PERGAMON

Neural Networks 15 (2002) 1279–1288

Neural Networks

www.elsevier.com/locate/neunet

Extracting regression rules from neural networks

Kazumi Saito[a,*], Ryohei Nakano[b]

[a]NTT Communication Science Laboratories, NTT Corporation, 2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0237, Japan
[b]Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya 466-8555, Japan

Received 22 August 2001; accepted 10 June 2002

Abstract

This paper proposes a new framework and method for extracting regression rules from neural networks trained with multivariate data containing both nominal and numeric variables. Each regression rule is expressed as a pair of a logical formula on the conditional part over nominal variables and a polynomial equation on the action part over numeric variables. The proposed extraction method first generates one such regression rule for each training sample, then utilizes the *k*-means algorithm to generate a much smaller set of rules having more general conditions, where the number of distinct polynomial equations is determined through cross-validation. Finally, this method invokes decision-tree induction to form logical formulae of nominal conditions as conditional parts of final regression rules. Experiments using four data sets show that our method works well in extracting quite accurate and interesting regression rules.

Keywords: Polynomial equations; Nominal variables; Regression rules; Rule extraction; Vector quantization; Cross-validation; Decision trees

1. Introduction

Modeling complex phenomena such as price tendency of goods is a challenging but important research issue. Such modeling tasks will be greatly promoted if we can efficiently and adequately find some hidden but intrinsic relationships from previously observed data. To this end, we have proposed a method called RF5[1] (Saito & Nakano, 1997a,b), which employs neural networks of a polynomial type for producing explicitly understandable learning results from multivariate numeric data.

However, the data observed in many real fields usually contains both nominal and numeric values. By adequately encoding nominal values into numeric ones, we can straightforwardly apply RF5 to such data, but the resulting polynomial equations will not be explicitly understandable, i.e. the numeric equations over such encoded nominal values cannot be directly interpretable for our intuition. To overcome this problem, this paper proposes a new framework and method for extracting regression rules from trained neural networks, where the conditional part of each regression rule is expressed as a logical formula over nominal variables, and the action part is expressed as a polynomial equation over numeric variables.

Although there exists a large amount of work on rule extraction from trained neural networks (Andrews, Diederich, & Tickle, 1995; Ishikawa, 2000; Mitra & Hayashi, 2000), the work has emphasized classification tasks rather than regression models that predicts some continuous variable. In fact, it was stated in a survey paper by Tickle, Andrews, Golea, and Diederich (1998) that one future direction for knowledge-extraction techniques is enabling to deal with neural networks of real-valued outputs. We believe that our proposing framework and method will take an important step along this research direction.

This paper is organized as follows. Section 2 formalizes our extraction problem and explains the basic framework associated with extracting regression rules from neural networks trained with data containing both nominal and numeric values. Section 3 gives details of the rule extraction method called RN2, which adopts the *k*-means algorithm for finding representative values, the cross-validation error as a criterion for model selection, and the c4.5 program for generating nominal conditions. Section 4 reports experimental results using one artificial and three real data sets. Section 5 discusses related work and directions for future research.

* Corresponding author. Tel.: +81-774-93-5137; fax: +81-774-93-5155.
E-mail addresses: saito@cslab.kecl.ntt.co.jp (K. Saito), nakano@ics.nitech.ac.jp (R. Nakano).

[1] RF5 denotes *Rule extraction from Facts, version 5*.

0893-6080/02/$ - see front matter
PII: S0893-6080(02)00089-8

Nomenclature

x_k	numeric input variable
K	number of numeric input variables
y	numeric output (target) variable
w_j	weight between hidden unit j and output unit
w_{jk}	weight between input unit for x_k and hidden unit j
J	number of hidden units
q_l	nominal input variable
L	number of nominal input variables
q_{lm}	dummy variable for q_l
M_l	number of categories (dummy variables) for q_l
Q_l^i	non-empty subset of categories associated with q_l appearing in rule i
I^*	number of regression rules
$y(\mathbf{q},\mathbf{x};\boldsymbol{\Theta})$	neural networks considered in this paper
$\boldsymbol{\Theta}$	vector consisting of all weights
P	number of weights
v_{jkl}	weight between input unit q_{kl} and hidden unit j
D	training data
N	number of training samples
T	test data
$\mathscr{G}(\boldsymbol{\Theta})$	generalization error
$\mathscr{L}(\boldsymbol{\Theta})$	objective function for least-squares estimate
$\mathscr{E}(\boldsymbol{\Theta})$	penalized objective function
λ_p	penalty factor for weight θ_p
c_j^μ	activation value for hidden unit j from nominal value of sample μ
r_j	representative value for hidden unit j
G_i	set of partitioned vectors
$\mathscr{VQ}$	objective function for quantization
N_i	number of vectors belonging to G_i
I	number of representatives
S	number of data segments for cross-validation
$I(\mathbf{q})$	indexing function
$\mathscr{CV}$	objective function for cross-validation
I'	number of terminal nodes in decision tree

2. Framework for rule extraction

2.1. Polynomial expressions

Let $(x_1,\ldots,x_K,y)$ or $(\mathbf{x},y)$ be a vector of variables describing a sample, where x_k is a numeric input variable and y is a numeric output (or target) variable. Then the RF5 algorithm (Saito & Nakano, 1997a,b) searches for multivariate polynomial equations of the following form, whose power values are not restricted to integers

$$y = w_0 + \sum_{j=1}^{J} w_j \prod_{k=1}^{K} x_k^{w_{jk}} = w_0 + \sum_{j=1}^{J} w_j \exp\left(\sum_{k=1}^{K} w_{jk} \ln x_k\right), \tag{1}$$

where J denotes the number of terms appearing in polynomial equations.

Clearly, Eq. (1) can be regarded as a feedforward computation of three-layer neural networks where the activation function of each hidden unit is $\exp(s) = e^s$. These hidden units are computationally powerful and referred to as *product units* (Durbin & Rumelhart, 1989). Moreover, such functional relations can represent many of the numeric laws found by previous scientific discovery systems like BACON (Langley, Simon, Bradshaw, & Żytkow, 1987). By using Eq. (1) that directly express multivariate polynomial equations, available domain knowledge will be straightforwardly embedded within the neural networks, and we can easily interpret the learned results. Therefore, by repeating such interactions with domain experts, we can expect to build practical models using past observed data.

2.2. Regression rules

Let $(q_1,\ldots,q_L,x_1,\ldots,x_K,y)$ or $(\mathbf{q},\mathbf{x},y)$ be a vector of variables describing a sample, where q_l is a nominal input variable. Here, by adding extra categories, if necessary,

without losing generality, we can assume that q_l exactly matches the only one category. Therefore, for each q_l we introduce a *dummy variable* expressed by q_{lm} as follows:

$$q_{lm} = \begin{cases} 1 & \text{if } q_l \text{ matches the } m\text{th category,} \\ 0 & \text{otherwise.} \end{cases}.$$

Here $m = 1, \ldots, M_l$, and M_l is the number of distinct categories appearing in q_l. Hereafter, $\mathbf{q}$ denotes a vector constructed by arranging all dummy variables.

As a class of target relationships, we consider a set of *regression rules*, where the conditional part of each rule is expressed as a logical formula over nominal variables, and the action part is expressed as a polynomial over numeric variables. Let Q_l^i be a non-empty subset of all the categories (dummy variables) associated with q_l, appearing in the conditional part of the ith rule. In this paper, regression rule sets of the following form are considered:

$$\text{if } \bigwedge_{l=1}^{L} \bigvee_{q_{lm} \in Q_l^i} q_{lm} \text{ then } y = w_0^i + \sum_{j=1}^{J^i} w_j^i \prod_{k=1}^{K} x_k^{w_{jk}^i}, \qquad i = 1, \ldots, I^*, \tag{2}$$

where I^* is the number of rules. Note that when Q_l^i contains all the categories associated with q_l, the corresponding disjunctive condition $\bigvee_{m=1}^{M_l} q_{lm}$ becomes always true, and we usually omit it from the condition.

We consider that Eq. (2) is general enough and widely applicable to many problems. For example, Coulomb's law $F = 4\pi\epsilon e_1 e_2 / r^2$ relating the force of attraction F of two particles with charges e_1 and e_2, respectively, separated by a distance r depends on ϵ, the permittivity of surrounding medium; i.e. if substance is 'water' then $F = 8897.352 e_1 e_2 / r^2$, if substance is 'air' then $F = 111.280 e_1 e_2 / r^2$, and so on.

2.3. Equivalence of representations

As a natural extension of Eq. (1), we can consider neural networks of the following form, including both nominal and numeric input variables

$$y(\mathbf{q}, \mathbf{x}; \boldsymbol{\Theta}) = w_0 + \sum_{j=1}^{J} w_j g\left(v_{j0} + \sum_{l=1}^{L} \sum_{m=1}^{M_l} v_{jlm} q_{lm} \right) \times \exp\left(\sum_{k=1}^{K} w_{jk} \ln x_k \right), \tag{3}$$

where $\boldsymbol{\Theta}$ denotes a P-dimensional parameter vector constructed by arranging all weights w_j, v_{jlm} and w_{jk}. As the activation function g, this paper adopts an exponential one ($g(\cdot) = \exp(\cdot)$) in conformity with g for numeric variables. Since we can eliminate the bias weights v_{j0} by regarding $w_j \exp(v_{j0})$ as new weights, neural networks of the following form are obtained:

$$y(\mathbf{q}, \mathbf{x}; \boldsymbol{\Theta}) = w_0 + \sum_{j=1}^{J} w_j \exp\left(\sum_{l=1}^{L} \sum_{m=1}^{M_l} v_{jlm} q_{lm} \right) \times \exp\left(\sum_{k=1}^{K} w_{jk} \ln x_k \right). \tag{4}$$

As shown below, Eqs. (2) and (4) are almost equivalent if the nominal conditions in Eq. (2) are mutually disjoint. Firstly, by replacing *true* and *false* of the truth values with 1 and 0, respectively, prediction values based on Eq. (2) are equivalently calculated as follows:

$$y = \sum_{i=1}^{I^*} \left(\prod_{l=1}^{L} \sum_{q_{lm} \in Q_l^i} q_{lm} \right) \left(w_0^i + \sum_{j=1}^{J^i} w_j^i \prod_{k=1}^{K} x_k^{w_{jk}^i} \right). \tag{5}$$

Thus, by using the following approximation with a large positive β,

$$\left(\prod_{l=1}^{L} \sum_{q_{lm} \in Q_l^i} q_{lm} \right) \approx \prod_{l=1}^{L} \exp\left(\sum_{m=1}^{M_l} v_{lm}^i q_{lm} \right), \quad \text{where } v_{lm}^i = \begin{cases} 0 & \text{if } q_{lm} \in Q_l^i, \\ -\beta & \text{otherwise,} \end{cases}. \tag{6}$$

Eq. (5) can be almost equivalently transformed into

$$y = \sum_{i=1}^{I^*} w_0^i \exp\left(\sum_{l=1}^{L} \sum_{m=1}^{M_l} v_{lm}^i q_{lm} \right) + \sum_{i=1}^{I^*} \sum_{j=1}^{J^i} w_j^i \exp\left(\sum_{l=1}^{L} \sum_{m=1}^{M_l} v_{lm}^i q_{lm} \right) \exp\left(\sum_{k=1}^{K} w_{jk}^i \ln x_k \right). \tag{7}$$

Clearly, Eq. (7) can be regarded as a special case of Eq. (4). Note that the above approximation can be arbitrarily accurate by using a large enough β.

Conversely, by selecting some category number $m(l)$ for each nominal variable q_l and putting $Q_l = \{q_{lm(l)}\}$, we can extract the following regression rule from Eq. (4)

$$\text{if } \bigwedge_{l=1}^{L} \bigvee_{q_{lm} \in Q_l} q_{lm} \text{ then } y = w_0 + \sum_{j=1}^{J} w_j \prod_{l=1}^{L} \exp(v_{jlm(l)}) \prod_{k=1}^{K} x_k^{w_{jk}}, \tag{8}$$

where $w_j \prod_{l=1}^{L} \exp(v_{jlm(l)})$ is regarded as a polynomial coefficient. Thus, by extracting individual rules for each possible combination of the category numbers, we can obtain a set of regression rules, which is equivalent to Eq. (4).

Therefore, it was shown that Eqs. (2) and (4) have the same representational capability. Moreover, it was suggested that a set of regression rules can be trained by using a neural network, and the trained neural network can be transformed into a set of regression rules. However, this extraction approach inherently produces $\prod_{l=1}^{L} M_l$ regression

rules, i.e. it will suffer from combinatorial explosion when the number of nominal variables gets large. To obtain a reasonable rule set, this paper proposes an extraction method, which is described in Section 2.4.

2.4. Learning of neural networks

After employing some adequate objective function like a sum of squared errors, we can use any learning algorithm for training neural networks defined in Eq. (4). However, since there exist several desirable characteristics, we adopt the following to train neural networks.

Let $D = \{(\mathbf{q}^\mu, \mathbf{x}^\mu, y^\mu) : \mu = 1, \ldots, N\}$ be a set of training data, where N is the number of samples. Here, we assume that each training sample $(\mathbf{q}^\mu, \mathbf{x}^\mu, y^\mu)$ is independent and identically distributed. Now, our goal in learning with neural networks is defined as a problem of minimizing the generalization error, that is, finding the optimal estimator $\boldsymbol{\Theta}^*$ that minimizes

$$\mathcal{G}(\boldsymbol{\Theta}) = E_D E_T (y^\nu - y(\mathbf{q}^\nu, \mathbf{x}^\nu; \boldsymbol{\Theta}(D)))^2, \tag{9}$$

where $T = (\mathbf{q}^\nu, \mathbf{x}^\nu, y^\nu)$ denotes test data independent of the training data D. The *least-squares estimate* of $\boldsymbol{\Theta}^*$, denoted by $\hat{\boldsymbol{\Theta}}$, minimizes the error sum of squares

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{2} \sum_{\mu=1}^{N} (y^\mu - y(\mathbf{q}^\mu, \mathbf{x}^\mu; \boldsymbol{\Theta}))^2. \tag{10}$$

However, this estimation is likely to overfit to noise of the training data, especially when we employ non-linear models such as neural networks. Thus, by using Eq. (10) as our criterion we cannot obtain good results in terms of the generalization performance defined in Eq. (9) (Bishop, 1995).

It is widely known that adding some penalty term to Eq. (10) can lead to significant improvements in network generalization (Bishop, 1995). To improve both the generalization performance and the readability of the learning results, we adopt a method to learn a distinct penalty factor for each weight as a minimization problem over the cross-validation error, called the minimum cross-validation (MCV) regularizer (Saito & Nakano, 2000b). Let $\hat{\lambda}_p$ (>0) be a learned penalty factor for a weight θ_p, then we can obtain the resulting neural network subject to Eq. (4) by finding the $\boldsymbol{\Theta}$ that minimizes the following objective function for weights

$$\mathcal{E}(\boldsymbol{\Theta}) = \mathcal{L}(\boldsymbol{\Theta}) + \frac{1}{2} \sum_{p=1}^{P} \hat{\lambda}_p \theta_p^2. \tag{11}$$

In order to efficiently and constantly obtain good learning results, we employ a second-order learning algorithm called *BPQ* (Saito & Nakano, 1997a). By adopting a quasi-Newton method (Luenberger, 1984) as a basic framework, we calculate the descent direction on the basis of a partial BFGS (Broyden–Fletcher–Goldfarb–Shanno) update and then efficiently calculate a reasonably accurate step-length as the minimal point of a second-order approximation. Note that in our own experiments (Saito & Nakano, 2000a), the combination of the squared penalty term and the BPQ algorithm drastically improves the convergence performance in comparison to other combinations, and at the same time, brings about excellent generalization performance.

3. Method for rule extraction

3.1. Overview of the method

Assume that we have already obtained a trained neural network. In order to find a set of regression rules as described in Eq. (2), we need a suitable efficient method to extract the nominal conditions from the trained neural network.

We can straightforwardly extract a regression rule for each training sample, and simply assemble them to obtain a rule set. Namely, the jth hidden unit calculates the following activation value from the encoded nominal values of the μth training sample

$$c_j^\mu = \exp\left(\sum_{l=1}^{L} \sum_{m=1}^{M_l} \hat{v}_{jlm} q_{lm}^\mu \right), \tag{12}$$

where $\hat{v}_{jlm}$ denotes the weights of the trained neural network. Thus, by putting $Q_l^\mu = \{q_{lm}^\mu : q_{lm}^\mu = 1\}$, we can obtain the following set of regression rules from the training data and the trained neural network

$$\text{if } \bigwedge_{l=1}^{L} \bigvee_{q_{lm} \in Q_l^\mu} q_{lm} \text{ then } y = \hat{w}_0 + \sum_{j=1}^{J} \hat{w}_j c_j^\mu \prod_{k=1}^{K} x_k^{\hat{w}_{jk}}, \tag{13}$$

$\mu = 1, \ldots, N.$

However, the results of this naive method can still be far from desirable because they contain a large number of similar rules, and each nominal condition is too specific representing only one training sample.

Based on the above considerations, we propose a new extraction method called RN2[2]; i.e. the number of distinct polynomial equations is reduced by finding representative values of Eq. (12), an adequate number of representatives is determined by using a criterion for model selection and a set of nominal conditions is determined by solving a standard classification problem by using decision trees.

3.2. Finding representative vectors

In order to find representative vectors, a set of vectors $\{\mathbf{c}^\mu = (c_1^\mu, \ldots, c_J^\mu)^{\mathrm{T}} : \mu = 1, \ldots, N\}$ calculated from the encoded nominal values is quantized into a set of representative vectors $\{\mathbf{r}^i = (r_1^i, \ldots, r_J^i)^{\mathrm{T}} : i = 1, \ldots, I\}$,

[2] RN2 denotes *Rule extraction from Neural networks, version 2*.

where I is the number of representatives. Among several vector quantization (VQ) algorithms, we employ the k-means algorithm (Lloyd, 1982) due to its simplicity.

In the k-means algorithm, all of the vectors are assigned simultaneously to their nearest representative vectors, each representative vector is moved to the group's mean and this process is repeated until there is no further change in the grouping of representative vectors. Consequently, all of the vectors are partitioned into I disjoint subsets $\{G_i : i = 1,\ldots,I\}$ so that the following sum-of-squares error function $\mathscr{VQ}$ is minimized:

$$\mathscr{VQ} = \sum_{i=1}^{I} \sum_{\mu\in G_i} \sum_{j=1}^{J} (c_j^{\mu} - r_j^i)^2. \tag{14}$$

Let N_i be the number of vectors belonging to G_i, then, each element of the representative vector is calculated as follows:

$$r_j^i = \frac{1}{N_i} \sum_{\mu\in G_i} c_j^{\mu}. \tag{15}$$

3.3. Criterion for representative number selection

For a given data set and trained neural network, since we do not know the optimal number of distinct polynomial equations in advance, we must evaluate the plausibility of the number of representatives by changing I. For this purpose, we employ the procedure of cross-validation (Stone, 1974) which was also mentioned in Section 3.2. This procedure divides the data D at random into S distinct segments $\{D_s : s = 1,\ldots,S\}$. $S-1$ segments are used for the training, and the remaining one is used for the test. This process is repeated S times by changing the remaining segment. The extreme case of $S = N$ is known as the *leave-one-out* method, which is often used for a small size of data (Bishop, 1995).

Now, we introduce a function $i(\mathbf{q})$ that returns the index of the representative vector minimizing the distance, i.e.

$$i(\mathbf{q}^{\mu}) = \arg\min_i \sum_{j=1}^{J} (c_j^{\mu} - r_j^i)^2. \tag{16}$$

By placing $i(\mathbf{q})$ on the conditional parts, we can consider the following set of rules using the representative vectors:

$$\text{if } i(q) = i \text{ then } y = \hat{w}_0 + \sum_{j=1}^{J} \hat{w}_j r_j^i \prod_{k=1}^{K} x_k^{\hat{w}_{jk}}, \qquad i = 1,\ldots,I. \tag{17}$$

Since each element of $\mathbf{c}$ is calculated as

$$c_j = \exp\left(\sum_{l=1}^{L} \sum_{m=1}^{M_l} \hat{v}_{jlm} q_{lm} \right). \tag{18}$$

Eq. (17) can be applied to a new sample, as well as the training samples. Thus, by using the final weights $\hat{\boldsymbol{\Theta}}^{(s)}$ calculated from the cross-validation procedure excluding one segment D_s, the output value with respect to a test sample ν can be calculated as

$$\hat{y}^{\nu} = \hat{w}_0^{(s)} + \sum_{j=1}^{J} \hat{w}_j^{(s)} r_j^{i(\mathbf{q}^{\nu})} \prod_{k=1}^{K} (x_k^{\nu})^{\hat{w}_{jk}^{(s)}}. \tag{19}$$

Therefore, we can define the following cross-validation error function

$$\mathscr{CV} = \frac{1}{N} \sum_{s=1}^{S} \sum_{\nu\in D_s} (y^{\nu} - \hat{y}^{\nu})^2. \tag{20}$$

3.4. Generating conditional parts

Finally, the indexing function $i(\mathbf{q})$ described in Eq. (17) must be transformed into a set of nominal conditions as described in Eq. (2). One reasonable approach is to perform this transformation by solving a simple classification problem whose training samples are $\{(\mathbf{q}^{\mu}, i(\mathbf{q}^{\mu})) : \mu = 1,\ldots,N\}$, where $i(\mathbf{q}^{\mu})$ indicates the class label of a training sample $\mathbf{q}^{\mu}$. Alternatively, a set of training samples belonging to class i is $\{\mathbf{q}^{\mu} : \mu \in G_i\}$. For this classification problem, we employ the c4.5 decision tree generation program (Quinlan, 1993) due to its wide availability.

In the induced decision tree, samples are passed from the root node to a terminal node that assigns the corresponding class label, with decisions being made at each non-terminal node. By concatenating such decisions at non-terminal nodes for the path to each terminal node, we can produce a set of conjunctive conditions. A procedure for constructing Q_l^i from the ith condition is as follows: Q_l^i is initialized to $\{q_{lm} : m = 1,\ldots,M_l\}$ for each l, then it is replaced with $\{q_{lm}\}$ (or $Q_l^i - \{q_{lm}\}$) according to each appearance of $q_{lm} = 1$ (or $q_{lm} = 0$) in the ith conjunctive condition. Therefore, we can obtain the following set of regression rules:

$$\text{if } \bigwedge_{l=1}^{L} \bigvee_{q_{lm}\in Q_l^i} q_{lm} \text{ then } y = w_0 + \sum_{j=1}^{J^i} w_j r_j^{\zeta(i)} \prod_{k=1}^{K} x_k^{w_{jk}}, \tag{21}$$

$$i = 1,\ldots,I',$$

where $\zeta(i)$ indicates the class label of the ith terminal node, and I' is the number of terminal nodes. In general, the number I' of terminal nodes is greater than or equal to the number $\hat{I}$ of distinct polynomial equations because the same equation may appear on more than one action parts. Note that it is possible to perform some simplifications of Eq. (21) such as the removal of $\bigvee_{m=1}^{M_l} q_{lm}$ and possible disjunctive integration of some nominal conditions having the same polynomial equations. Moreover, by using a special condition 'else', we can simplify one of nominal conditions and reduce the number of regression rules in many cases.

3.5. Summary of the method

The above procedure is summarized as follows:

Step 1. Calculate I representative vectors which minimize Eq. (14).
Step 2. Select the optimal $\hat{I}$ which minimizes Eq. (20).
Step 3. Generate nominal conditions using c4.5, and output the final rule set.

Clearly, these steps can be executed within the computational complexity of polynomial order with respect to the numbers of training samples, variables, hidden units, representatives, iterations performed by the k-means algorithm, and data segments used for cross-validation. Here note that the polynomial equations produced by our method differ only in their coefficients. However, since some polynomial coefficients can be zero, a set of arbitrary polynomial equations can be obtained if the adequate number of hidden units is prepared.

4. Evaluation by experiments

4.1. Experimental settings

By using artificial, scientific, financial and automobile data sets, we evaluated the RN2 method.

The common experimental settings for training neural networks are as follows. The initial values for the weights v_{jkl} and w_{jk} are independently generated according to a normal distribution with a mean of 0 and a standard deviation of 1; the initial values for the weights w_j are set to 0, but the bias value w_0 is initially set to the average output value of all training samples. The initial values for penalty factors λ_p are set to 1. The iteration is terminated when the gradient vector is sufficiently small, i.e. each element of the gradient vector is less than 10^{-6}.

The common experimental settings for extracting regression rules are as follows. In the k-means algorithm, initial representative vectors $\{\mathbf{r}^i\}$ are randomly selected as a subset of vectors $\{\mathbf{c}^\mu\}$. For each I, trials are repeated 100 times with different initial values, and the best result minimizing Eq. (14) is used. The cross-validation error of Eq. (20) is calculated by using the leave-one-out method, i.e. $S = N$. The candidate number I of representative vectors is incremented in turn from 1 until the cross-validation error increases. The c4.5 program is used with the initial settings.

4.2. Experiment using artificial data set

We consider the following set of regression rules:

$$\begin{cases} \text{if } q_{21} \wedge (q_{31} \vee q_{33}) & \text{then } y = 2 + 3x_1^{-1}x_2^3 + 4x_3x_4^{1/2}x_5^{-1/3} \\ \text{if } (q_{22} \vee q_{23}) \wedge (q_{32} \vee q_{34}) & \text{then } y = 2 + 5x_1^{-1}x_2^3 + 2x_3x_4^{1/2}x_5^{-1/3}, \\ \text{else} & \text{then } y = 2 + 4x_1^{-1}x_2^3 + 3x_3x_4^{1/2}x_5^{-1/3} \end{cases} \tag{22}$$

where we have three nominal and nine numeric input variables, and the numbers of categories of q_1, q_2 and q_3 are set as $M_1 = 2$, $M_2 = 3$ and $M_3 = 4$, respectively. Clearly, variables $q_1, x_6, \ldots, x_9$ are irrelevant to Eq. (22). Each sample is generated as follows: each value of nominal variables q_1, q_2, q_3 is randomly generated so that only one dummy variable becomes 1 for each nominal variable, each value of numeric variables $x_1, \ldots, x_9$ is randomly generated in the range of (0,1), and we get the corresponding value of y by calculating Eq. (22) and adding Gaussian noise with a mean of 0 and a standard deviation of 0.1 The number of samples is set to 400 ($N = 400$).

In this experiment, a neural network was trained by setting the number of hidden units to 2. Table 1 compares the performance of the experimental results obtained by applying the k-means algorithm with the different number of representative vectors I, where the root mean squared error (RMSE) was used for the evaluation; the training error was evaluated by using Eq. (17); the cross-validation error was calculated by using Eq. (20); and the generalization error was also evaluated by using a set of noise-free 10,000 test samples generated independently of the training samples. This table shows that the training error almost monotonically decreased; the cross-validation error was minimized when $I = 3$ (indicating that an adequate number of representative vectors is 3); and the generalization error was also minimized when $I = 3$. Since the cross-validation and generalization errors were minimized with the same number of representative vectors, we can see that the desirable model was selected by using the cross-validation for this data set. Here, the generalization error of the trained neural network was 0.315, showing the reasonable fidelity of rules.

On the other hand, the experimental result of $I = 1$ is nothing but using only the numeric variables by ignoring all the nominal variables; i.e. this corresponds to the result obtained by applying RF5 to only numeric data. Table 1 shows that these nominal variables played an important role because the generalization RMSEs decreased drastically from $I = 1$ to 3.

By applying the c4.5 program, we obtained the following decision tree whose leaf nodes correspond to the following:

$q_{21} = 0$:
| $q_{34} = 1 : 2(83.0) \Leftrightarrow (w_1 r_1^2, w_2 r_2^2) = (+5.04, +2.13)$

Table 1
Influence of number I for artificial data set

RMSE type	$I=1$	$I=2$	$I=3$	$I=4$
Training	2.090	0.828	0.142	0.142
Cross-validation	2.097	0.841	0.156	0.160
Generalization	2.814	1.437	0.320	0.322

$$
\begin{array}{l}
\mid\ q_{34}=0: \\
\mid\ \mid\ q_{32}=0: 3(129.0) \Leftrightarrow (w_1 r_1^3, w_2 r_2^3)=(+3.96,+2.97) \\
\mid\ \mid\ q_{32}=1: 2(53.0) \Leftrightarrow (w_1 r_1^2, w_2 r_2^2)=(+5.04,+2.13) \\
q_{21}=1: \\
\mid\ q_{34}=1: 3(36.0) \Leftrightarrow (w_1 r_1^3, w_2 r_2^3)=(+3.96,+2.97)
\end{array}
$$

law described by Langley et al. (1987). This data set contained nine samples, and four variables were used in our experiments, i.e. two nominal (battery and wire) and one numeric (conductance) input variables and one output (current) variable.

In this experiment, since the number of samples was small, the number of hidden units was set to 1. Table 2 shows the experimental results, which indicates that the adequate number of representatives was 3. In addition, since the cross-validation RMSEs drastically decreased from $I=1$ to 3, the nominal variables are considered to be indispensable in this rediscovery problem. By applying the c4.5 program, the following rule set was found:
where the weight values were rounded off to the third decimal

$$
\begin{cases}
\text{if Battery} = \text{“}A\text{”} & \text{then Current} = 0.006 + 1.047(\text{Conductance})^{0.964} \\
\text{if Battery} = \text{“}B\text{”} & \text{then Current} = 0.006 + 1.198(\text{Conductance})^{0.964}, \\
\text{if Battery} = \text{“}C\text{”} & \text{then Current} = 0.006 + 1.676(\text{Conductance})^{0.964}
\end{cases}
\tag{24}
$$

$$
\begin{array}{l}
\mid\ q_{34}=0: \\
\mid\ \mid\ q_{32}=0: 1(73.0) \Leftrightarrow (w_1 r_1^1, w_2 r_2^1)=(+3.10,+4.07) \\
\mid\ \mid\ q_{32}=1: 3(26.0) \Leftrightarrow (w_1 r_1^3, w_2 r_2^3)=(+3.96,+2.97).
\end{array}
$$

Here the polynomial coefficients were rounded off to the second decimal place, and each number of training samples arriving at the corresponding leaf node is shown in parenthesis. Then, we obtained the following rule set by performing some simplifications:
Therefore, although some of the weight values were

$$
\begin{cases}
\text{if } q_{21} \wedge (q_{31} \vee q_{33}) & \text{then } y = 2.01 + 3.10 x_1^{-1.00} x_2^{+3.01} + 4.07 x_3^{+1.02} x_4^{+0.51} x_5^{-0.33} \\
\text{if } (q_{22} \vee q_{23}) \wedge (q_{32} \vee q_{34}) & \text{then } y = 2.01 + 5.04 x_1^{-1.00} x_2^{+3.01} + 2.13 x_3^{+1.02} x_4^{+0.51} x_5^{-0.33} \\
\text{else} & \text{then } y = 2.01 + 3.96 x_1^{-1.00} x_2^{+3.01} + 2.97 x_3^{+1.02} x_4^{+0.51} x_5^{-0.33}.
\end{cases}
\tag{23}
$$

slightly different, we can see that RN2 successfully found a set of regression rules almost equivalent to the true one.

4.3. Experiment using scientific data set

To evaluate the rediscovery performance of RN2, we used a historical scientific data set concerning Ohm's

Table 2
Influence of number I for scientific data set

RMSE type	$I=1$	$I=2$	$I=3$	$I=4$
Training	0.99397	0.23223	0.03963	0.02801
Cross-validation	1.11826	0.27996	0.05964	0.05987

place. Clearly, the obtained rule set tells us that the nominal variable corresponding to the wire was irrelevant, and the battery was used in changing the coefficients of the conductance. Note that one form of Ohm's law for electrical circuits is described as Current = Voltage × Conductance, and the voltage is determined by a battery type. Therefore, although the exponent was slightly different from 1 and a very small constant term was included, we can see that RN2 successfully rediscovered the fundamental relation of Ohm's law.

Note that the BACON systems attempt to discover laws from data containing nominal variables by postulating intrinsic properties (Langley et al., 1987). In the above scientific problem, the voltage is regarded as the intrinsic property associated with the nominal variable Battery. The representative vectors calculated from our method can be used in defining the intrinsic values. Actually, in an experiment to revise an existing ecosystem model using Earth science data, our method was successfully applied for finding improved values for an intrinsic property associated with vegetation type, which plays a central role in the model (Saito et al., 2001).

4.4. Experiment using financial data set

We performed an experimental study to find underlying rules of market capitalization using balance sheet (BS)

Table 3
Influence of number I for financial data set

RMSE type	$I=1$	$I=2$	$I=3$	$I=4$
Training	562571.4	420067.5	388081.8	385114.0
Cross-validation	572776.7	440071.6	404998.2	406696.5

items (Saito et al., 2000). Our experiments used data of 953 companies listed on the first section of the Tokyo Stock Exchange (TSE), where banks, and insurance, securities and recently listed companies were excluded. The market capitalization of each company was calculated by multiplying the shares of outstanding stocks by the stock price at the end of October 1999. As the first step in this study, we selected six fundamental items from all of the BS items (x_1 : current assets; x_2 : property and equipment; x_3 : total assets; x_4 : current liabilities; x_5 : long-term debt; and x_6 : total liabilities), and the values of these items were also calculated at the end of October 1999. Additionally, since market capitalization rules can differ according to the type of industry, the 33 classifications of the TSE were used as a nominal variable. Note that the actual number of categories amounted to 30 because three categories were excluded as described above.

In order to intuitively understand the effect of the nominal variable, the number of hidden units was fixed at 1. Table 3 shows the experimental results, indicating the adequate number of representatives was 3. Since the cross-validation RMSEs clearly decreased from $I=1$ to 3, the nominal variable was effectively used. The final rule set obtained was

$$\text{if} \bigvee_{q_m \in Q^i} q_m \text{ then } y = 12891.6 + w_1 r^i x_2^{+0.668} x_3^{+1.043} x_6^{-0.747},$$
$$i = 1, 2, 3, \tag{25}$$

where each polynomial coefficient was calculated as

$$w_1 r^1 = +1.907, \qquad w_1 r^2 = +1.122, \qquad w_1 r^3 = +0.657. \tag{26}$$

Each group of the nominal conditions was as follows:

$Q^1 = \{$'Pharmaceuticals', 'Rubber Products', 'Metal Products', 'Machinery', 'Electrical Machinery', 'Transport Equipment', 'Precision Instruments', 'Other Products', 'Communications', 'Services'$\}$.
$Q^2 = \{$'Foods', 'Textiles', 'Pulp and Paper', 'Chemicals', 'Glass and Ceramics', 'Nonferrous Metals', 'Maritime Transport', 'Retail Trade'$\}$.
$Q^3 = \{$'Fisheries', 'Mining', 'Construction', 'Oil and Coal Products', 'Iron and Steel', 'Electricity and Gas', 'Land Transport', 'Air Transport', 'Wearhousing', 'Wholesale', 'Other Financing Business', 'Real Estate'$\}$.

Since the second term on the right-hand side of the polynomial equation appearing in Eq. (25) is always positive, each of the polynomial coefficients $w_1 r^i$ can indicate the stock price setting tendency of industry groups in similar BS situations, i.e. industries appearing in Q^1 are likely to have a high setting, while those in Q^3 are likely to have a low setting. More specifically, when considering individual rules for each industry, the 'Pharmaceuticals' industry had the highest setting due to the largest coefficient (2.348), while the 'Electricity and Gas' industry had the lowest setting due to the smallest coefficient (0.374). It seems that the resulting rules extracted by RN2 are intuitively reasonable.

4.5. Experiment using automobile data set

By using the automobile data set from the UCI repository of machine learning databases, we attempted to extract regression rules for predicting the prices from the car and truck specifications in 1985. The data set had 159 samples with no missing values ($N=159$), and consisted of 10 nominal and 14 numeric input variables, and one target variable (price). Note that we ignored one nominal variable (engine location) having the same value through all the samples.

In this experiment, since the number of samples was small, the number of hidden units was also set to 1. Table 4 shows the experimental results, indicating that the adequate number of representatives was 3. Even in this problem, since the cross-validation RMSEs clearly decreased from $I=1$ to 3, the nominal variables played an important role. The polynomial part of the extracted rules was as follows:

$$y = 1163.16 + w_1 r^i x_2^{+1.638} x_4^{+0.046} x_5^{-1.436} x_6^{-0.997} x_9^{-0.245} x_{13}^{-0.071}, \tag{27}$$

where each polynomial coefficient was calculated as

$$w_1 r^1 = +1.453, \qquad w_1 r^2 = +1.038, \qquad w_1 r^3 = +0.763. \tag{28}$$

The relatively simple nominal conditions were obtained by using the c4.5 rules program. Similarly as described for the experiments using the financial data set, since the second term on the right-hand side of Eq. (27) is always positive, the polynomial coefficient $w_1 r^i$ can indicate the car price setting tendency for similar specifications. Actually, the regression rules told us that cars of each price setting are:

High price setting: '5-cylinder ones', 'BMW's', 'convertibles', 'VOLVO turbos', 'SAAB turbos', '6-cylinder turbos'.
Middle price setting: 'PEUGOT's', 'VOLVO non-turbos', 'SAAB non-turbos', 'HONDA 1bbl-fuel-system ones', 'MAZDA fair-risk-level ones', 'non-BMW non-turbos and 6-cylinder ones', 'non-5-cylinder turbos and fair-risk-level ones'.
Low price setting: other cars.

Table 4
Influence of number I for automobile data set

RMSE type	$I = 1$	$I = 2$	$I = 3$	$I = 4$
Training	3725.07	1989.67	1522.44	1370.38
Cross-validation	3757.56	2132.08	1665.48	1774.09

It seems that the resulting nominal conditions found by RN2 are natural for our intuition.

5. Related work and future directions

To date a large amount of work on rule extraction from trained neural networks has been done as surveyed by Andrews et al. (1995), Tickle et al. (1998) and Mitra and Hayashi (2000). However, as noted earlier, previous work has emphasized classification tasks rather than regression models that predicts some continuous variable. Although some methods for fuzzy reasoning or fuzzy control address continuous output values, substantially different problems are posed. Since our framework has some unique characteristics, it seems difficult to perform direct comparative studies using these existing methods. Whereas our future research should address enabling to express the m-of-n concept (Towell & Shavlik, 1993) on the nominal condition, and incorporating fuzziness (Mitra & Hayashi, 2000) in the proposed framework.

A few years ago, we proposed a preliminary extraction method (Nakano & Saito, 1999); i.e. a set of nominally conditioned terms is extracted from each hidden unit, and then each of these terms is in turn combined through all of the hidden units. When α denotes the average number of terms extracted from each hidden unit, the total number of combinations approximately amounts to α^I. Thus, as the number of hidden units or the number of nominal variables increases, this preliminary method will suffer from combinatorial explosion. Moreover, the user is required to determine an appropriate tolerance parameter by trial and error, where this parameter is used to regard a set of similar values of weights as the same. In contrast, RN2 can run within the computational complexity of polynomial order.

Apart from using neural networks, regression rules can be directly found from given data. As a prominent approach, we can consider some 'divide-and-conquer' methods including CART (Breiman, Friedman, Olshen, & Stone, 1984), ABACUS (Falkenhainer & Michalski, 1990), and Cubist (Rulequest, 2001). Clearly, we need to perform head-to-head comparisons to evaluate the relative strength and weakness of our approach, but reporting such experiments will be beyond the scope of this paper. Here, it should be emphasized that our approach can use available domain knowledge by encoding it as a network structure and/or a set of some weight values, while the divide-and-conquer approach is suitable for constructing entirely new structures from scratch. Thus, in general, our approach is expected to produce better learning results by using such domain knowledge, but this claim must be carefully inspected by further studies.

Our method can be considered one approach toward the computational scientific discovery, which has a long history within artificial intelligence. Early systems for numeric law discovery like BACON (Langley, 1979; Langley et al., 1987) employed a heuristic search through a space of new terms and simple equations. Its numerous successors like FAHRENHEIT (Żytkow, Zhu, & Hussam, 1990) incorporate more sophisticated and more extensive search through a larger space of numeric equations. Some methods like ABACUS (Falkenhainer & Michalski, 1990) can deal with both nominal and numeric values simultaneously. However, these methods are based on a combinatorial approach, and therefore likely to suffer from combinatorial explosion in search. We believe that our approach has great potential to overcome this problem. More recently, our methods have been applied for discovering some Web dynamics (Saito & Langley, 2002), as well as revising an existing ecosystem model (Saito et al., 2001) as noted earlier.

In addition, our research is also related to the system identification tasks based on group method of data handling (GMDH; Ivakhnenko, 1971), but there are at least two non-trivial differences. One is that the GMDH approach investigates polynomials with integer powers, while our method can find those with real powers such as 0.5, 1.5, or 0.333. The other is that the GMDH employs a combinatorial approach, while our method adopts a numerical optimization of neural networks. To improve the performance of such combinatorial search, several methods combined with evolutionary computation have been proposed (Fukumi, Mitsukuwa, & Akamatsu, 2000; Kargupta & Smith, 1991; Sano, 1992), and it has been shown that some simple chaotic behaviors can be modeled. Thus, our future research should also address examining possibilities that RN2 can be applicable to identification of a chaotic system.

6. Conclusion

Given data containing both nominal and numeric variables, we proposed a new framework and method, called RN2, for extracting regression rules from trained neural networks. The RN2 method used a combination of vector quantizers and decision trees for rule extraction. Two experiments using artificial and scientific data sets showed that RN2 can successfully extract regression rules almost equivalent to the original ones, even if the data contains irrelevant variables and a small amount of noise. In addition, the other two experiments using financial and automobile data sets showed that RN2 produces interesting regression rules.

References

Andrews, A., Diederich, J., & Tickle, A. B (1995). *A survey and critique of techniques for extracting rules from trained artificial neural networks*. Technical Report, Queensland University of Technology.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford: Clarendon Press.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth International Group.

Durbin, R., & Rumelhart, D. E. (1989). Product units: A computationally powerful and biologically plausible extension. *Neural Computation*, *1*, 133–142.

Falkenhainer, B. C., & Michalski, R. S. (1990). *Integrating quantitative and qualitative discovery in the ABACUS system* (*Vol. III*). *Machine learning: An artificial intelligence approach*, Cambridge, MA: Morgan Kaufmann, pp. 153–190.

Fukumi, M., Mitsukuwa, Y., & Akamatsu, N. (2000). A new rule generation method from neural networks formed using a genetic algorithm with virus infection. *Proceedings of the International Joint Conference on Neural Networks* (Vol. III, pp. 413–418). Como, Italy.

Ishikawa, M. (2000). Rule extraction by successive regularization. *Neural Networks*, *13*, 1171–1183.

Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, *1*, 364–378.

Kargupta, H., & Smith R. (1991). System identification using evolving polynomial networks. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 370–376). San Diego.

Langley, P. (1979). Rediscovering physics with BACON.3. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 505–507). Tokyo, Japan.

Langley, P., Simon, H. A., Bradshaw, G. L., & Żytkow, J. M. (1987). *Scientific discovery: Computational explorations of the creative processes*. Cambridge, MA: MIT Press.

Lloyd, S. P. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, *28*, 129–137.

Luenberger, D. G. (1984). *Linear and nonlinear programming*. Reading, MA: Addison-Wesley.

Mitra, S., & Hayashi, Y. (2000). Neuro-fuzzy rule generation: Survey in soft computing framework. *IEEE Transactions on Neural Networks*, *11*, 748–768.

Nakano R., & Saito K. (1999). Discovery of a set of nominally conditioned polynomials. *Proceedings of the Second International Conference on Discovery Science* (pp. 287–298). Tokyo, Japan.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Cambridge, MA: Morgan Kaufmann.

Rulequest (2001). Rulequest research web page. http://www.rulequest.com.

Saito, K., & Langley, P. (2002). Discovering empirical laws of Web dynamics. *The 2002 International Symposium on Applications and the Internet* (pp. 168–175). Nara, Japan.

Saito, K., & Nakano, R. (1997a). Partial BFGS update and efficient step-length calculation for three-layer neural networks. *Neural Computation*, *9*, 239–257.

Saito, K., & Nakano, R. (1997b). Law discovery using neural networks. *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (pp. 1078–1083). Nagoya, Japan.

Saito, K., & Nakano, R. (2000a). Second-order learning algorithm with squared penalty term. *Neural Computation*, *12*, 709–729.

Saito, K., & Nakano, R. (2000b). Discovery of relevant weights by minimizing cross-validation error. *Proceedings of the Fourth Pacific-Asia Conference on Knowledge Discovery and Data Mining* (pp. 372–375). Kyoto, Japan.

Saito, K., Langley, P., Grenager, T., Potter, C. S., Torregrosa, A., & Klooster, S. A. (2001). Computational revision of quantitative scientific models. *Proceedings of the Fourth International Conference on Discovery Science* (pp. 336–349). Washington, DC.

Saito, K., Ueda, N., Katagiri, S., Fukai, Y., Fujimaru, H., & Fujinawa, M. (2000). Law discovery from financial data using neural networks. *Proceedings of the IEEE/IAFE/INFORMS Conference on Computational Intelligence for Financial Engineering* (pp. 209–212). New York.

Sano, C. (1992). Hybird of (ID3 extension + backpropagation) hybrid & (case-based reasoner + Grossbreg net) hybrid with economics modeling controlled by genetic algorithm. *Applications of Artificial Intelligence X: Knowledge-Based Systems*, *SPIE-1707*, 180–194. Orlando.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions (with discussion). *Journal of the Royal Statistical Society B*, *64*, 111–147.

Tickle, A. B., Andrews, R., Golea, M., & Diederich, J. (1998). The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, *9*, 1057–1068.

Towell, G. G., & Shavlik, J. W. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, *13*, 71–101.

Żytkow, J. M., Zhu, J., & Hussam, A. (1990). Automated discovery in a chemistry laboratory. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 889–894). Boston, MA: AAAI Press.