Genetic Algorithm with Gradient Based Tuning for Constructing Fuzzy Rules

Zsolt Gera, József Dombi

Department of Computer Algorithms and Artificial Intelligence
University of Szeged
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: {gera|dombi}@inf.u-szeged.hu

Abstract: This paper presents a hybrid method to construct concise and comprehensible fuzzy rules from training data. The construction procedure consists of a genetic algorithm, which determines the rulebase, and a gradient based optimization for the tuning of the membership functions. An approximation of the well-known Lukasiewicz logic is used to describe both the fuzzy connectives and the memberships, which by this way have continuous derivatives.

1 Introduction

In the past decades neural networks were successfully used in input-output mapping with very good learning abilities. However the comprehensibility of neural networks are low, they lack of logical justification, one does not know why a trained network gives a certain answer. Its knowledge is distributed in its weights and structure and it cannot be directly translated into simple logical formulas. The problem of creating logical rules to describe a set of input-output data or a black box system's internal behavior is still an active area of Computational Intelligence. In this paper we propose a method to construct concise and comprehensible fuzzy rules from given training data. This paper is organized as follows. In section 2 we give the problem definition and outline the proposed solution method. In section 3 we define the squashing function (see [1]) which is intensively used later on and discuss some of its main properties. The proposed fuzzy rule constructing method is presented in sections 4 and 5. Section 6 contains an example for the application of the method.

2 Problem Definition and Solution Outline

Our main task is to create a system which produces fuzzy rules describing a set of training data. The training data is supposed to be a set of points x_i (i = 1…nData) in the n-dimensional space. A target class of c_i (i = 1...nClass) is assigned to every training data. The interval of the input values does not constrain applicability since all values can be scaled down to [0,1]. The target class labels (e.g. the color of the instance) should be transformed into binary valued vectors.

Comprehensibility and accuracy are the most important attributes of the rules. The first one is determined by the size of the rule set, and the number of antecedents per rule. A rule set is accurate if it can correctly classify previously unseen examples. The following restrictions are made on the structure of the rules to avoid too complex formulas. First, we are only concerned with disjunctions of conjunctions i.e. formulas in disjunctive normal forms. Second, we use the well-known Łukasiewicz fuzzy logic for the conjunctions and disjunctions. In short, the three-stage rule construction algorithm is the following.

- First, the training data is fuzzified by soft 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.

The above outlined approach is similar to radial basis function networks in the sense that each conjunction defines a region in feature space, and in an upper level the similar regions are united.

The third step of the rule construction algorithm requires that both the fuzzy membership functions and the logical connectives have a continuous gradient. The Łukasiewicz operator family and the widely used trapezoid membership functions do not fulfill this requirement, hence an approximation of them is needed before the description of the algorithm.

3 The Squashing Function

The Łukasiewicz fuzzy operator class (see e.g. Fodor and Roubens [5], Klement et al. [6], Ackerman [7], Cignoli et al. [8]) is commonly used for various purposes. In this well known operator family the cut function (denoted by square brackets: $[\cdot]$) plays an important role. We can get the cut function from x by taking the maximum of 0 and x and then taking the minimum of the result and 1.

Definition 1: *Let the cut function be:*

$$[x] = \min(\max(0, x), 1) = \begin{cases} 0, & \text{if} \quad x \leq 0 \\ x, & \text{if} \quad 0 < x < 1 \\ 1, & \text{if} \quad 1 \leq x \end{cases}$$

and let the generalized cut function be:

$$[x]_{a,\lambda} = \left[\lambda(x-a) + \frac{1}{2}\right] = \begin{cases} 0, & \text{if } x \leq a - \frac{1}{2\lambda} \\ 1, & \text{if } a + \frac{1}{2\lambda} \leq x \\ \frac{x-a}{\lambda} + \frac{1}{2} & \text{otherwise} \end{cases}$$

where $a \in R$ *is its center and* $\lambda \in R^+$ *is the tangent of its slope.*

The Łukasiewicz connectives can be constructed using the cut function. The formulas of the conjunction, disjunction, implication and negation are the following.

Definition 2: The Łukasiewicz connectives are

$$\begin{aligned} c(x,y) &= [x + y - 1], \\ d(x,y) &= [x + y], \\ i(x,y) &= [1 - x + y], \\ n(x) &= 1 - x, \end{aligned}$$

where $x,y \in [0,1]$.

The extension of the above expressions to n input values is simple because they are associative [9]. The n-ary formulas of the conjunction and the disjunction are:

$$c(x_1, x_2, \ldots, x_n) = \left[\sum_{i=1}^{n} x_i - (n-1)\right]$$

$$d(x_1, x_2, \ldots, x_n) = \left[\sum_{i=1}^{n} x_i\right]$$

As it can be seen from the above formulas, they have a common form

$$o(x_1, x_2, \ldots, x_n) = \left[\sum_{i=1}^{n} x_i - A\right]$$

where the type of the operator is determined only by the parameter A. If A = n-1 then the operator is conjunctive, and if A = 0 then the operator is disjunctive. The fact, that the type is determined only by a single parameter's value is a useful property of this operator family.

Triangular and trapezoidal membership functions are very common in fuzzy control and modeling because of their easy handling. The generalized cut function can be used to describe these piecewise linear membership functions. This way, the connectives and the membership functions are similar in the sense that both are constructed by the generalized cut function. A trapezoidal membership function can be constructed as the conjunction of two generalized cut functions as

$$\begin{aligned} c([x]_{a_1,\lambda_1}, n([x]_{a_2,\lambda_2})) &= [[x]_{a_1,\lambda_1} + 1 - [x]_{a_2,\lambda_2} - 1] \\ &= [[x]_{a_1,\lambda_1} - [x]_{a_2,\lambda_2}], \end{aligned}$$

where $a_1, a_2, \lambda_1, \lambda_2$ are real numbers and $a_1+1/(2\lambda_1) < a_2+1/(2\lambda_2)$. As a special case, by simple calculation, if $a_2 - a_1 = 1/(2\lambda_1) + 1/(2\lambda_2)$ then we get a triangular membership function with its core containing exactly one element $a_1+1/(2\lambda_1)$.

Previously we proposed an approximation of the generalized cut function in Dombi and Gera [4], where we investigated the properties of the squashing function in more detail. The definition of the squashing function is the following:

Definition 3: The squashing function is

$$\begin{aligned} S_{a,\lambda}^{(\beta)}(x) &= \ln\left(\frac{\sigma_{a+\frac{1}{2\lambda}}^{(-\beta)}(x)}{\sigma_{a-\frac{1}{2\lambda}}^{(-\beta)}(x)}\right)^{\lambda/\beta} \\ &= \ln\left(\frac{1+e^{\beta\left(x-a+\frac{1}{2\lambda}\right)}}{1+e^{\beta\left(x-a-\frac{1}{2\lambda}\right)}}\right)^{\lambda/\beta} \end{aligned}$$

where x, a, λ, β ∈ R and $\sigma_d^{(\beta)}(x)$ *denotes the sigmoid function* $(1+e^{-\beta(x-d)})^{-1}$.

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

$$\begin{aligned} \frac{\partial S_{a,\lambda}^{(\beta)}(x)}{\partial x} &= \lambda\left(\sigma_{a+\frac{1}{2\lambda}}^{(-\beta)}(x) - \sigma_{a-\frac{1}{2\lambda}}^{(-\beta)}(x)\right) \\ \frac{\partial S_{a,\lambda}^{(\beta)}(x)}{\partial a} &= \lambda\left(\sigma_{a+\frac{1}{2\lambda}}^{(\beta)}(x) - \sigma_{a-\frac{1}{2\lambda}}^{(\beta)}(x)\right) \\ \frac{\partial S_{a,\lambda}^{(\beta)}(x)}{\partial \lambda} &= \frac{1}{\lambda} S_{a,\lambda}^{(\beta)}(x) - \frac{\left(\sigma_{a+\frac{1}{2\lambda}}^{(\beta)}(x) + \sigma_{a-\frac{1}{2\lambda}}^{(\beta)}(x)\right)}{2\lambda} \end{aligned}$$

Using the squashing function one can approximate the Łukasiewicz operators by substituting the cut function to the squashing function.

Soft trapezoidal and triangular membership functions can be constructed by applying the approximated conjunction operator to two squashing functions. So both the membership functions and the Łukasiewicz connectives can be

approximated by the squashing function. Similarly to hard trapezoids, to describe a soft trapezoid membership function four parameters are needed, namely a_1, λ_1 and a_2, λ_2 where a_1 and λ_1 give the center and tangential of its left slope, and a_2 and λ_2 give the center and tangential of its right slope. The two β parameters of the squashing functions have to have opposite signs to form a trapezoid or triangle. For later use let us introduce a concise notation called which is part of a new framework called pliant logic.

Notation 5: Let the pliant notation of a soft trapezoidal membership function be the expression

$$\langle a_1 <_{\lambda_1} x <_{\lambda_2} a_2 \rangle_\beta$$

for a finite β, and

$$[a_1 <_{\lambda_1} x <_{\lambda_2} a_2]$$

for $\beta = \infty$.

So the approximation of a trapezoid membership function is the following:

$$\mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x) = S_{1/2,1}^{(\beta)}\left(S_{a_1,\lambda_1}^{(\beta)}(x) + S_{a_2,\lambda_2}^{(-\beta)}(x) - 1\right)$$

The derivatives of a soft trapezoidal can be easily calculated using the chain rule. Let us denote

$$y = S_{a_1,\lambda_1}^{(\beta)}(x) + S_{a_2,\lambda_2}^{(-\beta)}(x) - 1,$$
$$\mathcal{D}_+^{(\beta)}(x,u,v) = \sigma_{u+\frac{1}{2v}}^{(\beta)}(x) + \sigma_{u-\frac{1}{2v}}^{(\beta)}(x)$$
$$\mathcal{D}_-^{(\beta)}(x,u,v) = \sigma_{u+\frac{1}{2v}}^{(\beta)}(x) - \sigma_{u-\frac{1}{2v}}^{(\beta)}(x)$$

then

$$\frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial a_1} = \frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial y} \cdot \frac{\partial y}{\partial a_1} =$$
$$= \lambda_1 \cdot \mathcal{D}_-^{(-\beta)}(y, 1/2, 1) \cdot \mathcal{D}_-^{(\beta)}(x, a_1, \lambda_1)$$
$$\frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial \lambda_1} = \frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial y} \cdot \frac{\partial y}{\partial \lambda_1} =$$
$$= \frac{1}{\lambda_1}\mathcal{D}_-^{(-\beta)}(y, 1/2, 1) \cdot \left(S_{a_1,\lambda_1}^{(\beta)}(x) - \frac{1}{2}\mathcal{D}_+^{(\beta)}(x, a_1, \lambda_1)\right)$$
$$\frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial a_2} = \frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial y} \cdot \frac{\partial y}{\partial a_2} =$$
$$= \lambda_2 \cdot \mathcal{D}_-^{(-\beta)}(y, 1/2, 1) \cdot \mathcal{D}_-^{(-\beta)}(x, a_2, \lambda_2)$$

$$\frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial \lambda_2} = \frac{\partial \mu_{a_1,a_2}^{\lambda_1,\lambda_2}(x)}{\partial y} \cdot \frac{\partial y}{\partial \lambda_2} =$$
$$= \frac{1}{\lambda_2} \mathcal{D}_{-}^{(-\beta)}(y, 1/2, 1) \cdot \left(S_{a_2,\lambda_2}^{(-\beta)}(x) - \frac{1}{2} \mathcal{D}_{+}^{(-\beta)}(x, a_2, \lambda_2) \right)$$

The constructive rule learning method is based on the above defined squashing function and soft trapezoid functions.

4 The Structure and Representation of the Rules

The first step of the rule construction 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 k·n fuzzy membership values. From now on we will denote the fuzzified values as x_{ij} (i = 1...n, j = 1...k). The advantage of the initial fuzzification is that in the case of classification the output will be fuzzy valued, too. So the output has more information than simple crisp yes/no answers, we get classification reliability, too.

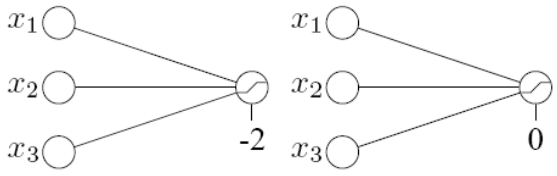

Figure 2
The same neuron with different biases −2 and 0, thus calculating the Łukasiewicz conjunction and disjunction of three inputs

A set of rules is represented by a constrained neural network. The network is constrained in the sense that the activation function of all neurons is the squashing function with fixed a = 1/2 and λ = 1, and that all the weights between the neurons are zero or one. The network is further restricted to have one hidden layer with any number of neurons. There are two kinds of neurons in the network: one functioning as a Łukasiewicz conjunction and one as a Łukasiewicz disjunction 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 (see Fig. 2). In a network 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. This structure resembles radial basis function networks, where the ridge functions over different dimensions are combined to form a local decision rule.

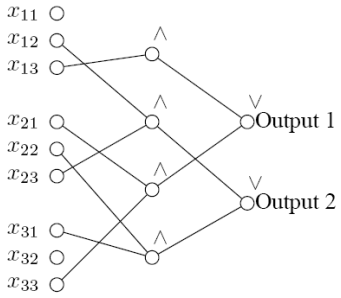

Figure 3
Illustration of an and-or neural network

These restrictions on the activation function, the weights and the structure are advantageous. First, a logical formula in Łukasiewicz logic can be directly assessed from the network. Second, the complexity of the represented logical formulas are greatly reduced (see e.g. [1] for the complexity of directly extracted formulas caused by real valued weights). However these restrictions affect the shape of the decision surface, too. The representable decision borders are parallel to the axises. See Fig. 3 for a typical network, where for example Output1 is x_{13} OR (x_{21} AND x_{33}). As it can be seen on Fig. 3, there can be input nodes which are not connected to any hidden neuron, thus their value does not count in the network's output. These nodes can be pruned to simplify the network structure.

So every output neuron corresponds to a rule. Because of the special structure of the network every rule is in disjunctive normal form (DNF). The advantage of this special constrained network structure is its high comprehensibility.

For multiclass classification 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. To sum up, the rule representation 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 and gives the cut function as the neurons'

activation function. So the value of β directly affects the smoothness of the decision surface. However it does not affect the actual decision borders.

- The number of fuzzy sets each input range is divided can be modified as necessary to get an adequately fine resolution of the feature space.

5 The Optimization Process

We use a similar approach to Pedrycz and Reformat [3] and Huang and Xing [2] 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 mixed learning method consists of two separate steps. 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 functioning of the underlying unknown system, so it has to be further refined. In the second step a gradient based local optimization method does the fine-tuning by optimizing the parameters of the fuzzy sets. Let us discuss these two steps 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 we have a synapse between two neurons in successive layers or not. In the first step this structure is optimized to give the best possible result. It is obvious to represent the network structure by a bit string, where a bit in the string means that whether there is a connection between the corresponding neurons. The fitness function of the genetic algorithm is the negative of the sum of squared errors between the network output and the target value.

$$F(\mathbf{x}) = -\sum_{i=1}^{n} ||\mathbf{z}_i - \mathbf{t}_i||^2,$$

where **z** denotes the output of the network and **t** denotes the desired output or target value and n is the number of training data. The network optimized by the genetic algorithm will only contain the necessary synapses to roughly describe the connection between the input and output data with the initial fuzzy sets. The final logical structure is coarse because the fuzzy sets most likely do not suit the problem well.

5.2 Gradient Based Optimization of Memberships

So these fuzzy sets must be refined. This refinement is achieved by fine tuning the parameters of the soft triangular or 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 logical 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 an output neuron. It is because the gradient of the not connected ones is zero, thus the optimization algorithm does not change their value.

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 gradient based optimization is not effective. After the two optimization steps (or at anytime) 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 logical formulas. The advantage of this rule extracting method is that the produced rules are easily interpretable fuzzy rules with expressed confidence in the result and that 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.

6 Example Rule Construction

In this section we show an example of the above defined rule construction method. The example problem set is the Wine dataset from the UCI machine learning repository.

The Wine dataset contains 178 instances of 13 dimensional real-valued input vectors. There are three types of wines to classify. The input data has been normalized into the interval [0,1]. The following rule set has been learned with 3 fuzzy sets per feature and 3 hidden nodes in the network:

- Wine 1: $[0.2 <_{10} x_5 <_{10} 0.7]$ AND $[0.4 <_{6.9} x_7 <_{4.4} 0.9]$
- Wine 2: $[x_{10} <_{5.6} 0.2]$
- Wine 3: $[x_7 <_{5.1} 0.2]$

These rules give 91.5% accuracy with 12 misclassified and 3 undecided samples. Note that only features x_7 and x_{10} are used in the rules. The average certainty factors of the three classes were 94%, 83% and 97.6%.

A better rulebase can be achieved by increasing the number of hidden neurons by one:

- Wine 1: $[0.3 <_{10} x_{13}]$ AND $[0.4 <_{6.5} x_7 <_{4.7} 1]$
- Wine 2: $([x_2 <_{5.8} 0.2]$ AND $[x_5 <_{3.2} 0.3])$ OR $[x_{10} <_{10} 0.2]$
- Wine 3: $[x_7 <_{5.1} 0.2]$

As it can be seen class Wine 2 uses two hidden neurons. These slightly more complex rules give 97.1% accuracy with 3 misclassified and 2 undecided samples, where the average certainty factors of the classes are 92%, 89.2% and 96.4%.

Summary and conclusions

In this paper a combined genetic algorithm – gradient based optimization method is introduced for fuzzy logical rule construction. 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 because of it. The rules are concise and easily understandable because of their disjunctive normal form which is guaranteed by the special network structure.

References

[1] J. L. Castro and E. Trillas, "The logic of neural networks," *Mathware & Soft Computing*, vol. 5, pp. 23–37, 1998

[2] S. Huang and H. Xing, "Extract intelligible and concise fuzzy rules from neural networks," *Fuzzy Sets and Systems*, vol. 132, pp. 233–243, 2002

[3] W. Pedrycz and M. Reformat, "Genetically optimized logic models," *Fuzzy Sets and Systems*, vol. 150, pp. 351–371, 2005

[4] József Dombi and Zsolt Gera, "The approximation of piecewise linear membership functions and Łukasiewicz operators," *Fuzzy Sets and Systems*, vol. 154, pp. 275–286, 2005

[5] J. Fodor and M. Roubens, *Fuzzy Preference Modelling and Multicriteria Decision Support*, Kluwer, 1994

[6] E. P. Klement, R. Mesiar and E. Pap, *Triangular Norms*, Kluwer, 2000

[7] R. Ackermann, *An introduction to many-valued logics*, Dover, 1967

[8] R. Cignoli, I. M. L. D'Ottaviano and D.Mundici, "Algebraic foundations of many-valued reasoning," *Trends in Logic*, vol. 7, 2000

[9] D. Dubois and H. Prade (Eds.), *Fundamentals of fuzzy sets*, Kluwer, 2000