Interpretation of Trained Neural Networks by Rule Extraction

Vasile Palade[1], Daniel-Ciprian Neagu[2], and Ron J. Patton[1]

[1] The University of Hull
Control and Intelligent Systems Engineering
Cottingham Road, HU6 7RX, Hull, United Kingdom
{V.Palade,R.J.Patton}@eng.hull.ac.uk
[2] University "Dunarea de Jos" of Galati
Department of Computer Science and Engineering
Domneasca Str. 111, Galati 6200, Romania
Dan.Neagu@ugal.ro

Abstract. The paper focuses on the problem of rule extraction from neural networks, with the aim of transforming the knowledge captured in a trained neural network into a familiar form for human user. The ultimate purpose for us is to develop human friendly shells for neural network based systems. In the first part of the paper it is presented an approach on extracting traditional crisp rules out of the neural networks, while the last part of the paper presents how to transform the neural network into a set of fuzzy rules using an interactive fuzzy operator. The rules are extracted from ordinary neural networks, which have not a structure that facilitate the rule extraction. The neural network trained with the well known Iris data set was considered as benchmark problem.

1 Introduction

Artificial neural networks represent an excellent tool that have been used to develop a wide range of real-world applications, especially in case when traditional solving methods fail. They exhibit advantages such as ideal learning ability from data, classification capabilities and generalization for situations not contained in training data set, computationally fastness once trained due to parallel processing, noise tolerance. There were these advantages that made neural networks to be successfully applied to various real-word problems, including: speech recognition, medical diagnosis, image computing, process control and modeling [11], [8], fault diagnosis (a recent survey on using different neural network based techniques in fault diagnosis can be found in [9]). The major shortcoming of neural networks is represented by their low degree of human comprehensibility [13]. Many authors have focused on solving this shortcoming of neural networks, by compiling the knowledge captured in the topology and weight matrix of a neural network, into a symbolic form; some of them into sets of ordinary if - then rules [7], [13], [14], [15], others into formulas from propositional logic or from non-monotonic logics [10], or into sets of fuzzy rules [1], [2], [3], [5], [6]. More transparency is offered by fuzzy neural networks [4], [5], [12], which represent a paradigm that combines the comprehensibility and capabilities of fuzzy reasoning to handle uncertainty and the capabilities of neural networks to learn from examples.

The paper has the following organization. Section 2 briefly presents an inverting of a neural network, more exactly given the output of the network, how to calculate the input of the network which produces the given output. This input calculation for a given network output is required in the rule extraction method presented in section 3. The inversion method presented in section 2 is a computationally fast inversion method, that optimize the searching for the network inputs which produce the desired network output. Section 3 proposes a method of traditional if-then rule extraction from neural networks, which have not a special structure that facilitates the rule extraction. Many other rule extraction methods reported in the literature rely on and need some special architectures for neural networks in order to be efficiently applied. Our method is based on interval propagation across the network, in a similar way as VIA method proceeds [13]. The rule extraction method is applied on the neural network trained with Iris data set. The main problem encountered when apply this method is the big number of rules required to satisfactory describe the network functioning. That’s why we tried to express the network behavior in a more concise way. Section 4 uses the same neural network as in section 3, and a fuzzy rule set is extracted, by introducing an interactive fuzzy operator [1], [5]. Conclusions of the paper are summarized in the last section.

2 Iterative Relation for Neural Network Inversion

An inversion method, which calculates the input of a neural network for a given output, was presented in [11]. An improved and a more computationally efficient version of this method was presented in detail in our previous papers [7], [8].

Given a three layered neural network, with q the number of the network inputs, h the number of hidden nodes, and r the number of network outputs. Within the q inputs of the network, f inputs are considered to be fixed inputs (past inputs and outputs of the process in case when neural network is used for process modeling), and we have to determine, by a searching procedure, the remaining p inputs (p+f=q). Given the output vector y, we have to find the input vector u which produces the output y. By u it is denoted the vector of the p unfixed inputs of the network, and by u_f the vector of the f fixed inputs of the network.

We have the following relations:

$$\begin{gathered} y' = f^{-1}(y) - \theta^{y} \\ W^{yx}\, x = y' \end{gathered} \tag{1}$$

where f is the nonlinear activation function of the network nodes, x the output vector of the hidden nodes, W^{yx} is the weight matrix between hidden layer and output layer, θ^{y} is the bias vector of the output layer. If W^{yx} is a squared matrix, then it is possible to calculate the input u as follows:

$$\begin{gathered} x = (W^{yx})^{-1} y' \\ x' = f^{-1}(x) - \theta^{x} \end{gathered}$$

$$W^{xu} u = x' - W_f^{xu} u_f \tag{2}$$

where W^{xu} is the weight matrix between unfixed inputs and hidden layer, W_f^{xu} is the weight matrix between fixed inputs and hidden layer, θ^x is the bias vector of the hidden layer. When the hidden and the output layers have exactly the same number of nodes (h=r), the inversion of the network consists of solving two linear equation systems ((1) and (2)). In the following, consider d the vector $d = W_f^{xu} u_f$.

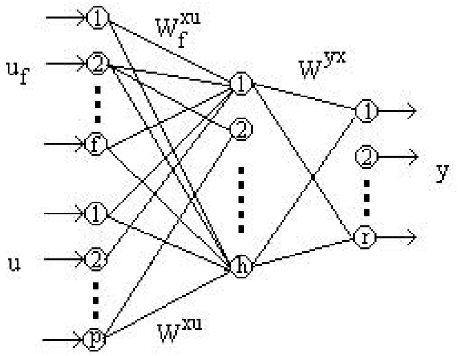

Fig. 1. The three-layered neural network

The general case of most neural models of real-world processes is represented by neural networks with h > r. In this case, it is possible to write the matrix W^{yx} and the linear system (1), by Jordan-Gauss elimination, in the following form:

$$\begin{array}{c} x_1 \; x_2 \quad x_r \; x_{r+1} \, x_h \;\; y' \\ \left[\begin{array}{c|c|c} 1 \; 0 \,.........0 & & \\ 0 \; 1 \,.........0 & & \\ .... & C & Y^* \\ .... & & \\ 0 \; 0 \,.........1 & & \end{array}\right] \end{array} \tag{3}$$

Partitioning the weight matrix W^{xu} and the vectors x, θ^x and d, as given below:

$$x = \begin{bmatrix} x' \\ x'' \end{bmatrix} \text{ where}$$

$$x' = [x_1, x_2, ..., x_r]^t \text{ and}$$

$$x'' = [x_{r+1}, ..., x_h]^t$$

$$W^{xu} = \begin{bmatrix} W^{x'u} \\ W^{x''u} \end{bmatrix} \quad \theta^{x} = \begin{bmatrix} \theta^{x'} \\ \theta^{x''} \end{bmatrix} \quad d = \begin{bmatrix} d^{x'} \\ d^{x''} \end{bmatrix} \text{where}$$

$W^{x'u}$ is a r x p matrix, and

$W^{x''u}$ is a (h - r) x p matrix

we have the relations:

$$x' = f(W^{x'u}u + d^{x'} + \theta^{x'})$$
$$x'' = f(W^{x''u}u + d^{x''} + \theta^{x''})$$

The input u, which produces the desired output y, is the solution of the equation:

$$x' = y^* - C x''$$

equivalent with:

$$y^* - Cf(W^{x''u}u + d^{x''} + \theta^{x''}) = f(W^{x'u}u + d^{x'} + \theta^{x'})$$

Expanding [8] the nonlinear function f, through a Taylor series around the point u_k, and solving, the following iterative relations is obtained:

$$\begin{aligned} u = u_k + [\, f'(W^{x'u}u_k + d^{x'} + \theta^{x'}) o W^{x'u} + \\ + C(f'(W^{x''u}u_k + d^{x''} + \theta^{x''}) o W^{x''u})]^{-1} \cdot \\ \cdot [y^* - Cf(W^{x''u}u_k + d^{x''} + \theta^{x''}) - f(W^{x'u}u_k + d^{x'} + \theta^{x'})] \end{aligned} \tag{4}$$

In the previous relation, f(⋅) and f'(⋅) are vectors of corresponding orders, where the function f and its derivative f' is applied on each component of the argument vector. The operator o multiplies each element of a matrix row with the corresponding element of the vector.

The matrix which must be inverted in relation (4) is a r x p matrix. If the number of unfixed network inputs is different than the number of network outputs, the inverse from the equation (4) must be replaced with the suitable pseudo-inverse.

3 Rule Extraction by Interval Propagation

In this section, we present a method for rule extraction from neural networks with continuous inputs and outputs. We named this method, presented also briefly in one of our previous paper [7], the *method of interval propagation*. The rules extracted by this method are crisp if – then rules, in the following form:

$$\textit{if} \quad (a_1 \le x_1 \le b_1) \text{ and } (a_2 \le x_2 \le b_2) \;\; .... \quad \text{and } (a_m \le x_m \le b_m)$$
$$\textit{then} \quad c_j \le y_j \le d_j$$

where $x_1, x_2, \ldots, x_m$ are the inputs of the network and y_j is the j output of the network, $j = 1, 2, \ldots, n$.

A similar method which tries to extract rules in the same form, out of a trained neural network, is the VIA method, developed by Thrun in [13]. VIA method refines the intervals of all units in the network, layer by layer, by techniques of linear programming, such as Simplex algorithm, propagating the constraints forward and backward through the network. The problem is that, VIA method may fail sometimes to decide if a rule is compatible or not with the network. Also the intervals obtained by VIA method are not always optimal. Our method continues the background ideas of VIA method and eliminates the drawbacks of this method.

Given P a layer in the network, and S the next layer. Every node in layer S calculates the value $x_i = f(\sum_{k \in P} w_{ik} x_k + \theta_i)$, where x_k is the output (activation value) of node k in layer P, x_i the output of node i in the layer S, w_{ik} the weight of the link between node k in layer P and node i in layer S, θ_i the bias of node i, and f the transfer function of the units in the network. The following relations can be written:

$$(\forall)\, k \in P \qquad x_k \in [a_k; b_k]$$
$$(\forall)\, i \in S \qquad x_i' = \sum_{k \in P} w_{ik} x_k + \theta_i$$
$$x_i = f(x_i')$$

For every node $i \in S$, we note with w_{il}^{+}, $l \in P_i^{+}$, the positive weights, and with w_{il}^{-}, $l \in P_i^{-}$, the negative weights. ($P_i^{+} \cup P_i^{-} = P$). The interval of variation for x_i', $[a_i'; b_i']$, $(\forall) i \in S$, is determined in the following way:

$$a_i' = \sum_{l \in P_i^{+}} w_{il}^{+} a_l + \sum_{r \in P_i^{-}} w_{ir}^{-} b_r + \theta_i$$
$$b_i' = \sum_{l \in P_i^{+}} w_{il}^{+} b_l + \sum_{r \in P_i^{-}} w_{ir}^{-} a_r + \theta_i$$

and the variation interval for the activation value x_i of node i in layer S is $[a_i; b_i]$, where: $a_i = f(a_i')$ and $b_i = f(b_i')$. In this way, the intervals are propagated, layer by layer, from the input layer to the output layer. So, given the variation intervals for inputs, the intervals of variation for outputs are determined. This is the forward phase. Some of the inputs may be unconstrained, and in this case the intervals are propagated forward across the network layers, assigning the interval of maximum variation ([0; 1]) for unconstrained inputs.

The backward phase appears when it is given the interval of variation for output and eventually for some inputs, and it must be determined the interval for unconstrained inputs. Suppose $x_1, x_2, \ldots, x_k$ are the constrained inputs after renumbering, and $x_{k+1}, \ldots, x_m$ the unconstrained inputs, and we want to determine rules when $(a_1 \le x_1 \le b_1)$ and ... $(a_k \le x_k \le b_k)$ and $(c_j \le y_j \le d_j)$.

First, it is checked the compatibility of the following rule:

$$\textit{if} \quad (a_1 \le x_1 \le b_1) \text{ and } (a_2 \le x_2 \le b_2) \text{ .... and } (a_k \le x_k \le b_k)$$
$$\textit{then} \quad c_j \le y_j \le d_j$$

with the network, assigning the maximum interval ([0; 1]) for unconstrained inputs. By forward propagation, the variation interval $[c_j'; d_j']$ for output is determined. If $[c_j'; d_j'] \subset [c_j; d_j]$, then the rule given above is a general rule, and it does not have sense to look for the variation intervals of remained inputs. If the intersection $[c_j'; d_j'] \cap [c_j; d_j]$ is empty set, then the rule is incompatible with the network. Otherwise, be $y_j^* \in [c_j'; d_j'] \cap [c_j; d_j]$.

By inverting the neural network as given in section 2, it is determined the input $x^*=(x_1^*, x_2^*, \ldots\ x_m^*)$ of the network which produce the output y_j^*. The idea is to find the maximal intervals around the values x_l, l=k+1, …, m, so that the corresponding rule to be compatible with the network. For example, beginning with input x_l, the right margin b_l of the variation interval is set up to:

$$b_l = x_l^* + \frac{1 - x_l^*}{2}$$

If the rule with $x_l \in [x_l^*; b_l]$ is compatible with the network, then the interval is enlarged, otherwise is shrinking, with a technique of dividing intervals into two halves, until the right margin b_l and a_l are determined with a given error. The procedure continues until all the variation intervals for all unconstrained inputs are determined. The hyper-cubs determined at the input depend on the start position – x*, and on the order of the determination of the variation intervals for unconstrained inputs.

Using the method of inverting a neural network described in section 2, the backward phase in VIA method can be reduced, with a very simple calculus, to a forward propagation of the input intervals.

3.1 Case Study

The goal of well-known iris problem is to recognise the type of an iris plant to which a given instance belongs. The data set is composed of 150 records, equally distributed between three classes: setosa, versicolor, and virginica. Two classes are not linearly separable from each other, while the third is linearly separable from the others. The data set is characterised by four attributes: petal length, petal width, sepal length, and sepal width, hence the neural network has four input neurons.

The three possible classes are coded as: 0.1, 0.5 and, respectively, 0.9, such that the application requires a single output neuron. The activation function for network neurons was sigmoid atansig function - $f_{atansig}$.

We trained a neural network with three hidden neurons, shown in figure 2. The input/hidden weights matrix after supervised learning is:

$$W^T = [w_{ij}]^T = \begin{bmatrix} 1.8465 & 5.5406 & 2.7217 & 2.8102 \\ 2.3933 & 2.5549 & -1.6371 & 2.1335 \\ 2.1572 & -6.0267 & 3.2005 & -1.1336 \end{bmatrix}.$$

The hidden/output weights matrix is: $B^T = [\beta_{jk}]^T = [-2.6372 \quad -0.7893 \quad -4.7428]$, while the biases for hidden neurons are: $T^T = [\tau_{jk}]^T = [-3.6773 \quad 0.0046 \quad 3.4464]$.

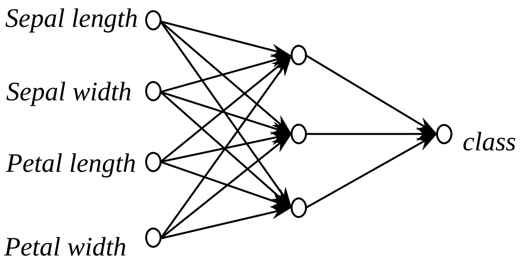

Fig. 2. The three layered neural network for Iris problem

A general form of a rule extracted from trained neural network, with the method presented in the previous section is:

If *sepal lenght* $\in$ $[sl_1;sl_2]$ and *sepal width* $\in$ $[sw_1;sw_2]$ and
petal lenght $\in$ $[pl_1;pl_2]$ and *petal width* $\in$ $[pw_1;pw_2]$
then y $\in$ [0;0.2] (for setosa)

For versicolor and verginica, the extracted rules have in the consequence the interval [0.4; 0.6], and respectively [0.8; 1.0].

We made studies on the balance between the number of extracted rules (comprehensibility) and the percentage of network functioning covering. First, we tried to extract a number of rules comparable with the total number of training instances. After extracting the more general 150 rules, 94% from the network functioning were covered. On the other hand, with just 25 general rules, it is possible to describe 42% of network functioning.

The extracted rule set proved also good generalization ability, comparable with that of the original network. But, generally, the generalization ability depends on the network training. If the network is properly trained, so that to provide good generalization ability, then the extracted set of rules will have also good generalization ability.

For a better approximation (more than 99%), and covering of almost entire part of the network functioning, the obtained number of extracted rules was very much increased. A big number of rules had to be wasted at the decision boundary between classes, especially at the non-linear border between the two not linearly separable classes.

In fact, this is one of the main drawbacks of approximation techniques by traditional crisp rules, the big number of required rules even for approximating the principal and the most general part of the network functioning (except the neighboring regions to the decision boundaries). In order to obtain a more compact way to describe the network, in the following section we extracted a set of fuzzy rules, which compiles in a more concise way the knowledge embedded in the neural network weights during training.

4 Fuzzy Rule Extraction from Neural Networks

The main disadvantage of most approximation techniques of neural networks by fuzzy rules is the exponential increase of required number of rules for a good approximation. In order to obtain a precise approximation, it must be increased the number of linguistic terms for each input/output variable and consequently the number of fuzzy rules [6]. This causes the lose of the significance of the linguistic terms, as well as of the extracted fuzzy rule set. Of course the number of extracted usual fuzzy rules is not as big as when we extract traditional crisp rules from the neural network, but it still remains high.

In the following part of this section, we introduced a fuzzy interactive operator, in order to express in a very few rules what the neural network learned during training. The meaning of the term “interactive” is given by the inputs correlation embedded in the behavior of the fuzzy operator. Based on the theoretical results presented in [1], it is possible to build a fuzzy rule based system which calculates the same function as a neural network. In this manner, the concept of *f*-duality [1], applied on a three layered feed-forward neural network trained to represent a set of data values, can be used to develop a new class of fuzzy connectives [5].

Let us consider the operation + in $\mathcal{R}$ and the sigmoid function atansig, (used commonly as activation function for neural network nodes):

$f_{atansig}(x) = \frac{1}{\pi}\operatorname{atan}(x) + \frac{1}{2}$, continuous (and bijective) application from $\mathcal{R}$ to (0;1).

The $f_{atansig}$-dual operator of + is o, defined on (0;1) as follows:

$$a \text{o} b = \frac{1}{\pi}\left(\frac{\pi}{2} + \operatorname{atan}\frac{\sin(\pi(a+b-1))}{\cos(\pi(a-0.5))\cos(\pi(b-0.5))}\right)$$

Indeed, it can be proved easily that $f_{atansig}(x_1+x_2) = f_{atansig}(x_1)$ o $f_{atansig}(x_2)$. The operator previously defined will be called the *interactive*$_{atan}$-OR operator (for short i_{atan}-OR).

Since f-duality is a general concept, then it could produce other interactive operators (e.g. *i*-OR [1], $i_{\tanh}$-OR [5]), which can be used in knowledge acquisition as well as for motivating the neural inferences.

Based on the properties of i_{atan}-OR [5] and the equivalence theorem proved in [1], it is possible to write the following set of fuzzy rules equivalent with a feed-forward neural network:

$$R_{jk}: \quad \text{IF} \quad \sum_{i=1}^{n} x_i w_{ij} + \tau_j \text{ is } A_{jk} \quad \text{THEN} \quad z_k = \beta_{jk}$$

The fuzzy expressing "x_i is A^i_{jk}" must be interpreted as follows:

"x_i is greater than approximately $r/w_{ij}-\tau_j$" (if $w_{ij}>0$), or

"x_i is lower than approximately $-(r/w_{ij}-\tau_j)$" (if $w_{ij}<0$),

where r is a positive real number obtained from an α-cut (for example 0.9).

The extracted fuzzy rule set equivalent to the neural network trained with Iris data in section 3, obtained by the mechanism given above, is:

R1. IF *sepal-length* is greater than approximately 2.160
i_{atan}-OR
sepal-width is greater than approximately 0.720
i_{atan}-OR
petal-length is greater than approximately 1.465
i_{atan}-OR
petal-width is greater than approximately 1.419
THEN y = -3.6773.

R2. IF *sepal-length* is greater than approximately 1.282
i_{atan}-OR
sepal-width is grater than approximately 1.201
i_{atan}-OR
petal-length is not greater than approximately 1.874
i_{atan}-OR
petal-width is greater than approximately 1.438
THEN y = 0.0046.

R3. IF *sepal-length* is greater than approximately 1.023
i_{atan}-OR
sepal-width is not greater than approximately 0.366
i_{atan}-OR
petal-length is greater than approximately 0.690
i_{atan}-OR
petal-width is not greater than approximately 1.948
THEN y = 3.4464.

The process of classification for a given input is determined by an aggregation computation using the interactive operator. The instance is matched against the rule premises, each rule being fired to a certain degree v_j. The global output is the weighted sum of these degrees: $y = -3.6773v_1 + 0.00462v_2 + 3.4464v_3$.

The class chosen for a given instance is that with the closest numerical value to *y*.

The three fuzzy rules given above present in a more compact way what the neural network learned by training. Because of the nature of the introduced interactive operator, this set of fuzzy rules is less comprehensible than the rules extracted in section 3, but it expresses in just 3 rules the knowledge captured within the neural network by training. The interactive operator enables us to reformulate fuzzy rules into a more compact way, but also still comprehensible.

5 Conclusions

The problem addressed in this paper is how to interpret the knowledge embedded in a trained neural network into a comprehensible as well as compact (concise) way for human user, using traditional and fuzzy rules. The methods presented in the paper are useful in neural based expert systems, and helps to explain the decisions of the neural network in a more familiar form for human expert. The result is an increased confidence of human user in the actions performed by a neural network. The methods can be used to develop human friendly shells for neural network based applications.

References

1. Benitez, J.M., Blanco, A., Delgado, M., Requena, I.: Neural methods for obtaining fuzzy rules. Mathware Soft Computing, vol.3 (1996) 371-382
2. Ishigami, H., Fukuda, T., Shibata, T., Arai, F.: Structure Optimization of Fuzzy Neural Networks by Genetic Algorithm. Fuzzy Sets and System 71 (1995) 257-265
3. Kosko, B.: Neural Networks and Fuzzy Systems. Prentice - Hall International Inc. (1992)
4. Lin, C.T., George Lee, C.S.: Neural - Network Based Fuzzy Logic Control and Decision System. IEEE Transactions on Computers, vol. 40 No. 12 (1991) 1320-1336
5. Neagu, C.D., Palade, V.: An interactive fuzzy operator used in rule extraction from neural networks. Neural Network World 4, Prague (2000) 675-684
6. Palade, V., Neagu, C.D., Negoita, M.: Genetic algorithm optimization of knowledge extraction from neural networks. Proceedings of the 6th International Conference on Neural Information Processing, Perth - Australia, vol. 2 (1999) 765-770
7. Palade, V., Neagu, C.D., Puscasu, G.: Rule extraction from neural networks by interval propagation. Proceedings of the 4th IEEE International Conference on Knowledge-Based Intelligent Engineering Systems & Allied Technologies-KES2000, Brighton - UK (2000) 217-220
8. Palade, V., Puscasu, G., Neagu, C.D.: Neural network-based control by inverting neural models. Control Engineering and Applied Informatics, no. 1, vol. 1, Bucharest (1999) 25-31
9. Patton, R.J., Uppal, F.J., Lopez-Toribio, C.J.: Soft computing aproaches to fault diagnosis for dynamic systems: a survey. Proceedings of the 4th IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes, Budapest, vol. 1 (2000) 298-311
10. Pinkas, G.: Logical Inference in Symmetric Connectionist Networks. Ph.D. thesis, Washington University (1992)
11. Scott, G.M.: Knowledge - Based Artificial Neural Networks for Process Modeling and Control. Ph.D. thesis, University of Wisconsin (1993)
12. Shann, J.J., Fu, H.C.: A fuzzy neural network for rule acquiring on fuzzy control systems. Fuzzy Sets and Systems 71 (1995) 345 – 357
13. Thrun, S.B.: Extracting Symbolic Knowledge from Artificial Neural Networks. Revised Version of Technical Research Report TR-IAI-93-5, Institut für Informatik III - Universität Bonn (1994)
14. Towell, G., Shavlik, J.W.: The Extraction of Refined Rules from Knowledge - Based Neural Networks. Machine-Learning, vol.13 (1993)
15. Yoo, J.H.: Symbolic Rule Extraction from Artificial Neural Networks. Ph.D. thesis, Wayne State University (1993)