Search-based Training for Logical Rule Extraction by Multilayer Perceptron

Włodzisław Duch
School of Computer Engineering
Nanyang Technological University
N4, Nanyang Avenue, Singapore 639798
and Department of Computer Methods
Nicholas Copernicus University
Grudziądzka 5, 87-100 Toruń, Poland
www.phys.uni.torun.pl/~duch

Mirosław Kordos
Institute of Computer Science
The Silesian University of Technology
Akademicka 16, 44-100 Gliwice, Poland

Abstract. **Search-based non-gradient training techniques are used to train an MLP-like neural network with quantized parameters. The network training is quite fast and the final network function is converted to crisp or fuzzy logical rules using a simple analysis of its weights. Various modifications of the method are presented, each generating a specific form of rules. Depending on the desired information one of the methods can be chosen. Feature selection and data discretization are automatically performed.**

***Index Terms*— quantized weights, neural networks, rule extraction, search algorithms**

I. INTRODUCTION

A good strategy in data mining is to extract simplest crisp logical rules first. They provide hyperrectangular decision borders in the feature space. This approximation may not be sufficient if complex decision borders are required, but it works quite well if the problem has an inherent logical structure. For many datasets crisp logical rules proved to be highly accurate, they are easy to understand by experts in a given domain, and they may expose problems with the data itself [1].

A general approach to classification and extraction of logical rules is proposed in this paper. The acronym of this approach, SMLP, may be interpreted either as "search-based MLP" or “simplified MLP”. The advantages of MLP neural networks are combined with rule based systems, allowing for extraction of simple logical rules. Instead of the gradient-based methods that run into problems for discontinuous, step-like transfer functions, the training algorithm is based on search methods. It leads to simplified network structures, with few connections between hidden and output layer. Various SMLP structures, trainings, and rule extraction algorithms are considered. Several sets of rules of similar accuracy may be generated, offering different advantages to domain experts. The search-based training methods have also been successfully tested with fully connected MLP networks [2], but this aspect is not discussed in this paper.

II. SMLP NETWORK

The simplest version of SMLP networks is based on a 3-layer MLP architecture. Neurons implement sigmoidal or step output functions with scalar product activation. The network requires discrete input data. If the data is continuous, it must be discretized prior to the training, or at the run-time by an additional network layer.

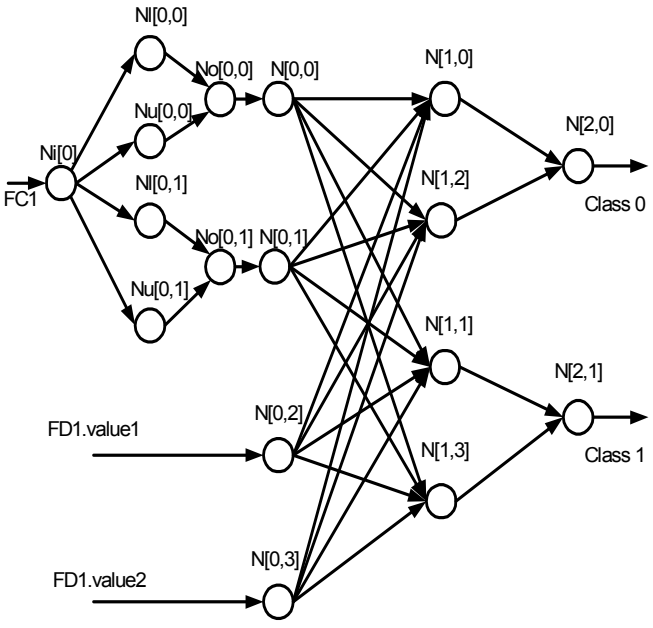

Fig. 1. SMLP Network Diagram, with some pre-processing L-units shown for FC1 feature

A separate input neuron for each discretized feature value is used. Thus the number of all input neurons equals the sum

of all distinct values for all features. The input values are 1 if the feature has the value represented by this neuron and 0 otherwise.

One hidden neuron per class is created at the beginning. The second hidden neuron per class is added, if the results with only one neuron are not satisfactionary, and then the whole network is re-trained or only the added neuron is trained. If the results are still unsatisfactory then the next hidden neuron is added. The number of hidden neurons per class should equal the number of data simplexes. The hidden layer performs M-of-N operation, which in most cases can be reduced to AND operation.

There is one output neuron per class that combines the partial rules given by hidden neurons for a given class (OR operation). The biases and weights of output neurons are constant (bias = ±0.5 each weight = ±1).

The network diagram is shown in Fig.1. Each value of a discrete feature (FD1.value1, FD2.value2) is given to a different input neuron. Continuous features (FC1) are discretized by logical units (L-units), as described in [1] and signals from L-units are given to the input neurons.

Only weights and biases of the hidden neurons are optimized. The weights can take integer values (only –1, 0, 1 if the transfer functions are step). The biases can take an integer –0.5 values (–0.5, 0.5, 1.5, 2.5,…., number_of_features+0.5).

At the beginning of the training all hidden neurons weights have the value of zero and biases of 0.5, so no data is assigned to any class. The value of 1 is added or subtracted from a single weight and bias. If the MSE error decreases then the change is kept, otherwise it is rejected. Then the value of 1 is added or subtracted from the next weight and again the error is calculated, until the change of all weights and biases in the hidden layer is examined. In some cases changing only one parameter at a time may not be sufficient for the algorithm to converge. Then modifying two or more parameters at a time can be used, though it is more time consuming.

The error can be calculated on the whole training set or, to speed up the operation, on a randomly chosen (different each time the error is calculated) subset of the training set. The signal of each output neuron should be 1 if the vector belongs to the class represented by this neuron, and 0 otherwise. We consider a vector to be classified correctly if the signal of the output neuron assigned to its class is higher than any other output neuron signal.

III. Rule Extraction and Feature Discretization

In general the hidden neurons give M-of-N rules (if M assumptions out of N are satisfied then the condition is true). If the sum of all inputs of a hidden neuron exceeds its bias a logical rule in the hidden layer is generated. In practice, after the training, biases often take the value equal to the sum of the incoming weights minus 0.5. Thus all N assumptions must be satisfied and M-of-N operations are reduced to AND operations. The output layer performs OR operation, combining rule conditions into final rules. This gives very plain and comprehensive rules. If a presence of a given value contributes to a given class, the hidden neuron weight will be positive. If the absence - then negative. If the value is irrelevant to this class then the weight is zero.

There are two objectives while discretizing continuous data: to have a few discrete values, to obtain a simple networks and simple rules, and to have enough discrete values for accurate rules and a reliable classification. Two discretization models are presented.

A. Prior to Training Discretization Based on Histogram Analysis

Initially each continuous feature space is divided into n equal width or equal frequency intervals (n = 10 is sufficient in most cases). Then each interval is assigned to the class to which the majority of its vectors belongs. Then adjacent intervals assigned to the same class are joined. If points from different classes overlap in many segments then that feature does not provide us with any information and is eliminated. This simplifies the rules significantly and does not influence classification accuracy.

This is the simplest and the quickest discretization method. It is tried as first and only if the results are inadequate it is followed by L-unit based discretization. More advanced discretization techniques may also be used, but so far have not been attempted [3].

C. Run-time L-unit Based Discretization

This discretization has been used in MLP2LN networks [1]. It is performed using a combination of three neurons, called L-unit, with frozen weights but adaptable biases. Since discretization and learning are done in the same network, results depend on the whole training set, not just on the single feature being discretized.

The initial interval boundaries obtained from some prior-to-training discretizations may be tuned using search techniques. An interval cut-off point in the most significant feature is shifted and the training is performed. If the error decreases then the shift was in a proper direction, otherwise in the wrong direction. The procedure may be repeated with each interval boundary point for all features. Features that are useless for discrimination of a given class are automatically removed.

IV. Training Methods and Their Influence on the Rules

First the training algorithm changes one weight at a time. If this does not work - then two weights are changed at a time. Changing more than three parameters is rather costly. We have also tested and successfully applied update of many parameters using genetic algorithms, which becomes more effective if changing many parameters at a time is required, but we do not report this here. However, for the real-world data sets changing two parameters at a time is usually sufficient.

In changing *n* parameters at a time, the order in which the weights are examined plays a role. It is an undesirable effect,

because the extracted rules depend on the training process in an unforeseen way. This problem is solved using feature selection based on the information included either in the single feature, or in the single feature value. The algorithm assesses the amount of information contained within a single feature (or a singe value), or jointly in two or more features (or values not necessarily within the same feature).

Searching first through values of a single feature is advantageous because it usually leads to the simplest and most comprehensive rules. Since this approach is not universal (e.g. it does not solve the XOR problem) also search through the feature values of vectors from one class can be performed. These searches usually produce more comprehensive rules then searching simultaneously throughout all feature values. Best First Search (BFS) or Beam Search (BES) search strategy may be used in all cases.

Weights of neurons that have already been trained may be frozen, minimizing calculation time and leading frequently to better results, since it corresponds to incremental learning, decomposing the task into learning general rules and than exception to these rules instead of trying to modify all rules to fit the data.

With biases of output neurons set to +0.5 the rules are positive - they express which conditions must be satisfied when a vector belongs to a given class (example: class 0 if petal-length < 3). If the biases are –0.5 then the rules are negative - they express which conditions must not be satisfied when a vector belongs to a given class (example: class 0 if not 3 <petal-length < 4.9 and not petal-length > 4.9).

V. Step Versus Sigmoidal Transfer Function

In most cases step transfer functions are used for logical rule extraction. In comparison with sigmoidal functions they produce simpler rules usually of the same, and sometimes even higher accuracy. Step functions give only information absolutely necessary to classify a vector. With step functions when a vector is classified - the error already equals zero and no additional incoming conditions can decrease it, so they do not come into the final rule.

Sigmoidal functions give also information about other feature values, specific to a given class but not required by the classification process. With sigmoidal functions adding more conditions to a rule may still decrease the MSE error, since the output signal is less than 1 and always can be increased. Moreover the number of additional conditions of the rule may be regulated by the required output accuracy, assuming that output values above some threshold are considered as 1.

On the Iris data trained with one feature only step transfer functions give:
Class 0 if petal-length < 3
Rules obtained with sigmoidal transfer functions have two conditions:
Class 0 if petal-length < 3 and not 3 < petal-length < 4.9

VI. Window Mechanism and Fuzzy Rules

The window function can be realized by three neural nodes (perceptrons), as shown in Fig. 1. Neurons Nl and Nu have step transfer functions, their output signal can take value –1 or 1. Ni has a linear transfer function. No has step transfer function with possible outputs 0 and 1. The original continuous signal is given to the Ni input. The discrete output signal from No is passed to the corresponding input neuron of an SMLP network. Weights and biases have the following values: Nl.weight = 1, Nu.weight = 1, Nl.bias = lower limit of the input signal, Nu.bias = upper limit of the input signal, No.weight(Nl) = 1, No.weight(Nu) = –1, No.bias = 1.5. No.signal = 1 only if Ni.bias < input signal < Nu.bias. If the input signal should be bounded only from one side then Nl.output or a Nu.output is given to the input layer neuron.

Fig. 2 shows a two-dimensional projection of the Appendicitis data. Some areas can be assigned only to one class. Other areas contain points that can be assigned to two classes. Fuzzy rules can describe points in areas where crisp rules overlap. The value of the membership function of such a point can be proportional either to the probability density for a given class in this area, or to the distance from that point to the decision border.

VII. Examples of Rules Extracted from some Benchmark Data Sets

We include some examples of rules extracted from UCI databases using the SMLP network using 10-fold crossvalidation.

The rule set for Iris data (150 samples, 50 of class 0, 50 of class 1 and 50 of class 2). Rules were obtained with one hidden neuron per class, changing one parameter at a time, with prior to training discretization based on histogram analysis. Accuracy of the rules below is 96.0%

Class 0 if petal-length < 3.0
Class 1 if 3.0 < petal-length < 4.9 ∧ 0.9 < petal-width < 1.7
Class 2 if 4.9 < petal-length ∨ 1.7 < petal-width

Rule sets for Appendicitis data (small medical data set, 106 samples, 21 of class 0 and 85 of class 1) were also obtained with 1 hidden neuron per class. Changing one or two parameters at a time with prior to training discretization based on histogram analysis followed by run-time L-unit based discretization to adjust precisely interval cut-off points gave:

Rule 1: class 0 if mnea<6700 ∧ mbap<11 else class 1 (91.5%)
Rule 2: class 0 if hnea<5600 else class 1 (89.6%)
Rule 3: class 0 if mnea<6700 else class 1 (89.6%)
Rule 4: class 0 if (hnea<5600 ∧ mnea<6700)
class 1 if (hnea>5600 ∧ mnea>6700)
P(class 0) = P(class 1)=0.5 if (hnea<5600 xor mnea<6700) (89.2%)

Although the total accuracy of Rule 4 is in this case slightly lower than that of Rule 2 and 3, it probably better describes the properties of this set, providing more information about the structure of data, as can be seen in Fig. 2.

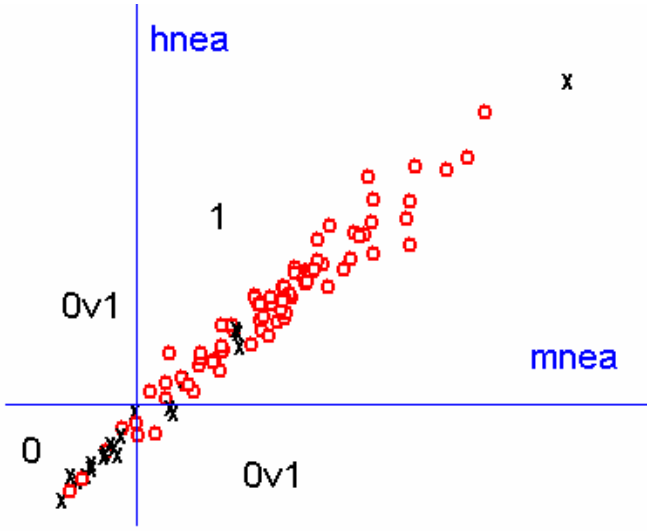

Fig. 2. Appendicitis data with decision borders, projection into two features

Rule sets for Ljubljana Breast Cancer data (medical data set: 286 samples, 202 of class 0, 84 of class 1) were obtained with 3 hidden neurons per class, changing 2 parameters at a time. The original data set has already discrete values, dividing continuous values into 9 bins for age, 13 bins for the number of nodes involved and 3 for the degree of malignancy.

Class 0 if (involved nodes ∉ (0,2) ∧ degree malignant = 3 and tumor size = 45-49 ∧ age ∉ 10-19) ∨
(involved nodes = (9-11) ∧ age ∉ (40-49)) or (tumor size ∈ (35-39) ∧ age ∈ (30-39)) else Class 1

These rules have already overfitted the data, accounting for accidental correlations rather then important factors. The best methods reduce the error from 29.7% (default) by a few percent. In this case a simpler solution is required [1], the rules clearly expose a problem with the reliability of this data.

Table 1. 10-fold cross-validation results for Appendicitis and Ljubljana Breast Cancer data

Method	Accuracy (%) for Appendicitis data	Accuracy (%) for Breast Cancer data
SMLP	89.6	78.3
C-MLP2LN [4]	89.6	78.0
MLP +BP [4]	83.9	71.5
CART [5]	84.9	77.3
C4.5 [5]	84.9	76.9
Bayes Rule [2]	83.0	75.9
FSM [4]	84.9	71.6
AQ15 [6]	-	73.5
Default	80.2	70.3

VIII. Conclusions

A neural network approach to classification and rule extraction, called SMLP was proposed. The model combines the advantages of MLP neural networks with the possibility of extracting simple rules in a comprehensive way. The training model is much simpler than gradient-based algorithms. Due to the perceptron properties, the rules given by hidden neurons are in the M-of-N form. Since the prepositional form of logical rules is usually preferred, M-of-N rules are reduced to AND + OR operations if possible.

As the experiments showed, the accuracy of results on the popular benchmark data sets is comparable with the best results obtained from other methods, while the network and rules are simpler and the training process is quicker. It cannot be said that the only criterion of the rule quality is the classification accuracy either using crossvalidation or a separate test set. Sometimes rules which are simpler, or which reflect data structure better, may be preferred, although their accuracy is lower. It is possible to obtain several sets of rules by the modification of network parameters and training process. A set of SMLP networks can be built to give users the possibility of choosing rules that are most suitable for their purpose.

Although many issues require further investigation this search-based approach has some potential that seems to be largely unexplored.

References

[1] Duch W, Adamczak R, Grąbczewski K. (2001): A new methodology of extraction, optimization and application of crisp and fuzzy logical rules, IEEE Transactions on Neural Networks **12**, 277-306

[2] Duch W, Adamczak R, Grąbczewski K. (1999): Searching for optimal MLP. *4th Conf. on Neural Networks and their Applications*, Zakopane, Poland, pp. 65-70

[3] Hussain F, Liu H, Tan C.L, Dash M. (1999): Discretization: an enabling technique. Technical Report TRC6/99, School of Computing, National University of Singapore.

[4] Adamczak R. (2001): Application of neural networks to classificaiton of experimental data, PhD Thesis (in Polish), Nicholaus Copernicus University, Torun.

[5] Weiss S.M, Kapouleas I. (1990): An empirical comparition of pasttern recognition, neural nets and machine learning classification methods, In: J.W. Shavlik and T.G. Detterisch, *Readings in Machine Learning,* Morgan Kauffmann Publ, CA.

[6] Michalski R.S, Mozitec I, Hong J, Lavrac N. (1986): The Multipurpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. In: Proc 5th Intern. Conf. on AI, pp.1041-1045, Philadelphia, PA: Morgan Kauffmann