

Rule Based Fuzzy Classification Using Squashing Functions

József Dombi, Zsolt Gera*

University of Szeged, Institute of Informatics

H-6720 Szeged, Árpád tér 2.

tel.:+3662544222/3830, fax: +3662546397

e-mail: {dombi|gera}@inf.u-szeged.hu

* Corresponding author.

Abstract – In this paper we are dealing with the construction of a fuzzy rule based classifier. A three-step method is proposed based on Łukasiewicz logic for the description of the rules and the fuzzy memberships to construct concise and highly comprehensible fuzzy rules. In our method, a genetic algorithm is applied to evolve the structure of the rules and then a gradient based optimization to fine tune the fuzzy membership functions. The introduced squashing function allows us not only to handle the approximation of the operators and the memberships in the same way, but also to efficiently calculate the derivatives of the membership functions. We also show applications of the model on the UCI machine learning database.

Index terms – Łukasiewicz logic, squashing function, rule based classifier

1 Introduction

Fuzzy rule extraction/refinement methods based on knowledge based neural networks (KBANN) introduced by Towell and Shavlik [4] are proved to be very popular. KBANN is the most widely known hybrid learning framework with extraction algorithms like Subset and MofN.

Besides KBANN other rule extraction methods were proposed for example the RuleNet technique of McMillan, Mozer and Smolensky [2], the rule-extraction-as-learning technique of Craven and Shavlik [7], the VIA algorithm of Thrun [9], the DEDEC technique of Tickle, Orłowski and Diederich [1], the BRAINNE system of Sestino and Dillon [11], the RULENEG technique of Pop et al. [3] and the RULEX technique of Andrews and Geva [8]. The preceding techniques give crisp rules, thus one does not know the probability of correctness of a classified instance. So fuzzy rule extraction models were developed to overcome this loss of information.

Huang and Xing [10] represent the continuous valued input parameters of the network by linguistic terms and extract rules by dominance. Pedrycz and Reformat [12] at first apply a genetic algorithm to evolve a network with weights fixed to one, and then optimize it using standard backpropagation relaxing the weights to the interval $[0, 1]$. Although integer weights are lost during optimization (which are necessary for logical rules), they are corrected by rounding them to zero or one.

In this paper we propose a hybrid method to construct concise and comprehensible fuzzy rules from given training data. This paper is organized as follows. In Section 2 we overview the problem and outline the proposed solution method. Section 3 introduces the squashing function (see [6]) and soft trapezoidal memberships. The proposed fuzzy rule constructing method is presented in Section 4. Section 5 contains examples for the application of the method, and finally conclusions are in Section 6.

2 Problem definition and solution outline

Our main task is to learn fuzzy rules describing a set of training data. Comprehensibility and classification performance are the most important attributes of a rule set. The first one is determined by the size of the rule set, and the number of antecedents per rule. To avoid complex formulas, we are only concerned with disjunctions of conjunctions i.e. formulas in disjunctive normal forms. Despite the wide class of t-norms and t-conorms we will use an approximation of the well-known Łukasiewicz connectives in formulas.

The training data is supposed to be a set of n -dimensional real-valued vectors \mathbf{x}_i ($i = 1 \dots n_d$).

A target class c_i ($i = 1 \dots n_c$) is assigned to every training data. We suppose the input values are in the interval $[0, 1]$. This normalization does not constrain applicability. The target class labels are transformed into binary valued vectors.

In short, the three-stage rule construction algorithm is the following.

- First, the training data is fuzzified by the trapezoidal membership functions covering each input dimension.
- Second, the logical structures of the rules are evolved by a genetic algorithm.
- Third, a gradient based local optimization is applied in order to refine the membership functions.

3 Preliminaries

The third step of the rule construction algorithm requires that both the membership functions and the logical connectives have a continuous gradient. The Łukasiewicz operators and the widely used trapezoidal memberships do not fulfill this requirement. As a solution an approximation of them is needed.

Let us introduce a common notation for the Łukasiewicz connectives and the trapezoidal memberships. Let

$$[x] = \min(1, \max(0, x)) \quad (1)$$

and call it cut function, and let

$$[x]_{a,\lambda} = [(x - a)/\lambda + 1/2] \quad (2)$$

be the generalized cut function. So the Łukasiewicz conjunction and disjunction are

$$c(x, y) = [x + y - 1] \quad d(x, y) = [x + y]. \quad (3)$$

Trapezoidal (or triangular) membership functions are usually defined as the conjunction of two monotone functions. Here we use the simpler form

$$TR(x; a_1, \lambda_1, a_2, \lambda_2) = [x]_{a_1, \lambda_1} - [x]_{a_2, \lambda_2}, \quad (4)$$

where $a_1, a_2, \lambda_1, \lambda_2 \in \mathbb{R}$ are the left and right centers and widths of the slopes. Note that $a_1 + \lambda_1/2 < a_2 - \lambda_2/2$ should hold. As a special case, if $a_2 - a_1 = \lambda_1/2 + \lambda_2/2$ then we get a triangular membership function.

The extension of the connectives to n input values is simple by associativity. Note that they have the common form

$$o(\mathbf{x}) = \left[\sum_{i=1}^n x_i - A \right] \quad (5)$$

where A determines the type of the operator. If $A = n - 1$ then it is a t-norm, and if $A = 0$ then it is a t-conorm. The fact, that the type is determined only by a single parameter value is a useful property of this operator family.

Previously the authors proposed the so-called squashing function, an approximation of the generalized cut function. Its definition is the following:

Definition 1 *The squashing function is [6]*

$$S_{a,\lambda}^{(\beta)}(x) = \frac{1}{\lambda\beta} \ln \frac{1 + e^{\beta(x-(a-\lambda/2))}}{1 + e^{\beta(x-(a+\lambda/2))}} = \frac{1}{\lambda\beta} \ln \frac{\sigma_{a+\lambda/2}^{(-\beta)}(x)}{\sigma_{a-\lambda/2}^{(-\beta)}(x)}.$$

where $x, a, \lambda, \beta \in \mathbb{R}$ and $\sigma_d^{(\beta)}(x)$ denotes the logistic function:

$$\sigma_d^{(\beta)}(x) = \frac{1}{1 + e^{-\beta \cdot (x-d)}}. \quad (6)$$

By increasing the value of β , the squashing function approaches the generalized cut function. The parameters a and λ determine its center and width.

The error of the approximation can be upper bounded by c/β , which means that by increasing the parameter β , the error decreases by the same order of magnitude.

The derivatives of the squashing function are easy to calculate and can be expressed by sigmoid functions and itself:

$$\frac{\partial S_{a,\lambda}^{(\beta)}(x)}{\partial x} = \frac{1}{\lambda} \left(\sigma_{a-\lambda/2}^{(\beta)}(x) - \sigma_{a+\lambda/2}^{(\beta)}(x) \right) \quad (7)$$

$$\frac{\partial S_{a,\lambda}^{(\beta)}(x)}{\partial a} = \frac{1}{\lambda} \left(\sigma_{a+\lambda/2}^{(\beta)}(x) - \sigma_{a-\lambda/2}^{(\beta)}(x) \right) \quad (8)$$

$$\frac{\partial S_{a,\lambda}^{(\beta)}(x)}{\partial \lambda} = -\frac{1}{\lambda} S_{a,\lambda}^{(\beta)}(x) + \frac{1}{2\lambda} \left(\sigma_{a+\lambda/2}^{(\beta)}(x) + \sigma_{a-\lambda/2}^{(\beta)}(x) \right) \quad (9)$$

By using squashing functions one can approximate the Łukasiewicz operators and piecewise linear membership functions (i.e. trapezoidal or triangular) by substituting the cut function. The approximated membership functions are called soft and defined as:

$$ATR(x; \beta, a_1, \lambda_1, a_2, \lambda_2) = S_{a_1, \lambda_1}^{(\beta)}(x) - S_{a_2, \lambda_2}^{(\beta)}(x). \quad (10)$$

The derivatives of a soft trapezoidal are simply the derivatives of the proper squashing function.

The constructive rule learning method is based on these soft trapezoidal memberships and approximated Łukasiewicz operators.

4 The structure and representation of the rules

The first step of rule learning is a discretization procedure, the fuzzification of the training data. Every input interval is equally divided into k fuzzy sets, where each fuzzy set is a soft triangular or trapezoidal one. Each element of the input vector is fuzzified by these membership functions, so that an n -dimensional data is represented by kn fuzzy membership values. From now on we will denote the fuzzified input data as x_{ij} , ($i = 1 \dots n$, $j = 1 \dots k$). The advantage of the initial fuzzification is that the output will not only provide crisp yes/no answers but classification reliabilities, too.

A set of rules is represented by a constrained neural network in the following way. The activation functions of the neurons are squashing functions with fixed $a = 1/2$ and $\lambda = 1$, and all weights of the network are zero or one. The network is further restricted to one hidden layer with any number of neurons. There are two kinds of neurons in the network: one functioning as a Łukasiewicz t-norm and one as a Łukasiewicz t-conorm (eq. 3), both approximated by the squashing function. Since the activation function of a neuron is given, *its type is determined by the neuron's bias*. A neuron is conjunctive if it has a bias of $n - 1$ (where n is the number of its input synapses), and disjunctive if it has a zero bias. With a given network structure these biases are constant, but for every new network with a different structure these biases must be recalculated to preserve the types of the neurons. The network is additionally constrained so that the hidden layer contains only conjunctive neurons and the output layer contains only disjunctive neurons. These restrictions affect the shape of the decision surface, too. The representable decision borders are parallel to the axes, and the decision surface is a union of local ridges.

Every output neuron corresponds to one rule. Because of the special structure of the network every rule is in disjunctive normal form (DNF). For multi class problems several networks (with one output

node) can be trained, one network per class. The output class is decided by taking the maximum of the activations of the networks' output.

These restrictions on the activation function, the weights and the structure of the network are advantageous for the following reasons. First, a fuzzy logical formula (rule) can be directly assessed from the network. Second, the complexity of the represented logical formulas are greatly reduced. See e.g. [5] for the high complexity of directly extracted formulas from neural networks caused by real valued weights. The third advantage of this special network structure is its high comprehensibility, which means that the learned rules are easily human-readable.

The model has three global parameters.

- The number of conjunctive neurons in the hidden layer. Because a hidden neuron corresponds to one local decision region, this is mainly determined by the complexity of the problem.
- The technical parameter denoted by β controls the power of the approximation. A small β gives smooth membership and activation functions, while a large β gives a better approximation of triangular and trapezoidal membership functions. So the value of β directly affects the smoothness of the decision surface.
- The number of fuzzy sets each input range is divided. It can be modified as necessary to get an adequately fine resolution of the feature space.

5 The optimization process

The model defined in the previous section is able to arbitrarily approximate a function with sufficiently many fuzzy sets and hidden neurons. Our aim is to give a good approximation of the input-output relation by a modest number of parameters.

We use a similar approach to Pedrycz and Reformat [12] and Huang and Xing [10] for the description of the rule set but the optimization process is different. The main differences are the fixed network weights and the gradient based fine tuning of the memberships.

The proposed hybrid learning method consists of three separate steps. After the initial fuzzification, first we fix the fuzzy sets of the input and by using a genetic algorithm the synapses of the network are optimized. This optimization gives rules that roughly describe the relation between the input/output training data, so it has to be further refined. In the third step a gradient based local optimization method does the fine-tuning by optimizing the parameters of the fuzzy sets. The latter two steps are discussed in more detail.

5.1 Rule optimization by GA

The network is defined so that its weights can be only zero or one. In other words it means that either there is a synapse between two neurons in successive layers or not. In the first step this structure is optimized by a genetic algorithm to give the best possible result. It is obvious to represent the network structure by a bit string, where a bit corresponds to a connection between the neurons. A simple fitness function of the genetic algorithm is the negative of the sum of squared errors between the network output and the target value.

$$F(\mathbf{x}) = -\sum_{i=1}^n (z_i - t_i)^2, \quad (11)$$

where \mathbf{z} denotes the output of the network and \mathbf{t} denotes the desired output or target value and n is the number of training data. Of course, other fitness functions are reasonable too, for example by subtracting from F a value proportional to the number of synapses. This way rewarding structures with less synapses.

The network optimized by the genetic algorithm will contain the necessary synapses to roughly describe the connection between the input and output data with the initial fuzzy sets. This rule set is coarse because the initial fuzzy sets most likely do not suit the problem well.

5.2 Gradient based local optimization of memberships

The refinement of the initial fuzzy sets is achieved by fine tuning the parameters of the soft trapezoidal membership functions. Our purpose of using soft membership functions was to have the opportunity to use a simple gradient based local optimization algorithm. The optimization is the following: modify the parameters of the fuzzy sets so that the overall error of the network decreases. By applying this optimization the resulting set of rules will possibly have a better description of the underlying system. We note that only those fuzzy sets are optimized which have (an indirect) connection to the output neuron. It is because the gradient of the not connected ones is zero, thus the optimization algorithm does not change their value.

In order to examine the gradient of the error of the network we must introduce some notations. Let W_h denote the matrix of weights between the input and the hidden layer, W_o the vector of weights between the hidden and the output layer (since there is only one output neuron), and \mathbf{b} the biases of the hidden layer. Let \mathbf{x}_i denote the input of the network, where $i = 1 \dots n_d$. Let \mathbf{y}_i denote the activation of the hidden neurons. Like above z_i denotes the network output.

The error of the network is the following.

$$E = \frac{1}{2} \sum_{i=1}^n (z_i - t_i)^2 \quad (12)$$

Let us denote the set of parameters by \mathbf{p} . These are the parameters of the trapezoidal fuzzy sets, four for each one: the center and width of its left and right sides. The partial derivative of E by \mathbf{p} is

$$\frac{\partial E}{\partial \mathbf{p}} = \sum_{i=1}^n \left(\frac{\partial z_i}{\partial \mathbf{p}} \right)^T (z_i - t_i). \quad (13)$$

In the network z_i is calculated by the following formula

$$z_i = S_{1/2,1}^{(\beta)}(W_o \mathbf{y}_i) \quad (14)$$

because all the biases of the output layer are zero. Its partial derivative according to \mathbf{p} is

$$\frac{\partial z_i}{\partial \mathbf{p}} = \frac{\partial S_{1/2,1}^{(\beta)}(W_o \mathbf{y}_i)}{\partial (W_o \mathbf{y}_i)} W_o \frac{\partial \mathbf{y}_i}{\partial \mathbf{p}}, \quad (15)$$

The partial derivatives of the hidden neurons' activation can be calculated similarly because

$$\mathbf{y}_i = S_{1/2,1}^{(\beta)}(W_h \mathbf{x}_i - \mathbf{b}), \quad (16)$$

so

$$\frac{\partial \mathbf{y}_i}{\partial \mathbf{p}} = \text{Diag} \left(\frac{\partial S_{1/2,1}^{(\beta)}(W_h \mathbf{x}_i - \mathbf{b})}{\partial (W_h \mathbf{x}_i - \mathbf{b})} \right) W_h \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}}, \quad (17)$$

where $\text{Diag}(\xi)$ denotes a diagonal matrix constructed from the vector ξ .

Because the network's inputs are fuzzified values according to given trapezoidal fuzzy memberships, its partial derivatives $\frac{\partial \mathbf{x}_i}{\partial \mathbf{p}}$ can be easily calculated.

The role of the parameter β is very important in the learning process. If its value is too low, there is no real distinction between the different fuzzy sets on the same input interval. If its value is too high (i.e. the squashing function approximates the generalized cut function very well), the optimization is not effective since the gradient of it is either zero or a non-zero constant. For these reasons this optimization step is realized as an iterative process with increasing β values. As a result the final approximation is negligible and the fuzzy sets are represented by piecewise-linear trapezoidal or triangular memberships.

After the two optimization steps the set of rules can be easily extracted from the network. There is a one-to-one correspondence between a network structure and a set of rules. The advantage of this rule learning method is twofold. First, the rules are easily interpretable fuzzy rules (because of the disjunctive normal form) with expressed confidence in the result. However, interpretability could be further increased by constraining the possible settings of membership functions, or by assigning linguistic variables to final membership functions. Second, there are no real valued weights in the network during the optimization which would have to be rounded (and thus losing information) to get a logic interpretation of the input/output relation of the training data.

6 Applications

In this section we show some examples of the above defined rule construction method. The example problem sets are the Iris, Wine, Ionosphere and the Thyroid datasets from the UCI machine learning repository. In all four experiments the genetic algorithm was run with the following setting:

population:	100
max. generations:	100
crossover method:	scattered
mutation prob.:	2%

The following shorthand notation will be used for the description of membership functions.

Notation 2 *Let us denote a trapezoidal membership function by*

$$[a_1 <_{\lambda_1} x <_{\lambda_2} a_2], \quad (18)$$

where a_i denote the centers and λ_i denote the widths of the left and right slopes. If one side of the trapezoid is outside of the corresponding input interval then it is omitted.

The Iris dataset is the following. The input feature space is four dimensional, which comprises of the sepal length, the sepal width, the petal length and the petal width of an Iris flower. One has to decide the class of the Iris flower i.e. whether it is an Iris Setosa, an Iris Virginica or an Iris Versicolor. There are 150 entries in the dataset. Three networks were trained, one for each class. Each input interval was divided by three trapezoidal fuzzy sets, the number of hidden neurons was one in each case. The learned rules for the Iris problem are:

- Iris Setosa: $[x_3 <_{1.7} 3.8]$

- Iris Virginica: $[1.5 <_{0.5} x_4]$
- Iris Versicolor: $[0.35 <_{3.76} x_3 <_{1.55} 6.6]$
AND $[0.27 <_{1.28} x_4 <_{0.32} 1.9]$

These rules give 96% accuracy with 5 misclassified samples. Only two features are used, and the average certainty factors are [98% 92% 96%] for the classes.

The Wine dataset contains 178 instances of 13 dimensional real-valued input vectors. There are three types of wines to classify, so three separate networks were trained. The following rule set has been learned with three fuzzy sets for each input:

- Wine 1: $[435 <_{683} x_{13}]$
- Wine 2: $[x_{10} <_{3.36} 5.9]$
- Wine 3: $[x_7 <_{1.26} 1.74]$

These rules give 95% accuracy with 6 misclassified and 3 undecided samples. Note that only three features are used (x_7, x_{10}, x_{13}) in the rules. The average certainty factors are [88% 85% 85%].

The Ionosphere dataset is a binary classification problem which contains 351 instances of 34 dimensional radar data. One has to decide whether there is evidence of some type of structure in the ionosphere. The following rule has been learned with only one hidden neuron:

$$[0.69 <_{0.5} x_1] \text{ AND } [-0.19 <_{0.013} x_5]$$

With 1/2 threshold, this simple rule gives 88% accuracy.

The Thyroid gland dataset contains 215 instances of 5 dimensional real-valued input vectors. There are three classes (normal, hypo and hyper), each class was classified with only one hidden neuron:

- Normal: $[4.92 <_{2.46} x_2 <_{6.95} 14.57]$
- Hypo: $[x_2 <_{3.75} 6.2]$
- Hyper: $[10.95 <_{4.31} x_2 <_{12.77} 36.8]$

These rules give 94.8% accuracy with 11 misclassifications. Note that only x_2 is used. The average certainty factors are [95% 88% 94%].

7 Summary and conclusions

In this paper a hybrid method with genetic algorithm and gradient based local optimization is introduced for fuzzy logical rule learning. The genetic algorithm is used to find those features of the input with which the separation of classes is optimal. The second step of the method refines the initial fuzzy membership functions in order to give better accuracy. The model is novel in the sense that logical information is directly available and that the fuzzy membership functions are optimized instead of the network weights, so that there is no need to round the weights to integers and thus lose information. The rules are concise and easily understandable because of their disjunctive normal form which is guaranteed by the special network structure. Tests show that the method is very useful in revealing hidden input/output relations.

Future work includes further extending the model, for example by preprocessing data or by adding the capability to handle nominal and ordinal data or missing values, too.

References

- [1] A. B. Tickle, M. Orłowski, J. Diederich. Dedec: decision detection by rule extraction from neural networks. Technical Report NRC QUT September 1994, Queensland University, 1994.
- [2] C. McMillan, M. C. Mozer and P. Smolensky. The connectionist scientist game: rule extraction and refinement in a neural network. In *Proc. 13th Annual Conference of the Cognitive Science Society. Hillsdale, NJ, USA, 1995*.
- [3] E. Pop, R. Hayward and J. Diederich. Ruleneg: extracting rules from a trained ann by stepwise negation. Technical Report NRC QUT December 1994, Queensland University, 1994.
- [4] G. Towell and J. Shavlik. The extraction of refined rules from knowledge based neural networks. *Machine Learning*, 131:71–101, 1993.
- [5] J. L. Castro and E. Trillas. The logic of neural networks. *Mathware & Soft Computing*, 5:23–37, 1998.
- [6] József Dombi and Zsolt Gera. The approximation of piecewise linear membership functions and Łukasiewicz operators. *Fuzzy Sets and Systems*, 154:275–286, 2005.
- [7] M. W. Craven and J. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning: Proc. of the 11th International Conference, San Francisco, CA, USA, 1994*.

- [8] R. Andrews and S. Geva. Inserting and extracting knowledge from constrained error back propagation networks. In *Proc. 6th Australian Conference on Neural Networks, Sydney*, 1995.
- [9] S. B. Thrun. Extracting provably correct rules from artificial neural networks. Technical Report IAI-TR-93-5, Institut for Informatik III Universitat Bonn, Germany, 1994.
- [10] S. Huang and H. Xing. Extract intelligible and concise fuzzy rules from neural networks. *Fuzzy Sets and Systems*, 132:233–243, 2002.
- [11] S. Sestino and T. Dillon. Automated knowledge acquisition of rules with continuously valued attributes. In *Proc. 12th International Conference on Expert Systems and their Applications, Avignon, France*, pages 645–656, 1992.
- [12] W. Pedrycz and M. Reformat. Genetically optimized logic models. *Fuzzy Sets and Systems*, 150:351–371, 2005.

Figure captions:

Fig. 1: On the left the squashing function with parameters $a = 0$, $\lambda = 1$ and $\beta = 16$. On the right soft trapezoidals with various β values

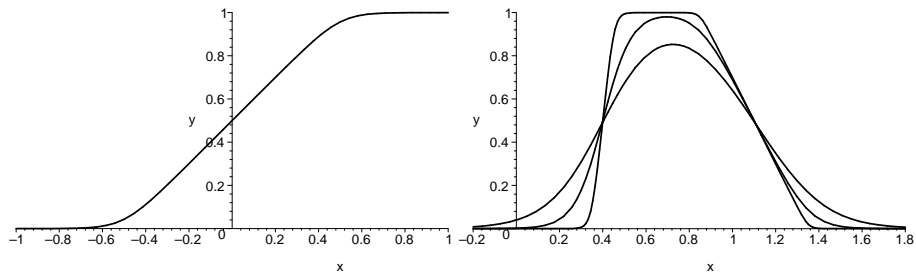


Figure 1: On the left the squashing function with parameters $a = 0$, $\lambda = 1$ and $\beta = 16$. On the right soft trapezoidals with various β values