# A New Methodology of Extraction, Optimization and Application of Crisp and Fuzzy Logical Rules

Włodzisław Duch, *Member, IEEE,* Rafał Adamczak, and Krzysztof Grąbczewski

***Abstract*—A new methodology of extraction, optimization, and application of sets of logical rules is described. Neural networks are used for initial rule extraction, local, or global minimization procedures for optimization, and Gaussian uncertainties of measurements are assumed during application of logical rules. Algorithms for extraction of logical rules from data with real-valued features require determination of linguistic variables or membership functions. Context-dependent membership functions for crisp and fuzzy linguistic variables are introduced and methods of their determination described. Several neural and machine learning methods of logical rule extraction generating initial rules are described, based on constrained multilayer perceptron, networks with localized transfer functions or on separability criteria for determination of linguistic variables. A tradeoff between accuracy/simplicity is explored at the rule extraction stage and between rejection/error level at the optimization stage. Gaussian uncertainties of measurements are assumed during application of crisp logical rules, leading to "soft trapezoidal" membership functions and allowing to optimize the linguistic variables using gradient procedures. Numerous applications of this methodology to benchmark and real-life problems are reported and very simple crisp logical rules for many datasets provided.**

***Index Terms*—Backpropagation, data mining, decision trees, feature selection, fuzzy systems, logical rule-based systems, neural networks.**

## I. Introduction

ADAPTIVE systems, such as the multilayered perceptron (MLP) and other neural networks, adjust their internal parameters performing vector mappings from the input to the output space. Although they may achieve high accuracy of classification, the knowledge acquired by such systems is represented in a large number of numerical parameters and network architectures, in a way that is incomprehensible for humans. The *a priori* knowledge about the problem to be solved is frequently given in a symbolic, rule-based form. Extraction of knowledge from data, combining it with available symbolic knowledge and refining the resulting knowledge-based expert systems is a great challenge for computational intelligence. Reasoning with logical rules is more acceptable to human users than recommendations given by black box systems [1], because such reasoning is comprehensible, provides explanations, and may be validated by human inspection, increasing confidence in the system, important relationships and features may be discovered in the data.

Manuscript received June 15, 1999; revised December 20, 1999 and June 15, 2000. This work was supported by the Polish Committee for Scientific Research, Grant 8 T11F 014 14.

The 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).

Publisher Item Identifier S 1045-9227(00)09855-6.

Comprehensibility is often regarded in machine learning (ML) as the most desired characteristic of inductive methods (i.e., methods that learn from examples). Michalski, one of the ML pioneers, formulated it in the following way: "The results of computer induction should be symbolic descriptions of given entities, semantically and structurally similar to those a human expert might produce observing the same entities. Components of these descriptions should be comprehensible as single "chunks" of information, directly interpretable in natural language, and should relate quantitative and qualitative concepts in an integrated fashion" [2].

Many methods to find logical description of the data have been designed in the past using statistical, pattern recognition [3] and machine learning [4] approaches. Rule-based systems should be preferred over other methods of classification only in cases when the set of logical rules is not too complex and their predictive accuracy is sufficiently high. Hundreds of logical rules produced by some algorithms provide opaque description of the data and therefore are not more comprehensible than any black-box classification system. 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 medicine, finances, commerce, and other applications.

A good strategy in data mining and classification tasks is to use the simplest description of the data that does not compromise accuracy: extract crisp logical rules first, use fuzzy rules if crisp rules are not sufficient, and only if the number of logical rules required for high accuracy of classification is too large use other, more sophisticated tools. In many applications simple crisp logical rules proved to be more accurate and were able to generalize better than many machine and neural learning algorithms [5]. In other applications fuzzification of logical rules gave more accurate results [6]. Crisp logical rules may be converted to a specific form of fuzzy rules (Section VIII) and optimized using gradient procedures, providing higher accuracy without significant increase of the complexity or decrease of comprehensibility of the rule-based system.

Are neural methods competitive to other methods in providing simple and accurate sets of logical rules? There are two issues here: understanding what neural networks really do, and using neural networks to extract logical rules describing the data. Many neural rule extraction methods have been devised in the past, but there are very few comparisons with other methods and explicit logical rules are almost never published. Several neural methods have been compared experimentally [1] on the mushroom and the three monk problems benchmark datasets [7], but no comparison with machine learning methods has

been given. There is a strong competition from decision trees [8], which are fast, accurate, and can easily be converted to sets of logical rules, from inductive methods of machine learning [4], and from systems based on fuzzy [9], [10] and rough sets [11], [12].

Despite this competition neural networks seem to have important advantages, especially for problems with continuous-valued inputs. Good linguistic variables may be determined simultaneously with logical rules, selection and aggregation of features into smaller number of more useful features may be incorporated in the neural model, adaptation mechanisms for continuously changing data are built in, and wide-margin classification provided by neural networks leads to more robust logical rules.

In this paper we do not introduce "a new neural method" for rule extraction, but rather present a complete methodology for extraction, optimization, and application of sets of logical rules. An overview of neural rule extraction methods is made in the next section, followed by some comments on types of logical rules used in inductive methods. The first step in the rule-based data analysis requires selection of initial linguistic variables, as described in Section IV. Several new neural rule extraction methods are presented in Section V and a pedagogical example of the actual process of rule extraction, based on the well-known Iris flower data [7], is given in Section VI. Once initial rules are extracted simplification and optimization of linguistic variables for real-valued attributes is done. In Section VII, the accuracy/rejection tradeoff for sets of rules is explored. A new error function is defined allowing to create hierarchical sets of rules, starting from rules that are very reliable but reject many cases (assigning them to the "unknown" class), to rules that classify all data but are less reliable.

Crisp logical rules assign a given input vector to a single class with probability equal one, even in cases when similar probability for two or more classes should be reported. In Section VIII, a method for calculation of probabilities for rule-based classifiers is presented. Assuming Gaussian uncertainties of the measured features analytical formulas for classification probabilities are derived. Such approach is equivalent to the use of fuzzy rules with "soft trapezoid" membership functions applied to crisp input vectors. This enables optimization of linguistic variables for very large sets of rules using efficient gradient procedures and preserves the ease of interpretation of crisp logical rules. Illustration of the optimization and probability calculation steps is done in Section IX, while in Section X many applications on well-known data and some real-world examples are presented and, whenever possible, compared with other approaches. Explicit form of rules are given, in most cases the simplest and most accurate reported in the literature so far for these datasets. Section XI contains summary and conclusions.

## II. An Overview of Neural Rule Extraction Methods

A taxonomy of the neural rule extraction algorithms may characterize different methods using five dimensions [13]: 1) the "expressive power" of the extracted rules (types of rules extracted); 2) the "quality" of the extracted rules (accuracy, fidelity comparing to the underlying network, comprehensibility and consistency of the extracted rules); 3) the "translucency" of the method, based on local-global use of the neural network (analysis of individual nodes versus analysis of the total network function); 4) the algorithmic complexity of the method; 5) specialized network training schemes. One should add one more dimension to this scheme; and 6) the treatment of linguistic variables: some methods work only with binary variables, other with discretized inputs, and yet other with continuous variables that are converted to linguistic variables automatically.

In the simplest case the inputs are binary and the network gives logical outputs. After training the network performance is equivalent to a set of logical rules that may be found by giving as input all possible combinations of features. For $n$ binary features the number of conjunctive rules is $3^n$ (since each feature may either be absent, present or its negation may be present in the rule antecedent). To limit the number of nodes in the search graph one may try to limit the number of literals in the antecedents of extracted rules. In one of the first neural rule extraction methods Saito and Nakano [14] restricted the maximum number of positive and negative literals and the depth of the breadth-first search process, additionally restricting the search tree to those combinations of literals that were present in the training set. Due to these restrictions their method could sometimes accept a rule that was too general. This drawback has been removed in the method developed by Gallant [15]. The difficulty comes from the inputs that are not specified in the rule provided as a candidate by the search procedure. Gallant takes all possible values for these inputs and although his rules are always correct they may be too specific.

The validity interval analysis (VIA) method developed by Thrun [16] is a further extension of the global approach. A validity interval, specifying the maximum activation range for each input, may be found using linear programming techniques. These intervals may be propagated backward and forward through the network. Arbitrary linear constraints may be applied to input as well as output units, giving the method the ability to check the validity of nonstandard form of rules, such as the $M$-of-$N$ rules, i.e., logical expressions in which at least $M$ of $N$ literals are true. VIA can handle also continuous-valued input features, starting from the training values and replacing them with intervals that are increased to achieve good generalization of rules. The method may be applied to any neural network with monotonic transfer functions. Unfortunately it has a tendency to extract rules that are too specific and rather numerous.

These methods are global, based on analysis of outputs of the whole network for various inputs. Local, or "decompositional" methods [1] analyze fragments of the network, usually single hidden nodes, to extract rules. Such networks are either based on sigmoidal functions (step functions in the logical limit), or on localized functions. Using step functions the output of each neuron becomes logical (binary) and since sigmoidal transfer functions are monotonic and activations are between zero and one it is enough to know the sign of the network weight to determine the contribution to activation of a given unit. Search for rules has now $2^n$ possible combinations of input features (irrelevant or relevant feature, with negation of literal determined by the weight sign), while in the global approach monotonicity

does not, in general, hold. Rules corresponding to the whole network are combined from rules for each network node.

Local methods for extraction of conjunctive rules were proposed by Fu [17]–[20] and Gallant [15]. As with the global methods depth of search for good rules is restricted. The weights may be used to limit the search tree by providing the evaluation of contributions of inputs that are not specified in rule antecedents. As shown by Sethi and Yoo [21] the number of search nodes is then reduced to $O(2^n/\sqrt{n})$. In the `Subset` algorithm of Towell and Shavlik [22] inputs with largest weights are analyzed first, and if they are sufficient to activate the hidden node of the network irrespectively of the values on other inputs, a new rule is recorded. Combinations of the two largest weights follow, until the maximum number of antecedent conditions is reached. A fuzzy version of this approach has been proposed by Hayashi [23].

All these methods still have a problem with exponentially growing number of possible conjunctive prepositional rules. Towell and Shavlik [22] proposed to use the $M$-of-$N$ rules, since they are implemented in a natural way by network nodes. In some cases such rules may be more compact and comprehensible than conjunctive rules. To avoid combinatorial explosion of the number of possible input combinations for each network node groups of connections with similar weights are formed. Weights in the group are replaced by their averages. Groups that do not affect the output are eliminated and biases reoptimized for frozen weights. Such a simplified network has effectively lower number of independent inputs, therefore it is easier to analyze. If symbolic knowledge is used to specify initial weights, as it is done in the knowledge-based artificial neural networks (KBANNs) of Towell and Shavlik [24], weights cluster before and after training. The search process is further simplified if the prototype weight templates (corresponding to symbolic rules) are used for comparison with the weight vectors [25] (weights are adjusted during training to make them more similar to templates). The RuleNet method based on templates has also been used to find the best $M$-of-$N$ rules in $O(n^2)$ steps and the best sets of nested $M$-of-$N$ rules in $O(n^3)$ steps [26], exploring large spaces of candidate rules. The method handles only discrete-valued features, therefore initial discretization is necessary for continuous features. The network has only one hidden layer with a specific architecture to inject symbolic rules into the network and refine them iteratively.

Several authors noticed the need for simplification of neural networks to facilitate rule extraction process. Setiono and Liu [27] use a regularization term in the cost function to iteratively prune small weights. After simplification the network is discretized by clustering activation values of the hidden unit obtained during presentation of the training set. The method does not guarantee that all rules will be found, but results for small networks were encouraging. The method of successive regularization [28] is based on a similar idea, with Laplace regularization (sum of absolute weight values) in the error function, inducing a constant decay of weights. Only weights smaller than some threshold are included in the regularizing term (this is called "selective forgetting"). Hidden units are forced to become fully active or completely inactive. After training a skeletal network structure is left and the dominant rules extracted. Keeping this skeletal network frozen small connections are revived by decreasing the regularization parameters. After training of the more complex network additional logical rules are obtained from analysis of new nodes/connections. Another simple method belonging to that group has been presented by Geczy and Usui [29]. Weights in the MLP network with one hidden layer are mapped after training into zero, $+1$ or $-1$ values, simplifying the rule search step. In our own MLP2LN approach [30] described below such a mapping is incorporated in the learning scheme.

Rule extraction as learning (REAL) is a rather general technique introduced by Craven and Shavlik [31] for incremental generation of new rules (conjunctive or $M$-of-$N$). If a new example is not classified correctly by the existing set of rules a new rule, based on this example, is added and the fidelity of the extended set of rules is checked against the neural network responses on all examples used so far. The RULENEG algorithm [1], [32], [94] is based on a similar principle: one conjunctive rule per input pattern is generated and if a new training vector is not correctly classified by the existing set of rules $\mathcal{R}$ a new rule is created as a conjunction of all those inputs literals that have influence on the class of the vector. This is determined by consecutive negation of each input value followed by checking (using the neural network) if the predicted class has changed.

In the BRAINNE algorithm [33] a network of $m$ inputs and $n$ outputs is changed to a network of $m+n$ inputs and $n$ outputs and retrained. Original inputs that have weights which change little after extension and retraining of the network correspond to the most important features. The method can handle continuous inputs and has been used in several benchmark and real-life problems, producing rather complex sets of rules. Logical rule extraction has also been attempted using self-organizing ART model [34] and fuzzy ARTMAP architecture [35]. In the last case a certainty factors for each rule are provided. Simpler self-organizing architectures may also be used for rule extraction [36], although accuracy of the self-organized mapping for classification problems is rather poor.

The DEDEC algorithm [1], [37] extracts rules by finding a minimal information sufficient to distinguish, from the neural network point of view, between a given pattern and all other patterns. To achieve this a new set of training patterns is generated. First, inputs are ranked in order of their importance, estimated by inspection of the influence of the input weights on the network outputs. Second, clusters of vectors are selected and used instead of original cases. Only those features ranked as important are used to derive conjunctive rules, which are found by searching.

Since our goal is to get the simplest logical description of the data, rather than description of the network mapping, we are in favor of using specialized training schemes and architectures. Of course any rule extraction method may be used to approximate the neural-network function on some training data. The network is used as an "oracle," providing as many training examples as one wishes. This approach has been used quite successfully by Craven and Shavlik in their TREPAN algorithm [38], combining decision trees with neural networks. Decision trees are induced by querying neural network for new examples, adding

tree nodes that offer the best fidelity to the classification by the network. New branches of the tree are created only after a large number of queries has been answered. Therefore the method is more robust than direct decision tree approach, which suffers from small number of cases in the deeper branches. Classifiers based on ensembles of different models, similarity-based classifiers, statistical methods or any other classifiers that produce incomprehensible models of the data may be approximated by rule-based systems in the same way.

Neural networks based on separable localized transfer function are equivalent to fuzzy logic systems [39]. Each node has a direct interpretation in terms of fuzzy rules and there is no need for a search process. Gaussian functions were used for inserting and extracting knowledge into the radial basis set type of networks [40]. More general proposal for neurofuzzy system based on separable functions was made by Duch [41], [42]. Discussion of rule extraction using localized transfer functions has been given by Andrews and Geva [43], [95]. These authors developed a quite successful approach called RULEX [44], based on constrained MLP networks with pairs of sigmoidal functions combined to form "ridges," or "local bumps." Rules may in this case be extracted directly from analysis of weights and thresholds, without the search process, since disjoint regions of the data correspond to one hidden unit. In effect the method is similar to a localized network with rectangular transfer functions. The method works for continuous as well as discrete inputs.

Methods of combining neural and symbolic knowledge, refining probabilistic rule bases, scientific law discovery and data mining are closely related to applications of neural networks for extraction of logical rules. Symbolic rules may be converted into RAPTURE networks [45] and trained using a modified backpropagation algorithms for optimization of certainty factors. The network prunes small connections and grows adding new nodes if classification accuracy becomes too low.

It may seem that neurofuzzy systems should have advantages in application to rule extraction, since crisp rules are just a special case of fuzzy rules. Quite many neurofuzzy systems are known and some indeed work rather well [42], [46], [96], [47]–[49]. However, there is a danger of overparametrization of such systems, leading to difficulty of finding optimal solutions [10], [50], even with the help of genetic algorithms or other global optimization methods. Systems based on rough sets [11] require additional discretization procedures which may determine the quality of their performance. We have included a few results obtained by fuzzy and rough systems in Section X presenting applications. Simpler rule extraction systems based on neural networks may have advantages over the fuzzy, rough or neurofuzzy systems, although a good empirical comparison of their capabilities is certainly needed. Many rule extraction methods have been tested on rather exotic datasets, therefore their relative advantages are hard to judge.

Most papers on the rule extraction are usually limited to the description of new algorithms, presenting only a partial solution to the problem of knowledge extraction from data. Control of the tradeoff between comprehensibility and accuracy, optimization of the linguistic variables and final rules, and estimation of the reliability of rules are almost never discussed. In practical applications it may be quite useful to have rough, low accuracy, simple description of the data and to be able to provide more accurate, but more complex description, in a controlled manner. Neural methods of rule extraction may provide initial rules, but that should not be the end of the story.

## III. Types of Rules

In this section types of logical rules are discussed, stressing the importance of decision borders they are able to provide in multidimensional feature spaces. Although nonstandard form of rules, such as $M$-of-$N$ rules ($M$ out of $N$ antecedents should be true), fuzzy rules, decision trees [4] and more complex forms of knowledge representation are sometimes used in this paper we will consider only standard IF…THEN prepositional rules. Since these rules are the simplest and most comprehensible they should be tried first.

A very general form of prepositional rule is

$$\text{IF } X \in K^{(i)} \qquad \text{THEN Class}(X) = C_i \tag{1}$$

i.e., if $X$ belongs to the cluster $K^{(i)}$ then its class is $C_i = \text{Class}(K^{(i)})$, the same as for all vectors in this cluster. This general approach does not restrict the shapes of clusters used in logical rules, but unless the clusters are visualized in some way (a difficult task in high-dimensional feature spaces) it does not give more understanding of the data than any black box classifier. Therefore some assumptions regarding the shapes of clusters should be made, with the goal of obtaining the smallest number of comprehensible rules in mind.

For clusters with decision borders that have simple convex shapes several conjunctive rules of the type

$$\text{IF } (x_1 \in \mathcal{X}_1 \wedge x_2 \in \mathcal{X}_2 \wedge \ldots x_N \in \mathcal{X}_N) \quad \text{THEN Class} = C_k \tag{2}$$

may be sufficient. If $\mathcal{X}_i$ are sets of symbolic values, discrete numerical values, or intervals for continuous features, crisp logic rules are obtained. They provide hyperrectangular decision borders in the feature subspaces corresponding to variables appearing in rule conditions. This approximation may not be sufficient if complex decision borders are required, but it may work quite well if the problem has inherent logical structure.

A fruitful way of looking at logical rules is to treat them as an approximation to the posterior probability of classification $p(C_i|X; M)$, where the model $M$ is composed of the set of rules. Crisp rules give $p(C_i|X; M) = 0, 1$ but if clusters belonging to different classes overlap this is obviously wrong. A soft interpretation of the $\in$ operator requires "membership" functions and leads to fuzzy rules, for example in the form

$$p(C_k|X; M) = \frac{\mu^{(k)}(X)}{\sum_i \mu^{(i)}(X)} \tag{3}$$

where

$$\mu^{(k)}(X) = \prod_i \mu_i^{(k)}(X_i) \tag{4}$$

and $\mu^{(k)}(X)$ is the value of the membership function defined for cluster $k$. Such **context-dependent** or **cluster-dependent membership functions** are rarely used in classification systems

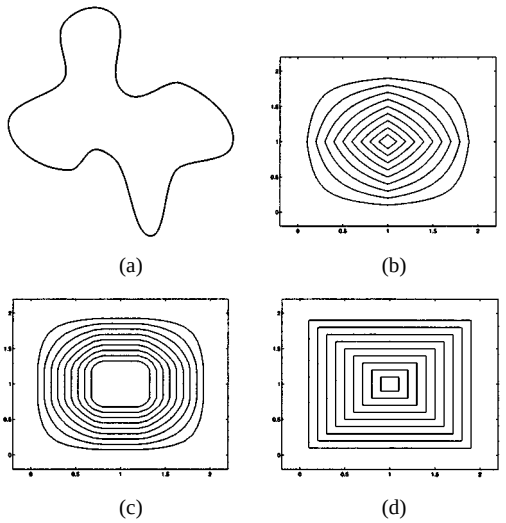


Fig. 1. Shapes of decision borders for (a) general clusters, (b) fuzzy rules (using product of membership function), (c) rough rules (trapezoidal approximation), and (d) crisp logical rules.

based on fuzzy logics, although they are quite natural in the neurofuzzy systems [42].

The flexibility of the fuzzy approach depends on the choice of membership functions. Fuzzy logic classifiers frequently use a few membership functions per input feature [10]. Triangular membership functions provide oval decision borders, similar to those provided by Gaussian functions (cf. Fig. 1). Therefore results should be similar to that of the radial basis function (RBF) networks and indeed they are formally equivalent [39]. Triangular membership functions may be regarded as piece-wise linear approximation to Gaussian membership functions, while trapezoidal membership functions are similar approximations to the soft trapezoid functions obtained from combinations of two sigmoidal transfer functions (cf. next section).

Thus decision borders provided by the fuzzy rules, although of different shape than those of crisp rules, do not allow for more flexible partitioning of the input space. Their big advantage is the ability to provide classification probabilities instead of yes/no answers. From the accuracy and simplicity point of view the ability to deal with oblique distribution of data may be more important than softer decision borders. Rotation of decision borders requires new linguistic variables, formed by taking linear combination, or by making nonlinear transformations of input features. The meaning of such rules is sometimes difficult to comprehend (cf. proverbial "comparing apples with oranges"). Another form of incomprehensible rules is obtained from a union of halfspaces defined by hyperplanes, forming a convex, polyhedral shapes.

The rough set theory [11] is also used to derive crisp logic prepositional rules. In this theory for two-class problems the lower approximation of the data is defined as a set of vectors, or a region of the feature space containing input vectors that belong to a single class $C_k$ with probability $p(C_k|X;\, M) = 1$, while the upper approximation covers all instances which have a nonzero chance to belong to this class [i.e., probability is $p(C_k|X;\, M) > 0$]. In practice the shape of the boundary between the upper and the lower approximations depends on the indiscernibility (or similarity) relation used. Linear approximation to the boundary region leads to trapezoidal membership functions, i.e., the same shapes of decision borders as obtained by fuzzy systems with such membership functions. The crisp form of logical rules is obtained when trapezoidal membership functions are changed into rectangular functions. Rectangles allow to define logical linguistic variables for each feature by intervals or sets of nominal values.

Crisp, fuzzy, and rough set decision borders are special cases of more general decision borders provided by neural networks based on localized separable transfer functions [42]. Although individual fuzzy, rough and neurofuzzy systems differ in their approach to logical rule discovery, their ultimate capability depends on the decision borders they may provide for classification.

## IV. Context-Dependent Linguistic Variables

Logical rules require symbolic inputs, called linguistic variables. The input data has to be quantized first, i.e., features defining the problem should be identified and their subranges (sets of symbolic values, integer values, or continuous intervals) labeled. For example a variable "size" has the value "small" if the continuous variable $x_k$ measuring size falls in some specified range, $x_k \in [a,\, b]$. Using one input variable several binary (logical) variables are created, for example $s_1 = \delta(size,\, small)$ equal to one (true) only if variable "size" has the value "small."

Linguistic variables used by us are **context dependent**, i.e., they may be different in each rule (cf. [51]). For real-valued attributes intervals defining linguistic variables used in logical rules are needed. Determination of these intervals is done by analysis of histograms (only in simple cases), information-based criteria like those used for decision-trees [4], using feature space mapping (FSM) constructive neural network [42], using special "linguistic units" (L-units) in multilayer perceptron (MLP) networks [51] or using an explicit separability criterion [52]. Since it is hard to overestimate the importance of good linguistic units these methods are described below in some details.

A symbolic attribute *color* may take values *green, red, blue* and appear in a rule as logical condition, for example *color* $=$ *red*. An alternative way is to use a predicate function *color*$(x)$. Depending on the type of variable $x$ the predicate function may have a different interpretation. For example, if $x$ is the wavelength of light and $x \in [600\ \text{nm},\, 700\ \text{nm}]$ then *color*$(x)$ is *red*, i.e., logical condition *color*$(x)$ $=$ *red* is true. One may also introduce predicates for each color defined by logical functions *color-green*$(x)$, *color-red*$(x)$, *color-blue*$(x)$. Such logical predicate functions are linguistic variables, mapping symbolic or real values of $x$ into binary zero, one or *false, true*.

If the input $x \in \mathcal{X}$, where $\mathcal{X}$ is the subset of real numbers, or a large set of integers or symbolic values, linguistic variables are created dividing the data $\mathcal{X}$ into distinct (for crisp logic) subsets $\mathcal{X}_i$. Linguistic variables are introduced as

$$s_i(x) = F, \qquad \text{unless } x \in \mathcal{X}_i, \qquad \text{then } s_i(x) = T.$$

For $\mathcal{X} \subseteq R$ sets $\mathcal{X}_i$ are usually intervals and linguistic variables are binary functions mapping $x$ into zero or one. A typical linguistic variable associated with the attribute "tire pressure"

will be *low* if $x < 1.7$, *normal* if $1.7 \leq x \leq 2.2$ and *high* if $x \geq 2.2$. A rule may then have conditions of the form *high*$(x)$, which is usually written as $x =$ *high*, meaning that $x \geq 2.2$.

Introducing a *color-red*$(x)$ predicate that has values in the [0, 1] range, instead of the binary 0, 1 values, one may interpret it as estimation of similarity of color that $x$ has to the typical red color. Using such predicate functions as logical conditions is equivalent to some form of fuzzy logic, depending on the way logical functions are mapped on arithmetic functions [9]. Thus soft predicate functions play the role of membership functions: binary valued functions are used in crisp logic and real valued functions in fuzzy logic (for multistep values multivalued logic conditions are defined). For $\mathcal{X} \subseteq R$ crisp membership functions are rectangular while fuzzy membership functions have triangular, trapezoid, Gaussian, or other shapes that are useful for evaluation of similarities.

In many applications of fuzzy sets a common set of linguistic variables is assumed for all rules. Such membership functions are **context-independent**, identical for all regions of the input space. Defining, for example, three triangular membership functions per attribute, $\mu_1(x_i)$, $\mu_2(x_i)$, $\mu_3(x_i)$, rules for combinations

$$\text{IF } (\mu_{k_1}(x_1) \wedge \mu_{k_2}(x_2) \ldots \wedge \mu_{k_N}(x_N))$$

are sought [9], with $k_i = 1, 2, 3$. Unfortunately, the number of combinations grows exponentially with the number of attributes (here like $3^N$), and the method works only for two or three dimensions. Covering of a complex cluster may require a large number of such membership functions. In both crisp and fuzzy cases linguistic variables should be **context dependent**, i.e., optimized in each rule. Small tire pressure for bicycle is different than for a car or a truck. For example if $x_1 =$ *broad* for $1 \leq x_1 \leq 4$, $x_1 =$ *average* for $2 \leq x_1 \leq 3$, and $x_2 =$ *small* for $1 \leq x_2 \leq 2$, $x_2 =$ *large* for $3 \leq x_2 \leq 4$ then two simple rules

$$\begin{aligned} &\text{IF } (x_1 = broad \wedge x_2 = small) && \text{THEN } C = great \\ &\text{IF } (x_1 = average \wedge x_2 = large) && \text{THEN } C = great \\ &\text{ELSE } C = so\text{-}so \end{aligned}$$

would be more complex if written using linguistic variables that partition $x_1$ into distinct or just partially overlapping subsets. In the context of $x_2 =$ *large* linguistic variable $x_1 =$ *average*, rather than *broad*, should be used. Instead of using a fixed number of linguistic variables one should rather use rule-dependent linguistic variables, optimized for each rule.

The simplest way to select initial linguistic variables is to analyze histograms, displaying data for all classes for each feature. Histograms should be smoothed, for example by assuming that each data vector is really a Gaussian or a triangular fuzzy number. Unfortunately histograms for all features frequently overlap. Therefore we have developed several methods for determination of initial linguistic variables.

### A. Selection using Density Networks

Feature space mapping (FSM) is a constructive neural network [42], [53], [54] that estimates the probability density $p(C|X, Y; M)$ of input $X$-output $Y$ pairs in each class $C$. Nodes of this network use localized, separable transfer functions, providing good linguistic variables. Crisp decision regions are obtained by using rectangular transfer functions; if this is not sufficient Gaussian, trapezoidal or other separable transfer functions are used.

The network is initialized using a decision tree or a clusterization method based on dendrograms [53], and adapted to the incoming input data by moving the transfer functions centers, decreasing and increasing their dispersions, or by adding more transfer functions (new network nodes) if necessary. The initialization process is robust and may already lead to reasonable intervals for the initial linguistic variables. In some cases results after initialization, before the start of learning, were better than final results of other classification systems [53]. The FSM network may use an arbitrary separable transfer function, including triangular, trapezoidal, Gaussian, or the bicentral combinations of sigmoidal functions [55] with soft trapezoidal shapes. Two simple bicentral-type functions are constructed as the difference of two sigmoids, $\sigma(x) - \sigma(x - \theta)$ or the product of pairs of sigmoidal functions $\sigma(x)(1 - \sigma(x))$ for each dimension. For logistic functions of the form $\sigma(x) = 1/(1 + e^{-x})$ after normalization the two forms become identical

$$\frac{\sigma(x+b)(1-\sigma(x-b))}{\sigma(b)(1-\sigma(-b))} = \frac{\sigma(x+b)-\sigma(x-b)}{\sigma(b)-\sigma(-b)}. \quad (5)$$

The proof is not difficult if one notes the following identities:

$$\sigma(b)/\sigma(-b) = e^b; \qquad \sigma(b) = 1 - \sigma(-b). \quad (6)$$

If the gain of sigmoidal functions $\sigma(x)$ is slowly increased during learning rectangular functions are smoothly recovered from products $\prod_i(\sigma(x_i - b_i) - \sigma(x_i + b'_i))$. After training nodes of the FSM network are analyzed, providing good intervals for logical variables. To encourage broad intervals, increasing stability of rules and facilitating selection of features, the lower and the upper values defining linguistic variables are moved away from the center of the function during iterative training (the same effect may be achieved by adding penalty terms to the cost function). To obtain initial linguistic variables for rule extraction we start with rectangular transfer functions which may be fuzzified by using soft trapezoidal functions.

### B. Linguistic Neural Units

Linguistic neural units (L-units) automatically analyze continuous inputs and produce linguistic variables [51]. The basic scheme of such unit is shown in Fig. 2. An input $x_i$ is connected via $W_1$, $W_2$ weights to two neurons, each with its own separate bias, $b_i$ and $b'_i$. All transfer functions are sigmoidal. At the end of the training they should be very steep, although at the beginning they may be quite smooth, allowing for fuzzy approximation of classification borders. The two hidden neurons of the L-unit are connected to its output neuron using weights $S_1$, $S_2$.

Experiments showed that learning is faster if connections from the two hidden L-unit neurons to other hidden neurons are added. All weights have values constrained at the end of the training to $0, \pm 1$. The network (Fig. 3) composed of L-units and hidden units (called R-units, since they provide logical rules) is an MLP network with specific (constrained)

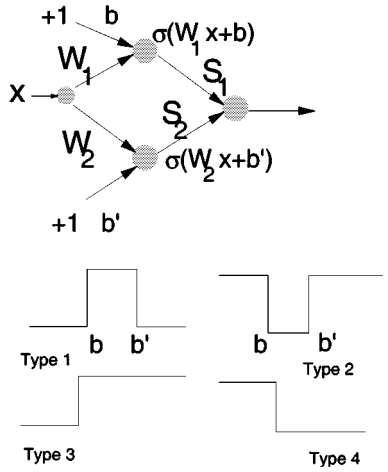


Fig. 2. Construction of a linguistic unit converting continuous inputs to linguistic variables.

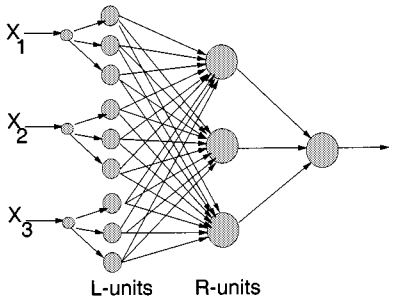


Fig. 3. MLP network with linguistic and rule units. An additional aggregation layer may be added between the input and L-units.

architecture. Since L-units have only one input, one output, and four constrained weights as parameters, functions realized by these units belong to one of the four types shown in the limit of large gain in Fig. 2.

The first of these functions (Type 1) is obtained as a difference of two sigmoids and represents a typical linguistic variable $s_k$ equivalent to $x_i \in [b_i, b_i']$, the second (Type 2) denotes negation $\neg s_k$ while the other two (Type 3 and 4), with only one nonzero weight, correspond to $x_i \geq b$ or $x_i \leq b$. The borders $b_i$ and $b_i'$ defining linguistic variables and the four constrained weights are treated as adaptive parameters of our network.

The threshold of the output unit is kept fixed at one. Input weights $W_1$, $W_2$, and the weights $S_1$, $S_2$, each taking values constrained to 0, $\pm 1$, may take at most 81 values. Only a few combinations give different L-unit transfer functions (Table I). Most combinations are identically zero—in this case the feature does not contribute to the rule. One could also use a single neuron with rectangular or bicentral transfer function instead of the L-unit. The network structure would then look simpler but it would not be a constrained MLP network, easy to implement using conventional neural-network programs.

In practice training L-units separately from R-units leads to faster convergence. When the L-unit weights are trained (optimizing linguistic variables) R-unit weights are kept frozen and vice versa. The output L-unit neurons have frequently both weights $S_1, S_2 = 0$ and are deleted, because open intervals realized by other hidden L-unit nodes are sufficient.

TABLE I
EXAMPLES OF POSSIBLE FUNCTIONS REALIZED BY L-UNITS, $b > b'$, TYPE 1–TYPE 4 AS IN FIG. 2

| $W_1$ | $W_2$ | $S_1$ | $S_2$ | Function type |
|---|---|---|---|---|
| +1 | +1 | +1 | -1 | Type 1 |
| -1 | +1 | +1 | +1 | Type 2 |
| +1 | 0 | +1 | 0 | Type 3 |
| -1 | 0 | -1 | 0 | Type 4 |

In some applications with a large number of features **an aggregation** of some types of features is possible and should lead to better linguistic variables. Groups of features that are of the same type may be combined together by an additional layer of neurons between input and L-units. These aggregation units (A-units) are either trained without any regularization, or trained with initial enforcement of zero connections followed by training without any regularization. The A-units should be designed incorporating knowledge about the type of input features. We have used this approach only in a few difficult cases, when hundreds of features are present.

The L-units take as input continuous vectors $X^{(p)} = (x_1^{(p)}, \ldots, x_N^{(p)})$ and give as output a vector of linguistic variables $\mathbf{L}^{(p)} = \mathbf{L}(X^{(p)}) = (l_1^{(p)}, \ldots, l_K^{(p)})$. 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 ("conflicts" in the rough set terminology) that other network nodes 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. One solution is to minimize the number of identical linguistic variables corresponding to vectors that belong to different classes

$$E(\mathbf{B}, \mathbf{B}') = \sum_{\substack{p,p' \\ C^p \neq C^{p'}}} \delta\left(\mathbf{L}^{(p)}, \mathbf{L}^{(p')}\right) \tag{7}$$

where $C^p = C(X^{(p)})$ is the class the $X^{(p)}$ vector belongs to and $\mathbf{B}$, $\mathbf{B}'$ are the intervals defining linguistic variables $\mathbf{L}$. To enable gradient minimization $\delta$ functions may be replaced by narrow Gaussian distributions. The total error function should be summed over all intervals $B$, $B'$. Such explicit conditions enforcing distinguishability may be desirable, but may also lead to creation of too many linguistic variables handling noise in the data.

### C. Separability Criterion

Another approach to selection of linguistic variables is based on a general separability criterion introduced by us recently [52]. The best "split value" for an open interval should separate the maximum number of pairs of vectors from different classes. Among all split values which satisfy this condition the one which separates the smallest number of pairs of vectors belonging to the same class is selected. The criterion is applicable to both continuous and discrete features. Since one feature is

treated at a time the minimization process is easier than either trying to minimize classification error or (7) in respect to all intervals at the same time.

The *split value* (or *cutoff point*) is defined differently for continuous and discrete features. In the case of continuous features the *split value* is a real number, in other cases it is a subset of the set of alternative values of the feature. In all cases the *left side* (LS) and the *right side* (RS) of a split value $s$ of feature $f$ for given dataset $D$ is defined as

$$\begin{aligned}\mathrm{LS}(s,\, f,\, D) &= \begin{cases}\{x \in D\colon f(x) < s\} & \text{if } f \text{ is continuous}\\ \{x \in D\colon f(x) \notin s\} & \text{otherwise}\end{cases}\\ \mathrm{RS}(s,\, f,\, D) &= D - \mathrm{LS}(s,\, f,\, D)\end{aligned} \tag{8}$$

where $f(x)$ is the $f$'s feature value for the data vector $x$.

The *separability of a split value* $s$ is defined as

$$\begin{aligned}&\mathrm{SSV}(s)\\ &\quad = 2\sum_{c\in C} |\mathrm{LS}(s,\, f,\, D) \cap D_c| \cdot |\mathrm{RS}(s,\, f,\, D) \cap (D - D_c)|\\ &\qquad - \sum_{c\in C} \min(|\mathrm{LS}(s,\, f,\, D) \cap D_c|,\, |\mathrm{RS}(s,\, f,\, D) \cap D_c|)\end{aligned} \tag{9}$$

where $C$ is the set of classes and $D_c$ is the set of data vectors from $D$ which belong to class $c$. The higher the separability of a split value the better. Points beyond the borders of feature values existing in the dataset have the SSV (separability split value) equal to zero, while separability of all points between the borders is positive. This means that for every dataset containing vectors which belong to at least two different classes, for each feature which has at least two different values, there exists a split value of maximal separability.

When the feature being examined is continuous and there are several different split values of maximal separability close to each other, a reasonable heuristics is to select the split value closest to the average of all of them. To avoid such situations split values which are natural for a given dataset are examined, i.e., values that are between adjacent feature values. If there are two maxima with smaller split values in between, or if the feature is discrete, then the selection of the best split value may be arbitrary.

The separability criterion can be used in several different ways to discretize a continuous feature, if context-independent linguistic variables are desired. For instance, the same algorithm can be followed as for the construction of a decision tree, but the possible cut points should be checked only for the feature being discretized. The recursive process stops when the subsequent splits do not significantly improve the separability or when a sufficient number of cut points is obtained. The recursive process is necessary, because usually features have just one or two maximal cut points. When the data is split into two parts at least one best split value for each of the parts will certainly be found in the next stage.

Sometimes all split values of a given feature have very low separability. This either means that the feature is not important or that it should be taken into account in conjunction with discretization of another feature. The separability of a single split value can easily be generalized to the separability of a set of all split values for a given feature, which can be used for the feature selection. If separability measures for all features are low context dependent linguistic variables are necessary. Search for the best separability of a pair or a combination of several features is performed quite efficiently using beam search techniques. For a pair of features the search complexity is quadratic in the number of split values considered, enabling in practice exhaustive search. Searching for all feature split values at the same time takes into account mutual interaction of features, therefore it may significantly improve results, but since the search complexity is high the width of the beam search should be selected to make it practical.

## V. Rule Extraction Algorithms

After initial definition of linguistic variables methods to find logical rules are needed. Neural methods that we will use for that purpose focus on analysis of parameters (weights and biases) of trained networks. Since in many cases inductive bias of neural networks may not be the most appropriate for a given data methods described below may either be used to extract logical rules directly from the data or to find a set of logical rules that approximates the mapping generated by a neural network. These and other methods of rule extraction are useful to generate initial form of rules that should be further simplified and optimized together with the linguistic variables.

### A. MLP2LN: Changing MLP into Logical Network

To facilitate extraction of logical rules from an MLP network one should transform it smoothly into a network performing logical operations (logical network, LN). This transformation, called here MLP2LN [56], may be realized in several ways. Skeletonization of a large MLP network is the method of choice if our goal is to find logical rules for an already trained network. Otherwise starting from a single neuron and constructing the logical network using training data directly (called the C-MLP2LN method) is faster and more accurate. Since interpretation of the activation of the MLP network nodes is not easy [57] a smooth transition from MLP to a logical-type of network performing similar functions is advocated. This transition is achieved during network training by:

1) gradually increasing the slope $\beta$ of sigmoidal functions $\sigma(\beta x)$ to obtain crisp decision regions;
2) simplifying the network structure by inducing the weight decay through a penalty term;
3) enforcing the integer weight values 0 and $\pm 1$, interpreted as $0$ = irrelevant input, $+1$ = positive and $-1$ = negative evidence. These objectives are achieved by adding two additional terms to the standard mean square error function $E_0(W)$

$$\begin{aligned}E(W) = \frac{1}{2}\sum_p \sum_k \left(Y_k^{(p)} - \mathbf{F}_k\left(X^{(p)}; W\right)\right)^2 + \frac{\lambda_1}{2}\sum_{i,j} W_{ij}^2\\ + \frac{\lambda_2}{2}\sum_{i,j} W_{ij}^2 (W_{ij}-1)^2 (W_{ij}+1)^2.\end{aligned} \tag{10}$$

The first part is the standard mean square error measure of matching the network output vectors $\mathbf{F}(X^{(p)}; W)$ with the de-

sired output vectors $Y^{(p)}$ for all training data samples $p$. The second term, scaled by $\lambda_1$, is frequently used in the weight pruning or in the Bayesian regularization method [58], [59] to improve generalization of the MLP networks.

A naive interpretation why such regularization works is based on observation that small weights and thresholds mean that only the linear part of the sigmoid around $\sigma(0)$ is used. Therefore the decision borders are rather smooth. On the other hand for logical rules we need sharp decision borders and as simple skeletal network as possible. To achieve these objectives the first regularization term is used at the beginning of the training to force some weights to become sufficiently small to removed them. The second regularization term, scaled by $\lambda_2$, is a sum over all weights and has minimum (zero) for weights approaching zero or $\pm 1$. The first term is switched off and the second increased in the second stage of the training. This allows the network to increase the remaining weights and together with increasing slopes of sigmoids to provide sharp decision borders.

The sixth-order regularization term in the cost function may be replaced by one of the lower order terms

$$\begin{aligned} &|W_{ij}||W_{ij}^2 - 1| \text{ cubic} \\ &|W_{ij}| + |W_{ij}^2 - 1| \text{ quadratic} \\ &\sum_{k=-1}^{+1} |W_{ij} + k| - \left|W_{ij} - \tfrac{1}{2}\right| - \left|W_{ij} + \tfrac{1}{2}\right| - 1. \end{aligned} \quad (11)$$

These extra terms lead to the additional change of weights in the backpropagation procedure, for example for the sixth-order term

$$W_{ij} \leftarrow \lambda_1 W_{ij} + \lambda_2 W_{ij}(W_{ij}^2 - 1)(3W_{ij}^2 - 1). \quad (12)$$

Although nonzero weights have values restricted to $\pm 1$ increasing the slopes $\beta$ is equivalent to using one, large nonzero weight value $\pm W$. One could consider several different maximal values of $W$ in the final network, for example by adding, after skeletonization of the network, the following penalty term:

$$\sum_{i,j} (\sigma(W_{ij} + 1) - \sigma(W_{ij} - 1)). \quad (13)$$

This term will not restrict the weights to $\pm 1$ but will allow them to grow beyond these values. We have not explored yet this possibility because at the end of the training the slopes should be infinitely steep, corresponding to infinite nonzero weights. Such approach may be interesting if the final goal is a hybrid, network-rule based system.

Introduction of integer weights may also be justified from the Bayesian perspective [58], [59]. The cost function specifies our prior knowledge about the probability distribution $P(W|M)$ of the weights in our model $M$. For classification tasks, when crisp logical decisions are required, the prior probability of the weight values should include not only small weights, but also large positive and negative weights distributed around $\pm 1$. For example

$$\begin{aligned} P(W|M) &= Z(\alpha)^{-1} e^{-\alpha E(W|M)} \\ &\propto \prod_{ij} e^{-\alpha_1 W_{ij}^2} \prod_{ij} e^{-\alpha_2 |W_{ij}^2 - 1|} \end{aligned} \quad (14)$$

where the parameters $\alpha_i$ play a similar role for probabilities as the parameters $\lambda_i$ for the cost function. Using alternative cost functions amounts to different priors for regularization, for example using Laplace instead of the Gaussian prior. Initial 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.

The training proceeds separately for each output class. Although the method works with general multilayer backpropagation networks we recommend the C-MLP2LN constructive procedure that frequently leads to satisfactory solutions in a much faster way. As with all neural procedures for some data the network training may slow down and require some experimentation. Initially several constructive networks should be trained without regularization to determine the expected training error and the average number of epochs needed for convergence. Below typical values of parameters that work well in most cases are given.

1) Create one hidden neuron (R-unit neuron).
2) Train the neuron on data for the first class using backpropagation procedure with regularization. Start with small $\lambda_1 = 10^{-5}$ and $\lambda_2 = 0$ and the unit slope $\sigma(x/T)$, $T = 1$.
3) If convergence is slow (for example, for 10% of the maximum number of training epochs the decrease of the error is lower than $1/n$, where $n$ is the number of the training samples) try training two neurons simultaneously; in rare cases training more than two neurons simultaneously may significantly speed up the training.
    a) Train as long as the error decreases; then increase $\lambda_1 \leftarrow 10\lambda_1$ and the slope of sigmoidal functions $T \leftarrow T + 1$ and train further; repeat this step until sharp increase of the error (typical more than five times) is noticed when $\lambda_1$ is increased.
    b) Decrease $\lambda_1$ slightly until the error is reduced to the previous value and train until convergence.
    c) Remove weights smaller than $|W| < 0.1$.
    d) Take $\lambda_2 = \lambda_1$ and $\lambda_1 = 0$ and train slowly increasing the slopes and $\lambda_2$ until the remaining weights reach $0 \pm 0.05$ or $\pm 1 \pm 0.05$.
    e) Set very large slopes $T \approx 1000$ and integer weights $0 \pm 1$.
4) Analyze the weights and the threshold(s) obtained by checking the combinations of linguistic features that activate the first neuron(s). This analysis (see Section VI for an example) allows to write the first group of logical rules that cover the most common input–output relations.
5) Freeze the weights of existing neurons during further training. This is equivalent to training only new neurons (usually one per class at a time) on the data that has not been properly handled so far.
6) Add the next neuron and train it on the remaining data in the same way as the first one. Connect it to the output neuron for the class it belongs to.
7) Repeat this procedure until all data are correctly classified, or the number of rules obtained grows sharply, sig-

nifying overfitting (for example, one or more rules per one new vector classified correctly are obtained).

8) Repeat the whole procedure for data belonging to other classes.

Thus the network expands after a neuron is added and then shrinks after connections with small weights are removed. A set of rules $\mathcal{R}_1 \vee \mathcal{R}_2 \ldots \vee \mathcal{R}_n$ is found for each class separately. 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 (Fig. 3). Output neurons performing summation of the incoming signals are linear and have either positive weight $+1$ (adding more rules) or negative weight $-1$. The last case corresponds to those rules that cancel some of the errors created by the previously found rules that were too general. They may be regarded as exceptions to the rules.

Since each time only one neuron per class is trained the C-MLP2LN training is fast. Both standard MLP architecture with linguistic inputs or the L-R network may be used with the C-MLP2LN approach. Since the first neuron for a given class is trained on all data for that class the rules it learns are most general, covering the largest number of instances. Therefore rules obtained by this algorithm are ordered, starting with rules that have the largest coverage and ending with rules that handle only a few cases. This order allows for a very easy check of the quality of a set of rules by looking at the errors on the training 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 set to $\pm 1$. However, some rules obtained from analysis of the network may involve spurious conditions and therefore the optimization and simplification step is necessary (cf. Section VII).

Although constraints (10) do not change the MLP exactly into a logical network they are sufficient to facilitate logical interpretation of the final network function. $\lambda_1$ and $\lambda_2$ parameters determine the simplicity/accuracy tradeoff of the generated network and extracted rules. If a very simple network (and thus simple logical rules) is desired, giving only rough description of the data, $\lambda_1$ should be as large as possible: although one may estimate the relative size of the regularization term versus the mean square error (MSE) a few experiments are sufficient to find the largest value for which the MSE is still acceptable and does not decrease quickly when $\lambda_1$ is decreased. Smaller values of $\lambda_1$ should be used to obtain more accurate networks (larger sets of rules). The final value of $\lambda_2$ near the end of the training may grow larger than the maximum value of $\lambda_1$.

The only way to change MLP into a logical network is by increasing the slope of sigmoidal functions to infinity, changing them into the step-functions. Such a process is difficult since a very steep sigmoid functions leads to the nonzero gradients only in small regions of the feature space, and thus the number of vectors contributing to the learning process goes to zero. Therefore when convergence becomes slow for large slopes it is necessary to stop network training, extract logical rules and optimize the intervals of the linguistic variables. This optimization step, described in Section VII, is performed at the level of the rule-based classifier, not the MLP network. A direct method to obtain logical MLP network is described below.

### B. Search-Based MLP

Minimization and search methods share the same goal of optimizing some cost functions. Quantization of network parameters (weights and biases) allows to replace minimization by search. Increasing step by step the resolution of quantization from coarse to fine allows to find the network parameters with arbitrary precision. Search-based optimization allows to use step-like discontinuous transfer functions as well as any smooth functions. Replacing the gradient-based backpropagation training methods by global search algorithm to minimize the value of the error function is rather expensive, therefore some form of a heuristic search should be used, for example the best first search or the beam search [60]. Even if the best first search algorithm is used (corresponding to the steepest gradient descent) a good solution may be found gradually increasing the resolution of the discrete network parameters [61]. In backpropagation training this would roughly correspond to a period of learning with rather large learning constants, with some annealing schedule for decreasing the learning constant.

Given a network architecture the algorithm starts with all weights $W_{ij} = 0$ and biases $\theta_i = -0.5$, so that all data is assigned to the default class (corresponding to zero network output). At the beginning of the search procedure the step value $\Delta$ for weights (and biases) is set. This value is added or subtracted from weights and biases, $W_{ij} \pm \Delta$, $\theta_i \pm \Delta$. This significantly reduces the search space. The best first and the beam search strategies are used to modify one parameter at a time. Since computer experiments showed that sometimes such search is not sufficient computationally more demanding variants of the search methods modifying two weights at a time may be used. To speed up the search they are performed in two stages. First, all the single changes of parameters are tested and a number of the most promising changes (i.e., changes decreasing the value of the cost function) is selected (the beam width). Second, all pairs of parameter changes from the chosen set, or even all the subsets of this set, are tested, and the best combination of changes applied to the network. Since the first stage reduces the number of weights and biases that are good candidates for updating the whole procedure is computationally efficient.

The search-based training procedure is an interesting alternative to the gradient-based backpropagation training [61]. Adding some constraints to the optimized cost function can produce networks easily convertible to crisp logical rules or fuzzy logical rules with soft trapezoidal membership functions obtained by subtracting two sigmoidal functions (5). If all the weights are integers (which is the case when $\Delta = 1$) and the hidden neuron transfer function is sufficiently steep, then the resulting network can easily be converted to a set of $M$-of-$N$ rules. The rules are generated by simple analysis of network parameters. All the input combinations are checked and if their sum exceeds appropriate bias a logical rule is

generated. To obtain small number of conjunctive logical rules the space of weight values is searched assuming that biases are always equal to the sum of the incoming weights minus 0.5, i.e., $\theta_i = \sum_j |W_{ij}| - 0.5$. In such cases a single neuron is equivalent to just one logical rule, since only one combination of inputs gives a sum greater than the bias. For example, if the only nonzero weights for neuron 1 are $W_{11} = +1$, $W_{12} = -1$, the threshold is $+1.5$ and the rule is: IF $X_1 \wedge \neg X_2$ THEN True.

### C. Probability Density Networks

Although constructive C-MLP2LN algorithm and search based MLP method work very well, especially with the optimization of final rules described in Section VII, in complex cases FSM network with rectangular functions (or soft rectangular functions that are changed into rectangular during training) may be easier to use. FSM uses efficient clusterization procedures (based on dendrograms or decision trees) for initialization, frequently obtaining quite good results without any training (see [42], [53], [54] papers, where details of the training algorithm are described). Each network node covers a cluster of input vectors. The training procedure changes the node parameters (such as their positions in the input space) until the error function reaches a minimum. Nodes that cover only a few training vectors are removed and nodes that cover many training vectors are optimized.

The node that has the largest output most often when all training vectors are presented covers the largest number of input vectors. This node, assigned to a certain class $C_i$ (this is the class majority of the vectors it covers belong to), corresponds to the most general logical rule. The interval $[b_k, b'_k]$ for the selected node is adjusted to cover all $C_i$ class vectors that activate it. The value $b_k$ ($b'_k$) is set between the lowest (highest) value of the $x_k$ belonging to the training vectors of the $C_i$ class covered by this node and the $x_k$ value of the nearest vector from another class. Those features that cover the whole input data range are deleted since their contribution is always constant. For the remaining features further selection is done by checking the number of errors on vectors belonging to classes other than the class assigned to a given node. This procedure is repeated for all network nodes [54].

For radial membership functions, such as Gaussians, one could also use the RBF networks for extraction of crisp rules, although we are not aware of any papers in which the transition from Gaussian-like functions to rectangular function limit [for example by increasing exponent $n$ in $\exp(-x^{2n})$ function] has been studied.

### D. Rule Generation using Separability Criterion

SSV separability criterion defined in (9) has a natural application in construction of decision trees. The simplest method of building such a tree is to use the best first search method. The separability of each possible cut point of each continuous feature, or of each subset of the set of values of each discrete feature, is evaluated. The best splits are selected and the space (and dataset as well) is divided into two parts by the first two branches of the binary tree. The criterion is then applied recursively to each of the resulting parts of the input space (with their corresponding data subsets). The tree is finished when it classifies the data with maximal accuracy. 100% accuracy is possible only if there are no contradictory examples in the dataset.

The accuracy of 100% usually means overfitting. To avoid it a pruning technique is used **to maximize generalization** capacity of the resulting tree. Ten-fold crossvalidation for the training set is performed. In each crossvalidation pass unseen samples are used to find the best way to prune the tree. Leaves that lead to overfitting cannot be determined because the final tree may be quite different than the tree built for the training data available during crossvalidation (i.e., 90% of the data in ten-fold crossvalidation), since decision trees, as well as most other classifiers, are unstable [62]. Therefore an optimal *degree of pruning* is determined. Pruning with the degree of $n$ means cutting off all the pairs of leaves which reduce the number of errors of their parent by not more than $n$. In each pass of the crossvalidation the number of errors counted for the test part of the data is checked. The optimal degree of pruning is the maximal degree (natural number) corresponding to the minimal total crossvalidation test error (sum of all crossvalidation test errors).

Each step of the best first search grows the decision tree by splitting one of its leaves in two. So after each step we improve (or in the worst case preserve) the classification accuracy. It means that the best first search follows a single branch of the search tree: if at a given stage we choose the best split we will never try any alternative split although it can finally give much better (i.e., smaller) tree. To diminish this drawback we use beam search instead of best first search, capable of finding better results at a larger computational cost.

The decision tree is easily converted into a set of crisp logical rules (each branch of the tree represents one rule). However, the rules containing premises describing all the nodes from the root of the tree to its leaves can be more complex than necessary. Especially in bigger trees it may turn out that the decisions made at the very beginning are not important for classification of data vectors which end up in a leaf. They may be important for a large data set, but not necessarily for smaller, localized samples. Therefore redundant rule antecedents should be removed. To find out which premises are spurious they are deleted one by one and a check of the accuracy is made. If the accuracy is decreased the premise should be kept. We will refer to this method of generating rules as SSV, i.e., using the same name as for the separability criterion.

## VI. Extraction of Rules—Pedagogical Illustration

For pedagogical purposes we will illustrate the first steps of our methodology using the Fisher Iris dataset. The data has been taken from the UCI machine learning repository [7]. The Iris data has 150 vectors evenly distributed in three classes: 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 given in centimeters).

The simplest way to obtain linguistic variables, often used in design of fuzzy systems, is based on division of each feature data range into a fixed number of parts and use of the triangular (or similar) membership functions for each part [10]. The same approach may be used for crisp logic. Dividing the range of each

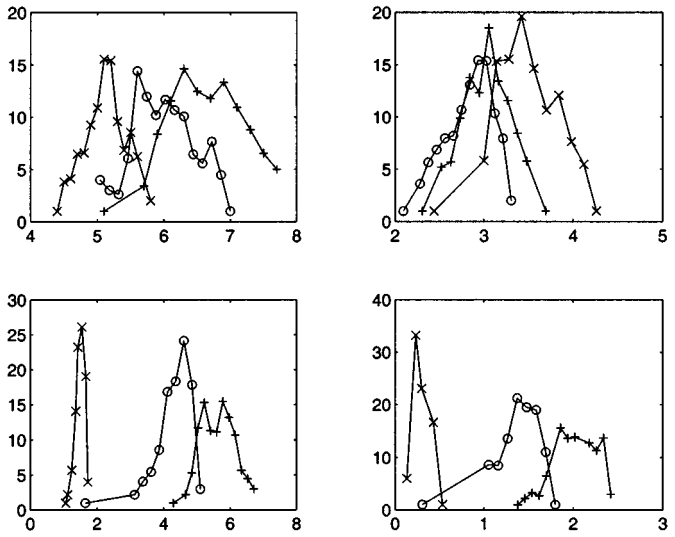

Fig. 4. Histograms of the four $x_1 - x_4$ Iris features. The $x_3$, $x_4$ features (lower part) allow for better discrimination than the first two features.

feature into three equal parts, called small $(s)$, medium $(m)$ and large $(l)$ the $x_1$ feature will be called small if it is in [4.3, 5.5] range, medium in (5.5, 6.7] and large in (6.7, 7.9]. Thus instead of four continuous-valued inputs a network with 12 binary inputs equal to $\pm 1$ is constructed. For example, the medium value of a single feature is coded by three input units $(-1, +1, -1)$. 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 some iris-virginica vectors and cannot be classified correctly. Therefore after discretization the maximum classification accuracy is 98.7%. Indistinguishable vectors should be removed from the training sequence.

Although there is no reason why such a procedure should provide good linguistic units for the Iris example by chance it is not so bad! The accuracy of classification using logical rules critically depends on selection of linguistic variables. Using two variables per feature, small and large, dividing the range of feature values in the middle, 13 vectors from Iris-setosa class get mixed with the vectors from two other classes. Using four linguistic variables per feature also decreases classification accuracy, mixing 16 Iris-versicolor cases with Iris-virginica. Evidently division into three classes is fortuitous. Analysis of the histograms of the individual features for each class, shown in Fig. 4 and Table II, proves that the division into three 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 (shown in Table II) was made by dividing the data range into 15 bins and smoothing these histograms by counting not only the number of vectors falling in a given bin, but also adding 0.4 to adjacent bins.

This discretization is quite useful for the initialization of L-units, although random initialization would, after some training, also lead to similar intervals. It may also be used for initialization of the FSM network nodes, although dendrogram-based methods work quite well. For the Iris case

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] |

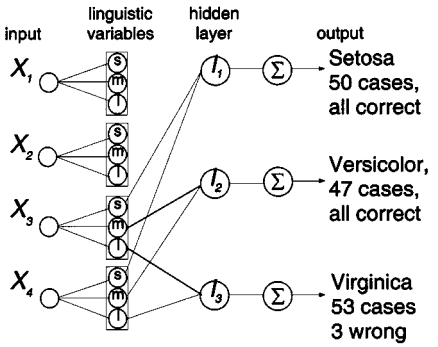


Fig. 5. Final structure of the network for the Iris problem.

dendrogram initialization with Gaussian nodes gives 95% correct answers without any training of the network or optimization of rules. The network has four nodes corresponding to four fuzzy rules. FSM initialization with rectangular functions gives 80% of correct answers and requires short training to improve the linguistic variable intervals [53].

A single neuron per class was sufficient to train the C-MLP2LN network, therefore the final network structure (Fig. 5) has 12 input nodes and three output nodes. Hidden nodes are only needed when more than one neuron is necessary to cover all the rules for a given class. The network was trained for about 1000 epochs and the final weights were within 0.05 from the desired $\pm 1$ or zero values. The following weights and thresholds for the three neurons were obtained (only the signs of the weights are written):

$$\begin{array}{lllll} \text{Setosa} & (0,0,0 & 0,0,0 & +,0,0 & +,0,0) \quad \theta = 1 \\ \text{Versicolor} & (0,0,0 & 0,0,0 & 0,+,0 & 0,+,0) \quad \theta = 2 \\ \text{Virginica} & (0,0,0 & 0,0,0 & 0,0,+ & 0,0,+) \quad \theta = 1. \end{array}$$

These weight vectors are so simple that there is no need for rule extraction. The corresponding rules are:

Iris-setosa if $x_3 = s \vee x_4 = s$;
Iris-versicolor if $x_3 = m \wedge x_4 = m$;
Iris-virginica if $x_3 = l \vee x_4 = l$.

Only two features, $x_3$ and $x_4$, are relevant since all weights for the remaining features become zero. The trained network structure is shown in Fig. 5. The first rule correctly classifies all samples from the Iris-setosa class. Together with the other two rules 147 vectors (98%) are correctly classified using only the $x_3$ and $x_4$ features.

Linguistic variables were not optimized in the example above. As a result the solution obtained is rather brittle (Fig. 6)—the decision borders are placed too close to the data.

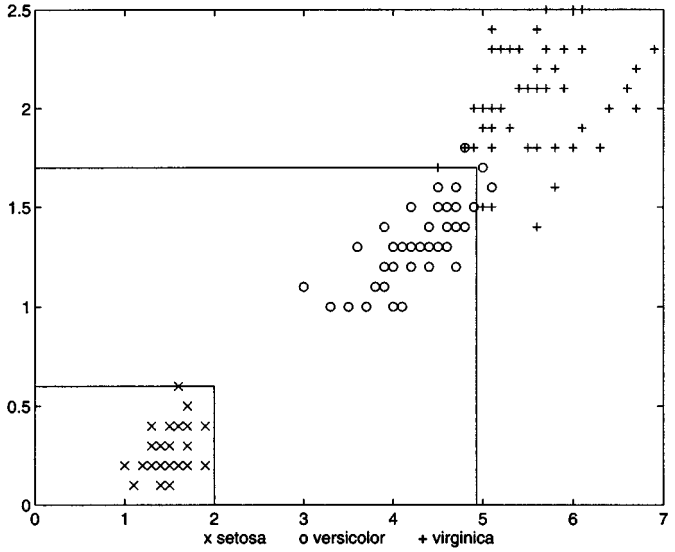


Fig. 6. Iris dataset displayed in $x_3$ and $x_4$ coordinates; decision regions (rules) for the three classes are also shown. Note the three Iris-versicolor cases that are incorrectly classified using these two features only. The brittleness of rules is illustrated by decision border that is placed too close to the setosa class.

Using L-R network several solutions with optimized linguistic variables are found, depending on the regularization parameters $\lambda$. The simplest rules involve only one attribute, petal length $x_3$.

$\mathcal{R}_1^{(1)}$: iris-setosa if $x_3 < 2.5$ $(100\%)$;
$\mathcal{R}_2^{(1)}$: iris-virginica if $x_3 > 4.8$ $(92\%)$;
$\mathcal{R}_3^{(1)}$: else iris-versicolor $(94\%)$

The first rule is accurate in 100% of cases since the setosa class is easily separated from the two other classes. The overall accuracy is 95.3% (seven errors). Slightly more accurate rules (96%) are obtained for smaller regularization parameters:

Iris-setosa if $x_3 \leq 2.56$;
Iris-virginica if $x_4 > 1.63$;
Iris-versicolor otherwise.

Similar solutions are found with search-based MLPs. All these rules are more robust than those obtained with linguistic variables from histograms. SSV criterion has found another simple set of rules, offering 98% accuracy:

Iris-setosa if $x_4 < 0.8$;
Iris-virginica if $x_3 > 4.95 \wedge x_4 > 1.65$;
Iris-versicolor otherwise.

What about more complex solutions? Using $\lambda_1 = 0$ and small value of $\lambda_2$ the following weights and thresholds are found:

| | | | | | |
|---|---|---|---|---|---|
| Setosa | $(+, 0, 0$ | $0, 0, +$ | $+, -, 0$ | $+, -, -)$ | $\theta = 2$ |
| Versicolor | $(0, 0, 0$ | $0, 0, 0$ | $0, +, -$ | $0, +, -)$ | $\theta = 3$ |
| Virginica | $(0, 0, 0$ | $0, 0, 0$ | $-, -, +$ | $-, -, +)$ | $\theta = 1.$ |

To analyze these vectors note that in MLP2LN or in search-based MLPs with discretized network parameters 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 $V_s \cdot W_s$. To find all rules that are compatible with a given set of weights and thresholds one has to perform a search process, considering combinations of all inputs to the activation of the network node. Since MLP2LN method guarantees that only relevant inputs have nonzero weights the search space has $2^n$ elements, where $n$ is the number of used features.

TABLE III
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 | | |

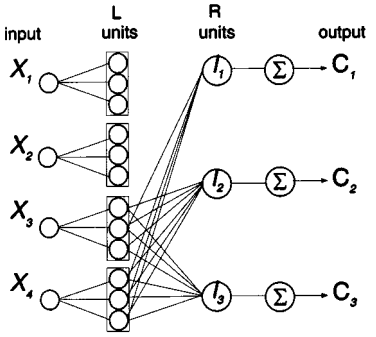


Fig. 7. Structure of the network trained with $\lambda_1 = 0$ on the Iris problem.

For the Iris-setosa vectors the weights for the first feature are $(+, 0, 0)$, therefore contribution from $x_1 = s$ is $\Delta = +1$. 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 summarized in the Table III and the structure of the network for this case is shown in Fig. 7.

Using Table III one can easily create a search tree (Fig. 8) with weights equal to the total contribution of each feature to the final activation. At the first level there are two branches, at the second level also two, for $x_3$ it is three and for $x_4$ it is two, giving a total of 24 leaves. At the first level contribution of $x_1$ is $+1$ for $x_1 = s$ or $-1$ for $x_1 = \neg s$. 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. Changing the order in which the levels are considered equivalent rules are obtained. A useful heuristic to find the simplest set of rules is to start with features that contribute the most to the activation (features four and three in this case). As shown in Fig. 8, if $x_4 = s$ the activation $\Delta$ is already three and if it is followed by $x_3 = s$ the activation $\Delta = 5$ and the two other features will not reduce the activation below three (since each may subtract at most one). 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

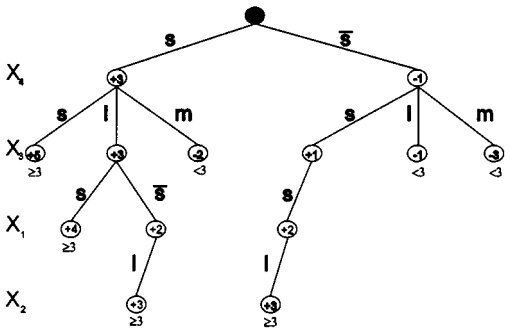


Fig. 8. Tree-based search for rules after network has been trained.

antecedents for class Iris-setosa, one rule for Iris-versicolor and one for Iris-virginica

$$\begin{aligned}&\text{IF } (x_3 = s \wedge x_4 = s)\\ &\quad \vee (x_1 = s \wedge x_3 = l \wedge x_4 = s)\\ &\quad \vee (x_1 = \neg s \wedge x_2 = l \wedge x_3 = l \wedge x_4 = s)\\ &\quad \vee (x_1 = s \wedge x_2 = l \wedge x_3 = s \wedge x_4 = \neg s)\\ &\qquad \text{THEN iris-setosa}\\ &\text{IF } (x_3 = m \wedge x_4 = m) \qquad \text{THEN iris-versicolor}\\ &\text{IF } (x_3 = l) \vee (x_4 = l) \qquad \text{THEN iris-virginica.}\end{aligned} \tag{15}$$

These rules allow for correct classification of 147 vectors, achieving the highest theoretical accuracy (98%) for the histogram discretization. Comparing them to simpler rules of the same accuracy presented above it is clear that they are too complex. Large thresholds may simplify the rule extraction process, leading to simpler search trees. One could implement additional conditions in the MLP2LN algorithm to encourage such large thresholds, but we have not tested this option yet, although we use it in SSV and search-based MLPs. The validity of all rules presented here has been confirmed with a Prolog program, which is also used to search for rules in complex cases.

Density networks provide logical rules without the need to check the combination of linguistic features. An FSM node implementing rectangular transfer function has the intervals defined for each relevant feature and is equivalent to a conjunctive rule. Using Gaussian or other soft transfer functions has direct interpretation in form of fuzzy rules, and the transition process between fuzzy and crisp rules may be studied by increasing the slopes of sigmoidal functions combined to create bicentral transfer function (5).

It is impossible to estimate statistical accuracy of the logical rules in cross-validation tests since for each training data set a different set of rules is obtained. Comparison of accuracy on datasets with separate training and test parts is done in Section X.

## VII. Optimization and Reliability of Rules

Rules obtained from analysis of neural networks or decision trees may involve spurious conditions, more specific rules may be contained in general rules or logical expressions may be simplified if written in another form. Therefore an important part of rule optimization involves simplification and symbolic operations on rules. We use a Prolog program for such simplifications. In addition optimal linguistic variables for continuous-valued features may be found for the sets of rules extracted. These optimized linguistic variables may be used to extract better rules in an iterative process, starting from initial values of linguistic variables, extracting logical rules, optimizing linguistic variables, and repeating the whole process with new linguistic variables until convergence is achieved. Usually two or three iterations are sufficient to stabilize the sets of rules.

Optimal linguistic variables (intervals) and other adaptive parameters may be found by maximization of a predictive power of a rule-based (or any other) classifier. Let $\mathcal{F}(C_i, C_j|M)$ be the confusion matrix, i.e., the number of instances in which class $C_j$ is predicted when the true class was $C_i$, given some parameters $M$. Then for $n$ samples $p_M(C_i, C_j) = \mathcal{F}(C_i, C_j|M)/n$ is the probability of (mis)classification. The best parameters $M$ are selected by maximizing the number (or probability) of correct predictions (called also the "predictive power" of rules)

$$\max_M [\mathrm{Tr}\mathcal{F}(C_i, C_j|M)] \tag{16}$$

over all parameters $M$, or minimizing the number of wrong predictions [possibly with some risk matrix $\mathbf{R}(C_i, C_j)$]

$$\min_M \left[\sum_{i\neq j} \mathbf{R}(C_i, C_j)\mathcal{F}(C_i, C_j|M)\right]. \tag{17}$$

Weighted combination of these two terms:

$$E(M) = \gamma \sum_{i\neq j} \mathcal{F}(C_i, C_j|M) - \mathrm{Tr}\mathcal{F}(C_i, C_j|M) \tag{18}$$

is bounded by $-n$ and should be minimized over parameters $M$ without constraints. For discontinuous cost function $E(M)$ this minimization may be performed using simulated annealing or multisimplex global minimization methods. If $\gamma$ is large the number of errors after minimization may become zero but some instances may be rejected (i.e., rules will not cover the whole input space). Thus optimization of the cost function $E(M)$ allows to explore the **accuracy-rejection rate tradeoff**.

Since rules discriminate between instances of one class and all other classes one can define a cost function for each rule separately

$$E_R(M) = \gamma(\mathcal{F}_{+-} + \mathcal{F}_{-+}) - (\mathcal{F}_{++} + \mathcal{F}_{--}) \tag{19}$$

and minimize it over parameters $M$ used in the rule $\mathcal{R}$ only ($+$ means here one of the classes, and $-$ means all other classes). The combination $\mathcal{F}_{++}/(\mathcal{F}_{++} + \mathcal{F}_{+-}) \in (0, 1]$ is sometimes called the sensitivity of a rule [75], while $\mathcal{F}_{--}/(\mathcal{F}_{--} + \mathcal{F}_{-+})$ is called the specificity of a rule. Some rule induction methods optimize such combinations of $\mathcal{F}_{x,y}$ values.

**Estimation of the reliability** of rules is very important in many applications. Tests of classification accuracy should be

performed using stratified ten-fold crossvalidation, each time including rule optimization on the training set. Changing the value of $\gamma$ will produce a series of models with higher and higher classification accuracy at the expense of growing rejection rate. A set of rules may classify some cases 100% correctly for all data partitionings; if some instances are not covered by this set of rules another set of rules of lower accuracy is used (the accuracy of rules is estimated on the training set only). High accuracy rules should give more confidence that they are reliable.

Most rule extraction procedures give only one set of rules, assigning to each rule a confidence factor, for example $c_i = p_M(C_i, C_i)/\sum_j p_M(C_i, C_j)$. This is rather misleading. A rule $\mathcal{R}^{(1)}$ that does not make any errors on the training set covers typical instances and its reliability is close to 100%. If a less accurate rule $\mathcal{R}^{(2)}$ is given, for example classifying correctly 90% of instances, the reliability of classification for instances covered by the first rule is still close to 100% and the reliability of classification in the border region $\mathcal{R}^{(2)} \backslash \mathcal{R}^{(1)}$ [cases covered by $\mathcal{R}^{(2)}$ but not by $\mathcal{R}^{(1)}$] is much less than 90%. Including just these border cases gives much lower confidence factors and since the number of such cases is relatively small the estimate itself has low reliability. A possibility sometimes worth considering is to use a similarity-based classifier (such as the $k$-NN method or RBF network) to improve accuracy in the border region. This may be useful if the optimal classification borders have complex shape that logical rules are not able to approximate.

Logical rules, similarly as any other classification systems, may become **brittle** if the decision borders are placed too close to the data vectors instead of being placed between the clusters (cf. Fig. 6). The brittleness problem is solved either at the optimization stage by selecting the middle values of the intervals for which best performance is obtained or, in a more general way, by adding noise to the data. Using the first method one determines the largest cuboid (in the parameter space) in which the number of errors is constant, starting from the values of the optimized parameters. The center of this cuboid is taken as the final estimation of the adaptive parameters. A better method to overcome the brittleness problem is presented in the next section.

## VIII. Probabilities From Crisp Rules

Neural systems have good generalization properties because they are wide margin classifiers. Their decision borders are obtained from the mean square error optimization of smooth function that extends over larger neighborhood contributing to the error. This allows for three important improvements: 1) the use of inexpensive gradient method instead of global minimization; 2) more robust rules with wider classification margins; 3) estimation of class probability, instead of 0–1 decisions.

Input values result usually from observations which are not quite accurate, therefore instead of the attribute value $x$ a Gaussian distribution $G_x = G(y; x, s_x)$ centered around $x$ with dispersion $s_x$ should be given. This distribution may be treated as a membership function of a fuzzy number $G_x$. To compute probabilities $p(C_i|X)$ a Monte Carlo procedure may be performed, sampling vectors from Gaussian distributions defined for all attributes. Analytical evaluation is based on the cumulative distribution function

$$\rho(a-x) = \int_{-\infty}^{a} G(y; x, s_x)\, dy = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{a-x}{s_x\sqrt{2}}\right)\right] \approx \sigma(\beta(a-x)) \tag{20}$$

where erf is the error function and $\beta = 2.4/\sqrt{2}s_x$ makes the erf function similar to the standard unipolar sigmoidal function with the accuracy better than 2%. A rule $R_a(x)$ with single crisp condition $x \geq a$ is fulfilled by a Gaussian number $G_x$ with probability

$$p(R_a(G_x) = T) = \int_{a}^{+\infty} G(y; x, s_x)\, dy \approx \sigma(\beta(x-a)). \tag{21}$$

Taking instead of the erf function a logistic function corresponds to an assumption about the error distribution of $x$ from Gaussian to $\sigma(x)(1-\sigma(x))$, approximating Gaussian distribution with $s^2 = 1.7$ within 3.5%. If the rule involves closed interval $[a, b]$, $a \leq b$ the probability that it is fulfilled by a sample from the Gaussian distribution representing the data is

$$p(R_{a,b}(G_x) = T) \approx \sigma(\beta(x-a)) - \sigma(\beta(x-b)). \tag{22}$$

Thus the probability that a given condition is fulfilled is proportional to the value of soft trapezoid function realized by L-unit. Crisp logical rules with assumption that data has been measured with finite precision lead to soft L-functions that allow to compute classification probabilities that are no longer binary. In this way we may either fuzzify the crisp logical rules or obtain fuzzy rules directly from neural networks. Crisp logical rules with the assumption of input uncertainties are equivalent to fuzzy rules with specific membership functions. The ease of interpretation favors crisp rules, while the accuracy and the possibility of application of gradient-based techniques to optimization favors fuzzy rules: we have the best of both worlds.

It is easy to calculate probabilities for single rule conditions of the form $x < a$, $x > a$ or $x \in (a, b)$

$$\begin{aligned} p(x < a) &= \int_{-\infty}^{a} G(y; x, s_x)\, dy \\ &= \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{a-x}{s_x\sqrt{2}}\right)\right] \\ p(x > a) &= \int_{a}^{+\infty} G(y; x, s_x)\, dy \\ &= \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{a-x}{s_x\sqrt{2}}\right)\right] \\ p(x \in (a, b)) &= \frac{1}{2}\left[\operatorname{erf}\left(\frac{b-x}{s_x\sqrt{2}}\right) - \operatorname{erf}\left(\frac{a-x}{s_x\sqrt{2}}\right)\right]. \end{aligned} \tag{23}$$

Notice that this interpretation does not differentiate between inequalities $\leq$ and $<$. To obtain reasonable probabilities rules with borders such that $\leq$ may be replaced by $<$ without loss of accuracy are required, i.e., borders should be placed between discrete values.

The probability that a vector $X$ belongs to a rule $R = r_1 \wedge \ldots \wedge r_N$ may be defined as the product of the probabilities of $X_i \in r_i$ for $i = 1, \ldots, N$. Such definition assumes that all the attributes which occur in rule $R$ are mutually independent, which is usually not the case. However, if a pair of strongly dependent attributes is used in linguistic variables that appear in a single rule one of these variables is dropped and the other reoptimized at the stage of rule simplification. Therefore the product should be very close to real probability. Obviously the rule may not contain more than one premise per attribute, but it is easy to convert the rules appropriately if they do not satisfy this condition.

Another problem occurs when probability of $X$ belonging to a class described by more than one rule is estimated. Rules usually overlap because they use only a subset of all attributes and their conditions do not exclude each other. Summing and normalizing probabilities obtained for different classes may give results quite different from real Monte Carlo probabilities. To avoid this problem probabilities are calculated as

$$p(C|X;\, M) = \sum_{R \in 2^{\mathcal{R}_C}} (-1)^{|R|+1} p\left(X \in \bigcap R\right) \tag{24}$$

where $\mathcal{R}_C$ is the set of the classification rules for class $C$, $2^{\mathcal{R}_C}$ is the set of all subsets of rules and $|R|$ is the number of elements in $R$. The probability $p(X \in \bigcap R)$ is calculated as a product of probabilities for single rule conditions according to (23) ($X \in \bigcap R$ may be treated as a single conjunctive rule). This formula takes care of overlapping rule regions, for example for two rules $R_1(X)$, $R_2(X)$ for class $C$ the probability $p(C|X;\, M)$ is $p(X \in R_1) + p(X \in R_2) - p(X \in R_1 \cap R_2)$.

Instead of the number of misclassifications the error function may include a sum over all probabilities

$$E(M,\, s_x) = \tfrac{1}{2} \sum_X \sum_i (p(C_i|X;\, M) - \delta(C(X),\, C_i))^2 \tag{25}$$

where

| | |
|---|---|
| $M$ | includes intervals defining linguistic variables; |
| $s_x$ | are Gaussian uncertainties of inputs; |
| $p(C_i\|X;\, M)$ | is calculated using (24). |

The confusion matrix computed using probabilities instead of the error counts allows for optimization of (18) using gradient-based methods. This minimization may be performed directly or may be presented as a neural network problem with a special network architecture.

Uncertainties $s_x$ of the values of features are additional adaptive parameters that may be optimized. We have used so far a very simple optimization with all $s_x$ taken as a percentage of the range of feature $x$ to perform one-dimensional minimization of the error function independently of other steps.

An alternative possibility that we have considered,[1] but not implemented yet, is to use the renormalized network outputs to compute probabilities

$$p(C_k|X) = \frac{o_k(X)}{\sum_i o_i(X)} \tag{26}$$

with output neurons for class $k$ summing the contributions of rule nodes

$$o_k(X) = \sigma\left(\sum_i R_{i,\,k}(X)\right). \tag{27}$$

Each of these rule nodes computes normalized products of L-unit outputs connected to it. Although results will not be equivalent to Monte Carlo simulations, $p(C_k|X)$ values behave like probabilities and may be useful.

This approach to soft optimization may be used with any set of crisp logical rules to overcome the brittleness problem and to obtain robust wide margin rule-based classifiers. Wide margins are desirable to optimize the placement of decision borders from generalization point of view. If a single parameter $s$ scaling all $s_x$ is used it may be hard to avoid an increase of the number of classification errors despite the fact that the overall probability of correct classification will increase. To avoid this problem a few iterative steps are used: after minimization $s$ is decreased and minimization repeated until $s$ becomes sufficiently small and probabilities almost binary. In the limit minimization of MSE becomes equivalent to minimization of the classification error, but the brittleness problem is solved because the intervals that are optimally placed for larger input uncertainties do not change in subsequent minimizations.

## IX. Optimization and Probabilities for Iris Data

In the MLP2LN method $\lambda_1$ and $\lambda_2$ constraint parameters allow to generate different sets of rules. If the L-R network architecture is used iterative optimization of linguistic variables is possible. The initial rules were derived in Section VI. The cost function in (18) allows for final optimization of linguistic variables. Fuzzy rules allow for direct gradient-based optimization. For crisp rules probabilities should be introduced first, as described in Section VIII, or nongradient optimization techniques should be used. Different values of the $\gamma$ and $\lambda_1$ parameters ($\lambda_2$ is not so important here) lead to a hierarchy of rules with increasing reliability.

This process is illustrated below on the Iris data. In the previous section the simplest set of rules $\mathcal{R}^{(1)}$ using only one feature, $x_3$, was found. Lowering the final hyperparameter $\lambda_1$ leads to the following set of rules:

$\mathcal{R}_1^{(2)}$: setosa if $(x_3 < 2.9 \vee x_4 < 0.9)$ (100%);
$\mathcal{R}_2^{(2)}$: versicolor if $(x_3 \in [2.9, 4.95] \wedge x_4 \in [0.9, 1.65])$ (100%);
$\mathcal{R}_3^{(2)}$: virginica if $(x_3 > 4.95) \vee (x_4 > 1.65)$ (94%).

The $\mathcal{R}^{(2)}$ set of rules classifies correctly 147 vectors, achieving the overall 98.0% accuracy. However, the first two rules have 100% reliability while all errors are due to the

[1]We are grateful to N. Jankowski for this idea.

third rule, covering 53 cases. Further decrease of constraint hyperparameters $\lambda$ allows to replace one of these rules 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. One cannot find more reliable rules this way.

100% reliability of all rules is achieved after optimization of $\mathcal{R}^{(2)}$ rules with increasing $\gamma \geq 0$ and minimizing Eq. (18). The smallest value of $\gamma$ for which all rules do not make any errors is found. For Iris this set of rules leaves 11 vectors, eight virginica and three versicolor, as unclassified.

$\mathcal{R}_1^{(3)}$: setosa if $(x_3 < 2.9)$ $(100\%)$;
$\mathcal{R}_2^{(3)}$: versicolor if $(x_3 \in [2.9,\ 4.9] \wedge x_4 < 1.7)$ $(100\%)$;
$\mathcal{R}_3^{(3)}$: virginica if $(x_3 \geq 5.3 \vee x_4 \geq 1.9)$ $(100\%)$.

The vectors rejected by $\mathcal{R}^{(3)}$ rules may be classified by $\mathcal{R}^{(2)}$ rules, but the reliability of classification for the vectors in the $\mathcal{R}^{(2)} \backslash \mathcal{R}^{(3)}$ border region is rather low: with $p = 8/11$ they should be assigned to the virginica class and with $p = 3/11$ to the versicolor class. It is possible to generate more specific rules, including more features, just for the border region, or to use in this region similarity-based classification system, such as $k$-NN, but for this small dataset we do not expect any real improvement since the true probability distributions of leave's sizes for the two classes of iris flowers certainly overlap.

The Iris example is too simple to see the full advantage of applying the optimization and probabilistic evaluation, since the number of parameters to optimize is small and optimal accuracy (98%) is achieved with crisp rules. For cases near the decision border between Iris virginica and Iris versicolor a more realistic probabilities $p(C|X;\ M)$ are calculated using formula (23). The natural uncertainties here are $\pm 0.1$, equal to the accuracy of measurements. Six vectors near that border have probabilities around 0.5, up to 0.75, the remaining vectors have higher probabilities. Calculation of probabilities was essential in our real-life application of rule extraction methods to psychometric data and NASA shuttle, presented below.

We have used the Iris example for pedagogical reasons only. Reclassification accuracy (in-sample accuracy for the whole dataset) of rules derived by several rule extraction systems are collected in Table IV. Unfortunately the statistical estimation of accuracy (out-of-sample accuracy) has not been given by the authors of these methods (such comparison is done on data with separate test parts). Nevertheless complexity and reclassification accuracy of rules found by different methods give some idea about their relative merits. The number of rules and conditions does not characterize fully the complexity of the set of rules, since fuzzy rules have additional parameters. "Else" condition is not counted as a separate rule.

The neurofuzzy NEFCLASS system [70] belongs to the best of its kind and if it had used context dependent linguistic variables it would probably achieve better results, but following the crowd the authors used three equally distributed fuzzy sets for each feature. The best seven fuzzy rules classified correctly 96.7% of data. The system is not able to reduce the number of features automatically, but if used with the last two iris features it will give the same performance using only three best rules (out of nine possible) with six conditions. Other neurofuzzy systems, such as FuNe–I [74], give even worse results. Kasabov [71] has used his neurofuzzy FuNN system partitioning each feature into five fuzzy linguistic variables, obtaining as a result 104 fuzzy rules with 368 conditions (for 150 data vectors)! Instead of compression of information that logical rules should provide a reverse process occurred. Ishibuchi *et al.* [66] report better results by combining several fuzzy systems and using various voting methods. Jagielska *et al.* [65] reports 100% reclassification accuracy with six genetically optimized weighted rules, which means that the data is overfitted and the method should give poor result in crossvalidation tests of classification accuracy. Unfortunately the main purpose of building rule-based systems, i.e., comprehensibility of data description, is lost in both cases.

TABLE IV
SUMMARY OF RULE EXTRACTION RESULTS FOR THE IRIS DATASET. F = FUZZY, C = CRISP, R = ROUGH, W = WEIGHTED

| Method | Rules/cond. features | Type | Reclassification accuracy |
|---|---|---|---|
| ReFuNN [10] | 9/26/4 | F | 95.7 |
| ReFuNN [10] | 14/28/4 | F | 95.7 |
| ReFuNN [10] | 104/368/4 | F | 95.7 |
| Grobian [72] | 118/-/4 | R | 100.0 |
| GA+NN [65] | 6/6/4 | W | 100.0 |
| NEFCLASS[70] | 7/28/4 | F | 96.7 |
| NEFCLASS[70] | 3/6/2 | F | 96.7 |
| FuNe-I[74] | 7/-/3 | F | 96.0 |
| C-MLP2LN | 2/2/1 | C | 95.7 |
| C-MLP2LN | 2/2/2 | C | 96.0 |
| C-MLP2LN | 2/3/2 | C | 98.0 |
| SSV | 2/2/2 | C | 98.0 |

Rough sets also do not produce comprehensible description of this simple data, producing a large number of rules. Grobian [72] uses 118 rules for perfect classification, clearly overfitting the data, reaching only 91–92% in ten-fold crossvalidation tests. Earlier application of rough sets to the Iris data [73] gave very poor results (77% accuracy), probably because four linguistic attributes per feature were used. This shows again the importance of optimization and the use of context-dependent linguistic variables instead of *ad hoc* partitions of input features. Thus even such a simple data seems to be difficult to handle for many rule extraction systems.

## X. ILLUSTRATIVE APPLICATIONS

We have analyzed a large number of datasets comparing our results with the results obtained by other methods whenever possible. Many results, including explicit logical rules, are collected at http://www.phys.uni.torun.pl/kmk/projects/rules.html. As we have already stressed, rules are useful if they are comprehensible and accurate. Although many sets of rules of various complexity have been found only the simplest and the most accurate sets of

rules are given here. They may be used as a reference or benchmark for other rule extraction systems.

Crossvalidation is useful as a measure of generalization capability since classifiers may overfit the training data. Such danger does not exist if a small number of simple rules is extracted. Accuracy on the training data should in such cases be similar as the accuracy on the test data and the differences tell us more about the statistical representativeness of the training and the test data than about the classification method itself (cf. results for larger datasets given below). Statistical tests, such as the stratified ten-fold crossvalidation or the leave-one-out tests, are difficult to perform since rules have to be extracted many times. Moreover, since different rules may be extracted for different data partitions it is impossible to present a single set of rules or to compare rules obtained by different methods.

The simplest form of rules is frequently quite stable when training on 90% of the data. In the mushroom case described below it is sufficient to use 10% of the data for training to find the first two rules that cover 99.4% of all the cases. During crossvalidation it may happen that the rare cases, covered by the two remaining rules, will be missing from the training part and thus the rules will not be found. Thus the averaged accuracy of the method will be below 100%, although the method is capable of finding 100% accurate rules for this data. Crossvalidation may not be so useful for evaluation of the rule extraction methods.

Quite frequently only the reclassification accuracy (in-sample or overall accuracy) on the whole dataset for extracted rules is quoted. This may not be sufficient to estimate statistical accuracy of rules, therefore in a few cases crossvalidation results are also given here. The best comparison of accuracy is offered on large dataset with the separate test part, such as the hypothyroid or the NASA shuttle problem. We have analyzed six databases with such separate test sets, allowing to judge generalization capability of the methods proposed in this paper.

Rule extraction methods should not be judged only on the basis of the accuracy of the rules but also on their simplicity and their comprehensibility. The simplest rules are usually rather stable in crossvalidation tests and for such rules reclassification accuracy is close to statistical estimations.

### A. Mushrooms

In the **mushroom problem** [1], [7] the database consists of 8124 vectors, each with 22 symbolic attributes with up to 12 different values, equivalent to 118 logical features. 51.8% of the cases represent 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. Using the C-MLP2LN algorithm with the cost function (10) the following disjunctive rules for poisonous mushrooms have been discovered:

$\mathcal{R}_1$) odor $=\neg$(almond $\vee$ anise $\vee$ none);
$\mathcal{R}_2$) spore-print-color $=$ green;
$\mathcal{R}_3$) odor $=$ none $\wedge$ stalk-surface-below-ring $=$ scaly $\wedge$ (stalk-color-above-ring $=\neg$brown);
$\mathcal{R}_4$) habitat $=$ leaves $\wedge$ cap-color $=$ white.

TABLE V
SUMMARY OF RULE EXTRACTION RESULTS FOR THE MUSHROOM DATASET; RECLASSIFICATION ACCURACY IS GIVEN IN PERCENTS

| Method | Rules/cond. features | Reclassification Accuracy |
|---|---|---|
| RULENEG[33] | 300/8087 | 91.0 |
| REAL [31] | 155/6603 | 98.0 |
| DEDEC [37] | 26/26 | 99.8 |
| TREX[1] | 3/13 | 100.0 |
| C4.5 (decision tree) | 3/3 | 99.8 |
| RULEX[44] | 1/3/1 | 98.5 |
| Successive Regulariz.[90] | 1/4/2 | 99.4 |
| Successive Regulariz.[90] | 2/22/4 | 99.9 |
| Successive Regulariz.[90] | 3/24/6 | 100.0 |
| C-MLP2LN, SSV | 1/3/1 | 98.5 |
| C-MLP2LN, SSV | 2/4/2 | 99.4 |
| C-MLP2LN | 3/7/4 | 99.9 |
| SSV | 3/7/4 | 99.9 |
| C-MLP2LN | 4/9/6 | 100.0 |
| SSV | 4/9/5 | 100.0 |

Rule $\mathcal{R}_1$ misses 120 poisonous cases (98.52% accuracy), adding rule $\mathcal{R}_2$ leaves 48 errors (99.41% accuracy), adding third rule leaves only eight errors (99.90% accuracy), and all rules $\mathcal{R}_1$ to $\mathcal{R}_4$ classify all poisonous cases correctly. 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 eight cases. We have also derived the same rules using only 10% of all data for training, therefore results from crossvalidation should be identical to the results given in Table V. This is the simplest systematic logical description of the mushroom dataset that we know of (some of these rules have probably been also found by the RULEX and TREX algorithms [1]) and therefore should be used as a benchmark for other rule extraction methods.

For the mushroom dataset SSV tree has found 100% accurate solution which can be described as four logical rules using only five attributes. The first of these is identical as found by the C-MLP2LN, but next two rules are different, using "gill-size" instead of stalk and cap related attributes. Since the last two rules cover only a small percentage of all cases many equivalent descriptions are possible. SSV rules give perhaps the simplest set of rules found so far.

$\mathcal{R}_1$: odor $=\neg$(almond $\vee$ anise $\vee$ none);
$\mathcal{R}_2$: spore-print-color $=$ green;
$\mathcal{R}_3$: gill-size $=$ narrow $\wedge$ (stalk-surface-above-ring $=$ (silky $\vee$ scaly) $\vee$ population $=$ clustered.

If odor is removed from the list of available features 13 rules are needed to reach 100% correct classification. This example

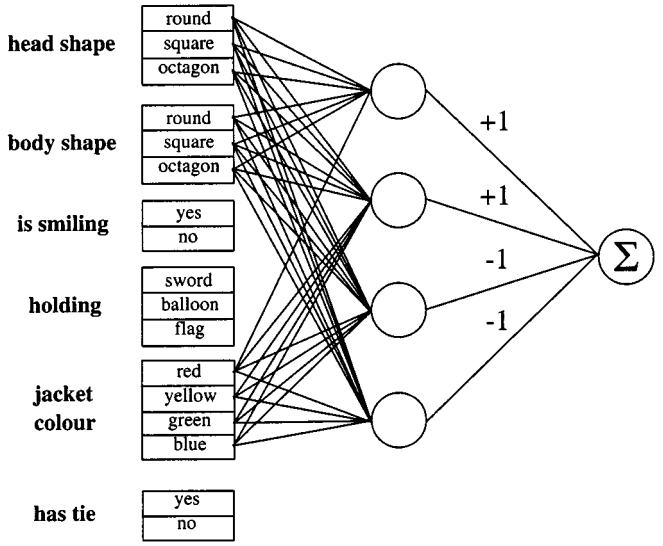


Fig. 9. The network for the Monk 1 problem. The first two neurons were taught simultaneously. The other two handle exceptions.

illustrates how important the simplicity of the rules is. Although neural and other methods may give a perfect solution logical rules derived here give probably the most comprehensible description of the data.

### *B. The Three Monk Problems*

The three monk problems are artificial, small problems designed to test machine learning algorithms [67], [1], [76]. Each of the three monks problems is to determine whether an object described by six features (shown in Fig. 9) is a monk or not. The three 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*).

There are 432 combinations of the six symbolic attributes. In the first problem 124 cases were randomly selected for the training set, in the second problem 169 cases and in the third problem 122 cases of which 5% were misclassifcations introducing some noise in the data. Such artificial data may be difficult to handle. Attempts to train the MLP2LN network with a single neuron were not successful—convergence was unacceptably slow and therefore the final error was too large. It was necessary to train two or more neurons in the hidden layer simultaneously. The number of neurons trained should be increased until convergence is fast (the definition of "fast" depends on the data, but it does not differ from evaluation of the convergence of standard MLP). After training on the Monk 1 data the weights for the two neurons were frozen (Fig. 9). This technique has also been used in the Monk 2 problem where up to four neurons were trained simultaneously (real data never required more than two neurons to be simultaneously trained).

Initial rules derived for the Monk problems were too general, i.e., each rule covered relatively large percentage of cases from a wrong class. The first two neurons in the Monk 1 problem classify properly all positive examples accepting some negative ones. The patterns which are not recognized properly are treated as exceptions to the general rules extracted from the network. The hidden layer had to be extended adding neurons with a negative contribution to the output node. After the whole rule extraction process is finished two separate sets of rules are obtained, one comprising information on positive examples, and the other describing exceptions, modifying the first set of rules. Below 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.

C-MLP2LN method applied to the Monk 1 problem needed three passes (one pass, or training stage, is a single process of training leading to convergence, finished with freezing the weights of all trained neurons). The two hidden neurons trained during the first pass recognized all the positive examples and 11 negative ones. In the second training pass one hidden neuron detected six exceptions and in the third pass another hidden neuron was taught the remaining five exceptions. Some statistics concerning all the stages of the algorithm for all three Monk problems is given in Table VI. 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 supports our claim that the method learns the most common rules first. The isolated cases are being recognized after subsequent stages.

In the Monk-1 problem four rules and two exceptions have been generated, altogether composed of 14 atomic formulas. They classify the training data without any errors.

Although the definition of the Monk-2 problem is very simple, the training process required much more effort. As shown in Table VI it needed the biggest number of passes of the algorithm. Each of the three first rule 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 trained in the final passes of our algorithm can specialize in recognizing patterns which do not resemble other patterns. Sixteen rules and eight exceptions were extracted from the resulting network. The number of atomic formulas which compose them is 132.

The third Monk problem also required one additional pass to find exceptions. Two neurons gave three rules, and two other neurons generated four exceptions. The whole logical system for this case contains 33 atomic formulas. Although the data has been deliberately contaminated with 5% noise it is well known [67] that rules giving 100% accuracy may be found.

TABLE VI
TRAINING STAGES IN THE 3 MONK PROBLEMS

| Problem | Pass No. | Neurons | Rules/Exc. | Examples |
|---|---|---|---|---|
| **Monk 1** | **1** | **2** | **rules** | **42** |
| 4 rules | *2* | *1* | *exceptions* | *6* |
| 2 exceptions | *3* | *1* | *exceptions* | *5* |
| **Monk 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** |
| **Monk 3** | **1** | **1** | **rules** | **57** |
| 3 rules | *2* | *2* | *exceptions* | *5* |
| 4 exceptions | **3** | **1** | **rules** | **3** |

FSM fuzzy rules obtained with Gaussian membership functions were not so good as the crisp rules from C-MLP2LN. For the Monk 1 problem 16 rules were generated, giving 97.9% accuracy on the training and 94.5% accuracy on the test set. For Monk 2 the number of generated rules was 32, and the accuracy was 94% on training and only 79.3% on the test set. Fifteen rules generated for Monk 3 gives 96.7% on the training and 95.5% on the test set. Soft decision borders are not appropriate for this problem, therefore fuzzy methods will not be as accurate as crisp rule extraction. Results obtained with many machine learning rule-based systems described in the original paper on the three monk problems [67] are compared in Table VII.

### C. The Appendicitis Data

The appendicitis data contains only 106 cases, with eight attributes (results of medical tests), and two classes: 88 cases with acute appendicitis and 18 cases with other problems. For this small dataset very simple classification rules have been found by Weiss and Kapouleas [80] using their predictive value maximization (PVM) approach. Since PVM makes exhaustive search testing all possible simple rules we may be sure that this is indeed the simplest solution. Using histograms for the two classes initial linguistic variables were found. Initially two simple rules have been found [81]

$$\text{MNEA} > 6650 \vee \text{MBAP} > 12. \tag{28}$$

The overall accuracy of these rules is 91.5%. Since these are essentially the same rules as found by Weiss and Kapouleas [80] using their PVM approach the leave-one-out accuracy should also be close to 89.6%. Rules are rather robust and do not change much if a single vector is removed from the training set in the leave-one-out procedure. Although we have improved classification accuracy by generating two more rules (adding a second neuron) the first of these rules covers just two cases and the second just one case. Such rules are more likely due to the noise

TABLE VII
COMPARISON OF RESULTS FOR THE 3 MONK DATASETS, ACCURACY ON THE TEST SET IN %

| Method | Monk-1 | Monk-2 | Monk-3 |
|---|---|---|---|
| AQ17-DCI | 100 | 100 | 94.2 |
| AQ17-HCI | 100 | 93.1 | 100 |
| AQ17-GA | 100 | 86.8 | 100 |
| Assistant Pro. | 100 | 81.5 | 100 |
| mFOIL | 100 | 69.2 | 100 |
| ID5R | 79.7 | 69.2 | 95.2 |
| IDL | 97.2 | 66.2 | - |
| ID5R-hat | 90.3 | 65.7 | - |
| TDIDT | 75.7 | 66.7 | - |
| ID3 | 98.6 | 67.9 | 94.4 |
| AQR | 95.9 | 79.7 | 87.0 |
| CN2 | 100 | 69.0 | 89.1 |
| AQR | 95.9 | 79.7 | 87.0 |
| CLASSWEB 0.10 | 71.8 | 64.8 | 80.8 |
| CLASSWEB 0.15 | 65.7 | 61.6 | 85.4 |
| CLASSWEB 0.20 | 63.0 | 57.2 | 75.2 |
| PRISM | 86.3 | 72.7 | 90.3 |
| ECOWEB, ext | 82.7 | 71.3 | 68.0 |
| Neural methods | | | |
| MLP | 100 | 100 | 93.1 |
| MLP+regularization | 100 | 100 | 97.2 |
| Cascade Corelation | 100 | 100 | 97.2 |
| FSM, fuzzy rules | 94.5 | 79.3 | 95.5 |
| SSV, crisp rules | 100 | 80.6 | 97.2 |
| C-MLP2LN rules | 100 | 100 | 100 |

in the data then to a highly specific and rare case of interest to an expert. Using L-units and random MLP initialization another set of rules giving 89.6% of accuracy has been found

$$\text{WBC1} > 8400 \vee \text{MBAP} \geq 42 \tag{29}$$

with the confusion matrix $P = \left(\begin{smallmatrix} 84 & 10 \\ 1 & 11 \end{smallmatrix}\right)$. Here column labels are of the true class and row labels of the assigned class, i.e., one real appendicitis case was classified as "other problem" and ten "other problems" as appendicitis. For comparison [75] $k$-NN in the leave-one out test gives 82.1% and with optimization of distance function and $k$ accuracy is about 89%, MLP reaches about 86% and Bayes rule 83%. C4.5 decision tree gives three rules correctly covering 91.5% of all cases. For this case we would expect about the same accuracy in the leave-one-out tests from our C-MLP2LN rules, PVM rules and CART or C4.5 decision trees since these methods consistently generate similar rules for this dataset. Twelve fuzzy rules from FSM achieve 84.5% accuracy in the leave-one-out test, and in the ten-fold crossvalidation accuracy is only slightly lower, 84.3%.

TABLE VIII
RESULTS FOR THE APPENDICITIS DATASET; RECLASSIFICATION AND THE LEAVE-ONE-OUT ACCURACY ARE GIVEN IN PERCENTS

| Method | Reclassific. | L-one-out |
|---|---|---|
| SSV, 2 crisp rules (our) | 94.3 | 89.6 |
| C-MLP2LN, 1 neuron (our) | 91.5 | 89.6 |
| PVM [75] | 91.5 | 89.6 |
| MLP+backprop [75] | 90.2 | 85.8 |
| CART [75] | 90.6 | 84.9 |
| FSM, 12 fuzzy rules (our) | 92.5 | 84.4 |
| Bayes [75] | 88.7 | 83.0 |
| k-NN [75] | – | 82.1 |
| C-MLP2LN, 2 neurons (our) | 94.3 | – |

The decision tree built with SSV converted into logical rules gives just two rules per class. Because there are no "don't know" answers, only the rules for one of the cases need to be presented, the other class can be summarized using the ELSE condition. The first rule obtained using separability criterion gives 91.5% accuracy. The second one is already unreliable, covering only three additional data vectors and increasing the accuracy of reclassification to 94.3%.

$\mathcal{R}_1$: HNEA $< 7520.5 \wedge$ MBAP $< 12$;
$\mathcal{R}_2$: HNEA $\in$ (9543.5, 9997.5).

Statistical accuracy is of course lower. In the leave-one-out test rules differ only slightly for different runs, achieving 89.6% of accuracy. In the 10-fold crossvalidation tests (repeated ten times) SSV rules achieve on average 86.3% accuracy (best results +2.6% and worst −1.1%). In Table VIII results of different methods for this dataset are compared. Twelve fuzzy rules from FSM were derived using Gaussian membership functions. We have not made the leave-one-out test with the more complex C-MLP2LN rules, but the results should be close to 89.6% obtained with a single neuron and with SSV rules.

### *D. Hepatitis*

This is another small medical database from UCI [7], containing only 155 samples belonging to two different classes (32 "die" cases, 123 "live" cases). There are 19 attributes, 13 binary, and six attributes with six to eight discrete values. This data is quite "dangerous" to use, since it contains many missing values—for some features almost half of the vectors have missing values. Using averages of these missing values leads to very good, but quite useless results. For example, using L-units to generate linguistic variables we were able to find one rule for the "die" class, achieving overall 96.1% of accuracy

$$\text{age} \geq 30 \wedge \text{sex} = \text{male} \wedge \text{fatigue} = \text{no} \wedge \text{protime} \leq 50. \quad (30)$$

The confusion matrix (live, die) is now: $P = \left(\begin{smallmatrix} 120 & 3 \\ 3 & 29 \end{smallmatrix}\right)$.

This single rule is very accurate but it uses variable "protime" which is missing in almost half of the cases. Rules discovered using the C-MLP2LN method do not contain such misleading attributes [81]

$$\text{age} > 52 \wedge \text{bilirubin} > 3.5$$
$$\text{histology} = \text{yes} \wedge \text{ascites} = \text{no} \wedge \text{age} \in [30, 51].$$

These rules classify correctly 14 of the 32 vectors representing the "die" class, giving 88.4% accuracy for the reclassifcation of the whole dataset. Further efforts to add new neurons to classify the remaining data lead to a large number of rules which is a clear indication of data overfitting.

TABLE IX
RESULTS FROM THE 10-FOLD CROSSVALIDATION FOR THE HEPATITIS DATASET

| Method | Accuracy | Remarks |
|---|---|---|
| k-NN, k=18, Manhattan | 90.2± 0.7 | our result |
| FSM + rotations | 89.7 | our results |
| LDA | 86.4 | Ref. [88] |
| Naive Bayes | 86.3 | Ref. [88] |
| IncNet + rotations | 86.0 | Ref. [69] |
| QDA | 85.8 | Ref. [88] |
| 1-NN | 85.3 | Ref. [88] |
| ASR | 85.0 | Ref. [88] |
| FDA | 84.5 | Ref. [88] |
| LVQ | 83.2 | Ref. [88] |
| CART | 82.7 | Ref. [88] |
| MLP with BP | 82.1 | Ref. [88] |
| ASI | 82.0 | Ref. [88] |
| LFC | 81.9 | Ref. [88] |
| Deafult | 79.4 | |

The highest accuracy, 90.2 $\pm$ 0.7% was obtained using $k$-nearest neighbors method, with only slightly lower accuracy of 89.7% obtained from FSM generating fuzzy rules, using Gaussian membership functions and allowing for rotation. Other classification methods give slightly lower accuracy, for example CART decision tree giving only 82.7%, $k$-NN 85.5% (for $k = 1$) and linear discriminants analysis 86.4%. A majority classifier is correct in 79.4% of cases.

Considering that $k$-NN has rather small variance of 0.7% the differences between the two best methods and the rest are significant. The two best methods provide quite complex decision borders, perhaps indicating that classification using simple rules cannot be accurate in this case. One may still argue that logical rules are a reasonable way to approach such small datasets. Although statistical accuracy offered is lower rules give at least some guidance and allow for validation of the classification model by experts. Table IX shows the results from the 10-fold crossvalidation of the dataset.

### *E. The Ljubliana Cancer Data*

The Ljubliana cancer data [7] contains 286 cases, of which 201 are no-recurrence-events (70.3%) and 85 are

recurrence-events (29.7%). There are nine attributes, with two to 13 different values each. A single logical rule for the recurrence-events

$$\text{involved nodes} = \neg[0, 2] \wedge \text{Degree-malignant} = 3$$

with ELSE condition for the second class, gives over 77% accuracy in crossvalidation tests. Such simple rule cannot overfit the data and is found on any crossvalidation partition. Therefore the 10-CV accuracy is identical to reclassification accuracy. This rule is easy to interpret: recurrence is expected if the number of involved nodes is bigger than two and the cells are highly malignant. More accurate optimized rules:

$\mathcal{R}_1$: Degree malignant $= 3 \wedge$ breast $=$ left $\wedge$ node caps $=$ yes;

$\mathcal{R}_2$: (Degree malignant $= 3 \vee$ breast $=$ left) $\wedge$ age $= \notin$ [30–49] $\wedge$ tumor size $=$ [35–54].

give slightly higher reclassification accuracy, but no increase in crossvalidation. Since the dataset is small many different sets of rules may give similar accuracy. Using the separability split values (SSV) to generate linguistic variables one rule for the class of recurrence-events is obtained

$$\text{involved nodes} \neg[0, 2] \wedge \text{Degree-malignant} \in [2, 4]$$

achieving 76.2% accuracy in reclassification of the data. More complex set of three rules obtained using SSV gives 77.6% accuracy and in the ten-fold crossvalidation tests an average of 73.5% (worst result $-0.8\%$, best $+1.0\%$), i.e., only a few percent above the default value, indicating that rules are already too complex and overfit the data. Several machine learning methods give results below the default, as shown in Table X.

It would be hard to improve upon result of these simple rules, which are easily understood by anyone. We doubt that there is any more information in this dataset. Most methods give significantly lower accuracy using more complex models. For example, FSM with 33 fuzzy rules gives results that are only insignificantly better than the default accuracy. LERS [78], a machine learning technique based on rough sets, gave after optimization almost 100 "certain" rules and about the same number of "possible" rules, achieving accuracy that is below the majority rate. Although it may not be the limit of accuracy for rough set systems the number of rules produced by such systems is usually quite large, and thus the insight into the knowledge hidden in the data is lost. FSM generates 33 rules with Gaussian membership functions, achieving 71.4% accuracy on the test part and 85.4% accuracy on the training part. CART decision tree gave the best results, 77.1% in the crossvalidation tests. Since CART reclassification results are not much better little difference between reclassification and crossvalidation accuracy of the MLP2LN and SSV rules should be expected.

### F. The Cleveland Heart Disease Data

The Cleveland heart disease dataset [7] (collected at V.A. Medical Center, Long Beach and Cleveland Clinic Foundation by R. Detrano) contains 303 instances, with 164 healthy (54.1%) instances, the rest are heart disease instances of various severity.

TABLE X
10-FOLD CROSSVALIDATION AND RECLASSIFCATION ACCURACY IN % FOR THE LJUBLIANA CANCER DATASET

| Method | 10-fold CV | Remarks |
|---|---|---|
| C-MLP2LN, 1 rule | 77.1 | our result |
| CART, PVM | 77.1 | Ref. [75] |
| Naive Bayes rule | 75.9 | Ref. [75] |
| SSV, 3 rules | 73.5±0.9 | our result |
| FSM, 33 fuzzy rules | 71.4 | our result |
| MLP + BP | 71.5 | Ref. [75] |
| Default | 70.3 | |
| AQ15 | 66-72 | Ref. [86] |
| Weighted network | 68-73.5 | Ref. [87] |
| LERS (rough rules) | 69.4 | Ref. [78] |
| k-NN | 65.3 | Ref. [75] |
| | Reclassification | |
| Assistant-86 | 78.0 | Ref. [85] |
| C-MLP2LN, 2 rules | 78.0 | our result |
| SSV, 3 rules | 77.6 | our result |
| SSV, 1 rule | 76.2 | our result |

While the database has 76 raw attributes, only 13 of them are actually used in machine learning tests, including six continuous features and four nominal values. There are many missing values of the attributes. Results obtained with various methods for this data set are collected in Table XI.

After some simplifications rules derived by the C-MLP2LN approach are

$\mathcal{R}_1$: (thal $= 0 \vee$ thal $= 1) \wedge$ ca $= 0.0$ $(88.5\%)$;

$\mathcal{R}_2$: (thal $= 0 \vee$ ca $= 0.0) \wedge$ cp $\neq 2$ $(85.2\%)$.

These rules give 85.5% correct answers on the whole set and compare favorable with the accuracy of other classifiers. Three rules describing the Cleveland heart data obtained using SSV method are 85.8% accurate (the first rule containing alternative is counted as two rules)

$\mathcal{R}_1$: ca $= 0.0 \wedge$ (thal $= 0 \vee$ exang $= 0$);

$\mathcal{R}_2$: cp $\neq 2 \wedge$ slope $\neq 2$.

These rules are quite similar to rules generated by C-MLP2LN. Ten-fold crossvalidation using SSV method gives an average of 81.0 $\pm$ 0.8% accuracy and the leave one out results are about 1% better. Twenty four fuzzy rules were generated by FSM, achieving 93.4% on the training and 82.5 $\pm$ 1.6% on the test part. These results are lower than those discriminant analysis, perhaps indicating the need to provide rotated sharp decision borders.

### G. Wisconsin Breast Cancer Data

The Wisconsin cancer dataset [68] contains 699 instances, with 458 benign (65.5%) and 241 (34.5%) malignant cases. Each instance is described by the case number, nine attributes with integer value in the range 1–10 (for example, feature $f_2$ is "clump thickness" and $f_8$ is "bland chromatin") and a binary

TABLE XI
RESULTS FROM THE 10-FOLD CROSSVALIDATION FOR THE CLEVELAND HEART DISEASE DATASET

| Method | Accuracy | Ref. |
|---|---|---|
| k-NN, k=28, 7 features | 85.1±0.5 | our result |
| Linear Discriminant Analysis | 84.5 | [88] |
| Fisher LDA | 84.2 | [88] |
| Naive Bayes | 83.4 | [88] |
| Bayes (pairwise dependent) | 83.1 | [88] |
| LVQ | 82.9 | [88] |
| k-NN, k=27, Manhattan | 82.8±0.6 | our result |
| MLP+backprop | 81.3 | [88] |
| CART (decision tree) | 80.8 | [88] |
| Quadratic Discriminants | 75.4 | [88] |
| LFC, ASI, ASR decision trees | 74.4-78.4 | [88] |
| FSM, 27 fuzzy rules | 82.0 | our result |
| SSV, 3 rules | 81.8±1.6 | our result |

class label. For 16 instances one attribute is missing (it was replaced by an average value). This data has been analyzed in a number of papers (Table XII).

The simplest rules obtained for the malignant class using C-MLP2LN are

$$f_2 \geq 7 \vee f_7 \geq 6 \qquad (95.6\%).$$

These rules cover 215 malignant cases and ten benign cases, achieving overall accuracy (including the ELSE condition) of 94.9%. More complex network gave five disjunctive rules for the malignant cases, with benign cases covered by the ELSE condition:

$\mathcal{R}_1$: $f_2 < 6 \wedge f_4 < 4 \wedge f_7 < 2 \wedge f_8 < 5$ (100)%;
$\mathcal{R}_2$: $f_2 < 6 \wedge f_5 < 4 \wedge f_7 < 2 \wedge f_8 < 5$ (100)%;
$\mathcal{R}_3$: $f_2 < 6 \wedge f_4 < 4 \wedge f_5 < 4 \wedge f_7 < 2$ (100)%;
$\mathcal{R}_4$: $f_2 \in [6,8] \wedge f_4 < 4 \wedge f_5 < 4 \wedge f_7 < 2 \wedge f_8 < 5$ (100)%;
$\mathcal{R}_5$: $f_2 < 6 \wedge f_4 < 4 \wedge f_5 < 4 \wedge f_7 \in [2, 7] \wedge f_8 < 5$ (92.3)%.

The first four rules achieve 100% accuracy (i.e., they cover cases of malignant class only), the last rule covers only 39 cases, 36 malignant and three benign. The confusion matrix is: $P = \begin{pmatrix} 238 & 3 \\ 25 & 433 \end{pmatrix}$, i.e., there are three benign cases wrongly classified as malignant and 25 malignant cases wrongly classified as benign, giving overall accuracy of 96%. Optimization of this set of rules [(18)] gives

$\mathcal{R}_1$: $f_2 < 6 \wedge f_4 < 3 \wedge f_8 < 8$ (99.8)%;
$\mathcal{R}_2$: $f_2 < 9 \wedge f_5 < 4 \wedge f_7 < 2 \wedge f_8 < 5$ (100)%;
$\mathcal{R}_3$: $f_2 < 10 \wedge f_4 < 4 \wedge f_5 < 4 \wedge f_7 < 3$ (100)%;
$\mathcal{R}_4$: $f_2 < 7 \wedge f_4 < 9 \wedge f_5 < 3 \wedge f_7 \in [4, 9] \wedge f_8 < 4$ (100)%;
$\mathcal{R}_5$: $f_2 \in [3,4] \wedge f_4 < 9 \wedge f_5 < 10 \wedge f_7 < 6 \wedge f_8 < 8$ (99.8)%.

These rules classify only one benign vector as malignant ($\mathcal{R}_1$ and $\mathcal{R}_5$, the same vector), and the ELSE condition for the benign class makes six errors, giving 99.0% overall accuracy. In all cases features $f_3$ and $f_6$ (both related to the cell size) were not important and $f_2$ with $f_7$ were the most important.

TABLE XII
RESULTS FROM THE TEN-FOLD CROSSVALIDATION AND RECLASSIFICATION FOR THE WISCONSIN BREAST CANCER DATASET

| Method | Accuracy | Ref. |
|---|---|---|
| IncNet | 97.1 | [69] |
| k-NN | 97.0±0.12 | our result |
| Fisher LDA | 96.8 | [88] |
| MLP+backprop | 96.7 | [88] |
| LVQ | 96.6 | [88] |
| Bayes (pairwise dependent) | 96.6 | [88] |
| Naive Bayes | 96.4 | [88] |
| DB-CART | 96.2 | [79] |
| LDA | 96.0 | [88] |
| LFC, ASI, ASR dec. trees | 94.4-95.6 | [88] |
| CART (dec. tree) | 93.5 | [79] |
| Quadratic DA | 34.5 | [88] |
| FSM, 12 fuzzy rules | 96.5 | our result |
| SSV, 3 crisp rules | 96.3±0.2 | our result |
| Results from reclassification | | Rules/type |
| C-MLP2LN | 99.0 | 5, C, our result |
| C-MLP2LN | 97.7 | 4, C, our result |
| SSV | 97.4 | 3, C, our result |
| NEFCLASS | 96.5 | 4, F [70] |
| C-MLP2LN | 94.9 | 2, C, our result |
| NEFCLASS | 92.7 | 3, F [70] |

Using L-units four more accurate rules for the malignant class are created (their reliability is in parenthesis):

$\mathcal{R}_1$: $f_4 < 3 \wedge f_5 < 4 \wedge f_7 < 6 \wedge f10 = 1$ (99.5)%;
$\mathcal{R}_2$: $f_2 < 7 \wedge f_5 < 4 \wedge f_7 < 6 \wedge f10 = 1$ (99.8)%;
$\mathcal{R}_3$: $f_2 < 7 \wedge f_4 < 3 \wedge f_7 < 6 \wedge f10 = 1$ (99.4)%;
$\mathcal{R}_4$: $f_2 < 7 \wedge f_4 < 3 \wedge f_5 < 4 \wedge f_7 < 6$ (99.4)%.

Including the ELSE condition they give 97.7% overall accuracy. The confusion matrix (benign, malignant) is $\begin{pmatrix} 447 & 5 \\ 11 & 236 \end{pmatrix}$. Only five malignant cases are misclassified as benign. Fuzzified rules predict with almost 100% confidence that these vectors belong to the wrong class, indicating that the data is slightly noisy.

Minimization of (18) allows to enforce 100% reliability of all rules. Eight rules were obtained, rejecting 51 cases (7.3% of all vectors). For malignant class these rules are

$\mathcal{R}_1$) $f_2 < 9 \wedge f_4 < 4 \wedge f_7 < 3 \wedge f_8 < 6$;
$\mathcal{R}_2$) $f_2 < 5 \wedge f_5 < 8 \wedge f_7 < 5 \wedge f_8 < 10$;
$\mathcal{R}_3$) $f_2 < 4 \wedge f_4 < 2 \wedge f_5 < 3 \wedge f_7 < 7$;
$\mathcal{R}_4$) $f_2 < 10 \wedge f_5 < 10 \wedge f_7 \in [1, 5] \wedge f_8 < 2$.

For the benign cases initial rules are obtained by negation of the above rules; after optimization the rule becomes: $\neg(\mathcal{R}_5 \vee \mathcal{R}_6 \vee \mathcal{R}_7 \vee \mathcal{R}_8)$, where

$\mathcal{R}_5$) $f_2 < 8 \wedge f_4 < 5 \wedge f_8 < 4$;
$\mathcal{R}_6$) $f_2 < 9 \wedge f_5 < 6 \wedge f_7 < 9 \wedge f_8 < 5$;
$\mathcal{R}_7$) $f_2 < 9 \wedge f_4 < 6 \wedge f_5 < 8 \wedge f_7 < 9$;
$\mathcal{R}_8$) $f_2 = 6 \wedge f_4 < 10 \wedge f_5 < 10 \wedge f_7 < 2 \wedge f_8 < 9$.

TABLE XIII
RESULTS FROM THE TEN-FOLD CROSSVALIDATION AND RECLASSIFICATION FOR THE DIABETES DATASET, ACCURACY IN %

| Method | Accuracy | Reference |
|---|---|---|
| Logdisc | 77.7 | Ref. [88] |
| IncNet | 77.6 | Ref. [69] |
| DIPOL92 | 77.6 | Ref. [88] |
| LDA | 77.5 | Ref. [88] |
| SMART | 76.8 | Ref. [88] |
| ASI | 76.6 | Ref. [88] |
| FDA | 76.5 | Ref. [88] |
| BP | 76.4 | Ref. [88] |
| LVQ | 75.8 | Ref. [88] |
| RBF | 75.7 | Ref. [88] |
| LFC | 75.8 | Ref. [88] |
| NB | 75.3 | Ref. [88] |
| SNB | 75.4 | Ref. [88] |
| DB-CART, 33 nodes | 74.4 | Ref. [79] |
| ASR | 74.3 | Ref. [88] |
| FSM, 50 fuzzy rules | 73.8 | our result |
| CART, 11 nodes | 73.7 | Ref. [79] |
| C4.5 | 73.0 | Ref. [88] |
| CART | 72.8 | Ref. [88] |
| Kohonen SOM | 72.2 | Ref. [88] |
| kNN | 71.9 | Ref. [88] |
| Reclasssification | | |
| C-MLP2LN, 2 rules | 77.7 | our result |
| C-MLP2LN, 1 rule | 75.0 | our result |

For the Wisconsin breast cancer data SSV generates a very simple set of three rules for the second class, achieving 97.4% of reclassification accuracy. In the 10-fold crossvalidation test SSV rules give on average 96.3% (worst $-0.2\%$, best $+0.2\%$) accuracy.

$\mathcal{R}_1$: $f_4 > 2.5 \wedge f_7 > 2.5$;
$\mathcal{R}_2$: $f_4 > 2.5 \wedge f_6 > 3.5 \wedge f_7 < 2.5$;
$\mathcal{R}_3$: $f_2 > 5.5 \wedge f_4 < 2.5 \wedge f_7 > 1.6$.

The NEFCLASS neurofuzzy system has also been applied to this data [70], removing 16 cases with missing values. The system was initialized with fuzzy clustering method and used three trapezoidal membership functions per input feature. Reclassification error using three rules (eight conditions each, since one feature has been deleted) gave 92.7% correct answers. Using four rules and the "best per class" rule learning results gave only 80.4% correct answers, showing the usefulness of prior knowledge from initial clusterization. If only two membership functions per feature are used better reclassification accuracy of 96.5% is obtained using four fuzzy rules. FSM generated 12 rules with Gaussian membership functions, providing 97.8% on the training and 96.5% on the test part in 10-fold crossvalidation tests. Thus crisp rules seem to offer simpler and more accurate description of this dataset.

### H. Diabetes

The "Pima Indian diabetes" dataset is stored in the UCI repository [7] and is frequently used as benchmark data. All patients were females at least 21 years old, of Pima Indian heritage. The data contains two classes, eight attributes, 768 instances, 500 (65.1%) healthy and 268 (34.9%) diabetes cases. Our first attempts at extracting rules for this dataset were not successful because histograms do not provide a useful starting point here. L-units and separability criterion provided good linguistic variables. This dataset was used in the Statlog project [8], with the best 10-fold crossvalidation accuracy around 77.7% obtained by logistic discriminant analysis. One simple rule for the healthy cases achieving 75% accuracy is

$$f_2 \leq 151 \wedge f_6 \leq 47 \tag{31}$$

where $f_2$ is the "plasma glucose concentration" and $f_6$ the body mass index [weight in kg/(height in m)$^2$]. The confusion matrix (healthy, diabetes) is $\left(\begin{smallmatrix} 467 & 159 \\ 33 & 109 \end{smallmatrix}\right)$. FSM neurofuzzy system with Gaussian functions generates 50 rules and achieves in the ten-fold crossvalidation 85.3% accuracy on the training part and only 73.8% on the test part. Since better results are achieved using linear discrimination sharp and rotated decision borders may be needed for optimal classification of this data. Table XIII shows the results from the 10-fold crossvalidation and reclassification for this dataset.

### I. Hepatobiliary Disorders

This data, used previously in [91], contains medical records of 536 patients admitted to a university affiliated Tokyo-based hospital, with four types of hepatobiliary disorders: alcoholic liver damage, primary hepatoma, liver cirrhosis and cholelithiasis. The records included results of nine biochemical tests and sex of the patient. The same 163 cases as in [91] were used as the test data. In the previous work three fuzzy sets per each input were assigned using recommendation of the medical experts. A fuzzy neural network was constructed and trained until 100% correct answers were obtained on the training set. The accuracy on the test set varied from less than 60% to a peak of 75.5%. Although we quote this result in the Table XIV below it seems impossible to find good criteria that will predict when the training on the test set should be stopped. Fuzzy rules equivalent to the fuzzy network were derived but their accuracy on the test set was not given. This data has also been analyzed by Mitra *et al.* [92] using a knowledge-based fuzzy MLP system with results on the test set in the range from 33% to 66.3%, depending on the actual fuzzy model used.

For this dataset crisp rules were not too successful. The initial 49 rules obtained by C-MLP2LN procedure gave 83.5% on the training and 63.2% on the test set. Optimization did not improve these results significantly. On the other hand fuzzy rules derived using the FSM network, with Gaussian as well as with triangular functions, gave similar accuracy of 75.6–75.8%. Fuzzy neural

TABLE XIV
RESULTS FOR THE HEPATOBILIARY DISORDERS. ACCURACY ON THE TRAINING AND TEST SETS, IN %, ALL CALCULATIONS ARE OURS

| Method | Training set | Test set |
|---|---|---|
| 1-NN, weighted (ASA) | 83.4 | 82.8 |
| 1-NN, 4 features | 76.9 | 80.4 |
| K* method | – | 78.5 |
| kNN, k=1, Manhattan | 79.1 | 77.9 |
| FSM, Gaussian functions | 93 | 75.6 |
| FSM, 60 triangular functions | 93 | 75.8 |
| IB1c (instance-based) | – | 76.7 |
| FSM, Gaussian functions | 93 | 75.6 |
| C4.5 decision tree | 94.4 | 75.5 |
| Fuzzy neural network | 100 | 75.5 |
| Cascade Correlation | – | 71.0 |
| MLP with RPROP | – | 68.0 |
| Best fuzzy MLP model | 75.5 | 66.3 |
| C4.5 decision rules | 64.5 | 66.3 |
| DLVQ (38 nodes) | 100 | 66.0 |
| LDA (statistical) | 68.4 | 65.0 |
| 49 crisp logical rules | 83.5 | 63.2 |
| FOIL (inductive logic) | 99 | 60.1 |
| T2 (rules from decision tree) | 67.5 | 53.3 |
| 1R (rules) | 58.4 | 50.3 |
| Naive Bayes | – | 46.6 |
| IB2-IB4 | 81.2-85.5 | 43.6-44.6 |

network used over 100 neurons to achieve 75.5% accuracy, indicating that good decision borders in this case are quite complex and many logical rules will be required. Various results for this dataset are summarized in Table XIV.

FSM gives about 60 Gaussian or triangular membership functions achieving accuracy of 75.5–75.8%. Rotation of these functions (i.e., introducing linear combination of inputs to the rules) does not improve this accuracy. We have also made 10-fold crossvalidation tests on the mixed data (training plus test data), achieving similar results. Many methods give rather poor results on this dataset, including various variants of the instance-based learning (IB2-IB4, except for the IB1c, which is specifically designed to work with continuous input data), statistical methods (Bayes, LDA) and pattern recognition methods (LVQ). The best results were obtained with the $K^*$ method based on algorithmic complexity optimization, giving 78.5% on the test set, and $k$NN with Manhattan distance function, $k = 1$ with selection of features, giving 80.4% accuracy (for details, see [6]).

### *J. The Hypothyroid Data*

This is a somewhat larger dataset [7], with 3772 cases for training, 3428 cases for testing, 22 attributes (15 binary, six continuous), and three classes: primary hypothyroid, compensated hypothyroid, and normal (no hypothyroid). The class distribution in the training set is 93, 191, 3488 vectors and in the test set 73, 177, 3178. Initially four rules were found, with 99.68% accuracy on the training set and 99.07% accuracy on the test set. For the first class two rules are sufficient (all values of continuous features are multiplied here by 1000)

$\mathcal{R}_{11}$: FTI $< 63 \wedge$ TSH $\geq 29$;
$\mathcal{R}_{12}$: FTI $< 63 \wedge$ TSH $\in [6.1,29) \wedge$ T3$< 20$.

For the second class one rule is created

$\mathcal{R}_2$: FTI $\in [63, 180] \wedge$ TSH $\geq 6.1 \wedge$ on thyroxine $=$ no $\wedge$ surgery $=$ no

and the third class is covered by the ELSE condition. With these rules we get 99.68% accuracy on the training set and 99.07% error on the test set. Optimization of these rules leads to slightly more accurate set of rules

$\mathcal{R}_{11}$: TSH $\geq 30.48 \wedge$ FTI $< 64.27$ $(97.06\%)$;
$\mathcal{R}_{12}$: TSH $\in [6.02,29.53] \wedge$ FTI $< 64.27 \wedge$ T3$< 23.22$ $(100\%)$;
$\mathcal{R}_2$: TSH $\geq 6.02 \wedge$ FTI $\in [64.27,186.71] \wedge$ TT4 $\in [50,150.5) \wedge$ on thyroxine $=$ no $\wedge$ surgery $=$ no $(98.96\%)$.

The ELSE condition has 100% reliability on the training set. These rules make only four errors on the training set (99.89%) and 22 errors on the test set (99.36%). They are similar to those found using heuristic version of PVM method by Weiss and Kapouleas [80]. The differences among PVM, CART and C-MLP2LN are for this dataset rather small (Table XV), but other methods, such as well-optimized MLP (including genetic optimization of network architecture [89]) or cascade correlation classifiers, give results that are significantly worse. Poor results of $k$-NN are especially worth noting, showing that in this case, despite large amount of reference vectors, similarity-based methods are not competitive. Ten fuzzy rules obtained using FSM with Gaussian membership functions are also less accurate than the three crisp rules.

The C-MLP2LN solution seems to be close to optimal [77]. Similar rules were obtained from the SSV separability criterion:

$\mathcal{R}_1$: TSH $> 6.05 \wedge$ FTI $< 64.72 \wedge$ thyroid-surgery $=$ no;
$\mathcal{R}_2$: TSH $> 6.05 \wedge$ FTI $> 64.72 \wedge$ TT4 $< 150.5 \wedge$ thyroid-surgery $=$ no $\wedge$ on-thyroxine $=$ no.

TABLE XV
RESULTS FOR THE HYPOTHYROID DATASET

| Method | % train | % test | Ref. |
|---|---|---|---|
| C-MLP2LN | 99.89 | 99.36 | our result |
| CART | 99.79 | 99.36 | [80] |
| PVM | 99.79 | 99.33 | [80] |
| SSV rules | 99.79 | 99.33 | our result |
| FSM 10 rules | 99.60 | 98.90 | our result |
| Cascade correl. | 100.00 | 98.5 | [89] |
| MLP+backprop | 99.60 | 98.5 | [89] |
| 3-NN, 3 features used | 98.7 | 97.9 | our result |
| Bayes | 97.0 | 96.1 | [80] |
| k-NN | – | 95.3 | [80] |

TABLE XVI
SUMMARY OF RESULTS FOR THE NASA SHUTTLE DATASET

| Method | Train | Test | Ref. |
|---|---|---|---|
| SSV, 32 rules | 100.00 | 99.99 | our result |
| NewID dec. tree | 100.00 | 99.99 | [8] |
| FSM, 17 rules | 99.98 | 99.97 | our result |
| $k$-NN + feature sel. | – | 99.95 | our result |
| C4.5 dec. tree | 99.96 | 99.90 | [8] |
| $k$-NN | – | 99.56 | [8] |
| RBF | 98.40 | 98.60 | [8] |
| MLP+BP | 95.50 | 96.57 | [8] |
| Logistic discrimination | 96.07 | 96.17 | [8] |
| Linear discrimination | 95.02 | 95.17 | [8] |

These rules match our best results and have been found with fully automatic rule extraction approach. Results are summarized in Table XV. It is worth noting that the error of the best neural network classifiers is still twice as large (1.5%) as the error made by these simple rules. Excellent results of rule-based classifiers for this dataset show the need to provide sharp decision borders instead of soft borders provided by the fuzzy and neural systems. This may be an artefact of providing sharp division into three output classes.

### K. NASA Shuttle

The Shuttle dataset from NASA contains nine continuous numerical attributes related to the positions of radiators in the Space Shuttle. There are 43 500 training vectors and 14 500 test vectors, divided into seven classes in a very uneven way: about 80% from class 1 and only six examples from class 6 in the training set. This data has been used in the Stalog project [8], therefore accuracy of our rules may be compared with many other classification systems (Table XVI).

We have used the FSM network with rectangular membership functions and SSV criterion here. Initialization of the network gives seven nodes achieving already 88% accuracy. Increasing accuracy (using constructive learning algorithm) on the training set to 94%, 96%, and 98% leads to a total of 15, 18, and 25 nodes and accuracies on the test set of 95.5%, 97.8%, and 98.5%. Backpropagation network reached an accuracy of 95.5% on the training set. $k$-NN is very slow in this case, requiring all 43 500 training vectors as reference for computing distances, reaching on the test set 99.56% but with feature selection improving to 99.95%. Optimization of the FSM rules generated 15 logical rules. For example, for the third class rules are

$$F2 \in [-188.43, -27.50] \wedge F9 \in [1, 74]$$
$$F2 \in [-129.49, -21.11] \wedge F9 \in [17, 76].$$

The set of 17 rules makes only three errors on the training set (99.99% accuracy), leaving eight vectors unclassified, and no errors on the test set but leaving nine vectors unclassified (99.94%). After Gaussian fuzzification of inputs (very small, 0.05%) only three errors and five unclassified vectors are obtained for the training and three vectors are unclassified and one error is made (with the probability of correct class for this case close to 50%) for the test set. Rules from SSV gave even better results: 100% correct on the training and only one error on the test set.

These results are much better than those obtained from the MLP or RBF networks (as reported in the Stalog project [8]) and comparable with the results of the best decision trees which work very well for this problem. It is interesting to note that in the Stalog project the NewID tree (descendant of the ID3 tree), which gave the best results here, has not been among the first 3 best methods for any other of the 22 datasets analyzed. Results of the C4.5 decision tree are already significantly worse. Our rule extraction approach has consistently been giving top results. Logical rules provide highly accurate and quite simple description of Shuttle dataset.

### L. Psychometric Data

Our methodology of extraction and optimization of logical rules has been used by us in several real-life projects. One of these projects concerns the psychometric data collected in the Academic Psychological Clinic of our University. Minnesota Multiphasic Personality Inventory (MMPI) test was used, consisting of 550 questions with three possible answers (yes, no, don't know) each. MMPI evaluates psychological characteristics reflecting social and personal maladjustment, including psychological dysfunction. Hundreds of books and papers were written on the interpretation of this test (cf. review [93]). Many computerized versions of the MMPI test exist to assist in information acquisition, but evaluation of results is still done by an experienced clinical psychologist. Our goal is to provide automatic psychological diagnosis.

The raw MMPI data is used to compute 14 real-valued coefficients (this corresponds to manual aggregation of input data), called "psychometric scales." These coefficients are often displayed as a histogram (called "a psychogram") allowing skilled psychologists to diagnose specific problems, such as neurosis, drug addiction or criminal tendencies. First four coefficients are just the control scales (measuring consistency of answers, allowing to find malingerers, etc.), with the rest forming clinical scales. These scales were developed to measure tendencies toward hypochondria, depression, hysteria, psychopathy, paranoia, schizophrenia, etc. A large number of simplification schemes has been developed to make the interpretation of psychograms easier. They may range from rule-based systems derived from observations of characteristic shapes of psychograms, Fisher discrimination functions, or systems using a small number of coefficients, such as the three Goldberg coefficients. Unfortunately there is no comparison of these different schemes and their relative merits have not been tested statistically. Our goal was to provide an automatic psychological diagnosis.

Rule based system is most desirable because a detailed interpretation, including description of personality type, may be assigned to each diagnosis. We have worked with two datasets, one for woman, with 1027 cases belonging to 27 classes (normal, neurotic, drug addicts, schizophrenic, psychopaths, organic problems, malingerers etc.) determined by expert psychologists, and the second for man, with 1167 cases and 28 classes. Rules were generated using C4.5 classification tree [83] and the FSM system.

For the first dataset C4.5 created 55 rules, achieving 93.0% of correct responses. Assuming about 1% inaccuracy of measurements improves results to 93.7%. FSM (with rectangular membership functions) generated 69 rules agreeing in 95.4% with diagnosis by human experts. Gaussian fuzzification at the level of 1.1–1.5% increases accuracy to 97.6%. For the second dataset C4.5 created 61 rules giving 92.5% accuracy (93.1% after fuzzification), while FSM generated 98 rules giving 95.9% accuracy and after fuzzification 96.9%. Some rules cover only few cases from the database, therefore further pruning and re-optimization is desirable.

Rules involve from two to nine attributes. For most classes there were only a few errors and it is quite probable that they are due to the psychologists interpreting the psychogram data. Two classes, organic problems and schizophrenia, are difficult since their symptoms are easily confused with symptoms belonging to other classes. Each rule has detailed interpretation associated with it by psychologists. Fuzzification leads to additional adjectives in verbal interpretation, like "strong tendencies," or "typical." An expert system using these rules should be evaluated by clinical psychologist in the near future. A typical rule has the form:

$$\text{If } f_7 \in [55, 68] \wedge f_{12} \in [81, 93] \wedge f_{14} \in [49, 56]$$
$$\text{Then Paranoia}$$

where $f_7$ is the hysteria scale, etc. An example of a psychogram with rule conditions shown as vertical bars is shown in Fig. 10. The rule has five conditions and the actual case is accepted by that rule with 71.8% probability, calculated with assumption of Gaussian uncertainties shown on the vertical bars for each conditions. The rule condition for the Ps (psychostenia) scale fits with only 72.2% to the measured value, which means that the value is close to the interval boundary. An expert system based on our logical rules is under evaluation by clinical psychologists.

## XI. Summary and Conclusion

Methodology of extraction of crisp and fuzzy logical rules from data and black box classifiers (such as neural networks) has been described. This methodology includes:

1) determination and optimization of linguistic variables;
2) initial generation of rules of different complexity using constrained MLP networks, search-based MLP's, FSM networks or separability criterion;
3) optimization of rules and exploration of the rejection/accuracy tradeoff;
4) calculation of probabilities, enabling also estimation of reliability of classification, gradient optimization of large sets of rules, creating more robust logical rules and providing additional adaptive parameters.

Extraction of crisp logical rules is advantageous independently of the final classifier used. First, in our tests logical rules proved to be highly accurate; second, they are easily understandable by experts in a given domain; third, they may expose problems with the data itself. This became evident in the analysis of a real-world medical datasets we were involved in. Some research groups reported very good results using this data, but after extraction of logical rules it became clear that missing features in the data were replaced by their averages for a given class. Crossvalidation tests on such data are quite accurate but in a real application averages for a given class can be added only after, not before the diagnosis.

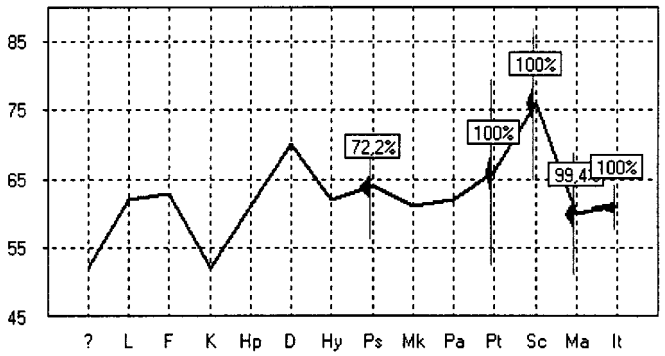


Fig. 10. Psychogram with rule conditions and fuzzified input displayed.

From geometrical point of view crisp logic rules correspond to a division of the feature space with hyperplanes perpendicular to the axes, into areas with symbolic names (corresponding to class and rule numbers). If the classes in the input space are correctly separated with such hyperplanes accurate logical description of the data is possible and worthwhile. Otherwise accuracy of logical description of the data may increase slowly with the number of linguistic variables and generalization ability of a rule-based system (measured by crossvalidation tests) may even decrease. If the number of logical rules is too high or the accuracy of classification is too low, other classification methods should be attempted. Fuzzy logic may offer better approximation with smaller number of rules, including simple piecewise linear approximation rules and more complex membership functions. However, fuzzy rules based on triangular or Gaussian membership functions provide oval decision borders that do not approximate correctly sharp decision boundaries necessary for description of data with inherent logical structure. Complex membership functions are provided by neurofuzzy systems, such as the FSM network [42]. As long as separable transfer functions are used network nodes are equivalent to fuzzy rules. Although fuzzy rules are symbolic their comprehensibility is lower than crisp rules. Finding a global optimum of the error function for sophisticated classification systems is usually more difficult than for sets of crisp rules. Therefore a good strategy is to start with extraction of crisp rules first and use fuzzy rules only if the results are not satisfactory.

The problem of determination of linguistic variables is not separable from the rule extraction itself. An iterative algorithm has been proposed, improving in turns linguistic variables and then rules based on these variables. We have stressed the importance of context-dependent linguistic variables since an unwarranted assumption that the whole range of attribute values should be partitioned into intervals corresponding to linguistic variables is frequently used. Histograms are helpful to determine initial linguistic variables only in simple cases. Good linguistic variables are found using probability density networks, special neural linguistic units, or separability criterion.

Four groups of methods for extraction of logical rules have been introduced in this paper. MLP2LN method of converting the MLP into a network performing logical operations has been quite successful. The constructive C-MLP2LN version, with L-R structure of the MLP network, composed of linguistic units and rule units (with possible addition of aggregation units) is quite fast. A search-based MLP algorithm as an alternative to backpropagation training is particularly easy to implement and analyze, giving a single logical rule per neuron. MLP2LN methods in complex cases require an additional rule extraction step, with search for combination of inputs that lead to activations exceeding the thresholds. Feature space mapping (FSM) probability density networks are used for fuzzy rule extraction, creating also crisp rules if a transition to rectangular membership function is made. SSV separability criterion combined with the beam search techniques finds optimal separation values for interacting features, creating decision trees that are easily converted to sets of logical rules. The last two methods allow to extract rules by inspection of network or tree nodes.

After extraction of rules modified predictive power cost function for additional optimization of linguistic variables is used, creating hierarchical sets of logical rules with different reliability-rejection rate. A great advantage of fuzzy logic is the soft evaluation of probabilities of different classes, instead of binary yes or no crisp logic answers. Gaussian fuzzification of the input values may give the same probabilities as the Monte Carlo procedure performed for input vectors distributed around measured values. Thus simple interpretation of crisp logical rules is preserved, accuracy is improved by using additional parameters for estimation of measurement uncertainties, and gradient procedures instead of costly global minimization may be used. Gaussian uncertainties are equivalent to "soft trapezoid" fuzzification of the rectangular crisp membership functions. Sets of crisp logical rules may then be used to calculate probabilities. Novel vectors that would either be rejected or assigned to the default class are assigned to the most probable class. Application to psychometric data analysis combines comprehensibility of data description, allowing for verbal interpretation, with high accuracy and soft probabilities of different diagnoses.

Using this methodology we have analyzed many medical and technical datasets obtaining simple and accurate logical rules. For several benchmark problems simplest logical description known so far was obtained. For some problems, such as the hypothyroid or NASA Shuttle, logical rules are more accurate than any other classification method [6], including neural networks. Possible explanations of this empirical observation are

1) the inability of soft transfer functions (sigmoidal or Gaussian) to represent sharp, rectangular edges that may be necessary to separate two classes defined by a crisp logical rule;
2) the problem of finding globally optimal solution of the nonlinear optimization problem for neural classifiers—in some cases we have used a global optimization method to improve our rules, in other optimization of linguistic variables and optimization of rules has been separated, leading to better solutions than gradient-based neural classifiers were able to find;
3) the problem of finding an optimal balance between the flexibility of adaptive models and the danger of overfitting the data. Bayesian regularization based on priors leading to weight decay [58] helps in case of some neural and statistical classification models, but it has an adverse effect if sharp decision borders are needed. Sharp decision borders require large weights and thresholds while regularization terms decrease all weights. Logical rules give much better control over the complexity of the data representation and elimination of outliers—rules that cover only a few new data vectors are easily identified and removed;
4) for medical data labeling the cases "sick" or "healthy" introduces implicitly crisp logical rules. Forced to make yes–no diagnosis human experts may fit the results of tests to specific intervals.

Although we are pleased with the results obtained so far several challenges still remain: aggregation of large number of input features (some data mining problems we work on have more than 1000 features and less than 1000 cases), construction of hierarchical systems when a large number of features contain missing data, automatization of the whole process of logical data description and creation of expert systems, going beyond prepositional logic and simple linguistic variables. We are sure that neural networks will play an important role in this field.

Please note that many papers of our group are available at http://www.phys.uni.torun.pl/kmk/publications.html.

## REFERENCES

[1] R. Andrews, J. Diederich, and A. B. Tickle, "A survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-Based Syst.*, vol. 8, pp. 373–389, 1995.
[2] R. Michalski, "A theory and methodology of inductive learning," *Artificial Intell.*, vol. 20, pp. 111–161, 1983.
[3] R. Schalkoff, *Pattern Recognition. Statistical, Structural and Neural Approaches*. New York: Wiley, 1992.
[4] T. Mitchell, *Machine Learning*. New York: McGraw Hill, 1997.
[5] W. Duch, R. Adamczak, K. Grąbczewski, and G. Żal, "Hybrid neural-global minimization logical rule extraction method for medical diagnosis support," in *Intelligent Information Systems VII*, Malbork, Poland, 1998, 15-19.06, pp. 85–94.
[6] W. Duch, R. Adamczak, K. Grąbczewski, G. Żal, and Y. Hayashi, "Fuzzy and crisp logical rule extraction methods in application to medical data," in *Computational Intelligence and Applications. Springer Studies in Fuzziness and Soft Computing*, P. S. Szczepaniak, Ed., 1999, vol. 23.
[7] UCI repository of machine learning databases, C. J. Mertz and P. M. Murphy. [Online]. Available: http://www.ics.uci.edu/pub/machine-learning-data-bases
[8] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. London, U.K.: Ellis Horwood, 1994.
[9] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[10] N. Kasabov, *Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering*. Cambridge, MA: MIT Press, 1996.
[11] Z. Pawlak, *Rough Sets—Theoretical Aspects of Reasoning About Data*. Boston, MA: Kluwer, 1991.
[12] S. K. Pal and A. Skowron, *Rough Fuzzy Hybridization A New Trend in Decision-Making*: Springer-Verlag, 1999.
[13] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich, "The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks," *IEEE Trans. Neural Networks*, vol. 9, pp. 1057–1068, 1998.
[14] K. Saito and R. Nakano, "Medical diagnostic expert system based on PDP model," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, San Diego, CA, 1988, pp. 255–262.

[15] S. Gallant, *Neural Network Learning and Expert Systems*. Cambridge, MA: MIT Press, 1993.

[16] S. Thrun, "Extracting rules from artificial neural networks with distributed representations," in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D.S. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995.

[17] L. M. Fu, "Rule learning by searching on adapted nets," in *Proc. 9th Nat. Conf. Artificial Intell.*, Anaheim, CA, 1991, pp. 590–595.

[18] ——, "Knowledge-based connectionism for revising domain theories," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 173–182, 1993.

[19] ——, *Neural Networks in Computer Intelligence*. New York: McGraw Hill, 1994.

[20] ——, "Rule generation from neural networks," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, pp. 1114–1124, 1994.

[21] I. K. Sethi and J. H. Yoo, "Symbolic approximation of feedforward neural networks," in *Pattern Recognition in Practice*, E. S. Gelsema and L. N. Kanal, Eds. New York: North-Holland, 1994, vol. 4.

[22] G. Towel and J. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, pp. 71–101, 1993.

[23] Y. Hayashi, "A neural expert system with automated extraction of fuzzy if-then rules," in *Advances in Neural Information Processing Systems*, R. Lippmann, J. Moody, and D. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, vol. 3.

[24] G. Towell and J. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intell.*, vol. 70, pp. 119–165, 1994.

[25] C. McMillan, M. C. Mozer, and P. Smolensky, "Rule induction through integrated symbolic and subsymbolic processing," in *Advances in Neural Processing Systems*, J. Moody, S. Hanson, and R. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, vol. 4.

[26] J. A. Alexander and M. C. Mozer, "Template-based algorithms for connectionist rule extraction," in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995, vol. 7.

[27] R. Setiono and H. Liu, "Understanding neural networks via rule extraction," in *Proc. 14th Int. Joint Conf. Artificial Intell*. San Mateo, CA: Morgan Kaufmann, 1995, pp. 480–485.

[28] M. Ishikawa, "Rule extraction by successive regularization," in *Proc. 1996 IEEE Int. Conf. Neural Networks*, Washington, DC, 1996, pp. 1139–1143.

[29] P. Geczy and S. Usui, "Rule extraction from trained neural networks," in *Int. Conf. Neural Inform. Processing*, vol. 2, New Zealand, Nov. 1997, pp. 835–838.

[30] W. Duch, R. Adamczak, and K. Grąbczewski, "Extraction of logical rules from training data using backpropagation networks," in *Proc. 1st Online Workshop Soft Comput.*, Aug. 19–30, 1996, http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/, pp. 25–30.

[31] M. W. Craven and J. W. Shavlik, "Using sampling and queries to extract rules from trained neural networks," in *Proc. 11th Int. Conf. Machine Learning*. New Brunswick, NJ: Morgan Kaufmann, 1994, pp. 37–45.

[32] E. Pop, R. Hayward, and J. Diederich, "RULENEG: Extracting rules from a trained ANN by stepwise negation,", QUT NRC technical report, December 1994.

[33] S. Sestito and T. Dillon, *Automated Knowledge Acquisition*, Australia: Prentice Hall, 1994.

[34] M. J. Healy and T. P. Caudell, "Acquiring rule sets as a product of learning in a logical neural architecture," *IEEE Trans. Neural Networks*, vol. 8, pp. 461–474, 1997.

[35] A.-H. Tan, "Rule learning and extraction with self-organizing neural networks," in *Proc. 1993 Connectionist Models Summer School*. Hillsdale, NJ: Lawrence Erlbaum, 1994, pp. 192–199.

[36] A. Ultsch, "Knowledge extraction from self-organizing neural networks," in *Information and Classification*, O. Opitz, B. Lausen, and R. Klar, Eds. Berlin: Springer, 1993, pp. 301–306.

[37] A. B. Tickle, M. Orlowski, and J. Diederich, "DEDEC: Decision detection by rule extraction from neural networks,", QUT NRC Tech. Rep., Sept. 1994.

[38] M. W. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," in *Advances in Neural Information Processing Systems*, D. Touretzky, M. Mozer, and M. Hasselmo, Eds. Cambridge, MA: MIT Press, 1996, vol. 8.

[39] J.-S. R. Jang and C. T. Sun, "Functional equivalence between radial basis function neural networks and fuzzy inference systems," *IEEE Trans. Neural Networks*, vol. 4, no. 1, pp. 156–158, 1993.

[40] V. Tresp, J. Hollatz, and S. Ahmad, "Network structuring and training using rule-based knowledge," in *Advances in Neural Information Processing Systems*, J. Moody, S. Hanson, and R. Lippmann, Eds. San Mateo, CA: Morgan Kaufmann, 1992, vol. 4.

[41] W. Duch, "Floating Gaussian mapping: A new model of adaptive systems," *Neural Network World*, vol. 4, pp. 645–654, 1994.

[42] W. Duch and G. H. F. Diercksen, "Feature space mapping as a universal adaptive system," *Computer Physics Communication*, vol. 87, pp. 341–371, 1995.

[43] R. Andrews and S. Geva, "Rules and local function networks," in *Rules and Networks, Proc. Rule Extraction From Trained Artificial Neural Networks Workshop—AISB96*, R. Andrews and J. Diederich, Eds. Brighton, U.K., April 1996.

[44] ——, "Rule extraction from a constrained error backpropagation MLP," in *Proc. 5th Australian Conference on Neural Networks*, Brisbane, Australia, 1994, pp. 9–12.

[45] J. J. Mahoney and R. J. Mooney, "Combining neural and symbolic learning to revise probabilistic rule bases," in *Advances in Neural Inform. Processing Syst.*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, vol. 5, pp. 107–114.

[46] D. Nauck, F. Klawonn, and R. Kruse, *Foundations Neuro-Fuzzy Syst.*. New York: Wiley, 1997.

[47] D. Nauck, U. Nauck, and R. Kruse, "Generating classification rules with the neuro-fuzzy system NEFCLASS," in *Proc. Biennial Conf. North Amer. Fuzzy Inform. Processing Soc. (NAFIPS'96)*, Berkeley, CA, 1996.

[48] S. K. Halgamuge and M. Glesner, "Neural networks in designing fuzzy systems for real-world applications," *Fuzzy Sets Syst.*, vol. 65, pp. 1–12, 1994.

[49] J. M. Żurada and A. Łozowski, "Generating linguistic rules from data using neuro-fuzzy framework," in *Proc. 4th Int. Conf. Soft Comput., IIZUKA'96*, vol. 2, Iizuka, Japan, 1996, pp. 618–621.

[50] N. Kasabov, R. Kozma, and W. Duch, "Rule extraction from linguistic rule networks and from fuzzy neural networks: Prepositional versus fuzzy rules," in *Prof. 4th Int. Conf. Neural Networks Applicat.*, Marseille, France, March 11–13, 1998, pp. 403–406.

[51] W. Duch, R. Adamczak, and K. Grąbczewski, "Neural optimization of linguistic variables and membership functions," in *Int. Conf. Neural Inform. Processing*, vol. 2, Perth, Australia, Nov. 1999, pp. 616–621.

[52] K. Grąbczewski and W. Duch, "A general purpose separability criterion for classification systems," in *Proc. 4th Conf. Neural Networks Applicat.*, Zakopane, May 1999, pp. 203–208.

[53] W. Duch, R. Adamczak, and N. Jankowski, "Initialization of adaptive parameters in density networks," in *Proc. 3rd Conf. Neural Networks*, Kule, Oct. 1997, pp. 99–104.

[54] ——, "New developments in the feature space mapping model," in *3rd Conf. Neural Networks*, Kule, Poland, Oct. 1997, pp. 65–70.

[55] W. Duch and N. Jankowski, "Bi-radial transfer functions," in *Proc. 2nd Conf. Neural Networks Applicat.*, vol. I, Orle Gniazdo, Poland, 1996, pp. 131–137.

[56] W. Duch, R. Adamczak, and K. Grąbczewski, "Extraction of logical rules from backpropagation networks," *Neural Processing Lett.*, vol. 7, pp. 1–9, 1998.

[57] J. M. Żurada, *Introduction to Artificial Neural Systems*. St Paul, MN: West, 1992.

[58] D. J. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Comput.*, vol. 4, pp. 448–472, 1992.

[59] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon, 1995.

[60] L. Kanal and V. Kumar, Eds., *Search in Artificial Intelligence*: Springer-Verlag, 1988.

[61] W. Duch and K. Grąbczewski, "Searching for optimal MLP," in *Proc. 4th Conf. Neural Networks Applicat.*, Zakopane, May 1999, pp. 65–70.

[62] L. Breiman, "Bias-Variance, regularization, instability and stabilization," in *Neural Networks and Machine Learning*, C. Bishop, Ed. New York: Springer-Verlag, 1998.

[63] W. Duch, R. Adamczak, and K. Grąbczewski, "Optimization of logical rules derived by neural procedures" (in Paper 741), in Proc. Int. Joint Conf. Neural Networks, Washington, DC, June 10–16, 1999, to be published.

[64] A. Łozowski, T. J. Cholewo, and J. M. Żurada, "Crisp rule extraction from perceptron network classifiers," in *Proc. IEEE Int. Conf. Neural Networks*, Washington, DC, 1996, pp. 94–99.

[65] I. Jagielska, C. Matthews, and T. Whitfort, "The application of neural networks, fuzzy logic, genetic algorithms and rough sets to automated knowledge acquisition," in *Proc. 4th Int. Conf. Soft Computing, IIZUKA'96*, vol. 2, Iizuka, Japan, 1996, pp. 565–569.

[66] H. Ishibuchi, T. Morisawa, and T. Nakashima, "Combining multiple fuzzy-rule-based classification systems," in *Proc. 4th Int. Conf. Soft Comput., IIZUKA'96*, vol. 2, Iizuka, Japan, 1996, pp. 822–825.

[67] S. B. Thrun *et al.*, "The MONK's problems: A performance comparison of different learning algorithms," Carnegie Mellon Univ., Pittsburgh, PA, CMU-CS-91-197, Dec. 1991.
[68] K. P. Bennett and O. L. Mangasarian, "Robust linear programming discrimination of two linearly inseparable sets," *Optimization Methods and Software*, vol. 1, pp. 23–34, 1992.
[69] N. Jankowski and V. Kadirkamanathan, "Statistical control of RBF-like networks for classification," in *Proc. 7th Int. Conf. Artificial Neural Networks*, Lausanne, Switzerland, 1997, pp. 385–390.
[70] D. Nauck, U. Nauck, and R. Kruse, "Generating classification rules with the neuro-fuzzy system NEFCLASS," in *Proc. Biennial Conf. North Amer. Fuzzy Inform. Processing Soc. (NAFIPS'96)*, Berkeley, CA, 1996.
[71] N. Kasabov, "Connectionist methods for fuzzy rules extraction, reasoning and adaptation," in *Proc. Int. Conf. Fuzzy Syst., Neural Networks, and Soft Comput.*. Iizuka, Japan: World Scientific, 1996, pp. 74–77.
[72] C. Browne, I. Düntsch, and G. Gediga, "IRIS revisited: A comparison of discriminant and enhanced rough set data analysis," in *Rough Sets in Knowledge Discovery*, L. Polkowski and A. Skowron, Eds. Heidelberg, Germany: Physica Verlag, 1998, vol. 2, pp. 345–368.
[73] J. Teghem and M. Benjelloun, "Some experiments to compare rough sets theory and ordinal statistical methods," in *Intelligent Decision Support: Handbook of Applications and Advances of Rough Set Theory*, R. Slowinski, Ed. Dordrecht, The Netherlands: Kluwer, 1992, vol. 11, System Theory, Knowledge Engineering and Problem Solving, pp. 267–284.
[74] S. K. Halgamuge and M. Glesner, "Neural networks in designing fuzzy systems for real world applications," *Fuzzy Sets Syst.*, vol. 65, pp. 1–12, 1994.
[75] S. M. Weiss and C. A. Kulikowski, *Computer Systems That Learn*. San Mateo, CA: Morgan Kauffman, 1990.
[76] W. Duch, R. Adamczak, and K. Grąbczewski, "Extraction of crisp logical rules using constrained backpropagation networks," in *Int. Conf. Artificial Neural Networks (ICNN'97)*, Houston, 1997, 9-12.6, pp. 2384–2389.
[77] W. Duch, R. Adamczak, K. Grąbczewski, and G. Żal, "Hybrid neural-global minimization method of logical rule extraction," Int. J. Advanced Comput. Intell., to be published.
[78] J. W. Grzymała-Busse and T. Soe, "Inducing simpler rules from reduced data," in *Intell. Inform. Syst. VII*, Malbork, Poland, 1998, 15-19.06, pp. 371–378.
[79] N. Shang and L. Breiman, "Distribution based trees are more accurate," in *Int. Conf. Neural Inform. Processing*, vol. 1, Hong Kong, 1996, pp. 133–138.
[80] S. M. Weiss and I. Kapouleas, "An empirical comparison of pattern recognition, neural nets and machine learning classification methods," in *Readings in Machine Learning*, J. W. Shavlik and T. G. Dietterich, Eds, CA: Morgan Kaufmann, 1990.
[81] W. Duch, R. Adamczak, and K. Grąbczewski, "Constraint MLP and density estimation for extraction of crisp logical rules from data," in *Int. Conf. Neural Inform. Processing*, vol. 2, New Zealand, Nov. 1997, pp. 831–834.
[82] ——, "Constrained backpropagation for feature selection and extraction of logical rules," in *Proc. 'Colloquia in AI'*, Łódź, Poland, 1996, pp. 163–170.
[83] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo: Morgan Kaufman, 1993.
[84] W. Duch and N. Jankowski, "New neural transfer functions," *Appl. Math. Comp. Sci.*, vol. 7, pp. 639–658, 1997.
[85] G. Cestnik, I. Konenenko, and I. Bratko, "Assistant-86: A knowledge-elicitation tool for sophisticated users," in *Progress in Machine Learning*, I. Bratko and N. Lavrac, Eds. Wilmslow, U.K.: Sigma, 1987, pp. 31–45.
[86] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, "The multipurpose incremental learning system AQ15 and its testing application to three medical domains," in *Proc. 5th Nat. Conf. AI*. San Mateo, CA: Morgan Kaufmann, 1986, pp. 1041–1045.
[87] M. Tan and L. Eshelman, "Using weighted networks to represent classification knowledge in noisy domains," in *Proc. 5th Int. Conf. Machine Learning*, Ann Arbor, MI, 1988, pp. 121–134.
[88] B. Ster and A. Dobnikar, "Neural networks in medical diagnosis: Comparison with other methods," in *Proc. Int. Conf. EANN'96*, A. Bulsari, Ed., 1996, pp. 427–430.
[89] W. Schiffman, M. Joost, and R. Werner, "Comparison of optimized backpropagation algorithms," in *Proc. Europ. Symp. Artificial Neural Networks*. Brussels, Belgium: De facto, 1993, pp. 97–104.
[90] W. Duch, R. Adamczak, K. Grąbczewski, M. Ishikawa, and H. Ueda, "Extraction of crisp logical rules using constrained backpropagation networks—Comparison of two new approaches," in *Proc. Europ. Symp. Artificial Neural Networks (ESANN'97)*, Bruge, Belgium, 1997, 16-18.4., pp. 109–114.
[91] Y. Hayashi, A. Imura, and K. Yoshida, "Fuzzy neural expert system and its application to medical diagnosis," in *Proc. 8th Int. Congr. Cybern. Syst.*, New York, 1990, pp. 54–61.
[92] S. Mitra, R. De, and S. Pal, "Knowledge based fuzzy MLP for classification and rule generation," *IEEE Trans. Neural Networks*, vol. 8, pp. 1338–1350, 1997.
[93] J. N. Butcher and S. V. Rouse, "Personality: Individual differences and clinical assessment," *Annu. Rev. Psychol.*, vol. 47, p. 87, 1996.
[94] R. Hayward, C. Ho-Stuart, J. Diederich, and E. Pop, "RULENEG: Extracting rules from a trained ANN by stepwise negation,", QUT NRC Tech. Rep., Jan. 1996.
[95] R. Andrews and S. Geva, "Refining expert knowledge with an artificial neural network," in *Int. Conf. Neural Inform. Processing*, vol. 2, New Zealand, Nov. 1997, pp. 847–850.
[96] D. Nauck and R. Kruse, "Designing neuro-fuzzy systems through backpropagation," in *Fuzzy Modeling: Paradigms and Practice*, W. Pedrycz, Ed. Boston, MA: Kluwer, 1996, pp. 203–228.

**Włodzisław Duch** (M'96) received the M.Sc. degree in physics in 1977 and the Ph.D. degree in quantum chemistry in 1980 from Nicholas Copernicus University, Toruń, Poland.

He worked as a Postdoctoral Fellow at the University of Southern California (USC), Los Angeles, from 1980 to 1982. From 1985 to 1987, he was an Alexander von Humboldt Fellow in the Max Planck Institute of Astrophysics, Munich, Germany, an institute which he visits every year for a few months. He wrote a monograph *Graphical Representation of Model Spaces* (New York: Springer-Verlag, 1986) that was the basis of his habilitation thesis (D.Sc.). He was an Associate Professor at the Nicholas Copernicus University, Toruń, Poland, and in 1997 he was granted the title of a Full Professor. He was a "Visiting Professor" in Japan, Canada, Germany, France, and the United States. He is currently the Head of the Department of Computer Methods, an independent, interdisciplinary unit within the Faculty of Physics and Astronomy of Nicholas Copernicus University. His scientific interest moved from computational methods of physics and chemistry (he is still on the Editor's board of *Computer Physics Communication* journal), through foundations of physics, to artificial intelligence, neural networks and cognitive science (he has co-founded and is the Head Scientific Editor of the Polish *Cognitive Science* journal). He has written three books, coauthored and edited four books, and written about 200 scientific papers.

**Rafał Adamczak** received the M.Sc. degree in physics from Nicholas Copernicus University, Toruń, Poland, in 1995 and is currently pursuing the Ph.D. degree in the Department of Computer Methods at the same university.

His main interests include artificial intelligence, neural networks, machine learning, and applications of computational intelligence methods to various problems.

**Krzysztof Grąbczewski** received the M.Sc. degree in mathematics from Nicholas Copernicus University, Toruń, Poland, in 1994.

He spent ten months in 1994 to 1995 at the University of Cambridge, U.K., working on automated theorem proving. Since 1995 he is a Research and Teaching Assistant in the Department of Computer Methods, Nicholas Copernicus University. His main interests include artificial intelligence, data mining, machine learning, and understanding data through logical, rule-based description.