Fourth International Conference on knowledge-Based Intelligent Engineering Systems & Allied Technologies, 30th Aug-1st Sept 2000, Brighton,UK

Rule extraction from neural networks by interval propagation

Vasile Palade, Daniel-Ciprian Neagu, Gheorghe Puscasu

University of Galati, School of Electrical Engineering,
Domneasca Str., Nr. 47, Galati 6200, Romania
E-mail: {Vasile.Palade, Dan.Neagu, Gheorghe.Puscasu}@ugal.ro

Abstract: This paper proposes a method of rule extraction from ordinary Backpropagation neural networks, which have not a structure that facilitates the rule extraction. This method is based on interval propagation across the network. The method of rule extraction uses a procedure of inverting a neural network, also presented in the paper. A common benchmark control problem was used to test the rule extraction and inverting methods.

Introduction

With their learning, classification and generalization capabilities, neural networks have been applied with remarkable success to a variety of real-word problems, such as: process control and fault diagnosis, speech recognition, medical diagnosis, image computing. The major shortcoming of neural networks is represented by their low degree of human comprehensibility [9]. 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; most of them into sets of ordinary if - then rules [10][11][9], others into formulas from propositional logic or from nonmonotonic logics [6], or into sets of fuzzy rules [1][2][5]. More transparency is offered by fuzzy neural networks [3][4][8], 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 is structured as follows. Section 2 presents a method of inverting a neural network which represents the forward model of a process. Given the output of the network, the method calculates the input of the network which produces the given output. The inversion method developed in the paper is the most computationally fast inversion method, optimizing the searching of the network inputs which produce the desired network output. The calculus required by the inversion operation can be performed on-line with a regular computer. Section 3 presents a method of traditional if-then rule extraction from neural networks, which have not a special structure that facilitates the rule extraction. The rule extraction method is tested, in section 4, on a common benchmark control problem, the liquid tank. The paper is ending with some conclusions.

Inverting neural networks

The inversion of a neural network is needed especially in neural control structures, such as in inverse and internal model neural control structures. An inversion method was presented in [7]. We improved this method and we obtained a more computationally efficient inversion method of a neural network.

Given a three layered neural network, which represents the forward neural model of a process, 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), 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:

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

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:

$$x = (W^{yx})^{-1} y'$$
$$x' = f^{-1}(x) - \theta^x$$
$$W^{xu} u = x' - W_f^{xu} u_f \qquad (2)$$

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 $d = W_f^{xu} u_f$

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}, by Jordan-Gauss elimination, in the following form:

$$\begin{array}{cccccc} x_1 & x_2 & x_r & x_{r+1} & x_h & y' \\ \end{array}$$
$$\left[\begin{array}{c|c|c} 1\ 0 \ldots\ldots 0 & & \\ 0\ 1 \ldots\ldots 0 & & \\ \ldots\ldots\ldots & C & Y^* \\ \ldots\ldots\ldots & & \\ 0\ 0 \ldots\ldots 1 & & \end{array}\right]$$

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

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

$$x' = [x_1, x_2, \ldots\ldots, x_r]^t \text{ and}$$
$$x'' = [x_{r+1}, \ldots., x_h]^t$$

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

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

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

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 [7] the nonlinear function f, through a Taylor series around the point u_k, the following relations is obtained:

$$y^* - C(f(W^{x''u}u_k + d^{x''} + \theta^{x''}) + f'(W^{x''u}u_k + d^{x''} + \theta^{x''})$$
$$(W^{x''u}u - W^{x''u}u_k)) = f(W^{x'u}u_k + d^{x'} + \theta^{x'}) +$$
$$f'(W^{x'u}u_k + d^{x'} + \theta^{x'})(W^{x'u}u - W^{x'u}u_k) \quad ,$$

where:

$$f'(W^{x''u}u_k + d^{x''} + \theta^{x''}) = \text{Diag}\begin{bmatrix} f'(W_1^{x''u}u_k + d_1^{x''} + \theta_1^{x''}) \\ f'(W_2^{x''u}u_k + d_2^{x''} + \theta_2^{x''}) \\ . \\ . \\ f'(W_{h-r}^{x''u}u_k + d_{h-r}^{x''} + \theta_{h-r}^{x''}) \end{bmatrix}$$

$$f'(W^{x'u}u_k + d^{x'} + \theta^{x'}) = \text{Diag}\begin{bmatrix} f'(W_1^{x'u}u_k + d_1^{x'} + \theta_1^{x'}) \\ f'(W_2^{x'u}u_k + d_2^{x'} + \theta_2^{x'}) \\ . \\ . \\ f'(W_r^{x'u}u_k + d_r^{x'} + \theta_r^{x'}) \end{bmatrix}$$

The notation DiagV, where V is a vector, specifies a diagonal matrix with the components of vector V on the main diagonal and all other matrix elements zero. Solving the equation given above, the following iterative relation is obtained:

$$u = u_k + [f'(W^{x'u}u_k + d^{x'} + \theta^{x'})W^{x'u} + \qquad (3)$$
$$Cf'(W^{x''u}u_k + d^{x''} + \theta^{x''})W^{x''u}]^{-1}$$
$$[y^* - Cf(W^{x''u}u_k + d^{x''} + \theta^{x''}) - f(W^{x'u}u_k + d^{x'} + \theta^{x'})]$$

The matrix which must be inverted in the relation given above 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 (3) must be replaced with the suitable pseudo-inverse. Changing the notations as follows:

$$f'(W^{x''u}u_k + d^{x''} + \theta^{x''}) = \begin{bmatrix} f'(W_1^{x''u}u_k + d_1^{x''} + \theta_1^{x''}) \\ f'(W_2^{x''u}u_k + d_2^{x''} + \theta_2^{x''}) \\ . \\ . \\ f'(W_{h-r}^{x''u}u_k + d_{h-r}^{x''} + \theta_{h-r}^{x''}) \end{bmatrix}$$

$$f'(W^{x'u}u_k + d^{x'} + \theta^{x'}) = \begin{bmatrix} f'(W_1^{x'u}u_k + d_1^{x'} + \theta_1^{x'}) \\ f'(W_2^{x'u}u_k + d_2^{x'} + \theta_2^{x'}) \\ . \\ . \\ f'(W_r^{x'u}u_k + d_r^{x'} + \theta_r^{x'}) \end{bmatrix}$$

the iterative relation (3) becomes:

$$u = u_k + [f'(W^{x'u}u_k + d^{x'} + \theta^{x'}) o W^{x'u} + \qquad (4)$$
$$C(f'(W^{x''u}u_k + d^{x''} + \theta^{x''}) o W^{x''u})]^{-1}$$
$$[y^* - Cf(W^{x''u}u_k + d^{x''} + \theta^{x''}) - f(W^{x'u}u_k + d^{x'} + \theta^{x'})]$$

The operator o from the relation given above multiplies a row of a matrix with the corresponding element of the vector. In this way the complexity of the calculus is significantly reduced, when the iterative relation (4) is implemented, instead of relation (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 our method, the *method of interval propagation*. The rules extracted by this method are crisp if – then rules, in the following form:

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

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

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 [9]. 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. We can write the relations:

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

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 $x_1, x_2, \dots, x_k$ are the constrained inputs after renumerotation, and $x_{k+1}, \dots, x_m$ 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:

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

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 $x^*=(x_1^*, x_2^*, \dots x_m^*)$ of the network which produce the output y_j^*. The idea is to find the maximal intervals around the values x_l, $l=k+1, \dots, m$, 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 established 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 hypercubs 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.

Case study

In order to test the rule extraction method presented in previous section, we have taken the neural network, trained to replace the inverse controller, shown in figure 1, for the well known control problem of liquid tank. The process is described by the following equation, where K=7, A=30, $u \in [0;40]$, $y \in [0;10]$:

$$\frac{dy}{dt} = \frac{1}{A}[u - K\sqrt{y}]$$

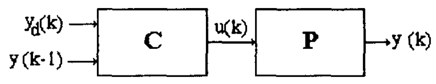

Figure 1. Inverse neural control structure

y_a	y_b	y_a^d	y_b^d	u_a	u_b
0.5600	0.5700	0.5599	0.5701	0.5090	0.5400
0.5600	0.5700	0.5593	0.5696	0.4700	0.5096
0.5150	0.5250	0.5143	0.5251	0.4400	0.4896
0.5150	0.5250	0.5151	0.5257	0.4888	0.5200
0.5209	0.5800	0.4600	0.5209	0	0
0.4600	0.5640	0.5640	0.5800	1	1

The 6 extracted rules, for example, from the table given above, cover just a small part of the network functioning, and they were generated in order to have good setpoint changes performances on a given range of the setpoint variation. For a complete description of the neural controller by a set of if-then rules, we have to extract much more rules, more precisely hundreds or thousands of rules. Using the rule extraction method presented in section 2, we randomly extracted 500 hundreds rules, which covers 98% of the network functioning. Consequently, we replaced the neural controller with a rule based controller. The performances on setpoint changes of the obtained rule based controller are given in figure 2.

Every line in the table represents a rule in the following form:

$$\textit{If} \;\; y(k-1) \in [y_a; y_b] \text{ and } y_d(k) \in [y_a^d; y_b^d]$$
$$\textit{then} \;\; u(k-1) \in [u_a; u_b]$$

A rule is activated if the condition in the premise of the rule is true. An inference cycle consists of finding the set of activated rules, and then the intersection of the intervals from the consequences of the activated rules. The output of the controller will be the median value of the interval previously found.

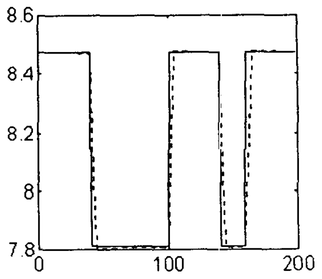

Figure 2. The performances of the rule based controller on setpoint changes

Conclusions

A method of rule extraction from neural networks was proposed in this paper. This method is useful in neural based expert systems, and helps to explain the decisions of the neural network in a familiar form for human users. Also the paper presents an efficient method of inverting a neural network, which can be used in the inverse neural control structure and neural model based control structure.

References

[1] Ishigami H., T. Fukuda, T. Shibata and F. Arai, *Structure Optimization of Fuzzy Neural Network by Genetic Algorithm*, Fuzzy Sets and System 71, pp. 257-265, 1995.

[2] B. Kosko, *Neural Networks and Fuzzy Systems*, Prentice - Hall International Inc., 1992.

[3] Lin C.T. and C.S. George Lee, *Neural - Network Based Fuzzy Logic Control and Decision System*, IEEE Transactions on Computers, vol. 40 No. 12, pp. 1320-1336, 1991.

[4] Neagu D., M. Negoita and V. Palade, *Aspects of integration of explicit and implicit knowledge in connectionist expert systems*, Proc. of the 6th International Conference on Neural Information Processing, Perth - Australia, vol. 2, pp. 759-764, 1999.

[5] V. Palade, G.Puscasu and D. Neagu, *GA optimization of knowledge extraction from neural networks*, Proc. of the 6th International Conference on Neural Information Processing, Perth - Australia, vol. 2, pp. 765-770, 1999.

[6] Pinkas G., *Logical Inference in Symmetric connectionist Networks*, PhD. thesis, Washington University, 1992.

[7] G. M. Scott, *Knowledge - Based Artificial Neural Networks for Process Modelling and Control*, PhD. thesis, University of Wisconsin, 1993.

[8] Shann J.J. and H.C. Fu, *A fuzzy neural network for rule acquiring on fuzzy control systems*, Fuzzy Sets and Systems vol. 71, pp. 345 - 357, 1995.

[9] 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.

[10] Towell G., J.W. Shavlik, *The Extraction of Refined Rules from Knowledge - Based Neural Networks*, Machine-Learning, vol.13, 1993.

[11] Yoo J.H., *Symbolic Rule Extraction from Artificial Neural Networks*, PhD thesis, Wayne State University, 1993.