Concise Papers

Generalized Analytic Rule Extraction for Feedforward Neural Networks

Amit Gupta,
Sang Park, Member, IEEE, and
Siuwa M. Lam

Abstract—This paper suggests the "input-network-training-output-extraction-knowledge" framework to classify existing rule extraction algorithms for feedforward neural networks. Based on the suggested framework, we identify the major practices of existing algorithms as relying on the technique of generate and test, which leads to exponential complexity, relying on specialized network structure and training algorithms, which leads to limited applications and reliance on the interpretation of hidden nodes, which leads to proliferation of classification rules and their incomprehensibility. In order to generalize the applicability of rule extraction, we propose the rule extraction algorithm Generalized Analytic Rule Extraction (GLARE), and demonstrate its efficacy by comparing it with neural networks per se and the popular rule extraction program for decision trees, C4.5.

Index Terms—Classification, neural network, rule extraction.

1 INTRODUCTION

Many recent studies have confirmed the effectiveness of neural networks for a variety of applications including classification problems, computer vision, time series analysis, and natural language recognition [4], [5], [9], [17], [18]. Despite the advantages of neural networks such as prediction accuracy, robustness, no requirements on data distribution assumptions, and model-free estimation procedure, there are still difficulties in determining the architecture, parameter settings, and training algorithms. In this work, we identify the major practices of existing algorithms as relying on the technique of generate and test, which leads to exponential complexity, relying on specialized network structure and training algorithms, which leads to limited applications and reliance on the interpretation of hidden nodes, which leads to proliferation of classification rules and their incomprehensibility. In order to generalize the applicability of rule extraction, we propose the rule extraction algorithm Generalized Analytic Rule Extraction (GLARE), and demonstrate its efficacy by comparing it with neural networks per se and the popular rule extraction program for decision trees, C4.5.

2 A CLASSIFICATION FRAMEWORK FOR RULE EXTRACTION ALGORITHMS

Because of the significance of the "lack of explanation" problem in neural networks, we have seen more and more rule extraction algorithms in the literature. There exists the need to organize rule extraction algorithms into perspectives using some classification schemes. Andrews et al. [1] are among the first to propose such a scheme, which utilizes five factors to classify rule extraction algorithms including the expressive power of extracted rule, translucency of extraction technique, utilization of specialized training regimes, quality of extracted rules, and algorithmic complexity. However, the suggested scheme has overlapping and missing areas. For example, the expressive power of rules can be considered as a partial indicator of quality of rules, and there is no coverage on algorithms which require specialized network structure and domain knowledge. After examining different rule extraction algorithms, we propose the classification framework "input-network-training-output-extraction-knowledge." The remaining of this section will explain the framework and identify some major practices in existing rule extraction algorithms.

Fig. 1 summarizes the characteristics of some existing rule extraction algorithms in terms of the proposed classification framework. Network input, the first component of the framework, concerns about the input requirement to the network. Input attributes can be boolean, nominal, or continuous; and there can be the input requirement of demand knowledge. Network structure considers whether an algorithm requires specialized network architecture to facilitate the rule extraction process. Training algorithm considers whether a rule extraction algorithm requires specialized training methods. Network output considers whether the output from the trained network has to be preprocessed before submitting as input to the rule extraction process. Extraction process considers the methodology as well as the computational complexity of an algorithm. There are two main types of extraction methodologies: Generate and test, which is search based; and analytic, which is nonsearch based. The analytic approach extracts rules by directly interpreting the strengths of connection weights in addition to the explanation capability, extracted rules may also have the merit of predicting new cases more accurately than the neural network per se. The above conjecture is based on the principle of minimum description length for theory formulation. A "good" rule extraction algorithm should compress and transform the set of distributed connection weights from the neural network into a set of succinct, essential, and comprehensible rules for explanation and prediction purposes.

The next section of this paper presents an "input-network-training-output-extraction-knowledge" framework to classify existing rule extraction algorithms. The examination of existing rule extraction algorithms identifies several common practices, which motivates the development of a new rule extraction algorithm Generalized Analytic Rule Extraction (GLARE). Section 3 describes the specifications of GLARE using the classification framework presented in Section 2. An illustrative example for GLARE is given in Section 4. In order to demonstrate the efficacy of GLARE, experiments are carried out to compare the rule performance of GLARE with neural networks per se, as well as with C4.5, which is a popular rule extraction algorithm for decision trees. Section 5 explains the experimental methodology. Section 6 presents and discusses the experimental results. The last section concludes the paper by discussing some future research for this research area.

A. Gupta is with Anderson Consulting, 3773 Willow Road, Northbrook, IL 60062. E-mail: amit.gupta@ac.com.
B. S. Park is with the Department of Industrial Managament, KAIST, 373-1 Kuwon-Dong, Yusong-Gu, Taejon, Korea 305-701. E-mail: sangpark@kaist.ac.kr.
C. S. Lam is with the Department of Management Information Science, California State University-Sacramento, 6001 J Street, Sacramento, CA 95819-6088. E-mail: lam@csas.edu.

Manuscript received 12 June 1997; revised 4 Dec. 1998.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and refer to IEEECS Log Number 105247.
Fig. 1. The framework for classifying rule extraction algorithms.

3 SPECIFICATIONS OF GLARE

This section describes the proposed rule extraction algorithm GLARE. Fig. 2 summarizes the specifications of GLARE based on the "Input-Network-Training-Output-Extraction-Knowledge" framework: GLARE is designed for networks with only one hidden layer. The next section provides an illustrative example for the
In this section, we propose a new approach to knowledge extraction from neural networks, called Generalized Analytic Rule Extraction (GLARE). GLARE is designed to extract meaningful rules from trained neural networks, which can be used for interpretation and understanding of the underlying knowledge captured by the network.

### Network Structure
- Standard feedforward neural network.
- One hidden layer.
- No restriction on the number of hidden nodes.

### Training Algorithm
- Standard backpropagation algorithm.

### Network Output or Input to Extraction
- No special treatment to the set of connection weights from the trained network.

### Rule Extraction Process
For each output node \(m\) (each class in the data set), do the following:

1. Create \(RWIH_n\) which are rankings of all input nodes \(X_i\) (category \(j\) of input attribute \(i\)) based on the total number of input attributes \(X_i\) and the total number of category values \(C_j\) for all attributes. Note that different attributes may have different numbers of category values. \(H_n\) is hidden node \(n\) where \(n = 1, 2, \ldots, N\), and \(N\) is the total number of hidden nodes.

2. Create \(RRWIH_n\) which are reduced \(RWIH_n\) from step 1, based on the parameter \(NWIH = \alpha\).

3. Calculate \(II(H_n)\) which are importance indexes for hidden node \(n\) to output node \(m\).

4. Create \(RWHO_m\) which is the ranking of importance of hidden nodes to output node \(m\) in descending order based on \(II(H_n)\) from step 3.

5. Create \(ATTR\) which is an \(N \times 1\) matrix consisting of elements \(1, 1, \ldots\) based on \(ATTR\) from step 4.

6. Create \(RATTR\) which is an \(N \times N\) matrix consisting of elements \(1\) based on \(RWHO_m\) from step 5.

7. Create the classification rule for node \(m\), based on \(RATTR\) from step 6.

### Output Rules
- Form at: If attribute \(0\) is category \(0\) or 1 AND attribute \(1\) is category \(2\) or 3 AND ...
- Then class = 0.
- Symbolic, one composite rule for each class, and each rule as a conjunction of disjunctives.
- The length of the conjunction is the number of input attributes, and disjunctive \(i\) contains no more than the total categories in attribute \(i\).

---

**Fig. 2. Specifications of Generalized Analytic Rule Extraction (GLARE)**

Algorithm. Fig. 3 provides an example for a fully connected and feedback neural network trained by backpropagation. For clarity purposes, only part of the connection weights are shown. \(X_i\) represents category value \(j\) of input attribute \(i\). \(X_i\) equals to 1 (0) if attribute \(i\) has (does not have) category value \(j\). \(\Sigma_i\) is the total number of input attributes, and \(\Sigma_j\) is the total number of category values for all attributes. Note that different attributes may have different numbers of category values. \(H_n\) is hidden node \(n\) where \(n = 0, 1, \ldots, N\), and \(N\) is the total number of hidden nodes. \(C_m\) is output node \(m\) representing class \(m\) in the data set. For each output class \(m\), GLARE performs the following seven steps for extracting the composite rule:

**Step (1): Create \(RWHI_n\)**

For each hidden node \(n\) in the network, create ranking \(RWHI_n\). \(RWHI_n\) 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 \(RWHI_n\) to indicate whether the connection weight between the input node and hidden node is positive or negative. The output of step (1) is a set of rankings consisting of \(N\) different \(RWHI_n\).

**Step (2): Create \(RRWHI_n\)**

This step creates \(RRWHI_n\), reduced \(RWHI_n\), by reducing the length of \(RWHI_n\) from step (1) as determined by the parameter \(NWIH\). \(NWIH\) means the number of adopted weights between input and hidden layer. Set the parameter \(NWIH\) to \(\alpha\) where \(1 \leq \alpha \leq \Sigma_i\). Some heuristics for choosing \(NWIH\) will be discussed in Section 6. Then, the first \(\alpha\) input nodes in \(RWHI_n\) will be retained for further processing, and remaining input nodes in \(RWHI_n\) will be deleted. The output of this step is a set of \(N\) different \(RRWHI_n\) rankings.
The purpose of this step is to select several largest connection weights for rule extraction.

**Step (3): Calculate importance indexes for hidden nodes.**

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 \( H_n \), record the activation level of each resubmitted training case, calculate the average activation level, then calculate the importance index using the following equation:

\[
II(H_n) = \text{ABS}(\text{AAI}_n * \text{WHO}_n) \quad (1)
\]

where:

- \( II(H_n) \) is the importance index of hidden node \( n \) to output node \( m \),
- \( \text{ABS}(\cdot) \) indicates absolute value,
- \( \text{AAI}_n \) is the average activation level of hidden node \( n \) for training cases of class \( m \), and
- \( \text{WHO}_n \) 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 preserve the learning power of partially activated hidden nodes. The output of step (3) is numeric values \( II(H_n) \) for all hidden nodes indicating the influential power of each hidden node in determining the output for output node \( m \).

**Step (4): Create RWHO\(_m\).**

Create the ranking RWHO\(_m\). RWHO\(_m\) 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 RWHO\(_m\) to indicate whether the connection weight between the hidden node and output node is positive or negative. The output of step (4) is a ranking of hidden nodes based on their importance on determining the output for output node \( m \).

**Step (5): Create ATTR.**

RWHO\(_m\) (one ranking) from step (4) and RWHI\(_m\) (N rankings) from step (2) are used to construct an \( N \times a \) matrix ATTR. ATTR consists of RWHI\(_m\) reordered and adjusted based on RWHO\(_m\). First, we reorder RWHI\(_m\) from step (2) according to the order of hidden nodes in RWHO\(_m\). Second, for hidden nodes with negative signs in RWHO\(_m\) we flip the sign of all input nodes in the corresponding RWHI\(_m\). The rationale of flipping the signs of \( X_s \) is explained as follows. For a hidden node which has a negative connection weight to an output node, the output of that hidden node must be low for the output node to generate a high output. Then, in order to have a low output from that hidden node, those input nodes which have negative connection weights to the hidden node must have the input value \( 1 \) and those input nodes which have positive connection weights to the hidden node must have the input value \( 0 \). This reasoning demands that the signs of all input nodes be reversed for hidden nodes with negative connection weights to the output node. The output of step (5) is a matrix ATTR in which the important input nodes for class \( m \) occupy the top rows and left columns. An \( +X_i \) (\( -X_i \)) in ATTR indicates that in order for output node \( C_m \) to have a high output (so that a certain case will be classified as class \( m \)), input node \( X_i \) must have the input value \( 1 \) (0).
Step (6): Create RATTR.

Create an $|I| \times |Y|$ matrix RATTR based on the ATTR from step (5). Following the directions of top to down and left to right, we use elements in ATTR to determine element values in RATTR. RATTR is initialized to 0. Then, an $+X_{ij}$ ($-X_{ij}$) in ATTR will set category $j$ of attribute $i$ in RATTR to 1 ($-1$). Once an element in RATTR is set, it will not be reset. In other words, less important elements in ATTR have only residual power to determine element values in RATTR. The output of step (6) is a matrix RATTR that can be used to construct the classification rule for class $m$.

Step (7): Create the classification rule for class $m$.

Based on element values in RATTR, we construct a composite rule for class $m$. An element of $1$ ($-1$) in RATTR 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 treats 0 as 1. The format of the extracted rule is:

If attribute 0 = category 0 or 1 AND attribute 1 = category 2 or 3 AND ...

Then class = 0.

Each rule is a conjunctive of disjunctives. The length of the conjunctive is the number of input attributes, and disjunctive $i$ contains no more than the total categories in attribute $i$.

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 RATTR 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, we may not be necessary to exhaust all elements in ATTR. Notice that since element 0 in row 3 cannot reset that to 1, according to the residual power principle for elements in ATTR.

4 AN ILLUSTRATIVE EXAMPLE

The network in Fig. 3 has nine input nodes ($i = 0, 1, 2$, and $j = 0, 1, 2, 3$ for each $i$), four hidden nodes ($n = 0, 1, 2, 3$), and three output nodes ($m = 0, 1, 2$). To extract the composite rule for class 1, GLARE carries out the following steps:

Step (1): Create $RWIH_i$.

Note that the connection weights for $H_1$, $H_2$, and $H_3$ are not shown in Fig. 1, and are assumed to generate $RWH_i$ as follows:

$RWH_0: +X_{10} +X_{11} -X_{20} +X_{21} +X_{22} +X_{23} -X_{30} -X_{31} -X_{32} -X_{33}$

$RWH_1: +X_{10} +X_{11} +X_{12} -X_{20} -X_{21} -X_{22} -X_{23} +X_{30} +X_{31} +X_{32} +X_{33}$

$RWH_2: +X_{10} -X_{12} -X_{20} +X_{21} +X_{22} +X_{23} -X_{30} -X_{31} -X_{32} -X_{33}$

$RWH_3: -X_{12} -X_{20} +X_{21} +X_{22} +X_{23} +X_{30} -X_{31} -X_{32} -X_{33}$

Step (2): Create $RRWH_{i10}$.

Suppose we set $NW_{i10}$ to 2. The following are the $RRWH_{i10}$ for output node 0:

$RRWH_{i10} : +X_{20} +X_{21}$

Step (3): Calculate importance indexes for hidden nodes.

$I(H_{i10})$, $I(H_{i20})$, $I(H_{i30})$, and $I(H_{i40})$ are calculated using (1). Suppose the calculated values are $I(H_{i10}) = 10.5$, $I(H_{i20}) = 19.8$, $I(H_{i30}) = 9.8$, and $I(H_{i40}) = 4.5$.

Step (4): Create $RWH_{i20}$.

Based on the importance indexes from step (3), the following $RWH_{i20}$ is constructed:

$RWH_{i20} : +X_{11} -X_{12} -X_{22} -X_{32}$

Note that we attach the positive sign to $H_1$ and $H_2$, and the negative sign to $H_3$ and $H_4$, based on the signs of connection weights from these hidden nodes to $C_0$. The above $RWH_{i20}$ indicates that $H_4$ is most important for determining the output value of $C_0$, followed by $H_0$, then $H_2$, and $H_3$ is the least important.

Step (5): Create ATTR.

Using $RWH_{i20}$, we generate the following ATTR for class 0:

Using ATTR from step (5), the following RATTR is constructed:

$$
\begin{array}{ccc}
0 & 1 & 2 \\
0 & +X_{10} & +X_{11} \\
row 1 & -X_{20} & +X_{11} \\
row 2 & +X_{10} & -X_{22} \\
row 3 & +X_{22} & +X_{11} \\
\end{array}
$$

The procedure of filling the above RATTR as follows. We start with element 0 in row 0 from ATTR. Since element is $+X_{10}$, we set category 0 of attribute 0 in RATTR to 1. Then, we use element 1 in row 0 from ATTR, and since that element is $+X_{11}$, we set category 1 of attribute 0 to 1. The filling procedure will go on until we exhaust all elements in ATTR. Notice that since element 0 in row 1 from ATTR has set category 2 of attribute 2 to -1, element 0 in row 3 cannot reset to 1, according to the residual power principle for elements in ATTR.

Step (6): Create RATTR.

Using the ATTR from step (5), the following rule for class 0 is constructed:

If attribute 0 = 0 or 1 and attribute 1 = 1 or 2 and attribute 2 = 0 or 1

Then class = 0.
TABLE 1
Correct Classification Rates (in Percentages) for Training Sets and Test Sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Tree</th>
<th>Tree-Rule</th>
<th>Neural Network</th>
<th>GLARE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post-p</td>
<td>72.41</td>
<td>71.43</td>
<td>72.41</td>
<td>87.93</td>
</tr>
<tr>
<td>Balloon-p</td>
<td>64.29</td>
<td>66.67</td>
<td>85.71</td>
<td>78.57</td>
</tr>
<tr>
<td>Hepatitis-n</td>
<td>88.64</td>
<td>78.57</td>
<td>81.82</td>
<td>97.73</td>
</tr>
<tr>
<td>BUPA-c</td>
<td>32.00</td>
<td>58.33</td>
<td>32.00</td>
<td>98.00</td>
</tr>
<tr>
<td>Glass-c</td>
<td>96.00</td>
<td>75.00</td>
<td>96.00</td>
<td>98.00</td>
</tr>
<tr>
<td>Iris-c</td>
<td>97.06</td>
<td>97.92</td>
<td>97.06</td>
<td>100.00</td>
</tr>
<tr>
<td>BUPA-p</td>
<td>70.00</td>
<td>75.00</td>
<td>60.00</td>
<td>50.00</td>
</tr>
<tr>
<td>Glass-p</td>
<td>80.00</td>
<td>83.33</td>
<td>90.00</td>
<td>92.00</td>
</tr>
<tr>
<td>Iris-p</td>
<td>61.77</td>
<td>60.42</td>
<td>94.12</td>
<td>97.06</td>
</tr>
<tr>
<td>BUPA-q</td>
<td>50.00</td>
<td>62.50</td>
<td>72.00</td>
<td>74.00</td>
</tr>
<tr>
<td>Glass-q</td>
<td>78.00</td>
<td>62.50</td>
<td>86.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Iris-q</td>
<td>60.78</td>
<td>62.50</td>
<td>62.75</td>
<td>98.04</td>
</tr>
</tbody>
</table>

* n means data sets with nominal attributes.
* p means data sets with continuous attributes.
* c means converting continuous attributes into nominal attributes using c scaling.
* q means converting continuous attributes into nominal attributes using q scaling.

5 EXPERIMENTAL METHODOLOGY

The experiment adopted six data sets from the machine learning databases [10] at the University of California at Irvine (UCI). The Postoperative Patient (Post), Balloon (only the last data set in the repository), and Hepatitis data sets have only nominal attributes. The BUPA, Glass, and Iris data sets have only continuous attributes. To avoid missing values, we deleted attributes with missing values. For Hepatitis, only attributes 4, 5, 6, 7, 8, 11, 12, 13, 14, 15, and 20 were used. For Glass, only attributes 2, 3, 4, 5, 6, 7, and 8 were used. In order to avoid uneven class distribution, we used only classes 1 and 2 in the Glass data set. Then, we randomly selected 74 cases from Hepatitis as well as from Glass for the experiment. The number of training to test cases is 7:3 in each data set. To apply GLARE, continuous attributes in the BUPA, Glass, and Iris data sets have to be centered and converted into nominal attributes. We developed two scaling methods, p scaling and q scaling, for the conversion. The method of p scaling is to assign attribute values into five intervals of equal length along the line between the minimum and maximum values for each attribute. The method of q scaling is to assign the first greatest 20 percent attribute values into category 0, the next greatest 20 percent into category 1, and so on. After the conversion, there are 12 different data sets including Post-n, Balloon-n, Hepatitis-n, BUPA-c, Glass-c, Iris-c, BUPA-p, Glass-p, Iris-p, BUPA-q, Glass-q, and Iris-q where n indicates nominal attributes, c indicates continuous attributes, p indicates nominal attributes from p scaling, and q indicates nominal attributes from q scaling. The following experimental procedure was applied to each of the n, p, and q data sets:

1. Use the C4.5 program [13] to build 10 decision trees from the training set. Decision trees are used to predict the test set. Record the best test set CCR (correct classification rate as the percentage of correctly classified cases).

2. Apply the rule extraction procedure from C4.5 to decision trees from step (1). C4.5 chooses the best tree from step (1) based on predicted error rates. Classification rules are generated from the chosen tree. Rules are applied to the training and test set. Record the test set CCR.

3. Convert nominal attributes into binary input attributes for backpropagation training. Apply backpropagation [12] to the training set, and predict test set. Repeat the training 10 times with a new random set of initial weights for each trial. Record the best test set CCR.

4. Apply GLARE to trained networks from step (3). Extracted rules are used to predict the training and test set. Record the best test set CCR.

The above experimental procedure was also applied to data sets with continuous attributes except that step (4) was not carried out since GLARE cannot be applied to continuous attributes. On the other hand, step (2) was performed on data sets with continuous attributes since C4.5 can perform threshold testing for continuous attributes, which will test each midpoint between two adjacent continuous attribute values to select the best threshold for grouping continuous attributes into nominal attributes. The number of hidden nodes is set at about 50 percent to 75 percent of the number of input nodes. All backpropagation training were executed in C, and have 1,000 epochs, 0.5 learning rate, and 0 momentum rate.

6 EXPERIMENTAL RESULTS AND DISCUSSION

Table 1 records all training and test set CCRs for experimental steps (1) to (4) described in Section 5. The discussion focuses on test set CCRs. The test set results on nominal data sets (Post, Balloon, Hepatitis) show that GLARE achieves the same or higher CCRs than the other methods. In Post data set, GLARE achieves a CCR of 71.43 percent, which is the same as Tree and Tree-Rule.
is higher than Neural Network per se. In Ballon and Hepatitis data sets, GLARE achieves 100 percent and 85.71 percent respectively, which are higher than all other methods. Table 2 presents the best performance on continuous data sets (BUPA, Glass, and Iris). In BUPA, the best performer (75 percent) is Tree (p) and Neural Network (c). GLARE (q) achieves the medium result 66.67 percent. Tree-Rule has the lowest result 58.33 percent. In Glass, the best performer is Tree (p). GLARE (p) and Tree-Rule (p) achieve the same medium result of 83.88 percent. Neural network (c) has the lowest of 79.17 percent. In Iris, Tree (c), Tree-Rule (c), and Neural network (q) achieve the same result of 97.92 percent, and GLARE is 4.17 percent lower than the majority. Among the three continuous data sets, GLARE cannot achieve the best results, and tree is always among the best performers. This phenomenon may be due to the loss of information from converting continuous attributes into nominal attributes for GLARE. Overall, the experimental results show that GLARE outperforms other methods in nominal data sets, but not in continuous data sets. As for the parameters NMATTR and NWIH in GLARE, our experience suggests their values be set at 50 percent to 80 percent of the maxima for the two values.

7 Conclusion

This paper proposes an algorithm GLARE to extract classification rules from feedforward and fully connected neural networks trained by backpropagation. The major characteristics of GLARE include its analytic approach for rule extraction, being applicable to standard network structure and training method, and extracted rules as direct mapping between input and output nodes. Experimental results have shown GLARE's efficacy for prediction comparing with neural networks per se and C4.5 in nominal data sets. The "Input-Network-Training-Output-Extraction-Knowledge" framework also reveals that numeric analysis for extraction process and numeric function as extracted knowledge are underexplored. Recently, we have seen some efforts [11, [14] in these directions. As neural networks are inherently numeric, more research should be directed at designing algorithms for numeric analysis and numeric representation for knowledge in neural networks.

References