Extraction of logical rules from training data using backpropagation networks

Włodzisław Duch, Rafał Adamczak and Krzysztof Grąbczewski

Abstract— **Simple method for extraction of logical rules from neural networks trained with backpropagation algorithm is presented. Logical interpretation is assured by adding an additional term to the cost function, forcing the weight values to be ±1 or zero. Auxiliary constraint ensures that the training process strives to a network with maximal number of zero weights, which augmented by weight pruning yields a minimal number of logical rules extracted by means of weights analysis. Rules are generated consecutively, from most general, covering many training examples, to most specific, covering a few or even single cases. If there are any exceptions to these rules, they are being detected by additional neurons.**

The algorithm applied to the Iris classification problem generates 3 rules which give 98.7% accuracy. The rules found for the three monks and mushroom problems classify all the examples correctly.

Keywords— **Neural networks, MLP, backpropagation, logical rules.**

I. Introduction

EXTRACTION of logical rules from the data is an important problem that has so far eluded satisfactory solution (for a review see [1]). Adaptive systems A_W, such as the multi-layered perceptrons (MLPs), are useful classifiers that adjust internal parameters W performing vector mappings from the input to the output space $Y^{(p)} = A_W(X^{(p)})$. Some classification problems have an inherent logical structure. In such cases it is preferable to use logical rules instead of adaptive classifiers, because logical reasoning is more acceptable to human users than the recommendations given by a black box systems [1]. Although the class of problems with inherent logical structure simple enough to be manageable by humans may be rather limited nevertheless it covers some important applications, such as the decision support systems in financial institutions.

Straightforward approach to the extraction of logical rules from neural systems is to use fuzzy logic and localized neuron transfer functions. In such a case the rules are of the type:

$$\begin{array}{ll} \text{IF} & \left(x_1 \in X_1 \wedge x_2 \in X_2 \wedge ... x_N \in X_N\right) \\ \text{THEN} & \left(y_1 \in Y_1 \wedge y_2 \in Y_2 \wedge ... y_M \in Y_N\right) \end{array} \quad (1)$$

In particular the radial basis function (RBF) approach is equivalent to the fuzzy logic systems with the gaussian membership functions [2]. Other networks that use factorizable functions, such as the Feature Space Mapping (FSM) network [3], may be treated as neurofuzzy systems using arbitrary membership functions. In this paper we will show how to obtain logical rules from MLP networks, by far the most popular and successful of the neural network models. The algorithm for extraction of logical rules is presented in the next section and illustrated in the third section. Further improvements of the algorithm are discussed in the fourth section and the paper is finished with a short discussion.

Authors are with the Department of Computer Methods, Nicholas Copernicus University, Grudziądzka 5, 87-100 Toruń, Poland. E-mail: duch,raad,kgrabcze@phys.uni.torun.pl

II. The algorithm

Logical rules require symbolic inputs (linguistic variables), therefore the input data has to be quantized first, i.e. the features defining the problem should be identified and labeled. If the input data components x_i are given as real numbers in the preprocessing stage one has to divide the data in distinct (for crisp logic) sets and introduce new, logical (linguistic) input variables s_k:

$$\text{IF } \left(x_i \in X_{i,j}\right) \text{ THEN } \left(s_k = label(x_k) = \text{T}\right) \quad (2)$$

For example, $s_k = s$ may designate the fact that the feature s_k is small, and $s_k = \neg s$ that it is not small. The problem of optimal selection of input features is very important and may be solved in an adaptive way by analysis of the nodes developed by the FSM network [3]. Crisp decision regions may be obtained in an adaptive way by using the product $\prod_i \sigma(x_i - b_i)(1 - \sigma(x_i + b'_i))$ of functions as the neuron processing function [4] and slowly increasing the gain of the sigmoidal functions $\sigma(x)$ during learning. Here we will assume that the input is already quantized. Each quantized feature s will have two or more values represented by a vector $V_{s1} = (+1, -1, -1...)$ for the first value, $V_{s2} = (-1, +1, -1...)$ for the second value etc. Quantization is implemented by the input nodes of the network, each node receiving features x_i (real numbers) and converting them to the vectors representing linguistic variables s_k.

Interpretation of the activation of the MLP network nodes is not easy [5]. To facilitate such an interpretation we will use very steep (high gain) sigmoid functions, train only the input layer and enforce the integer weight values 0 and ±1, interpreted as 0 = irrelevant input, +1 = must be present and −1 = must not be present. This is achieved by modifying the error function:

$$\begin{aligned} E(W) = \quad & \frac{1}{2}\sum_p \sum_k \left(Y_k^{(p)} - \mathbf{A}_W\left(X^{(p)}\right)_k\right)^2 + \quad (3) \\ & \frac{\lambda}{2}\sum_{i>j} W_{ij}^2 (W_{ij}-1)^2 (W_{ij}+1)^2 \end{aligned}$$

The first part is the standard measure of matching the network outputs $\mathbf{A}_W(X^{(p)})$ with the desired outputs $Y^{(p)}$ for all training data p. The second term is a sum over all weights and has minimum (zero) for weights approaching zero or ± 1. Similarly as in the weight pruning technique case in the backpropagation algorithm this term leads to the additional change of weights:

$$W_{ij} \leftarrow W_{ij} + \lambda W_{ij}(W_{ij}^2 - 1)(3W_{ij}^2 - 1) \qquad (4)$$

where λ scales the relative importance of auxiliary conditions. The training proceeds separately for each output class. One hidden neuron is created and is trained on all data by the backpropagation procedure until convergence. The weights and the threshold obtained are then analyzed and the first group of logical rules is found, covering the most common input-output relations. The input data that is correctly handled by the first neuron will not contribute to the error function, therefore the weights of this neuron are kept frozen during further training. A second neuron is added and trained on the remaining data. After convergence the second weight vector is analyzed and corresponding rules found. This procedure is repeated until all data are correctly classified, weights analyzed and a set of rules $R_1 \vee R_2 ... \vee R_n$ is found, identifying $Class1$. The output neuron for the first class is connected to the hidden neurons. It performs a simple summation of the incoming signals (Fig. 1). The same procedure is repeated for the remaining classes. Each time only one neuron is trained, therefore the training is very fast.

Rules $\mathcal{R}_k$ implemented by trained neurons are written in the form of logical conditions by considering contributions of inputs for each linguistic variable. Such variable s is represented by a vector V_s and its contribution to the activation is equal to the dot product of the subset W_s of the weight vector $V_s \cdot W_s$. A combination of linguistic variables activating the hidden neuron above the threshold is a logical rule in the form:

$$\text{IF } (s_1 \wedge \neg s_2 \wedge ... \wedge s_k) \text{ THEN } Class1 \qquad (5)$$

The rules obtained by this algorithm are ordered, starting with rules that are used most often and ending with rules that handle only a few cases. The final solution may be presented as a set of rules or as a network of nodes performing logical functions, with hidden neurons realizing the rules and the hidden-output neuron weights all set to +1.

III. Two examples

The first, rather trivial example, is the XOR case. As expected, the algorithm handles the first two input cases (-1,-1) and (+1,+1) with one hidden neuron and $W_{11} = +1, W_{21} = -1$ and the next two input cases $(-1,+1)$ and $(+1,-1)$ with $W_{12} = +1, W_{22} = -1$.

In the second example the classical Iris dataset was used. The data has 150 vectors evenly distributed in three classes, called iris-setosa, iris-versicolor and iris-virginica. Each vector has four features: sepal length x_1 and width x_2, and petal length x_3 and width x_4 (all in cm). The input values (length) for each of these features were divided into three equal parts, called small (s), medium (m) and large (l). Thus x_1 is called small if it is in $[4.3, 5.5]$ range, medium if in $(5.5, 6.7]$ and large if in $(6.7, 7.9]$. Thus instead of four inputs a network with 12 inputs equal to ± 1 is constructed. For example, the medium value of a single feature is coded by $(-1, +1, -1)$. With this discretization of the input features three vectors of the iris-versicolor class (coded as (m, m, l, l), (m, l, m, l) and (m, s, l, m)) become identical with a number of iris-virginica vectors and cannot be classified correctly. These vectors were removed from the training sequence.

For the Iris dataset a single neuron per one class was sufficient to train the network, therefore the final network structure is 12 input nodes and 3 output nodes (hidden nodes are only needed when more than one neuron is necessary to cover all rules for a given class). The scaling parameter was increased from $\lambda = 0.001$ at the beginning of the training to $\lambda = 0.01 - 0.1$ near the end. The network needed about 1000 epochs on average and the final weights were within 0.05 from the desired ± 1 or 0 values. The following weights and thresholds are obtained (only the signs of the weights are written):

Iris-setosa: $(+, 0, 0; 0, 0, +; +, -, 0; +, -, -), \theta = 2$
Iris-versicolor: $(0, 0, 0; 0, 0, 0; 0, +, -; 0, +, -), \theta = 3$
Iris-virginica: $(0, 0, 0; 0, 0, 0; -, -, +; -, -, +), \theta = 1$

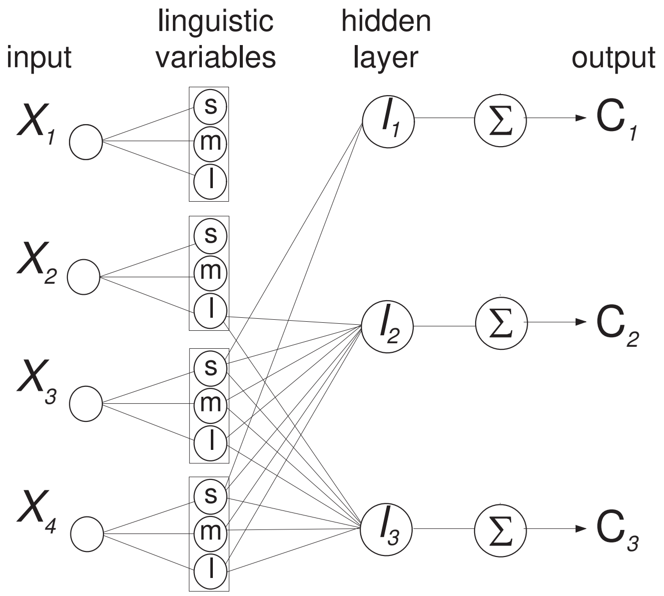

Fig. 1. Final structure of the network for the Iris problem.

Logical rules corresponding to these weights may be presented in several equivalent ways. In the simplest case analysis proceeds as follows: contributions from various values to the activation of the output neuron are considered. For the Iris-setosa vectors the weights for the first feature are $(+,0,0)$, therefore contribution from $x_1 = s$ is $\Delta = +1$ and from both $x_1 = m$ and $x_1 = l$, equivalent to $x_1 = \neg s$, contribution is $\Delta = -1$. Analysis of other features and weights is summed up in Table I.

TABLE I

Contributions of features for the first class (Iris-setosa).

No.	value	Δ	value	Δ	value	Δ
x_1	s	+1	$\neg s$	-1		
x_2	l	+1	$\neg l$	-1		
x_3	s	+2	m	-2	l	0
x_4	s	+3	$\neg s$	-1		

Using this table one can easily create a tree (Fig. 2) with weights equal to the total contribution of each feature to the final activation. At the first level contribution of x_4 is $+3$ for $x_4 = s$ or -1 for $x_4 = \neg s$; at the second level (for x_3) the branching factor is 3, for x_1 it is 2 and for x_2 it is 2, giving a total of 24 leaves. For Iris-setosa class only the leaves with activation equal to or larger than the threshold $\theta = 2$ should be considered. Logical rules are read directly from this tree. Equivalent rules are obtained if the order in which the levels are considered is changed. A useful heuristic to find the simplest set of rules is to start with features that contribute the most to the activation (in this case with features 4 and 3). As shown in Fig. 2, if $x_4 = s$ the activation Δ is already 3 and if it is followed by $x_3 = s$ the activation $\Delta = 5$ and the two other features will not reduce the activation below 3 (since each may subtruct at most 1). Therefore the activation is greater than the threshold $\Delta \geq \theta = 2$ for $x_3 = s \wedge x_4 = s$. In the same way other conditions consistent with the weights are found, giving a rule with four antecedents for class Iris-setosa, one rule for Iris-versicolor and one for Iris-virginica:

$$\begin{aligned} \text{IF} \quad & (x_3 = s \wedge x_4 = s) \vee \\ & (x_1 = s \wedge x_3 = l \wedge x_4 = s) \vee \\ & (x_1 = \neg s \wedge x_2 = l \wedge x_3 = l \wedge x_4 = s) \vee \\ & (x_1 = s \wedge x_2 = l \wedge x_3 = s \wedge x_4 = \neg s) \\ & \text{THEN iris-setosa} \end{aligned} \tag{6}$$

$$\text{IF} \quad (x_3 = m \wedge x_4 = m) \text{ THEN iris-versicolor} \tag{7}$$

$$\text{IF} \quad (x_3 = l) \vee (x_4 = l) \text{ THEN iris-virginica} \tag{8}$$

These rules allow for correct classification of the 147 vectors, achieving 98% of accuracy. The validity of the rules found was confirmed with a Prolog program. The accuracy of classification using logical rules critically depends on selection of features. For example, dividing each input value into two classes only, small and large, extending to the middle of [min,max] segment, 13 vectors from Iris-setosa class

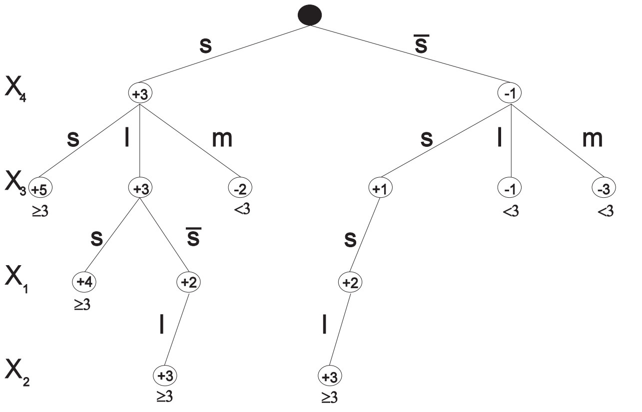

Fig. 2. Tree-based search for rules after network has been trained.

get mixed with the vectors from two other classes. Division into 4 classes also decreases classification accuracy, mixing 16 Iris-versicolor cases with Iris-virginica. Evidently division into 3 classes was fortuitous.

IV. Improvements

Analysis of the histograms of the individual features for each class shows that the division into 3 equal parts is almost optimal, cutting the histograms into the regions where values of features are most frequently found in a given class. For example, Iris-virginica class is more frequent for the value of x_3 above 4.93 and Iris-versicolor are more frequent below this value. Discretization based on histograms made by dividing the data range into 15 bins and smoothing the histogram by counting not only the number of vectors falling in a given bin but also adding 0.4 to adjacent bins is shown in Fig. 3. This discretization leads to the following table for linguistic variables:

TABLE II

Linguistic variables obtained by analysis of histograms.

	s	m	l
x_1	[4.3,5.5]	(5.5,6.1]	(6.1,7.9]
x_2	[2.0,2,75]	(2.75,3.2]	(3.2,4.4]
x_3	[1.0,2.0]	(2.0,4.93]	(4.93,6.9]
x_4	[0.1,0.6]	(0.6,1.7]	(1.7,2.5]

With this discretization and the rules given above only two vectors cannot be classified, i.e. classification accuracy is 98.7%.

The weight vector found does not guarantee that the simplest rules are found. There may be many zeros in the error function for different sets of weights. To enforce the simplest structure of the weight vector we will break the extra term in the error function into two parts:

$$\begin{aligned} E(W) = \quad & \frac{1}{2}\sum_p \sum_k \left(Y_k^{(p)} - \mathbf{A}_W\left(X^{(p)}\right)_k\right)^2 + \\ & \frac{\lambda_1}{2}\sum_{i>j} W_{ij}^2 + \frac{\lambda_2}{2}\sum_{i>j} W_{ij}^2 (W_{ij}-1)^2 (W_{ij}+1)^2 \end{aligned} \tag{9}$$

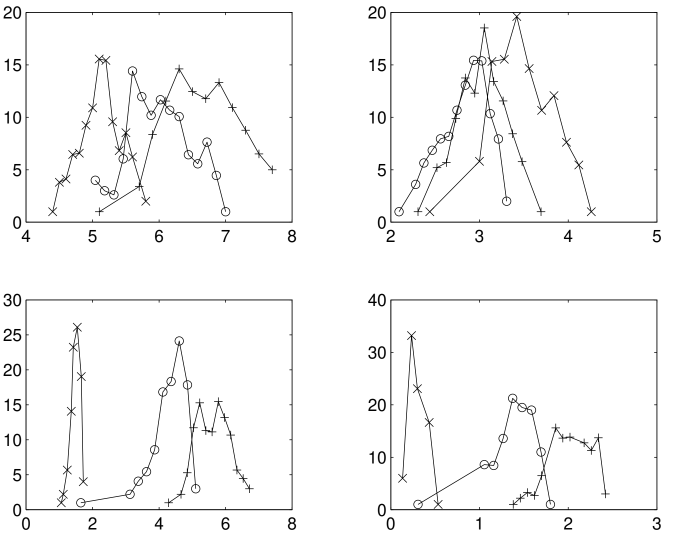

Fig. 3. Histograms of the four Iris features. The x_3, x_4 features (lower pictures) allow for better discrimination than the first two features.

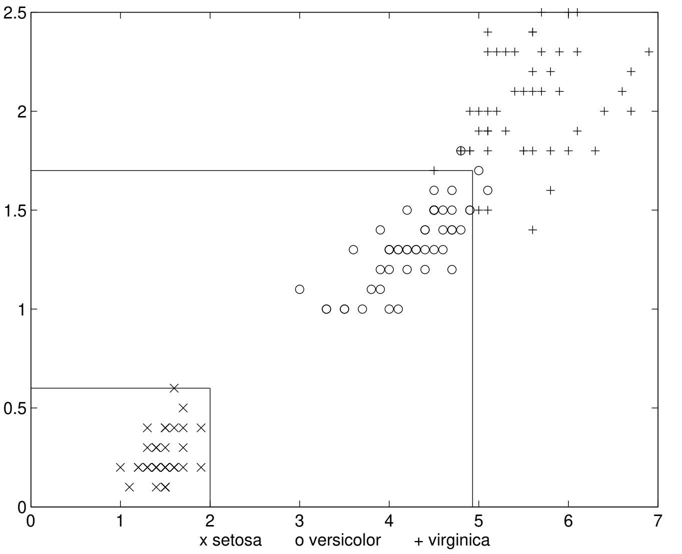

Fig. 4. Iris dataset displayed in x_3 and x_4 coordinates; decision regions (rules) for the three classes are also shown. Note the three Iris-versicolor cases that are incorrectly classified using these two features only.

The additional term with λ_1 gives backpropagation procedure an additional incentive to find solutions with large number of zero weights, i.e. to eliminate irrelevant features. In the Iris dataset case the weight vector for the neuron correctly classifying samples from the first class is now simplified to $(0,0,0;0,0,0;+,0,0;+,0,0), \theta = 1$ and only one rule is left!

$$\text{IF } (x_3 = s \vee x_4 = s) \text{ THEN } \text{ iris-setosa} \tag{10}$$

The three rules, Eq. 7, 8 and 10, one for each class, allow for correct classification of 147 vectors (98%) using only the x_3 and x_4 features. This is the simplest description of the Iris dataset that we know of (Fig. 4).

Discretization according to the histogram values requires rejection of two Iris-versicolor vectors and leads to the following weights:

Iris-setosa: $(0,0,0;0,0,0;+,0,0;+,0,0), \theta = 1$
Iris-versicolor: $(0,0,0;0,0,+;-,+,-;-,+,-), \theta = 3$
Iris-virginica: $(0,0,0;0,0,-;-,-,+;-,-,+), \theta = 2$

The final structure of the network correctly classifying 148 vectors (98.7%) is shown in Fig. 1. In addition to the rule (10) two logical rules are found:

$$\begin{aligned} \text{IF} \quad & (x_3 = m \wedge x_4 = m) \vee (x_2 = l \wedge x_3 = m) \vee \\ & (x_2 = l \wedge x_4 = m) \text{ THEN } \text{ iris-versicolor} \quad (11) \\ \text{IF} \quad & (x_3 = l \wedge x_4 = l) \vee (x_2 = \neg l \wedge x_3 = l) \vee \\ & (x_2 = \neg l \wedge x_4 = l) \text{ THEN } \text{ iris-virginica} \quad (12) \end{aligned}$$

V. More examples and further improvements to the algorithm

Despite of the difficulties with discretizing the input, the case of iris dataset turned out to be relatively simple. Each of the three classes was classified correctly by a network with single hidden neuron. The problems we describe in this section deal with objects described by discrete features. However to have them resolved we had to build more sophisticated networks. It revealed some limitations of the algorithm in the form described in section II and suggested the necessary improvements.

The new classification tasks we tried our method against are the three monks and mushrooms problems. They are described in detail in [1]. We state them here in a compact form:

Each of the three monks problems is to determine whether an object described by six features[1] is a monk or not. The problems define "being a monk" as having features satisfying the following formulae respectively:

1. head shape = body shape $\vee$ jacket color = *red*
2. exactly two of the six features have their first values
3. $\neg$ (body shape = *octagon* $\vee$ jacket color = *blue*) $\vee$ (holding = *sward* $\wedge$ jacket color = *green*)

The mushrooms problem is to classify mushrooms as poisonous or edible, given their 22 attributes. The attributes have between 2 and 12 different values.

Sometimes attempts to train a net with a single neuron are unsuccessful. In such cases it is reasonable to start with two neurons in the hidden layer and train them simultaneously. If this does not help the number of neurons is

[1] The features and their values can be found in Fig. 5.

increased until a convergeable network is constructed. After that the weights for all the trained neurons get frozen. We used this technique for instance in the monks 1 example (see Fig. 5) - the first two neurons were constructed this way. In some other tests we had to allow up to four neurons.

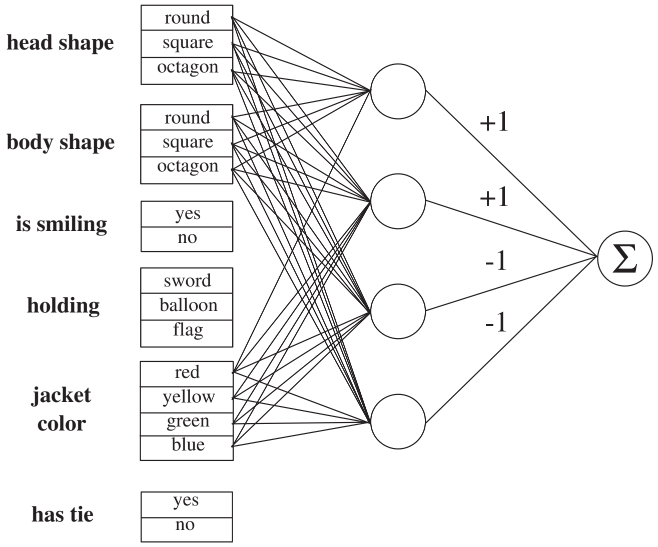

Fig. 5. The network for monks 1 problem. The first two neurons were taught simultaneously. The other two handle exceptions.

Another modification to the method had to be made to support the cases when the network we create learns more general patterns then needed. The monks 1 problem illustrates this. The first two neurons classify properly all the positive examples, however some negative ones are also accepted. It means that the patterns, which are not recognized properly are exceptions to the rules extracted from the network. To rectify this we have to extend the hidden layer by some neurons with a negative contribution to the output node. Two different ways leading to this result seem to be natural and simple. We can either use a sigmoidal function mapping onto the interval $(-1, 0)$ or set the weights of links connecting the new neurons with the output to -1. In both cases the training dataset must be changed so that only the exceptions give nonzero output (-1 and 1 respectively). Then our algorithm can be applied to it, as if a new task were defined. After the whole process is finished we have two separate sets of rules, one comprising information on positive examples, and the other describing exceptions. In further descriptions we will use the word "rules" to mean the rules of the first set, and the word "exceptions" for the members of the second set. To classify a pattern correctly, the first condition one ought to check is whether it is an exception, and then (only if it is not true) the basic classification rules can be applied to determine if the pattern belongs to the class.

Our method applied to the monks 1 problem needed three passes[2]. The two hidden neurons mentioned above, trained during the first pass, recognized all the positive examples and 11 negative ones. So, we had to weed the exceptions out. In the second pass, training one hidden neuron let us detect 6 of them. The third pass with another hidden neuron added was taught the remaining 5 exceptions. Some statistics concerning all the stages of the algorithm for all the problems presented in this section have been composed into Table III. Successive columns of the table have the following meaning: the first specifies problems and the final numbers of generated rules and exceptions, the second enumerates particular stages, the third gives the number of neurons trained simultaneously and fourth says if the aim was searching for rules or exceptions (to highlight the difference rules are printed in bold and exceptions in italic). The fifth column contains the numbers of instances classified properly thanks to rules generated during a given pass. The last column seems to confirm our note, that the method learns the most common rules first. The isolated cases are being recognized after subsequent stages.

TABLE III

Stages statistics for some classification tasks.

Problem	Pass No.	Neurons	Rules/Exc.	Examples
monks 1	**1**	**2**	**rules**	**42**
4 rules	*2*	*1*	*exceptions*	*6*
2 exceptions	*3*	*1*	*exceptions*	*5*
monks 2	**1**	**1**	**rules**	**33**
16 rules	*2*	*1*	*exceptions*	*5*
8 exceptions	**3**	**1**	**rules**	**16**
	4	*2*	*exceptions*	*6*
	5	**2**	**rules**	**10**
	6	*2*	*exceptions*	*3*
	7	**4**	**rules**	**5**
monks 3	**1**	**1**	**rules**	**57**
3 rules	*2*	*2*	*exceptions*	*5*
4 exceptions	**3**	**1**	**rules**	**3**
mushrooms	**1**	**1**	**rules**	**3868**
12 rules	**2**	**1**	**rules**	**40**
no exceptions	**3**	**1**	**rules**	**8**

In the monks 1 example we ended up with 4 rules and 2 exceptions, altogether composed of 14 atomic formulae. They classify the training data without any errors.

Although the monks 2 example definition is very simple, the training process required much effort. As shown in Table III it needed the biggest number of passes of the algorithm. Each of the three first rules searching stages ended with some exceptions and thence required additional stages. Moreover last stages made the impression that the relations among the training samples were very difficult to detect. Three passes trained networks with two hidden units, and the last one required even four units. It is worth to point out that the four nodes of the network constructed during the last pass are responsible for correct classification of just five examples. This shows how the neurons

[2]By one pass (or stage) we mean a single process of training leading to convergence and finished with freezing the weights of all trained neuron

trained at the final passes of our algorithm can specialize in recognizing patterns which do not resemble the others. We extracted 16 rules and 8 exceptions from the resulting network. The number of atomic formulae which compose them is 132.

The third monks problem also required one additional pass to find exceptions. Finally, two neurons gave three rules, and other two nodes generated four exceptions. The whole logical system for this case contains 33 atomic formulae.

The last, but by no means least result of the algorithm was achieved for the mushrooms classification dataset. This example is in some sense the richest one, as the database consists of 8124 vectors. No additional information about how to extract the training set is distributed with it, so we decided to train our network with all the data. We were quite surprised to see a single neuron capable of learning all the training samples. Unfortunately such a network was useless for us, as it had too many nonzero weights. The search tree used in the process of extracting rules grows exponentially with the number of "nonredundant" attributes describing objects. The tree (Fig. 2) built for the case of Iris-setosa was small and easy to trace. Thanks to the small number of features (4) it was possible to work out the rules with a pen and a piece of paper. It wouldn't be possible with a 22 levels tree as each feature can have up to three different values of possible contribution to the weighted sum which determines the signal sent out by the hidden unit. Our algorithm uses some heuristics, to avoid searching branches which do not give any rules and some simpler methods can check it out. Nevertheless, we have to assume, that the complexity of the algorithm is 3^N, where N is the number of nonredundant attributes. It means that in many cases we have to adjust relevant training coefficients to reduce to zero as many weights as possible. In the mushrooms example, the price we had to pay was increasing the number of passes (and neurons as well) to three, as one unit was no longer able to learn all the patterns with satisfiably small number of nonzero weights. However it made the analysis of the network much easier (i.e. possible) and reduced the number of rules to 12, and the number of atomic formulae to 27.

Rules extraction for Iris-setosa was so easy for one more reason: each feature had just three possible values. As a result of this each contribution of a feature to a weighted sum could be unambiguously represented by a feature value or its negation. In more complicated cases a contribution might be represented by a complex formula. Human make multiple mistakes during the analysis needed for extraction of rules, therefore at quite an early stage of the work we decided to write a piece of C++ code to make the complicated weights analysis for us. The accuracy of all the examples was also checked automatically by simple prolog programs.

VI. Discussion and summary

The problem of extracting rules from neural networks has a natural geometrical interpretation. Crisp logic rules correspond to a division of the input space with perpendicular hyperplanes into areas with symbolic names (Fig. 4). If the classes in the input space are correctly separated with such hyperplanes logical description of the data is possible. Logical approximation may become arbitrarily accurate by increasing the number of linguistic variables, but the number of rules may become unacceptably large. Fuzzy logic offers better approximation with smaller number of rules, including simple piecewise linear approximations rules and more complex membership functions. In RBF or FSM systems [3] after initial clusterization positions as well as the shapes of prototype data clusters are modified, giving complex membership functions for each class. As long as separable network functions are used the rules: IF $X \in X_C$ THEN the vector X is of the class C, may be analyzed in the fuzzy logic sense.

We have presented here a simple method of rule extraction based on the standard backpropagation technique with modified error function. Crisp logical rules are found automatically by analyzing nodes of trained networks. The method seems to outperform in many ways previous methods of rule extraction [1], [7], [8].

Acknowledgments

Support by the Polish Committee for Scientific Research, grant 8T11F 00308, is gratefully acknowledged.

References

[1] R. Andrews, J. Diederich, A.B. Tickle, "A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks," Knowledge-Based Systems vol. 8, pp. 373–389, 1995.

[2] J-S. R. Jang, C.T. Sun, "Functional Equivalence Between Radial Basis Function Neural Networks and Fuzzy Inference Systems," *IEEE Trans. on Neural Networks*, vol. 4, no. 1, pp. 156–158, 1993.

[3] W. Duch, G.H.F. Diercksen, "Feature Space Mapping as a universal adaptive system," *Computer Physics Communications,* vol. 87, pp. 341–371, 1995.

[4] W. Duch and N. Jankowski, "Bi-radial transfer functions," in *Proc. second conference on neural networks and their applications*, Orle Gniazdo, Poland, vol. I, pp. 131–137, 1996.

[5] J.M. Żurada, "Introduction to Artificial Neural Systems," West Publishing Company, St Paul, 1992.

[6] ftp.ics.uci.edu/pub/machine-learning-databases contains the Iris dataset.

[7] LiMiu Fu, "Neural Networks in Computer Intelligence," McGraw Hill Inc. 1994.

[8] A. Lozowski, T.J. Cholewo, J.M. Żurada, "Symbolic rule representation in neural network models," in *Proc. second conference on neural networks and their applications*, Orle Gniazdo, Poland, vol. II, pp. 300–305, 1996.