the final output nodes in the output layer, the number of rules extracted from the network can grow exponentially as the number of connection weights increases. Heuristics can be applied to limit the search at the expense of the accuracy of extracted rules.

Another method NofM [26, 27] uses groups of weights rather than individual weights as the building blocks for rule premises. Rules generated by NofM has the format: IF (N of the following M antecedents are true) THEN (class x). NofM method is suggested as the final step for a knowledge refinement process. The entire process is first to insert domain knowledge into a neural network, then train the network, and finally extract rules from the network using the NofM algorithm. To extract rules, NofM first collects similar connection weights for each hidden and output node into groups. Then, all the weights in the same group are set to the average of the group. Groups of weights which do not significantly affect the state (i.e., on or off) of the hidden or output node are deleted. After the above changes to connection weights, the network has to be retrained in order to re-optimize threshold units for hidden and output nodes. At last, one rule can be formed for each hidden and output node by collecting all contributing nodes into the M antecedent group. The value of N is determined by the magnitude of the connection weight from the threshold unit to the hidden or output node. NofM is developed for networks configured by domain knowledge.

Some other developments for rule extraction can be found in BRAINNE [20], RX [13], RuleX [1], and VIA [25], which share similar characteristics as the Subset and NofM methods in one or more aspects.

The GLARE Algorithm

The GLARE algorithm [9] was developed to extract symbolic classification rules from neural networks trained by backpropagation. GLARE has several unique and advantageous characteristics. First, GLARE does not require the insertion of rules into the network before rule extraction. This characteristic renders GLARE applicable to classification problems without domain knowledge. Second, GLARE preserves the learning power of partially activated nodes by interpreting both the node output and the connection weights. Because of this feature, GLARE avoids the problem of distorting the node output by forcing them into 0, 1, or –1. Third, GLARE extracts only one composite rule for each class, which simplifies the classification procedure for new cases and explains the classification procedure succinctly. Fourth, GLARE establishes a direct relationship between input attribute nodes and output class nodes, which eliminates the need for coding hidden nodes in classification rules.

There are two restrictions for the application of GLARE. First, since GLARE is designed for nominal input attributes, continuous input attributes have to be centered and converted to nominal attributes. Centering allows the elimination of threshold units from networks, and also ensures the comparability of connection weights. Before backpropagation training, nominal attributes have to be converted to dummy variables. For example, if an attribute has three category values, we will use the dummy variables "1 0 0" to represent category value 1, "0 1 0" to represent category value 2, and "0 0 1" to represent category value 3. Second, the current implementation of GLARE is restricted to neural networks with only one hidden layer. This should not be a severe handicap as it has been proved that one hidden layer with sufficient number of hidden nodes can approximate any Borel measurable function [3, 10].

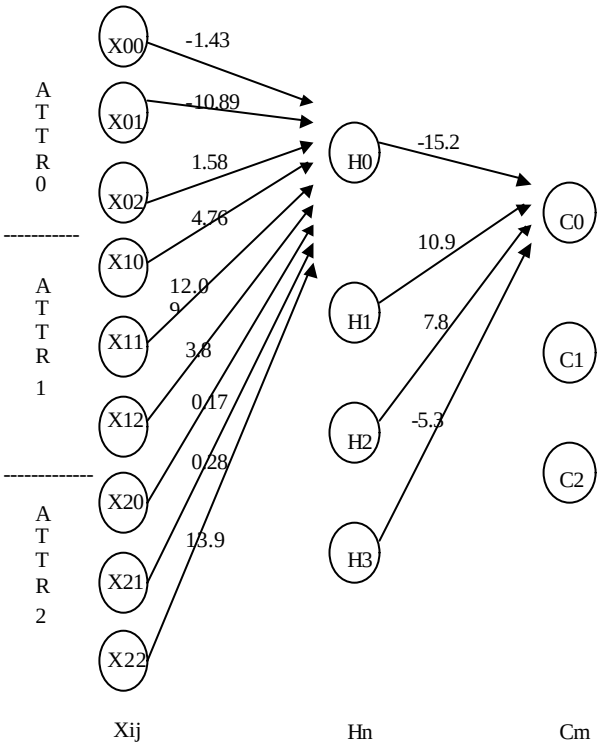

Figure 1 The GLARE algorithm for Rule Extraction

The remaining of this section summarizes the rule extraction procedure in GLARE, illustrated by extracting the classification rule for class 0 from the trained network in Figure 1. For the purpose of clarity, Figure 1 shows only connection weights to hidden node 0 and output node 0. X_{ij} represents category value j of attribute i. X_{ij} equals to 1 (0) if attribute i has (does not have) category value j. H_n is hidden node n, and C_m is output node m representing class m in the data set. The network in Figure 1 has 9 input nodes (i = 0, 1, 2, and j = 0, 1, 2 for each i), 4 hidden nodes (n = 0, 1, 2, 3), and 3 output nodes (m = 0, 1, 2). For each output node in the network, GLARE performs the following steps to extract one classification rule:

Step (1): Create RI_n – Ranking of Input Nodes for Hidden Node n.

For each hidden node n in the network, create ranking RI_n. RI_n is the ranking of all input nodes based on the descending order of absolute values of connection weights between input nodes to hidden node n. A positive or negative sign is added to each input node in RI_n to indicate whether the connection weight between the input node and hidden node n is positive or negative. The output of step (1) is a set of rankings RI_n where n = 0, 1, ..., N, and N is the total number of hidden nodes.

For example, in order to extract the classification rule for class 0 in Figure 1, we first create the following RI_n for output node C_0. Note that the connection weights for H_1, H_2, and H_3 are not shown in Figure 1, and are assumed to generate RI_1 to RI_3 as follows:

RI_0: $+X_{22}$ $+X_{11}$ $-X_{01}$ $+X_{10}$ $+X_{12}$ $+X_{02}$ $-X_{00}$ $+X_{21}$ $+X_{20}$
RI_1: $+X_{00}$ $+X_{01}$ $+X_{02}$ $+X_{10}$ $-X_{11}$ $-X_{12}$ $-X_{20}$ $-X_{21}$ $-X_{22}$
RI_2: $+X_{00}$ $-X_{02}$ $+X_{11}$ $-X_{20}$ $+X_{22}$ $-X_{01}$ $+X_{10}$ $-X_{12}$ $+X_{21}$
RI_3: $-X_{22}$ $-X_{21}$ $-X_{20}$ $-X_{12}$ $+X_{11}$ $+X_{10}$ $-X_{02}$ $-X_{01}$ $-X_{00}$.

Step (2): Create reduced RI_n.

Set the value of the parameter NW (number of connection weights) to p where p is greater than or equal to 1 and less than or equal to the total number of input nodes. Then, the first p input nodes in RI_n are retained for further processing, and remaining input nodes are deleted. The output of this step is a set of reduced rankings RI_n where n = 0, 1, ..., N, N is the total number of hidden nodes, and there are p input nodes in each reduced RI_n. The purpose of this step is to select several largest connection weights for rule extraction.

For example, suppose we set NW to 2, then we will have the following reduced RI_n for output node 0 in Figure 1:

RI_0: $+X_{22}$ $+X_{11}$
RI_1: $+X_{00}$ $+X_{01}$
RI_2: $+X_{00}$ $-X_{02}$
RI_3: $-X_{22}$ $-X_{21}$.

Step (3): Calculate $I(H_{mn})$ – Importance Index for Hidden Node n to output node m.

Resubmit all training cases of class m to the trained network. Notice that the network must be trained before the resubmission. For each hidden node, record the activation level of each resubmitted training case, calculate the average activation level, then calculate the importance index using the following equation:

$$I(H_{mn}) = ABS(L_{mn} * W_{mn}) \quad (1)$$

where $I(H_{mn})$ is the importance index of hidden node n to output node m, ABS(.) indicates absolute value, L_{mn} is the average activation level of hidden node n for training cases of class m, and W_{mn} is the connection weight from hidden node n to output node m. The purpose of this step is to take into consideration the partial activation level of a hidden node, and thus eliminate the practice of clamping the output of a partially activated node into on or off.

Following the example for extracting the rule for class 0, we calculate $I(H_{00})$, $I(H_{01})$, $I(H_{02})$, and $I(H_{03})$ using equation (1).

Step (4): Create RO_m – Ranking of all Hidden Nodes for Output Node m.

Create the ranking RO_m. RO_m is the ranking of all hidden nodes for output node C_m based on the descending order of the importance indexes from step (3). A positive or negative sign is added to each hidden node in RO_m to indicate whether the connection weight between the hidden node and output node m is positive or negative. The output of step (4) is a ranking of hidden nodes based on their importance on determining the output value of output node m.

Following the example for extracting the rule for class 0, suppose that the importance indexes from step (3) generate the following RO_0:

RO_0: $+H_1$ $-H_0$ $+H_2$ $-H_3$.

Note that we attach the positive sign to H_1 and H_2, and the negative sign to H_0 and H_3, based on the signs of connection weights from those hidden nodes to C_0. The above RO_0 indicates that H_1 is most important for determining the output value of C_0, followed by H_0, then H_2, and H_3 is least important.

Step (5): Create the Matrix A

RO_m (one ranking) from step (4) and RI_n (N rankings) from step (2) are used to construct the matrix *A*. The matrix *A* consists of RI_n reordered and adjusted based on RO_m. First, we reorder the rows of RI_n from step (2) according to the order of hidden nodes in RO_m. Second, for hidden nodes with negative signs in RO_m, we flip the sign of all input nodes in the corresponding RI_n. The output of step (5) is an $N \times p$ matrix. An element of $+X_{ij}$ ($-X_{ij}$) in *A* indicates that in order for output node C_m to have a high output (so that the case will be classified as class m), input node X_{ij} must have the input value 1 (0).

For the example of extracting the rule for class 0, we have the following matrix *A*:

		column 0	column 1
	0	$+X_{00}$	$+X_{01}$
row	1	$-X_{22}$	$-X_{11}$
	2	$+X_{00}$	$-X_{02}$
	3	$+X_{22}$	$+X_{21}$.

Notice that in the above matrix *A*, we put R_1 in row 0, R_0 in row 1, R_2 in row 2, and R_3 in row 3, as demanded by the order of hidden nodes in RO_0. We also flip the signs of the input nodes in R_0 and R_3 because H_0 and H_3 have negative signs in RO_0.

Step (6): Create the Matrix RA – Rule Matrix.

Create the matrix *RA* based on the *A* from step (5). Rows in *RA* represent input attributes (indexed by i) and columns

represent category values (indexed by j). We first initialize *RA* to 0. Notice that the cumulative effect of steps (1) to (5) is to arrange important attribute values (which affect the output of C_m significantly) to be in left columns and top rows in *A*. Following the directions of top to down and left to right, we use elements in *A* to determine element values in *RA*. An element of $+X_{ij}$ ($-X_{ij}$) in *A* will set category j of attribute i in *RA* to 1 (-1). Once an element in *RA* is set, it will not be reset. In other words, less important elements in *A* have only residual power to determine element values in *RA*. The matrix *RA* can be used to construct the classification rule for class m.

For the example of extracting the rule for class 0, we have the following *RA*:

		Categories (j)		
		0	1	2
	0	1	1	-1
Attributes (i)	1	0	-1	0
	2	0	1	-1 .

The procedure of filling in the above *RA* is as follows. We start with element 0 in row 0 from *A*. Since that element is $+X_{00}$, we set category 0 of attribute 0 in *RA* to 1. Then, we use element 1 in row 0 from *A*, and since that element is $+X_{01}$, we set category 1 of attribute 0 to 1. The filling in procedure will go on until we exhaust all elements in *A*. Notice that since element 0 in row 1 from *A* has set category 2 of attribute 2 to -1, element 0 in row 3 cannot reset that to 1, according to the residual power principle for elements in *A*.

Step (7): Create the Classification Rule for Class m.

Based on element values in *RA* from step (6), we construct a classification rule for class m. An element of 1 (-1) in *RA* indicates that attribute i must have (must not have) category value j for class m. An element of 0 indicates that it does not matter whether attribute i has category value j. The current implementation of GLARE algorithm treats 0 the same as 1. When there are two or more "1" for an input attribute from *RA*, attribute values are connected by the logical connector OR in the premise of the rule. Input attributes are connected by the logical connector AND.

Using the *RA* from step (6), we construct the following rule for class 0:

IF attribute 0 = 0 or 1 and
attribute 1 = 0 or 2 and
attribute 2 = 0 or 1
THEN class = 0.

The application order of rules to a new case can be very important for the correct classification of the case. The current implementation is to apply the most restrictive rule first, i.e., the rule with the most –1's in the *RA* matrix. For new cases to which no rule can be applied, the majority class in the training set is used as the default class. To avoid noise, it may not be necessary for a new case to match all attribute values in a rule in order to be labeled as the class indicated by the rule. The GLARE algorithm has a parameter NR that specifies the minimum number of attributes a case must match in order to be classified as the class for a rule.

EXPERIMENTAL DESIGN

This section describes the data set used in the experiment, the conversion process for continuous attributes, experimental procedure, and implementation details. The objective of the experiment is to verify the validity of extracted rules from GLARE for explanation and prediction purposes. We compared the performance of rules from GLARE with neural networks per se.

Table 1 Financial Data as Predictor Attributes

Variable	Definition
v1	Current Assets/Current Liabilities Proxy: (Cash + Marketable Securities + Net Receivables)/Current Liabilities
v2	Net Sales/Total Assets
v3	Net Income/Net Sales
v4	(Long Term Debt + Short Term Debt)/ Total Assets
v5	Total Sources of Fund/Total Uses of Fund Proxy: (Cash + Marketable Securities + Net Receivables)/Current Liabilities
v6	Research Expense
v7	Pretax Income/Net Sales
v8	Current Assets/Common Shareholders' Equity
v9	Common Shares Traded
v10	Capital Expenditure
v11	Earnings Per Share
v12	Dividend Per Share
v13	Depreciation Expense
v14	Tax Deferral and Investment Credit
v15	Market Capitalization = Stock Price × Common Shares Outstanding
v16	Relative Strength Index (RSI) = 100 - 100/(1 + RS) RS = Average of m periods' up closes/Average of m periods' down closes

The data set consists of 364 S&P companies for a period of 1985-1995 from the CompuStat database. We extract annual financial statement data from the Industrial Annual file. Based on recommendations from previous studies [11, 22, 21, 17, 14, 5, 15], we select 16 financial statement variables as the predictor attributes. The definitions of the variables are given in Table 1. In order to have cases with all the required variables, we eliminate cases with missing data or use proxies as much as we can. This process gives a sample of 364 companies from the S&P 500 list. The to-be-predicted variable is the rate of return on common shareholders' equity, which is defined as (Net Income - Preferred Dividend)/Common Shareholders' Equity. We

classify cases as high, medium, or low rate of return by selecting the top 120 companies as the high, the next 122 companies as the medium, and the last 122 companies as the low category. From the viewpoint of positive and negative examples, the high category represents positive examples, and the medium and low categories represent negative examples. For the purpose of this study, we are interested in selecting only positive cases.

The experiment procedure is a training and test process. Each training example or test case has an input vector which comprises the 16 financial statement variables, and an output classification which is either high, medium, or low rate of return. Because we are interested in selecting high return stocks, we calculate the average return of all the selected high return stocks from the neural network, and compare that average with the average of all companies in the test set. We also calculate the correct classification rates for reference purpose. If the trained network does not select any company as high return stock, the medium return stocks will be used, which happens in three training results. If the trained network does not select any company as high or medium return stock, the low return stocks will be used, which happens in one training result.

The experiment uses 1 year's financial data to predict the classification in the next year. The training set has financial data (input) from year n and classification (output) from year n+1. The test set has financial data from year n+1 and classification from year n+2. Each pair of training and test set involves financial data from 3 years. Starting from the beginning of the sample period, we have 85, 86, 87 as the first training and test set, 86, 87, 88 as the second training and test set, and so on. The sliding training and test window creates 9 sets of training and test samples for the experiment. Each training and test set has 364 companies.

The application of the GLARE algorithm requires continuous attributes be converted to nominal and then dummy variables. The conversion procedure is as follows. For each attribute, we first calculate the difference d between the maximum and minimum of all values. Then divide d by 5 receiving the quotient x. Attribute values which are greater than or equal to the minimum and less than (minimum + x) are classified into category 1, attribute values which are greater than or equal to (minimum + x) and less than (minimum + 2x) are classified into category 2, and so on. After conversion, each training case has a certain category for a certain attribute. The value of category 1 is further converted to the dummy values "1 0 0 0 0", the value of category 2 to "0 1 0 0 0", the value of category 3 to "0 0 1 0 0" and so on. The conversion process yields 80 input nodes (16 attributes × 5 dummy values) and 3 output nodes (high, medium, and low return) in a neural network. For test sets, they need to be converted to nominal values only for the purpose of applying classification rules. For each pair of the 9 training and test sets, we perform the following steps in the experiment:

(1) Apply conventional backpropagation to the training set with continuous input variables. Use the trained network to predict the test set with continuous input values. Calculate the average return of the high return stocks selected by the network.
(2) Apply conventional backpropagation to the training set with dummy input variables. Train the network with 1000 epochs. Apply GLARE to the trained network to extract classification rules. Use extracted rules to predict the test set with nominal input values. Calculate the average return of the high return stocks selected by the network.
(3) Apply conventional backpropagation to the training set with dummy input variables. Train the network with 5 epochs. Apply GLARE to the trained network to extract classification rules. Use extracted rules to predict the test set with nominal input values. Calculate the average return of the high return stocks selected by the network.

Paired t tests for mean differences are carried out to verify the significant differences between average return from the above steps and the average return of all companies. For all the neural network training sessions, we use 1 hidden layer, 30 hidden nodes, learning rate 0.5, and momentum rate 0.0. For the GLARE algorithm, we set NR (number of input attribute values a case must match to be classified as a certain class) to 10, and NW (number of weights used in building the reduced rankings for all input nodes to a hidden node) to 30. The backpropagation algorithm is implemented in the C language. All simulations are performed in a desk top computer.

EXPERIMENTAL RESULTS AND DISCUSSION

Tables 2 - 4 show the experimental results. Table 2 describes the correct classification rate (%) for conventional backpropagation training, extracted rules for 1000 training epochs, and extracted rules for 5 training epochs. Table 3 shows the average return from conventional backpropagation training, extracted rules for 1000 training epochs, and extracted rules for 5 training epochs. The conventional backpropagation, rules for 1000 epochs, and rules for 5 epochs achieve average returns of 0.1666, 0.19379, and 0.25398 respectively, which are all higher than the average of all companies 0.10851. In Table 4, we analyze the significant mean differences from Table 3. Using 0.05 as the level of significance, we find conventional backpropagation, rules for 1000 epochs, and rules for 5 epochs are all significantly higher than the average of all. Rules for 5 epochs achieve significantly higher return than rules for 1000 epochs. Comparing conventional backpropagation with extracted rules, rules for 5 epochs perform significantly better than conventional backpropagation while rules for 1000 epochs cannot be considered as better than conventional backpropagation statistically. Table 5 reports the symbolic classification rules extracted from the trained neural network for the prediction of year 1994 return.

CONCLUSION

This research project evaluated a rule extraction algorithm GLARE for neural networks trained using the backpropagation algorithm. The prediction performance of extracted rules is compared with neural networks per se. The experiment result indicates that extracted rules from GLARE perform significantly better than neural networks per se. The extracted rules also reveal the importance and interaction of input attributes, which provides explanation power to neural networks. One limitation of GLARE is its applicability to only the backpropagation training algorithm. Future research can focus on developing algorithms to extract rules from other types of networks and training algorithms.

Table 2 Correct Classification Rate (%) from Neural Network and Rule Extraction

Predicted Year	NN* Train	NN Test	Rule1K* Train	Rule1K Test	Rule5* Train	Rule5 Test
1987	95.05	35.44	87.36	31.87	92.31	32.14
1988	75.27	32.14	80.77	34.34	88.74	31.87
1989	76.37	34.34	81.32	25.55	89.29	31.32
1990	84.62	48.08	80.22	33.24	89.01	31.87
1991	89.84	59.89	87.36	37.09	90.38	29.12
1992	85.44	60.71	83.79	34.89	91.21	28.30
1993	89.56	53.57	79.12	36.54	91.48	33.52
1994	85.99	51.37	85.99	30.49	90.38	30.49
1995	85.44	49.45	85.16	30.77	91.21	31.59

* NN: Conventional backpropagation training.
Rule1K: Training has 1000 epochs before rule extraction.
Rule5: Training has 5 epochs before rule extraction.

Table 3 Rate of Return from Neural Network and Rule Extraction

Predicted Year	Average of All Companies	NN*	Rule1000*	Rule5*
1987	0.13041	0.13281	0.13843	0.23654
1988	0.10535	0.03268	0.24203	0.25158
1989	0.14292	0.13529	0.14281	0.28616
1990	0.15247	0.20769	0.12740	0.24372
1991	-0.00144	0.20036	0.23250	0.23006
1992	0.05856	0.14703	0.26326	0.25975
1993	0.08487	0.16121	0.11837	0.19832
1994	0.15980	0.23751	0.33356	0.33852
1995	0.14363	0.24482	0.14574	0.24119
Average	0.10851	0.16660	0.19379	0.25398

* NN: Conventional backpropagation training.
Rule1000: Training has 1000 epochs before rule extraction.
Rule5: Training has 5 epochs before rule extraction.

Table 4 P-value for 1-Tailed Paired t-Test (Data from Table 3)

	Compared with			
	Average of All	NN	Rule1000	Rule5
Average of All	-------	-------	-------	-------
NN	0.028	-------	-------	-------
Rule1K	0.024	0.229	-------	-------
Rule5	0.000	0.003	0.011	-------

Table 5 Classification Rules from Rule1000 and Rule5 Predicting Year 1994

Rule1K*	If	v1 = category 1, 3, or 4 and v2 = category 1, 2, or 3 and v3 = category 2, 3, or 5 and v4 = category 4 or 5 and v5 = category 3 and v6 = category 2 or 5 and v7 = category 1, 2, 4, or 5 and v8 = category 3 or 5 and v9 = category 1, 2, 3, or 4 and v10 = category 2 or 5 and v11 = category 2, 3, 4, or 5 and v12 = category 1, 2, 3, or 5 and v13 = category 2, 3, or 5 and v14 = category 1, 2, 4, or 5 and v15 = category 2, 3, or 4 and v16 = category 2, 4, or 5
	Then	high return stock.
Rule5*	If	v1 = category 2 or 4 and v2 = category 1, 2, or 4 and v3 = category 1, 3, or 4 and v4 = category 1, 2, or 4 and v5 = category 1, 2, 3, 4, or 5 and v6 = category 1, 4, or 5 and v7 = category 1, 2, or 3 and v8 = category 1, 2, or 3 and v9 = category 1 or 5 and v10 = category 3 and v11 = category 2, 3, or 5 and v12 = category 1, 4, or 5 and v13 = category 1, 3, or 5 and v14 = category 2, 3, or 5 and v15 = category 1, 4, or 5 and v16 = category 4 or 5
	Then	high return stock.

* Rule1000: Rule generated from neural network after 1000 training epochs.

* Rule5: Rule generated from neural network after 5 training epochs.

REFERENCES

[1] ANDREWS, R., DIEDERICH, J., and TICKLE, A. B. (1994) Rule extraction from a constrained error back propagation MLP. *Proceedings of Fifth Austrialian Conference on Neural Networks*, 9-12.

[2] BARNARD, E., and D. CASASENT (1989) A comparison between criterion functions for linear classifiers, with an application to neural nets. *IEEE transactions on System, Man, and Cybernetics*, **19**, 1030-1041.

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

[4] DECATUR, S. E. (1989) Application of neural networks to terrain classification. *Proceedings of the International Joint Conference on Neural Networks*, **1**, 283-288.

[5] DROPSY, V. (1996) Do macroeconomic factors help in predicting international equity risk premia? *Journal of Applied Business Research*, **12:3**, 120-132.

[6] DUTTA, S., and S. SHEKHAR (1988) Bond-rating: A non-conservative application of neural networks. *Proceedings of the IEEE International Conference on Neural Networks*, **2**, 443-450.

[7] FU, L. M. (1991) Rule learning by searching on adapted nets. *Proceedings of the Ninth National Conference on Artificial Intelligence*, 500-595.

[8] GALLANT, S. I. (1988) Connectionist expert systems. *Communications of the ACM*, **31**, 152-169.

[9] GUPTA, A., S. PARK, & S. M. LAM (1999) Generalized analytic rule extraction for feedforward neural networks. *IEEE Transactions on Knowledge and Data Engineering*, **11:6**, 985-991.

[10] HORNIK, K., M. STINCHCOMBE, and H. WHITE (1989) Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359-366.

[11] KIANG, M. Y., R. CHI, and K. Y. TAM (1993) DKAS: A distributed knowledge acquisition system in a DSS. *Journal of Management Information Systems*, **9:4**, 59-82.

[12] LAM, S. M. (1994) *Weight Decay Training, Fuzzy Set Techniques, and Rule Extraction for Backpropagation*. Ph.D. Thesis, University of Wisconsin-Madison.

[13] LU, H., SETIONO, R., and LIU, H. (1996) Effective data mining using neural networks. *IEEE Transactions on Knowledge and Data Engineering*, **8:2**, 957-961.

[14] MEYERS, T. (1989) *The Technical Analysis Course: A Winning Program for Stock and Future Traders & Investors.* Chicago: Probus.

[15] OU, J. A. (1990) The information content of nonearnings accounting numbers as earnings predictors. *Journal of Accounting Research*, **28**, 144-163.

[16] PAO, Y. H. (1989) *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley.

[17] RITCHIE, J. (1989) *Fundamental Analysis: A Back-to-the-Basics Investment Guide to Selecting Quality Stocks*. Chicago: Probus.

[18] ROSENBLATT, F. (1962) *Principles of Neurodynamics*. New York: Spartan.

[19] SAITO, K., and R. NAKANO (1988) Medical diagnostic expert system based on PDP model. *Proceedings of IEEE International Conference on Neural networks*, **1**, 255-262.

[20] SESTITO, S., and DILLON, T. S. (1994) Automated Knowledge Acquisition. Prentice Hall.

[21] SHARPE, W. F. (1966) Mutual fund performance. *Journal of Business*, **39**, 119-138.

[22] TREYNOR, J. L. (1965) How to rate management of investment funds. *Harvard Business Review*, **43:1**, 63-75.

[23] TAM, K. Y., and M. L. KIANG (1992) Managerial applications of neural networks: the case of bank failure predictions. *Management Science*, **38**, 926-947.

[24] TANG, Z., C. de ALMEIDA, and P. FISHWICK (1990) Time series forecasting using neural networks vs. Box-Jenkins methodology. *Proceedings of the First Workshop on Neural Networks: Academic/Industrial/NASA/Defense*, 95-100.

[25] THRUN, S. B. (1994) Extracting probably correct rules from artificial neural networks, Technical Report IAI-TR-93-5. Institute for Informatik III Universitat Bonn, Germany.

[26] TOWELL, G. G. (1991) *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* Ph.D. Thesis, University of Wisconsin-Madison.

[27] TOWELL, G. G., and J. W. SHAVLIK (1992) Interpretation of artificial neural networks: Mapping knowledge-based neural networks into rules. In J. E. Moody, S. J. Hanson, and R. P. Lippmann (Eds.), *Advances in Neural information Processing Systems 4*, San Mateo, CA: Morgan Kaufmann.