Extraction of crisp logical rules using constrained backpropagation networks

Włodzisław Duch, Rafał Adamczak and Krzysztof Grąbczewski
Department of Computer Methods, Nicholas Copernicus University, Grudziądzka 5,
87-100 Toruń, Poland. E-mail: duch,raad,kgrabcze@phys.uni.torun.pl

Abstract

The problem of extraction of crisp logical rules from neural networks trained with backpropagation algorithm is solved by transforming these networks into simpler networks performing logical functions. Two constraints are included in the cost function: regularization term inducing weight decay and additional term forcing the remaining weights to ± 1. Networks with minimal number of connections are created, leading to a small number of crisp logical rules. A constructive algorithm is proposed, in which rules are generated consecutively by adding more nodes to the network. Rules that are most general, covering many training examples, are created first, followed by more specific rules, covering a few cases only. Generation of new rules is stopped when their application on the test dataset does not increase the number of correctly classified cases. Our constructive algorithm applied to the Iris classification problem generates two rules with three antecedents giving 98.7% accuracy. A single rule for the mushroom problem leads to 98.52% accuracy while three additional rules allow for perfect classification. The rules found for the three monk problems classify all the examples correctly.

I. Introduction

Extraction of logical rules from the data is an important problem that has so far eluded satisfactory solution (for a recent review see [1]). There are many reasons why logical rules should be preferred over other methods of classification, provided that the complexity of the set of rules will not be too large and their accuracy will be sufficiently high. In some medical applications simple rules proved to be more accurate and were able to generalize better than many machine and neural learning algorithms [2]. Results presented in this paper give further support to the superiority of logical rules over other classification methods.

Adaptive systems M_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 $Y^{(p)} = M_W(X^{(p)})$. Although they may achieve high accuracy of classification the knowledge acquired by such neural systems is represented in a set of numerical parameters and architectures of networks in an incomprehensible way. Many methods to analyze trained neural networks, extract logical rules and select classification features have been devised in the past. Most important rule extraction methods have been reviewed and compared experimentally quite recently [1], therefore we will not discuss them here. These methods focus on analysis of parameters (weights) of trained networks, trying to achieve high fidelity of performance, i.e. obtaining identical classification results by extracted logical rules in comparison to the original networks.

Non-standard form of rules, such as M-of-N (M out of N antecedents should be true) or decision trees [3], are sometimes useful but in this paper we will consider only standard rules based on crisp logic. Our approach is quite straightforward: to extract rules from a trained neural network one should transform it smoothly into something resembling a logical network. We will discuss two approaches here. First, in which logical rules are extracted from MLP networks by gradually imposing constraints on the cost function, changing MLPs into logical networks. In the second approach one builds a logical network performing desired classifications. The first approach starts from larger network and simplifies it, while the second approach starts from a single neuron and constructs the network using the training data. These algorithms are presented in the next section. Performance of the constructive algorithm is illustrated on the three benchmark problems in the third section. The paper is finished with a short summary.

II. The algorithm

Logical rules require symbolic inputs (linguistic variables). 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 or other density networks [4]. Crisp decision regions are obtained in an adaptive way by using as the neuron processing function a pure product form of sigmoidal functions $\prod_i \sigma(x_i - b_i)(1 - \sigma(x_i - b_i'))$, product of differences $\prod_i(\sigma(x_i - b_i) - \sigma(x_i - b_i'))$ or a filtered combination of differences $\sigma\left(\sum_i(\sigma(x_i - b_i) - \sigma(x_i - b_i')) - B\right)$, slowly increasing the gain of the sigmoidal functions $\sigma(x)$ during learning. In this process fuzzy rules are transformed into crisp logical rules, i.e. complex decision regions

are transformed into simpler, hypercuboidal decision regions [5], [6]. We will not discuss this problem here. Logical (linguistic) input variables s_k for continuous input data components x_i may be obtained by dividing the data in distinct (for crisp logic) sets: IF $(x_i \in X_{i,j})$ THEN $(s_k = \mathrm{T})$. 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. 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.

Interpretation of the activation of the MLP network nodes is not easy [7]. To facilitate such an interpretation a smooth transition from MLP to a logical-type of network performing similar functions is advocated. This is achieved by: a) increasing the slope of sigmoidal functions to obtain crisp decision regions; b) simplifying the network structure by inducing the weight decay through a penalty term; c) enforcing the integer weight values 0 and ± 1, interpreted as 0 = irrelevant input, $+1$ = positive and -1 = negative evidence. These objectives are achieved by modifying the error function:

$$E(W) = \frac{1}{2}\sum_p \sum_k \left(Y_k^{(p)} - \mathbf{M}_W\left(X^{(p)}\right)_k\right)^2 + \frac{\lambda_1}{2}\sum_{i,j} W_{ij}^2 + \frac{\lambda_2}{2}\sum_{i,j} W_{ij}^2 (W_{ij}-1)^2 (W_{ij}+1)^2 \qquad (1)$$

Two additional terms are added to the standard mean square error. The first term, scaled by λ_1 hyperparameter, encourages weight decay, leading to skeletonization of the network and elimination of irrelevant features. The second term, scaled by λ_2, forces the remaining weights to approach ± 1, facilitating easy logical interpretation of the network function. In the backpropagation training algorithm these new terms lead to the additional change of weights: $\lambda_1 W_{ij} + \lambda_2 W_{ij}(W_{ij}^2 - 1)(3W_{ij}^2 - 1)$. This approach may also be justified from the Bayesian point of view [8]. The cost function specifies our prior knowledge about the probability distribution $P(W|M)$ of the weights in our model M. Since we model a network for classification tasks and expect crisp logical decision the prior probability of the weight values is:

$$P(W|M) = Z(\alpha)^{-1} e^{-\alpha E(W|M)} \propto \left(\prod_{ij} e^{-\alpha_1 W_{ij}^2}\right)\left(\prod_{ij} e^{-\alpha_2 W_{ij}^2 (W_{ij}-1)^2 (W_{ij}+1)^2}\right) \qquad (2)$$

where the parameters α play similar role for probabilities as the parameters λ for the cost function. Prior knowledge about the problem may also be inserted directly into the network structure, defining initial conditions modified further in view of the incoming data. Since the final network structure becomes quite simple insertion of partially correct rules to be refined by the learning process is quite straightforward.

Although the constraints (1) do not change the MLP exactly into a logical network they are sufficient to facilitate logical interpretation of the final network function. MLPs are trained with these constraints and the slopes of sigmoidal functions are gradually increased to obtain sharp decision boundaries. The weights and thresholds of the resulting networks are analyzed and logical rules written down. Rules $\mathcal{R}_k$ implemented by trained networks are obtained in the form of logical conditions by considering contributions of inputs for each linguistic variable s, represented by a vector V_s. Contribution of variable s to the activation is equal to the dot product $V_s \cdot W_s$ of the subset W_s of the weight vector corresponding to V_s. A combination of linguistic variables activating the hidden neuron above the threshold is a logical rule in the form: $R = (s_1 \wedge \neg s_2 \wedge ... \wedge s_k)$.

In the **constructive version** of our approach training proceeds separately for each output class. One hidden neuron per class is created and training proceeds until the total cost function reaches minimum. 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 are correctly handled by the first group of neurons will not contribute to the error function, therefore the weights of these neurons are kept frozen during further training. This is equivalent to training one neuron at a time on the remaining data. Each time after minimum of the total cost function is achieved the weight vectors are 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 the first class. The output neuron for a given class is connected to the hidden neurons created for that class – in simple cases only one neuron may be sufficient to learn all instances, becoming an output neuron rather than a hidden neuron. Output neurons perform summation of the incoming signals. The same procedure is repeated for the remaining classes.

Each time only one neuron per class is trained, therefore the training is very fast. Since the first neuron for a given class is trained on all data for that class the rules it learns are most general, covering largest number of instances. Therefore 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.

III. Three examples

A. Iris data

In the first example the classical Iris dataset was used (all datasets were taken from the UCI machine learning repository [9]). 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). Analysis of the histograms of the individual features for each class provided the linguistic variables. For example, Iris-virginica class is more frequent for the value of x_3 above 4.9 and Iris-versicolor are more frequent below this value. Since the number of vectors per class is rather small discretization based on smoothed histograms was made (Fig. 1). This discretization leads to the following table for linguistic variables:

TABLE I
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]

After such discretization two iris-versicolor vectors become identical to some iris-virginica vectors and therefore cannot be classified correctly. These vectors were removed from the training sequence. Instead of four continuous inputs a network with 12 binary inputs equal to ± 1 (features present or absent) is constructed. For example, the medium value of a single feature is coded by $(-1,+1,-1)$ vector. For the Iris dataset a single neuron per one class was sufficient to train the network, therefore the final network structure (Fig. 2) has 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 constraint hyperparameters were increased from $\lambda = 0.001$ at the beginning of the training to $\lambda = 0.01 - 0.1$ near the end, with stronger enforcement of weight decay than integer weights. On average the network needed about 1000 epochs for convergence. The final weights are taken to be exactly ± 1 or 0 while the final value of the slopes of sigmoids reaches 300. The following weights and thresholds were obtained (only the signs of the weights are written):

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

Interpretation of these weights and the resulting network function (Fig 2) is very simple. Only two features, x_3 and x_4 are relevant and a single rule per class is found:

$$\begin{aligned} &\text{IF} \quad (x_3 = s \ \wedge \ x_4 = s) \text{ THEN iris-setosa} \\ &\text{IF} \quad (x_3 = m \wedge x_4 = m) \text{ THEN iris-versicolor} \\ &\text{IF} \quad (x_3 = l) \ \vee \ (x_4 = l) \text{ THEN iris-virginica} \end{aligned} \tag{3}$$

These rules allow for correct classification of 147 vectors, achieving 98.0% of accuracy. Replacing the iris-versicolor rule with the condition ELSE, and noting that for the iris-setosa rule one may remove one antecedent without changing classification results one gets just two rules with three antecedents:
IF $(x_3 = s)$ iris-setosa, IF $(x_3 = l \vee x_4 = l)$ iris-virginica, ELSE iris-versicolor.

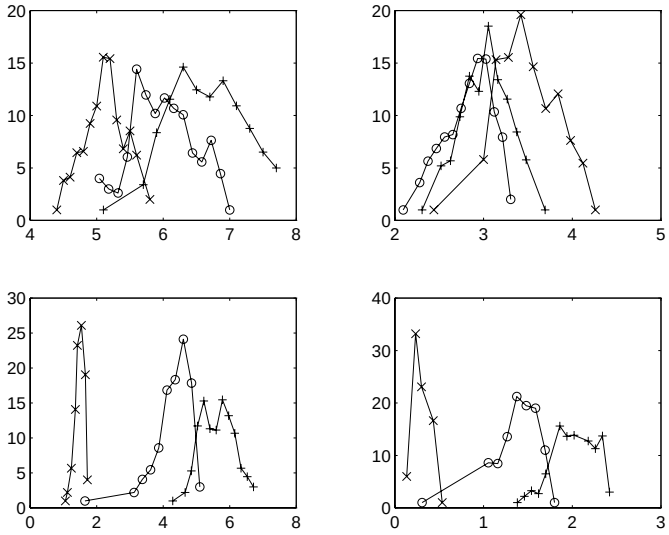

Fig. 1. Histograms of the four Iris features. The x_3, x_4 features (lower pictures) allow for better discrimination than the first two features.

Decreasing constraint parameters allows to replace one rule by four rules, with a total of three attributes and 11 antecedents, necessary to classify correctly a single additional vector, a clear indication that overfitting occurs. Increasing constraint hyperparameters further selects only one attribute, petal length x_3, and leaves two rules giving only 95.3% accuracy (7 errors): iris-setosa if $x_3 < 2.5$, iris-virginica if $x_3 > 4.9$, else iris-versicolor. This is the simplest description of the Iris dataset that we know of.

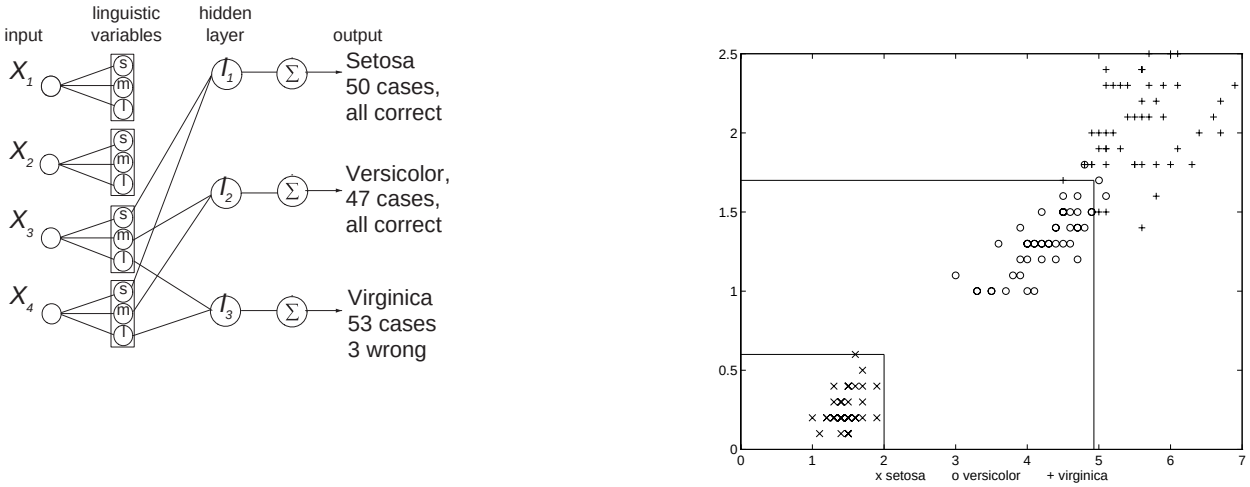

Fig. 2. Final structure of the network for the Iris problem and the decision borders in the space of the two relevant features x_3 and x_4.

B. Classification of mushrooms

In the mushroom problem [1], [9] the database consists of 8124 vectors, each with 22 discrete attributes, with 51.8% of the cases representing edible and the rest nonedible (mostly poisonous) mushrooms. A single neuron is capable of learning all the training samples (the problem is linearly separable), but the resulting network has many nonzero weights and is difficult to analyze from the logical point of view. One could use the subset algorithm for weight analysis [1], but the search tree grows exponentially with the number of "nonredundant" attributes describing objects, and for 22 attributes with up to 10 values each this is simply not feasible. Using constructive algorithm with the cost function Eq. 1 our algorithm has

discovered systematically the following disjunctive rules for poisonous mushrooms:

R_1) odor=¬(almond∨anise∨none), 120 poisonous cases missed, accuracy 98.52%
R_2) spore-print-color=green, 48 cases missed, accuracy 99.41%
R_3) gill-size=narrow∧stalk-surface-below-ring=scaly∧ odor=none∧ (stalk-surface-above-ring=silky ∨ population=clustered), 8 cases missed, accuracy 99.90%
R_4) population=clustered∧habitat=leaves∧cap-color=white∧ gill-spacing=¬close), all poisonous cases were correctly classified.

The first two rules are realized by one neuron. For large value of the weight-decay parameter only one rule with odor attribute is obtained, while for smaller hyperparameter values a second attribute (spore-print-color) is left. Adding a second neuron and training it on the remaining cases generates two additional rules, R_3 handling 40 cases and R_4 handling only 8 cases. The following single rule has been obtained for **edible** mushrooms, giving 48 errors, or 99.41% accuracy on the whole dataset: edible IF odor=(almond∨anise∨none)∧ spore-print-color=¬green This rule uses only two features and four antecedents. We have also tried to derive rules using only 10% of cases for training, achieving identical results. This is the simplest systematic logical description of the mushroom dataset that we know of, although some of these rules have probably been also found by RULEX and TREX algorithms [1].

Analysis of the graph representing possible contributions of the relevant attributes to the activation of neurons was done by a program written in C++. This program is useful for complicated weight analysis when the number of relevant attributes becomes large and paper-and-pencil analysis becomes too tedious. The accuracy of all the rules was checked automatically by a program written in Prolog. The mushroom problem illustrates also the capability of our algorithm to extract the dominant rules first and achieve 100% accuracy. This is done by decreasing the constraint hyperparameters, leading to more specific additional rules. Of course there is no need to obtain perfect classification on the training set: once the accuracy of classification on the test set starts to decrease the new rules handle only noise in the data.

C. The three monk problems

In each of the three monk problems one should determine whether an object described by six features is a monk or not [1]. The data for the Monk 3 problem is corrupted by adding 5% noise. "Being a monk" is defined by the following formulae in the three problems:
Monk 1: head shape = body shape ∨ jacket color = *red*
Monk 2: exactly two of the six features have their first values
Monk 3: ¬ (body shape = *octagon* ∨ jacket color = *blue*) ∨ (holding = *sward* ∧ jacket color = *green*)

Monk 1 problem. Two neurons were needed to learn all the training vectors identifying monks and select correct features. Unfortunately these neurons learned also to classify wrongly 5 other vectors as monks. The patterns which are not recognized properly should be treated as exceptions to the rules extracted from the network. To rectify this we have to extend the hidden layer adding neurons with a negative contribution to the output node. After the whole process is finished we have two separate sets of rules, one comprising information on positive examples, and the other describing exceptions. 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. The basic classification rules are applied to determine if the pattern belongs to a given class only if it does not belong to exceptions. For the Monks 1 problem one additional neuron handling exceptions has been generated, giving a total of 4 rules and one exception and classifying the data without any errors.

Monk 2 problem. The definition of this problem is very simple, but 15 logical rules are needed to describe it fully. Training in this case has generated 13 neurons, 8 of them handling rules and 5 handling exceptions to these rules. The four neurons added to the network in the final training stage are responsible for correct classification of just five examples. This shows how the neurons specialize in recognizing patterns which do not resemble other patterns. We extracted 16 rules and 8 exceptions from the resulting network. The number of atomic formulae which compose them is 132.

Monk 3 problem. In this problem two neurons handling rules and two neurons handling exceptions were generated. Although the training data for this problem has been corrupted it is still possible to obtain 100% accuracy [1]. Two neurons gave three rules, and the other two generated four exceptions. The whole logical system for this case contains 33 atomic formulae.

Some statistics concerning all the stages of the algorithm for the problems presented in this section

is shown in Table II. The first column specifies problem and gives the final numbers of generated rules and exceptions, the second enumerates particular stages of network expansion, the third gives the number of neurons trained simultaneously and the fourth informs whether the aim was searching for rules or exceptions (rules are printed in bold and exceptions in italic). The fifth column contains the numbers of instances classified properly thanks to rules generated at a given stage. The last column seems to confirm that the method learns the most common rules first.

TABLE II

STATISTICS FOR THE THREE MONK'S PROBLEMS.

Problem	Stage 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**

IV. SUMMARY

We have presented here a new approach to logical rule extraction based on the standard backpropagation technique with modified error function. Crisp logical rules are found automatically by analyzing networks trained with constraints that change MLPs into networks processing logical functions. Two versions of this approach have been presented, one aimed at simplification of typical MLPs and the other aimed at incremental construction of networks performing logical functions. The method of successive regularizations introduced by Ishikawa [10] has several features in common with our first approach and is capable of producing similar results. In this paper only the second, constructive method was disscussed in details since it requires less experimentation with various network structures. The constructive method has found the simplest logical description for the Iris and the mushroom test problems and shows a great promise as a general method for automatic rule extraction.

ACKNOWLEDGMENTS

Support by the Polish Committee for Scientific Research, grant 8T11F 00308, is gratefully acknowledged. W.D. is also grateful to the Heiwa Nakajima Foundation, Japan, for support, and to prof. Masumi Ishikawa for his hospitality at the Kyushu Institute of Technology.

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 8 (1995) 373–389

[2] S.M. Weiss, I. Kapouleas, "An empirical comparison of pattern recognition, neural nets and machine learning classification methods", in: J.W. Shavlik and T.G. Dietterich, Readings in Machine Learning, Morgan Kauffman Publ, CA 1990

[3] M.W. Craven, J. W. Shavlik, Extracting Tree-Structured Representations of Trained Networks, *Adv. in Neural Info. Processing* 8 (1996) 24–30

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

[5] 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.

[6] W. Duch, R. Adamczak, K. Grąbczewski, Constrained backpropagation for feature selection and extraction of logical rules, *Proc. of "Colloquiua in AI"*, Łódź, Poland 1996, p. xxx

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

[8] D.J. MacKay, "A practical Bayesian framework for backpropagation networks", Neural Computations 4 (1992) 448-472

[9] C.J. Mertz, P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/pub/machine-learning-databases.

[10] M. Ishikawa, "Rule extraction by succesive regularization", in: Proc. of the 1996 IEEE ICNN, Washington, June 1996, pp. 1139–1143.