Constrained backpropagation for feature selection and extraction of logical rules

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

A new architecture and method for feature selection and extraction of logical rules from neural networks trained with backpropagation algorithm is presented. The network consists of nodes that discover linguistic features and nodes that discover logical rules. Most weights are constrained to ± 1 or zero values. The relevant input features are automatically generated and selected by the network. Rules are generated consecutively, from the most general, covering many training examples, to the most specific, covering exceptions only. Automatic weight pruning ensures that a minimal number of logical rules is found. Results for the Iris classification problem illustrate the efficiency of this method.

Keywords

Neural networks, MLP, backpropagation, logical rule extraction, feature selection, Iris dataset.

I. Introduction

SOME classification problems have an inherent logical structure. In such cases it may be preferable to use logical rules instead of adaptive classifiers. Logical reasoning is more acceptable to human users than the recommendations given by black box adaptive classification systems [1]. If the set of logical rules is relatively small the explanation of the data structure that these rules offer is perceived to be satisfactory. 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. Therefore it is important to search for good methods of the extraction of logical rules from the preclassified training data (supervised learning problem) as well as from the data that has unknown structure of classes (unsuperivised training problem). Such methods should also determine relevant input information, i.e. select relevant input features. These problems have so far eluded satisfactory solution (for a review and further references see [1]). In this paper a new method for rule extraction and feature selection for the supervised classification case is presented.

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

$$\begin{aligned} \text{IF} \quad & (x_1 \in \mathcal{X}_1 \wedge x_2 \in \mathcal{X}_2 \wedge \ldots x_N \in \mathcal{X}_N) \\ \text{THEN} \quad & (y_1 \in \mathcal{Y}_1 \wedge y_2 \in \mathcal{Y}_2 \wedge \ldots y_M \in \mathcal{Y}_M) \end{aligned} \tag{1}$$

First Polish Conference on Theory and Applications of Artificial Intelligence, Łódź, December 1996

or simply IF ($X \in \mathcal{X}$) THEN $Fact_k = TRUE$. 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 for the description of classes $\mathcal{X}$, such as the Feature Space Mapping (FSM) network [4], may be treated as neurofuzzy systems using more complex membership functions. Since not all classes $\mathcal{X}$ may be described by a single factorizable function fuzzy logic approach is not so flexible as more general density-based approaches such as FSM [4]. Fuzzy logic requires "granularization" of class descriptions, i.e. summation of many products of membership functions to describe each class. Crisp logic imposes even more severe restrictions by admitting only combinations of step functions for feature description.

In our previous paper [3] we have found an interesting way to obtain logical rules from MLP neural networks, by far the most popular and successful of the neural network models. Our algorithm for extraction of logical rules works very well for discreet input values, or in the cases when continuous inputs are discretized by the user. This is not quite satisfactory since for some problems, especially when the number of classes is large, such discretization may be hard to find. In this paper we present quite novel solution to the problem, taking care of the rule extraction as well as the automatic feature determination and selection. The algorithm is presented in the next section and illustrated in the third section. The paper is finished with a short discussion.

II. The feature and rule extraction algorithm

The problem of extracting rules from neural networks has a natural geometrical interpretation. Crisp logical rules correspond to a division of the input space with perpendicular hyperplanes into cuboidal areas that are identified with classes or facts, while each division of the input variable is given a symbolic name and is known as a linguistic variable. If the classes in the input space may be correctly separated with the 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 variables and rules necessary for good description may become unacceptably large. Although fuzzy logic or clusterization techniques may offer a better approximation with smaller number of rules in this paper we shall be concerned only with the crisp logical rules.

Continuous valued inputs create a rather difficult problem for most methods of rule extraction. While discreet variables may be used directly in crisp logical rules, continuous inputs have to be quantized first. In some cases [3] it may be sufficient to divide the whole range of data into several sections, 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. In other cases a natural division of a given data attribute into discreet values is suggested by analysis of histograms. Such an analysis may become tedious and is not always done in an optimal way by humans. Therefore we propose special neural units, further called L-units, that should automatically analyze continuous inputs and produce linguistic variables.

The basic scheme of such units is shown in Figure 1. An input x_i is connected via two weights fixed to $+1$ weights to two neurons, each with it own separate bias, b_i and b'_i. The transfer functions of these neurons are of the sigmoidal type. At the end of training these sigmoids should become very steep, although at the beginning they may be quite smooth, allowing for fuzzy approximation of features. The two hidden neurons of L-unit are connected to the one output neuron using weights S_1, S_2 with values constrained to $0, \pm 1$. These constraints should be satisfied only at the end of the training. At the beginning of

training the whole network may be treated as an MLP network with specific architecture. The two input and one output neuron form together one logical unit designated as L. Such units, trained simultaneously with other units of the network, should discover linguistic variables. Since L-units have only one input, one output and two constrained weights as parameters functions realized by these units belong to one of the four types shown in the limit of large gain in Figure 1. The first of these functions is obtained as a difference of two sigmoids and represents a typical linguistic variable v_k designating the fact that $b_i \leq x_i \leq b'_i$, the second denotes negation of such variable. The other two functions, with only one non-zero weight (or two equal weights) are the unbounded (nonlocal) versions representing simpler inequalities of the form $b \leq x_i$ or $x_i \leq b$, where b is one of the biases of the two hidden neurons of given L-unit - which one to choose depends on the bias of the output neuron of the L-unit and the weights S_1, S_2. In practice if S_1 and S_2 are equal then both biases of hidden neurons have the same value. The borders b_i and b'_i defining linguistic variables and the two constrained weights and the bias of the output neuron are treated as adaptive parameters of our network.

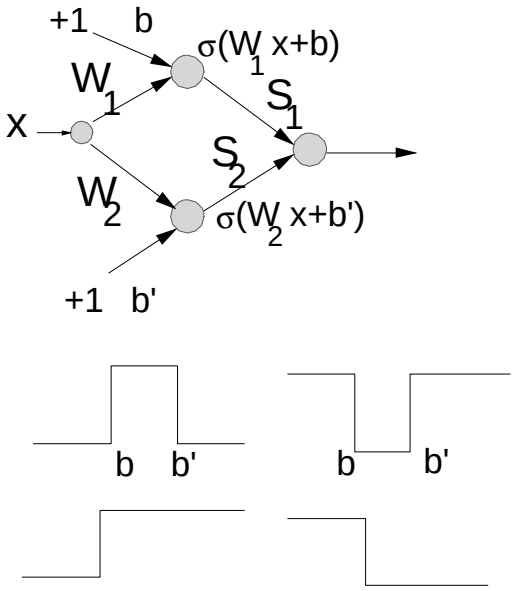

Fig. 1. Construction of a linguistic unit converting continuous inputs to linguistic variables.

Discrete variables taking many values are treated as continuous, but if they take only a few values than they may be treated as linguistic variables without any preprocessing - such inputs should be directly attached to the R-units.

An alternative adaptive way to obtain crisp decision regions is to use product of pairs of sigmoidal functions [5] instead of the linear combinations of sigmoidal functions:

$$s(X; B, B') = \prod_i \sigma(x_i - b_i)(1 - \sigma(x_i + b'_i)) \tag{2}$$

and slowly increase the gain of the sigmoidal functions $\sigma(x)$ during learning.

Having defined such input preprocessings we are ready to train the rule layer. Functions realized by L-units are defined in one dimension and in the hard limit they are step functions. To obtain multidimensional description of classes these functions are added or

subtructed from each other by the rule nodes. For simplicity of interpretation we will constrain the weights between L units and R or rule units (at the end of learning) also to $0, \pm 1$ values, although by softening of this condition we could obtain a fuzzy classification rules. Thus each neuron in the rule layer creates multidimensional combination of linguistic variables corresponding to class description. The thresholds of the rule neurons are adaptive parameters and their outputs are calculated using sigmoidal transfer functions. The threshold determines the type of function, and thus the type of rule, that the node embodies. Logical rules are obtained directly from graphical analysis of the trained network weights, in the same way as it was done in [3].

It is obvious that for one attribute several linguistic variables may be needed, therefore several L units for one input should be created. Unfortunately it is impossible to predict how many units would be needed. Since the complexity of the network is penalized by enforcing large number of zero weights one may start with spurious L-nodes and hope that if they are not needed their weights will vanish and the units will be dropped. More frequent result is that several L-units learn the same interval, which may seem confusing, but in fact it is very easy to identify and eliminate them during the rule extraction stage. One may also start with a smaller number of units and increase their number until the error drops to a satisfactory value.

The success of this method depends on our ability to find a global minimum of the classification error, corresponding to a set of simplest logical rules for classification. Since the method is just a simple modification of the standard backpropagation technique we do not need to derive the learning algorithm. The only modification is an additional term in the error function:

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

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:

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

where λ scales the relative importance of auxiliary conditions. An alternative form that could be used is:

$$E_a(W) = \lambda_1 \sum_{i,j} W_{ij}^2 + \lambda_2 \sum_{i,j} (W_{ij}-1)^2 + \lambda_2 \sum_{i,j} (W_{ij}+1)^2 \tag{5}$$

This form has two advantages: independent parameters control enforcing of zero and ± 1 weights, and an interpretation of this function from the Bayesian point of view [6] is straightforward. It defines our prior knowledge about the probability distribution $P(W|M)$ of the weights in our model M. Since we model a network that, trained on classification tasks, should give crisp logical decision we expect answers "yes", "no" or "irrelevant", therefore:

$$P(W|M) = Z(\alpha)^{-1} e^{-\alpha E_a(W|M)} \propto \left(\prod_{ij} e^{-\alpha_1 W_{ij}^2}\right) \left(\prod_{ij} e^{-\alpha_2 (W_{ij}-1)^2}\right) \left(\prod_{ij} e^{-\alpha_2 (W_{ij}+1)^2}\right) \quad (6)$$

Prior knowledge that we have about the problem may also be inserted directly into the network structure, defining initial conditions modified further in view of the incoming data. For example, instead of starting from completely random set of the network parameters we may start from parameters for our linguistic units estimated from the initial histogram analysis, allowing these units to optimize the final divisions of input features into linguistic variables. Since our network structure is so simple insertion of partially correct rules to be refined by the learning process is quite straightforward.

The easiest and fastest way of training would proceed separately for each output class. The fact that all the classes contribute to discretizing the input space makes training all the classes simultaneously more reasonable. Although the method works with general multilayered backpropagation networks we recommend to start with the following, simplified constructive procedure that frequently leads to satisfactory solutions. One hidden neuron for each class is created and 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 for a given class. 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. It implies that we have to freeze the generated L-units as well. If we are not satisfied with the accuracy of such a network we add second neurons for the classes which should be learned more precisely. After convergence the second weight vector is analyzed and corresponding rules found. If there are some exceptions from the extracted rules we catch them by adding neurons connected to the output neuron with the -1 weight and proceed as before. This procedure is repeated until all data are correctly classified, weights analyzed and a set of rules $\{R_1, R_2, \ldots, R_n\}$ is found, identifying $Class_1$. If the number of rules is too large or the rules found cover only a small number of training cases early stopping of the training is recommended to avoid overfitting or creating rules that classify noise in the data. The output neuron for the first class is connected to all hidden neurons created for that class. The output neuron performs a simple summation of the incoming signals (Fig. 2, where only one output neuron is shown).

An alternative procedure is to start with some network structure, randomize parameters and train the network until some weights become small enough to be removed and the structure is simplified. Although such regularization procedure may in some cases lead to similar results [7] it remains to be seen in complex classification problems which procedure is easier to use.

It should be clear that the network could have more than one rule layer and it is interesting to check if complex rules may be discovered this way. For example, Ishikawa [8] has shown that a logical function of the form $(a \vee b) \wedge (c \vee d)$ is discovered by a two hidden-layer network in its factorized form, so that only three neurons are left in the network, while if we allow only a single layer of rule neurons we may need four neurons, one for each of the four terms ($a \wedge c$ to $b \wedge d$). Therefore it may be worth experimenting with multilayered architectures, although if our goal is to find the simplest logical description of a given data we may achieve it by factorizing the rules in a final stage using an algorithmic program

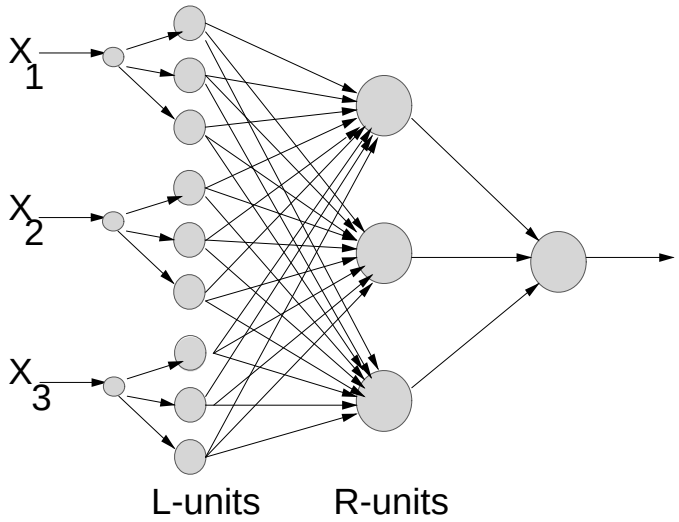

Fig. 2. Structure of the network for rule extraction and feature selection. L-units convert continuous inputs into linguistic variables, R-units form the rules.

similar to the simplifiers used in symbolic algebra packages.

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 of the form:

$$\text{IF } \; (s_1 \wedge \neg s_2 \wedge ... \wedge s_k) \; \text{ THEN } Class1 \tag{7}$$

The rules obtained by our algorithm are discovered in order of their importance, i.e. those that are used most often (classify more cases) are found first while rules that handle only a few cases are found at the end of the training. This ordering of rules allows for a very easy check of the quality of a set of rules by looking at the errors on the test data. An optimal balance between the number of rules and the generalization error is usually obtained when only the rules that cover larger number of cases are retained. 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 $+1$.

The L-layer takes as input continuous vectors $X^{(p)} = (x_1^{(p)}, ...x_N^{(p)})$ and gives as output a vector of linguistic variables $\mathbf{L}^{(p)} = (l_1^{(p)}, ...x_K^{(p)})$. Since this mapping is not one-to-one it may happen that two or more input vectors belonging to different classes are mapped to the same vector $\mathbf{L}^{(p)}$. This leads to classification errors that the R-units are not able to remove. If the network is not able to discover better features that prevent this kind of errors it may be worthwhile to explicitly force the distinguishability of all input vectors to avoid such situation. To obtain better discretization we may define the error function as:

$$E(B, B') = \sum_{p,p'} \delta\left(\mathbf{L}^{(p)}, \mathbf{L}^{(p')}\Big(1 - \delta\left(C(\mathbf{L}^{(p)}), C(\mathbf{L}^{(p')})\right)\right) \tag{8}$$

where $C(\mathbf{L}^{(p)})$ is the class the $X^{(p)}$ vector belongs to. To avoid problems with discontinuous δ functions we can replace them with a very narrow gaussian distributions. This leads to an untypical network structure and formally requires rather special training in which the output node receives, after each pattern presented, all other patterns for comparison, but there is no problem with implementations of such a scheme. Such explicit conditions enforcing distinguishability may be desirable, but may also lead to creation of too many linguistic variables that allow to handle noise in the data.

III. An example

As an example we will illustrate our algorithm on the classical Iris dataset. The data has 150 vectors evenly distributed in three classes, called iris-setosa, iris-versicolor and iris-virginica. Each vector has four continuous features: sepal length x_1 and width x_2, and petal length x_3 and width x_4 (all given in cm). In our previous paper 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 was called small if it was in $[4.3, 5.5]$ range, medium if in $(5.5, 6.7]$ and large if in $(6.7, 7.9]$. Such a division is close to the optimal from the point of view of linguistic variable selection and classification accuracy. Although analyzing one-dimensional histograms different division is obtained for the first two variables almost identical linguistic variables are found for the last two inputs.

Will our network produce similar linguistic variables as the ones defined above? Since for the final classification only inputs 3 and 4 are important we may expect that the weights to all L-units for the first two inputs should tend to zero while the linguistic variables for the last two inputs should resemble the one given above. To give the network a chance to discover similar features at least three linguistic units per input should be defined at the beginning. The simplest initialization for the B, B' parameters would be based on dividing the input data ranges into three parts. Since this is close to optimal we may expect in this case only small changes in these parameters.

For the Iris dataset a single neuron per one class was sufficient to train the network, therefore the final network is the network we started with. Each feature got three L-units to discretize its possible values. Three hidden neurons realized the three iris classes. In fact the hidden layer is redundant in this case, as it is necessary only if there is a class realized by more than one neuron (i.e. the first stage does not give satisfactory solutions and we have to use additional neurons to learn special cases or exceptions).

In the trained network only two of the twelve L-units are relevant (see Figure 3). One is used to distinguish the iris-setosa class, the other can recognize iris-virginica. If none of the two L-units gets activated the network points out the iris-versicolor class. The training process caused the network to minimize the effort by learning from examples just two of the three classes. The third class can be described by the word "otherwise".

Rule extraction from the simple network structure obtained above gave the following results:

Iris-setosa: $x_3 \leq 2.56$

Iris-virginica: $x_4 > 1.63$

Iris-versicolor: otherwise

These rules allow for correct classification of all but six vectors, achieving 96% of accuracy. Better accuracy could be obtained by adding more neurons to catch the exceptions. However, if good degree of generalization is our major aim the results presented here may

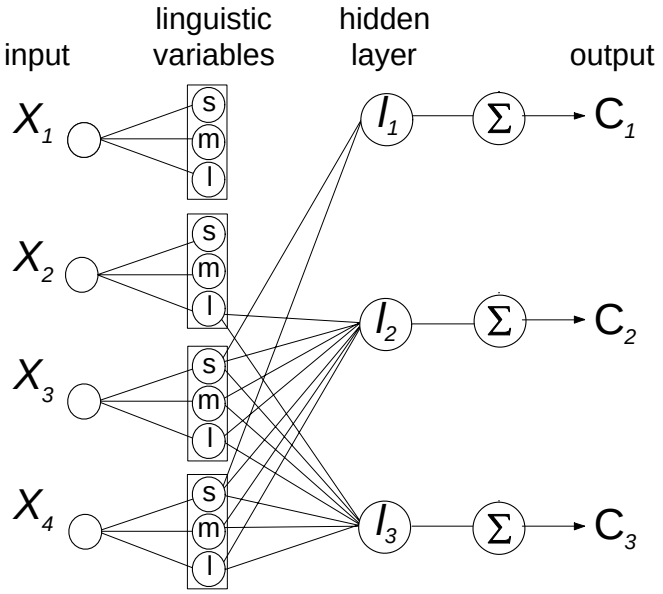

Fig. 3. Final structure of the network for the Iris problem.

be perfect - most simple and clear rules have been obtained (cf. [10]). The validity of the rules given above was confirmed with a Prolog program.

IV. Discussion and summary

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 automatically forms the local response units, similarly to the Constrained Error Backpropagation (CEBP) and RULEX algorithm of Andres and Geva [11], who use the local response units and constraints the output weights. In contrast to their approach our network has separate subnetworks, the L-units, devoted to linguistic variable extraction, and by enforcing zero and ± 1 weights takes the simplest possible structure. We do expect that in more complex problems it should give results of similar quality as we have already obtained [3] for the mushroom and Monk problems. Judging from this initial results we hope to analyze several real-world databases and obtain the simplest possible logical description of these databases. Similar results may also be obtained using the method of successive regularization introduced by Ishikawa [7]) and sharing with our method the idea of simplifying the network structure by enforcing zero weights, but missing automatic selection of linguistic features and simplifications due to weight constrains. We should report on a detail comparison of the three methods (ours, RULEX and successive regularization) in the near future.

Acknowledgments

This research has been partially supported by the Polish Committee for Scientific Research, grant 8T11F 00308. W.D. is grateful to prof. Masumi Ishikawa for numerous discussions and his hospitality at the Kyushu Institute of Technology, and to the Heiwa Nakajima Foundation that provided the funds for his visit there.

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] Włodzisław Duch, Rafał Adamczak and Krzysztof Grąbczewski, "Extraction of logical rules from neural networks", IEEE Transactions on Neural Networks (submitted)

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

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

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

[8] M. Ishikawa, "Learning of modular structured networks", Artificial Intelligence 75 (1995) 51-62

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

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

[11] R. Andrews, S. Geva, "Rule extraction from a constraint back propagation MLP", Proc. 5th Australian conference on Neural Networks, Brisbane, Queensland (1994), 9-12