Designing a Decompositional Rule Extraction Algorithm for Neural Networks

Jen-Cheng Chen[1], Jia-Sheng Heh[2], and Maiga Chang[3]

[1] Department of Electronic Engineering,
Chung Yuan Christian University, Chung Li, Taiwan, R.O.C.
32023 Chung-Li, Taiwan
clement@mcsl.ice.cycu.edu.tw
[2] Department of Information and Computer Engineering,
Chung Yuan Christian University, Chung Li, Taiwan, R.O.C.
32023 Chung-Li, Taiwan
jsheh@ice.cycu.edu.tw
[3] National Science and Technology Program for e-Learning, P.O. Box 9-187,
32099 Chung-Li, Taiwan
maiga@ms2.hinet.net
http://maiga.dnsalias.org/maiga/resume_e.htm

Abstract. The neural networks are successfully applied to many applications in different domains. However, due to the results made by the neural networks are difficult to explain the decision process of neural networks is supposed as a black box. The explanation of reasoning is important to some applications such like credit approval application and medical diagnosing software. Therefore, the rule extraction algorithm is becoming more and more important in explaining the extracted rules from the neural networks. In this paper, a decompositional algorithm is analyzed and designed to extract rules from neural networks. The algorithm is simple but efficient; can reduce the extracted rules but improve the efficiency of the algorithm at the same time. Moreover, the algorithm is compared to the other two algorithms, M-of-N and Garcez, by solving the MONK's problem.

1 Introduction

Although both expert systems and neural networks are typical systems in the domain of artificial intelligence, the basic components of these two kinds of systems are different. The knowledge base of expert systems is a set of rules which are stored in symbolic form, while neural networks encode learned knowledge within an established structure with adjustable weights in numerical form. Hence, it is difficult to transfer the training results of a neural network to the knowledge base of an expert system. The solutions represented by the knowledge base of an expert system could be explained and understood by users [10]. The explainable feature is the main advantage of expert system.

In contrast, neural networks have excellent abilities for classifying data and learning from inputs [7], but it is difficult to describe the decision process of a neural network or to merge more than one trained neural network [1][11]. The key for neural networks to overcome this deficiency is *rule extraction*. Rule extraction is the process of discovering the knowledge which is encoded within a neural network and representing these knowledge pieces in symbolic form, just as a rule base in an expert system. Once the rules which could be used to represent the neural network are extracted, the expert system then can use those rules as its knowledge base and gain the advantages of both Neural Networks and Expert Systems [15].

Although the neural-network-based systems performed well, it still can not make users feel comfortable due to the 'black box' effects of the neural networks. Therefore, some rule extraction methods for neural networks had been presented and classified into two categories, the pedagogical and the decompositional algorithms [18]. The pedagogical rule extraction algorithm aims to extract rules that map inputs directly into outputs [2]. Some typical algorithms of this category are "VIA"[17], "rule-extraction-as-learning"[6] and "RULENEG"[12]. Most of pedagogical algorithms are not effective when the size of the neural network increases, such as any real world problem. In order to improve the efficiency, decompositional algorithms focus on heuristically searching and extracting rules in neurons of neural networks individually. The existing decompositional algorithms include "KT"[8], "Subset"[19], "MofN"[20], "RULEX"[3], "Setiono"[14], and “Garcez”[9].

Some decompositional algorithms use weight pruning for more effectively processing [19][13][14], however, this kind of algorithms does not guarantee that the pruned network would be equivalent to the original one [9]. Therefore, the accuracy of extracted rules might be decreased. Most of decompositional algorithm are requested additional times to retrain the pruned network with original training data. This paper tries to develop a decompositional algorithm which could extract rules directly from trained neural network without retraining and keep high accuracy and low complexity at mean time. Such an algorithm is designed by using the concept of bound decomposition of neural network’s weights. The algorithm, Bound Decomposition Tree algorithm, offers an efficient method to extract rule from a neuron.

This paper analyzes the hyperspaces and hypercubes for the hyperplane of a neuron to develop an algorithm in order to extract positive, negative and uncertain Boolean rules from the neural network. After applying the Boolean logic these extracted rules will be simplified and be stored in the knowledge base. The users of the neural network-based systems then can understand the reasoning principle of the neural networks according to these human readable rules.

2 Rule-Extraction Mechanism

A typical *neuron* used in this paper is defined as

$$y = f(\mathbf{w}^T \mathbf{x} + w_0) = f(\textstyle\sum_{j \in N} w_j x_j + w_0) \tag{1}$$

in which N is the number of neuron inputs, $\boldsymbol{w} = [w_1, w_2, \ldots, w_n]^T$ is the *weight* vector of connection weights, w_0 is the *bias* and $x = [x_1, x_2, \ldots, x_n]^T$ is the inputs of the neuron. The *threshold function* f: $R \to [0,1]$ is monotonically increasing and may be linear, piecewise linear, hard-limited and sigmoid, as defined in a conventional neural network textbook. The weight vector and its threshold can be collected as a set of *augmented* *weights* $\mathbf{w}^* = [\mathbf{w}, w_0]^{\mathbf{T}}$, corresponding to the *augmented input* $\boldsymbol{x}^* = [\boldsymbol{x}, 1]^{\mathbf{T}}$. When the neuron inputs are binary, $\boldsymbol{x} = [x_j] \in \Xi = \{0, 1\}^N$, it is called *binary neuron*.

In essence, the outputs of a binary neuron are *linearly separable* and can divide the N-dimensional space (R^N) into two halfspaces as Fig.1 shows. Therefore, the input states can also be classified into positive state set $\Xi_{p,\mathbf{w}^*}$ and negative state set $\Xi_{n,\mathbf{w}^*}$. Take the input states in Fig. 1 as example :

$$\Xi_{p,\mathbf{w}^*} = \{110,111,100\} = \{11\text{-},1\text{-}0\} \tag{2}$$

and

$$\Xi_{n,\mathbf{w}^*} = \{010,000,011,001,101\} = \{0\text{--},\text{-}10\}, \tag{3}$$

in which '-' is called *don't care state*, meaning this state can be either '0' or '1' from the viewpoint of boolean logic. For example, 1-0 corresponds to x_1x_3' and {11-, 1-0} corresponds to $x_1x_2+x_1x_3'$.

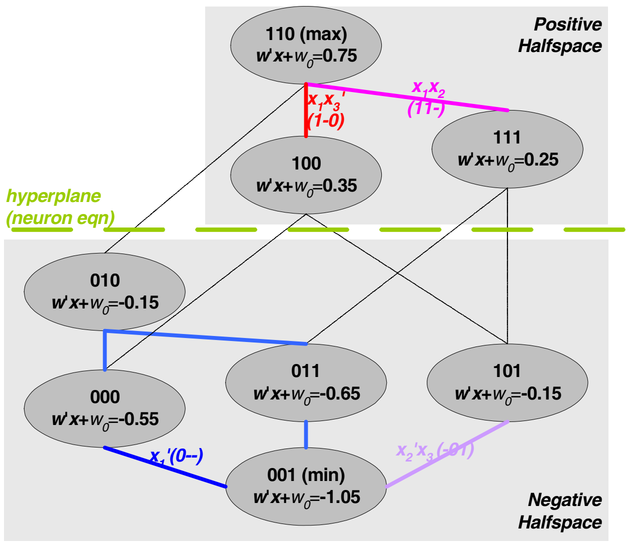

Fig. 1. The outputs of a binary neuron are linearly separable. According to the output, different input states can be divided to two halfspaces.

3 Bound Decomposition Tree Algorithm

For the example in Fig. 1, the initial bounds between the negative halfspace and the positive halfspace are -1.05 (001) and 0.75 (110). The initial bounds can be indicated as *bounds*(Ψ = '---') = [-1.05, 0.75] and *boundvar*(Ψ = '---') = 1.8. Moreover, a hypercube involves several input states and those states might come from different hyperspaces. Therefore, a hypercube could be divided into *positivecube*, *negativecube*, or *uncertain hypercube*. With these definitions, the rule extraction algorithm (extract boolean rules for a single neuron) is designed as following:

1. Given a set of weights and its threshold $\boldsymbol{w}$*=[$\boldsymbol{w}$,w_0] in Eq.(1) for a single neuron, and a rule margin Δ.
2. Sort these weights $\boldsymbol{w}$ = [w_j] by their absolute values to obtain an index sequence *IS* = {$j = arg\ max_{j\in N}\ |w_j|$}.
3. Set layer number $N_L = 0$, $\Psi = \Xi_{w*}$, *bound*(Ψ), and *boundvar*(Ψ).
4. Let $N_L = N_L+1$, *j* = the first index *j* in *IS*, and *IS* = *IS* \ {*j*}.
5. decompose each uncertain hypercube, Ψ = *positivecube*(·) ∪*negativecube*(·), in the deepest layer into positive hypercube and negative hypercube.
6. For each decomposed hypercubes, calculate the bounds [*lbound*(·), *ubound*(·)].
7. *boundvar*(·) = *boundvar*(Ψ) – $|w_j|$.
8. If all new hypercubes are positive or negative, or the sum of the rest weights less than Δ, then stop; otherwise go to step 4.
9. To transform all the calculated hypercubes into the positive rule $R_{p,w*}$, the negative rule $R_{n,w*}$, and the uncertain rule $R_{u,w*}$.

4 Experiments

In this section, a well-know problem, the MONK’s problem is taken into our system in order to verify the performance of the BDT algorithm [16]. The MONK’s problem has been used as a benchmark for machine learning systems and many rule extraction algorithms widely. Moreover, the comparisons between BDT, MofN, and Garcez’s algorithm are also made, All the three algorithms are implemented by both Matlab and Java, and the test results come from 100 execution cycles of MONK's Problem. The experiment environment is Intel Celeron 1.7GHz CPU and Microsoft Windows XP.

The experimental data of MONK’s problem comes from robots’ six attributes: head_shape ∈ {round, square, octagon}; body_shape ∈ {round, square, octagon}; is_smiling ∈ {yes, no}; holding ∈ {sword, balloon, flag}; jacket_color ∈ {red, yellow, green, blue}; and, has_tie ∈ {yes, no}. The neural networks used to verify the BDT algorithm and make comparisons in this paper for solving the MONK's problem are Thrun's trained BPNs [17]. The accuracy of the neural networks for training data is 100%. With the proposed algorithm in this paper, the corresponding rules are extracted with the 99.5% accuracy rate.

After getting the extracted rules by the BDT algorithm, the comparisons between BDT, MofN, and Garcez’s algorithm are made and listed in Table 1. Table 1 contains four parts, the first part is the algorithm name and the rule format; the second part is how many rules and related accuracy rates that are generated by the three different algorithms; the third part is the complexity of the extracted rules; and, the fourth part is how much time that the three algorithms needed to extract rules from the neural networks.

Before the experiment results of each part in Table 1 are discussed, there are several issues for the four parts from Table 1 are needed to clarify first. The first issue is the rule format, since the rule formats between the three algorithms are different, the extracted rule numbers will be difficult to compare. Therefore, in both second part and third part, the rules in M-of-N form are reconstructed to the rules in Boolean logic

Table 1. The experiment results for solving MONK's problem by using BDT, MofN, and Garcez's algorithms (*: The number in parenthesis is the rule number of Boolean logic format)

	BDT	MofN (kmean=7)	MofN (kmean=3)	MofN (kmean=2)	Garcez
Rule Format	Boolean Logic	M-of-N	M-of-N	M-of-N	Boolean Logic
Rule Number					
h_1	286	11(415)*	4(2555)*	3 (2865)*	45369
h_2	1969	45(2398)*	9(3475)*	3 (14960)*	50276
h_3	531	12(327)*	5(6286)*	2 (9250)*	54168
o_1	3	1(3)*	1(3)*	1(3)*	3
The Complexity of the Extracted Rules (Antecedent Rule Number)					
h_1	1909	36(2980)*	8(22820)*	5(21140)*	417131
h_2	15692	166(18621)*	20(27543)*	5(120250)*	454960
h_3	3685	35(2338)*	10(46922)*	3(73950)*	477476
o_1	6	1(6)*	1(6)*	1(6)*	6
Accuracy	**99.5%**	**99.5%**	**98.6%**	**98.1%**	**99.5%**
Execution Time (Running 1000 times) unit : second					
Matlab	71.0	76.7	25.16	16.6	35094
Java	26.1	3.14	1.26	0.59	969.5

form. The reconstructed Boolean logic rule number is listed inside the parenthesis just followed the M-of-N form rule number. The second issue is the accuracy rate, the accuracy rates come out from using the extracted rules to test the test data set of MONK's problem. The third issue is the complexity of the extracted rules, for example, a Boolean logic rule - $x_1x_2+x_3x_4'$. In this example, the extracted rule number is 2 (rule x_1x_2 and rule x_3x_4'), however, there are four antecedent rules (x_1, x_2, x_3, and x_4'). The last issue is the execution time, since the commercial programming languages have the benefits in reducing running time and commercialized, two different programming languages, Java and Matlab, are used to implement the three algorithms..

In Table 1, h_1-h_3 are 3 hidden nodes in the hidden layer and o_1 is the output neuron. "Rule Number" is the rule number extracted from a neuron and "The Complexity of the Extracted Rules" is the number of antecedents of all rules. Obviously, a smaller "Rule Number" represents fewer rules and a smaller "The Complexity of the Extracted Rules" represents more concise rules. Table 1 represents that the BDT algorithm makes "Rule Number" and "The Complexity of the Extracted Rules" greatly reduced and keep the accuracy rate high at meanwhile. As we can also find out that the execution time of BDT is less than the MofN with k-mean equals to 7. It is acceptable that the less time is needed for the MofN algorithm with k-mean equals to 3 and 2, because the value of k-mean is less the accuracy rate is lower. The only question mark is occurred in the execution time of Java application. As the last row in Table 1 shows that the Java version BDT algorithm takes too much time than the Java version MofN algorithm.

5 Future Works

Based on the analysis of the hyperplane, this paper proposed an algorithm to extract the positive and negative, even uncertain rules of a neuron at the same time. After the rules are extracted from a well-trained neural network, the users then could get more clear understandings about the neural network-based systems. Moreover, the extracted rules could be used as the knowledge base of expert systems [4][5]. From Table 1, the importance of the BDT algorithm is verified and competed with other two famous rule extraction algorithms, MofN and Garcez's algorithms. The BDT algorithm is getting better performance no matter in the extracted number, the accuracy rate, the rule complexity, and the execution time. Although the BDT algorithm could extract rules from a neural network with lower complexity, the system still needs to be enhanced. For example, the extracted rules are currently in Boolean logic format, in the near future, the BDT algorithm should be improved to deal with not only the binary inputs but also multi-level inputs and even the continuous inputs..

References

1. Andrews, R., Diederich, J., Tickle, A.B.: Survey and Critique of Techniques for Extracting rules from Trained Artificial Neural Networks. Knowledge Base Systems 8 (6) (1995) 373-389
2. Andrews, R., Geva, S.: Inserting and Extracting Knowledge from Constrained Error Back Propagation Networks. In Proc. of 6th Australian Conference on Neural Networks, Sydney, NSW (1995)
3. Andrews, R., Geva, S.: Rule Extraction from Local Cluster Neural Nets. Neurocomputing 47 (2002) 1-20
4. Chang, M., Chen, J.C., Chang, J.W., Heh, J.S.: Advanced Process Control Expert System of CVD Membrane Thickness Based on Neural Network. Materials Science Forum, 505-507 (2006) 313-318
5. Chen, J.C., Liu, T.S., Weng, C.S., Heh, J.S.: An Expert System of Coronary Artery Disease in Chinese and Western Medicine. In Proc. of 6th Asian-Pacific Conference on Medical and Biological Engineering, Tsukuba, Japan, Apr. 24-27, 2005 (2005) PA-3-06
6. Craven, M.W., Shavlik, J.W.: Using Sampling and Queries to Extract Rules from Trained Neural Networks. In Proc. of 11th International Conference on Machine Learning, New Brunswick, NJ (1994) 37-45
7. Freeman, J.A., Skapura, D.M.: Neural Networks. Addison-Wesley, New York (1992)
8. Fu, L.M.: Rule Generation from Neural Networks. IEEE Transactions on Systems, Man, and Cybernetics 28 (8) (1994) 1114-1124
9. Garcez, A.S. d'Avila, Broda, K., Gabbay, D.M.: Symbolic Knowledge Extraction from Trained Neural Networks: A Sound Approach. Artificial Intelligence 125 (2001) 155-207
10. Jackson, P.: Introduction to Expert Systems. Addison-Wesley, New York (1999)
11. Krishnan, R., Sivakumar, G., Bhattacharya, P.: A Search Technique for Rule Extraction from Trained Neural Networks. Pattern Recognition Letters 20 (1999) 273-280
12. Pop, E., Hayward, R., Diedrich, A.: RULENEG: Extracting Rules from a Trained Neural Network Using Stepwise Negation. Technical Report, Neurocomputing Research Centre, Queensland University of Technology (1994)
13. Setiono, R.: A Penalty Function Approach for Pruning Feedforward Neural Networks. Neural Computation 9 (1) (1997) 185-204
14. Setiono, R.: Extracting Rules from Neural Networks by Pruning and Hidden-unit Splitting, Neural Computation 9 (1) (1997) 205-225
15. Taha, I.A., Ghosh, J.: Symbolic Interpretation of Artificial Neural Networks. IEEE transactions on Knowledge and Data Engineering 11 (3) (1999) 448-463

16. Thrun, S.B. (ed.): The MONK's Problems: A Performance Comparison of Different Learning Algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University (1991).
17. Thrun, S.B.: Extracting Provably Correct Rules from Artificial Neural Networks. Technical Report IAI-TR-93-5, Institut für Informatik III, University of Bonn, Germany (1994).
18. Tickle, A.B., Andrews, R., Golea, M., Diederich, J.: The Truth Will Come to Light: Directions and Challenges in Extracting the Knowledge Embedded within Trained Artificial Neural Networks. IEEE Trans. Neural Networks 9 (6) (1998) 1057-1068
19. Towell, G.G., Shavlik, J.W.: The Extraction of Refined Rules from Knowledge-Based Neural Networks. Machine Learning 13 (1993) 71-101
20. Towell, G.G., Shavlik, J.W.: Knowledge-Based Artificial Neural Networks. Artificial Intelligence 70 (1994) 119-165