Comparison of Fuzzy Rule-based Learning and Inference Systems

Krisztián Balázs[1], László T. Kóczy[1,2], János Botzheim[1,3]

[1] Department of Telecommunication and Media Informatics, Budapest University of Technology and Economics, Hungary
[2] Faculty of Engineering Sciences, Széchenyi István University, Győr, Hungary
[3] Department of Automation, Széchenyi István University, Győr, Hungary
E-mail: krisztian.balazs.mail@gmail.com, {koczy, botzheim}@sze.hu, {koczy, botzheim}@tmit.bme.hu

Abstract: In our work we have compared various fuzzy rule based learning and inference systems. The base of the investigations was a modular system that we have implemented in C language. It contains several alternative versions of the two key elements of rule based learning – namely, the optimization algorithm and the inference method – which can be found in the literature. We obtained very different properties when combining these alternatives (changing the modules and connecting them) in all possible ways. The investigations determined the values of the quality measures (complexity and accuracy) of the obtained alternatives both analitically and experimentally where it was possible. Based on these quality measures the combinations have been ordered according to different aspects.

Keywords: fuzzy systems, rule based learning, optimization, complexity

1 Introduction

Because of their favorable properties, the scope of intelligent technical applications based on soft-computing methods is continuously expanding in problem fields accepting sub-optimal solutions. As a result, the use of fuzzy rule based learning and inference systems, as intelligent system components, is also growing. However, both theory and application practice still contain many unsolved questions, hence researching the theory and applicability of such systems is obviously an important and actual task. As the results of the investigations on applicability mean some kind of labelling for the involved methods, there are two outcomes of these investigations. One is the fact that they result in practical knowledge for industrial users covering which techniques offer better possibilities, which ones are worth to be selected for integration into their respective products.

The other one is a feedback to researchers regarding to in which direction should they continue their works.

Fuzzy rule based learning and inference systems are machine learning systems, so they can be characterized by the following basic properties: time and space complexity during learning and during inference based on the rules learned, furthermore, the accuracy of learning.

Our work aims the investigation methods of such systems. The starting point of this investigation is a modular system that we have implemented in C language. It contains several alternative versions of the two key elements of rule based learning, namely, the optimization algorithm and the inference method, which can be found in the literature. We have chosen deterministic gradient techniques as well as stochastic soft-computing algorithms for optimization, and we have chosen two inference methods, one using dense and the other one using spare rule bases. The former method is the Mamdani-inference [2], the latter is the stabilized KH-interpolation technique [3,4]. We obtained very different properties when combining these alternatives in all possible ways. The values of the above mentioned quality measures (complexity and accuracy) have been determined both analytically and experimentally where it was possible. As a result, fuzzy rule based learning and inference systems have been labelled and ordered according to different aspects.

Actually, our work is far from being complete, because we definitely have not implemented and compared all optimization and inference methods that can be found in the literature. This paper only tries to give a concept how such comparative investigations can be carried out.

The next section gives a brief overview of the algorithms and techniques used. After that, the structure of the modular system will be explained. The 4th section discusses the theoretical determination of complexity. The simulation results will be explained in the 5th section. Finally, we summarize our work and draw some conclusions.

2 Overview of the Algorithms and Techniques Used

In order to carry out this investigation, it is necessary to overview some theoretical points. Obviously, one of these is the theory of fuzzy rule based inference systems, and another one is machine learning. The former includes the basic mathematics of fuzzy systems, furthermore, the knowlegde of rule based inference methods, while the latter implies the familiarity to the techniques a wide scope of numerical optimization.

The following subsections aim to give a brief overview of some important points of these theoretical aspects, which will be referred to later repeatedly in the paper.

2.1 Fuzzy Rule-based Inference Systems

The structure of a fuzzy rule based inference system and the forms of fuzzy rules are well known from the literature. The idea was first proposed in [1] and then adopted in an easier computable frame in [2].

Mamdani-inference is so widely applied that its description will be omitted here, we just refer to some basic textbook on fuzzy control (e.g. [5,6]). The stabilized Kóczy-Hirota (in short KH-) interpolation method is based on a certain interpolation of a family of distances between fuzzy sets in the rules and in the observation [4]. Unlike the original KH-interpolation [3], it does not consider only the two closest surrounding rules, but it takes all the rules and computes the conclusion based on the consequent parts weighted by the distances.

Unfortunately, due to the characteristic of this technique, the stabilized KH-interpolation may result in abnormal fuzzy sets (in membership functions that do not represent fuzzy sets) as conclusions.

2.2 Machine Learning

Machine learning [7] means a process where parameters of a 'modelling system' are being adjusted so that its behavior becomes similar to the behavior of a 'system to model'. Since the behavior can be characterized by input-output pairs, the aim of the learning process can be formulated so that the modelling system should give similar outputs for the input as the original system does. If a function $\phi(\boldsymbol{x})$ denotes the system to model and $f(\boldsymbol{x},\boldsymbol{p})$ denotes the modelling system, where $\boldsymbol{x} \in \boldsymbol{X}$ is the input vector and $\boldsymbol{p}$ is the adjustable parameter vector, the previous requirement can be expressed as follows:

$$\forall x \in X : \phi(x) \approx f(x, p)$$

In a supervised case the learning happens using a set of training samples (input-output pairs). If the number of samples is m, the input in the i^{th} sample is x_i, the desired output is $d_i = \phi(\boldsymbol{x_i})$ and the output of the model is $y_i = f(\boldsymbol{x_i},\boldsymbol{p})$, the following formula can be used:

$$\forall i \in [1, m] : d_i \approx y_i$$

The error (ε) shows how similar the modelling system to the system to model is. Let us use a widely applied definition for the error, the Sum of Square Errors (SSE):

$$\varepsilon = \sum_{i=1}^{m} (d_i - y_i)^2$$

Obviously the task is to minimize this function ε. It can be done by optimization algorithms.

2.3 Optimization

In our case the optimum means minimum, as the error of the learning system must be minimized by choosing the proper parameter vector $\boldsymbol{p}$. Hereafter the optimization will be discussed in this sense.

There are several deterministic techniques as well as stochastic algorithms applied for the optimization. Some of them will be presented below.

2.3.1 Gradient Methods

The main idea of the gradient methods is to calculate the gradient of the objective function (in our case the error function) at the actual point and step towards lower values using it by modifying $\boldsymbol{p}$.

One of the most frequently used methods of this type is the backpropagation algorithm (BP) [7,8]. Each of its iterations contains the following steps: computing the gradient vector, multiplying it with a so-called bravery factor and finally, subtracting it from the actual position to obtain the new position. If the gradient vector function is given, the vector can be obtained by calling this function, otherwise by a pseudo-gradient computation.

A more advanced and effective technique is the Levenberg-Marquardt algorithm (LM) [9,10]. It computes not only the gradient vector, but a vector of the Taylor series estimation as well. The direction of the step that will be applied is between these vectors. For this, a Jacobian matrix needs to be computed. Each row of the matrix contains the gradient vector of a residual function, where a residual function is the difference of the given and the desired answer in the training sample. If the Jacobian matrix computing function is given, the matrix can be obtained by calling this function, otherwise by a pseudo-Jacobian computation.

A stopping condition can be integrated in the Levenberg-Marquardt method that can be checked in each iteration so that if it is fulfilled, it means that the algorithm cannot be continued effectively anymore, so further iterations are nearly useless. For the sake of latter refering, let us say that the stopping condition is active if the algorithm stops when it is fulfilled and inactive otherwise.

After a proper amount of iterations, as a result of the gradient steps, the algorithms find the nearest local minimum quite accurately. However, these techniques are very sensible to the location of the starting point, because in order to find the global optimum, the starting point must be located close to it.

2.3.2 Evolutionary Computation Methods

The evolutionary computation methods, like genetic algorithm (GA) [11] or bacterial evolutionary algorithm (BEA) [12], imitate the abstract model of the evolution observed in nature. Their aim is to change the ‘individuals’ in the

'population' by the evolutionary operators to obtain better and better ones. The goodness of an individual can be measured by its 'fitness'. If an individual represents a solution for a given problem, the algorithms try to find the optimal solution for the problem. Thus, in our case the individuals are potentially optimal parameter vectors, the fitness function is the error function and the best individual holds the (quasi-) optimal values for $\boldsymbol{p}$.

If an evolutionary algorithm uses an elitist strategy, it means that the best ever individual will always survive and appear in the next generation. As a result, at the end of the algorithm the best individual will be presented as the optimal solution.

2.3.3 Memetic Algorithms

Evolutionary computation techniques explore the whole error surface because of their characteristic, but they slowly approach to the local minima. Gradient and evolutionary methods may be combined [13,14], for example, if in each iteration for each individual a gradient procedure is applied. This combination has the advantage that every individual gets close to the nearest local minimum. This way we can unite the advantages of the gradient and the evolutionary techniques, namely, we find the local minima quite accurately on the whole error surface. Therefore we obtain the global minimum, i.e. the optimal parameter vector quite accurately.

3 Structure of the System

The system we have implemented has a modular structure. The modules compose hierarchical levels. The larger, logically more coherent modules will be called 'main modules', while the ones contained in the main ones will be 'sub-modules'. Due to the hierarchical structure, the modules are nested in each other, thus a main module can be the sub-module of another modul.

The outermost layer, which is the largest main module, is a learning system frame that has a connection to a sample generator program. The learning system module contains two sub-modules that are also main modules themselves: the inference techniques module and the optimization algorithms one. Detailed descriptions of the main modules follow with aspects of module planning.

3.1 Learning Module

During the planning of the learning system we had to determine the way the user can define the system to model, furthermore the type of learning and the architecture of the modelling system, namely, the $f(\boldsymbol{x}, \boldsymbol{p})$ function to adjust.

In this approach the system to model is static. It can be represented as a function to be approximated. The type of learning is supervised learning. This type needs a training sample set and (if possible) a test sample set from the user defined problem. If they are not previously available, they can be generated from any explicit definitions by the sample generator that has been implemented as an auxiliary program. It samples the function of the system to model and produces input-output pairs.

The modelling system is a fuzzy rule base whose rules contain trapezoidal membership functions. The components of parameter vector $\boldsymbol{p}$ represent the characteristic points (breakpoints) of the membership functions, therefore the learning process adjust these points by the help of the inference and optimization sub-modules. The number of rules, the type of the inference method and the intervals of the characteristic points are user defined values.

3.2 Inference Module

The inference (main) module contains two sub-modules. One implements Mamdani-inference while the other contains stabilized KH-interpolation. In both cases the defuzzification of the conclusions is made by Center Of Gravity (COG) algorithm.

For both techniques it is necessary before the inference, to verify whether the membership functions of the rules really describe fuzzy sets. The reason of this verification is that the order of the characteristic points can be changed during the application of the optimization algorithms which might result in abnormal fuzzy sets. This verification does not affect the computational demands significantly.

In case of stabilized KH-interpolation method the conclusion is covered by a convex hull before defuzzification, so that the possible abnormal conclusions become fuzzy sets. In not abnormal case this step has no effects, because the convex hull of a fuzzy set is itself.

3.3 Optimization Module

The optimization (main) module contains four sub-modules. These implement the backpropagation and the Levenberg-Marquardt method as deterministic gradient type, and the genetic and bacterial evolutionary algorithms as stochastic soft-computing methods as it was explained in subsection 2.3. The only deflection from the mentioned descriptions are in the evolutionary algorithms, in the initialization of the starting population and in the mutation parts. In the evolutionary methods the starting populations are generated in a way that avoids abnormal fuzzy sets. During the mutation attention must also be paid for the same problem.

By implementing the four optimization modules we have obtained four additional algorithms, because both the gradient and the evolutionary techniques could be combined into a memetic algorithm and there are apparently four combination possibilities.

Thus, the eight methods that will be investigated: backpropagation (BP), Levenberg-Marquardt algorithm (LM), genetic algorithm (GA), bacterial evolutionary algorithm (BEA), memetic algorithm with backpropagation steps (MEMBP), memetic algorithm with Levenberg-Marquardt steps (MEMLM), bacterial memetic algorithm with backpropagation steps (BMABP) and bacterial memetic algorithm with Levenberg-Marquardt steps (BMALM).

4 Theoretical Determination of Complexities

Although the error of the system cannot be determined theoriticaly, the other quality measures, namely both, the time and space complexities can be computed in some cases. Unfortunately, not in all cases though, because e.g., if we make the learning stop after reaching a fix error level, the number of iterations cannot be predicted due to the stochastic character of the methods. However, if the number of steps and generated rules are deterministic, because of the problem type or system parameters, the required complexities can be determined. These complexities can be easily calculated from the implemented program code.

Let us define some parameters that will be used:

N_{dim}	*number of input dimensions*
N_{sample}	*number of training samples*
N_{rule}	*number of rules*
N_{gen}	*number of generations*
N_{ind}	*number of individuals in the population*
N_{par}	*number of parameters to adjust (length of an individual)*
α_{sel}	*selection rate ($\in$[0,1])*
N_{clone}	*number of clones*
N_{iter}	*number of iterations*
N_{res}	*number of residual functions*
T_{fun}	*evaluation time complexity of the objective function*
$T_{fitness}$	*evaluation time complexity of the fitness function*
T_{res}	*evaluation time complexity of a residual function*
T_{grad}	*evaluation time complexity of the gradient function*
T_{jac}	*evaluation time complexity of the Jacobian calculating function*

4.1 Time Complexity

In case of optimization algorithms time complexities give the necessary evaluation number of objective, residual, fitness, gradient and Jacobian matrix calculating functions. Obviously each type of optimization does not use all of these functions.

As a matter of fact, learning is an optimization process, where the optimization algorithms call objective, residual and fitness functions. These functions do inferences, hence the time complexities of learning are determined mostly by the computational demands of optimizations and inferences. The rest of the complexities come from the gradient and the Jacobian matrix calculating functions.

Time complexities of optimization algorithms, inference techniques, function evaluations and learning methods are presented in Tables 1, 2, 3 and 4, respectively. In brackets *pg* and *pJ* denote the complexities when pseudo-gradient and pseudo-Jacobian computations are done. Since both Mamdani-inference and stabilized KH-interpolation have the same complexity (see Table 2), we do not distinguish the learning based on different inferences in Table 4.

Table 1
Time complexities of optimization algorithms

Algorithm	Time complexity
BP	$O(T_{fun} + N_{iter} * T_{grad})$
BP$^{(pg)}$	$O(N_{iter} * N_{par} * T_{fun})$
LM	$O(N_{iter} * (N_{res} * T_{res} + T_{jac}))$
LM$^{(pJ)}$	$O(N_{iter} * N_{res} * N_{par} * T_{res})$
GA	$O(N_{gen} * N_{ind} * T_{fitness})$
BEA	$O(N_{gen} * N_{ind} * N_{par} * N_{clone} * T_{fitness})$
MEMBP	$O(N_{gen} * N_{ind} * (T_{fitness} + \alpha_{sel} * N_{iter} * T_{grad}))$
MEMBP$^{(pg)}$	$O(N_{gen} * N_{ind} * (T_{fitness} + \alpha_{sel} * N_{iter} * N_{par} * T_{fun}))$
MEMLM	$O(N_{gen} * N_{ind} * (T_{fitness} + \alpha_{sel} * N_{iter} * (N_{res} * T_{res} + T_{jac})))$
MEMLM$^{(pJ)}$	$O(N_{gen} * N_{ind} * (T_{fitness} + \alpha_{sel} * N_{iter} * N_{res} * N_{par} * T_{res}))$
BMABP	$O(N_{gen} * N_{ind} * (N_{par} * N_{clone} * T_{fitness} + N_{iter} * T_{grad}))$
BMABP$^{(pg)}$	$O(N_{gen} * N_{ind} * N_{par} * (N_{clone} * T_{fitness} + N_{iter} * T_{fun}))$
BMALM	$O(N_{gen} * N_{ind} * (N_{par} * N_{clone} * T_{fitness} + N_{iter} * (N_{res} * T_{res} + T_{jac})))$
BMALM$^{(pJ)}$	$O(N_{gen} * N_{ind} * N_{par} * (N_{clone} * T_{fitness} + N_{iter} * N_{res} * T_{res}))$

Table 2
Time complexities of inference techniques

Technique	Time complexity
Mamdani	$O(N_{rule} * N_{dim})$
Stabilized KH	$O(N_{rule} * N_{dim})$

Table 3
Time complexities of function evaluations

Function	Time complexity
Objective	$O(N_{sample} * N_{rule} * N_{dim})$
Residual	$O(N_{rule} * N_{dim})$
Fitness	$O(N_{sample} * N_{rule} * N_{dim})$
Gradient	$O(N_{sample} * N_{rule} * N_{dim})$
Jacobian	$O(N_{sample} * N_{rule} * N_{dim})$

Table 4
Time complexities of learning methods

Method	Time complexity
BP	$O(N_{iter} * N_{sample} * N_{rule} * N_{dim})$
BP(pg)	$O(N_{iter} * N_{sample} * N_{rule}^2 * N_{dim}^2)$
LM	$O(N_{iter} * N_{sample} * N_{rule} * N_{dim})$
LM(pJ)	$O(N_{iter} * N_{sample} * N_{rule}^2 * N_{dim}^2)$
GA	$O(N_{gen} * N_{ind} * N_{sample} * N_{rule} * N_{dim})$
BEA	$O(N_{gen} * N_{ind} * N_{clone} * N_{sample} * N_{rule}^2 * N_{dim}^2)$
MEMBP	$O(N_{gen} * N_{ind} * (1 + \alpha_{sel} * N_{iter}) * N_{sample} * N_{rule} * N_{dim})$
MEMBP(pg)	$O(N_{gen} * N_{ind} * (1 + \alpha_{sel} * N_{iter} * N_{rule} * N_{dim}) * N_{sample} * N_{rule} * N_{dim})$
MEMLM	$O(N_{gen} * N_{ind} * (1 + \alpha_{sel} * N_{iter}) * N_{sample} * N_{rule} * N_{dim})$
MEMLM(pJ)	$O(N_{gen} * N_{ind} * (1 + \alpha_{sel} * N_{iter} * N_{rule} * N_{dim}) * N_{sample} * N_{rule} * N_{dim})$
BMABP	$O(N_{gen} * N_{ind} * (N_{rule} * N_{dim} * N_{clone} + N_{iter}) * N_{sample} * N_{rule} * N_{dim})$
BMABP(pg)	$O(N_{gen} * N_{ind} * (N_{clone} + N_{iter}) * N_{sample} * N_{rule}^2 * N_{dim}^2)$
BMALM	$O(N_{gen} * N_{ind} * (N_{rule} * N_{dim} * N_{clone} + N_{iter}) * N_{sample} * N_{rule} * N_{dim})$
BMALM(pJ)	$O(N_{gen} * N_{ind} * (N_{clone} + N_{iter}) * N_{sample} * N_{rule}^2 * N_{dim}^2)$

4.2 Space Complexity

In this section space complexities will be presented. Apparently, the order of magnitude of the complexities are determined only by the values in the parameters, because the other necessary memory allocations are constant. Therefore they are not needed to be expressed in the complexities.

Space complexities of optimization algorithms, inference techniques and learning methods are presented in Tables 5, 6 and 7, respectively. Since both Mamdani-inference and stabilized KH-interpolation have the same complexity (see Table 6), we do not distinguish the learning based on different inferences in Table 7.

Table 5
Space complexities of optimization algorithms

Algorithm	Space complexity
BP	$O(N_{par})$
LM	$O(N_{res} * N_{par})$
GA	$O(N_{ind} * N_{par})$
BEA	$O(N_{ind} * N_{par})$
MEMBP	$O(N_{ind} * N_{par})$
MEMLM	$O((N_{ind} + N_{res}) * N_{par})$
BMABP	$O(N_{ind} * N_{par})$
BMALM	$O((N_{ind} + N_{res}) * N_{par})$

Table 6
Space complexities of inference techniques

Technique	Space complexity
Mamdani	$O(N_{rule} * N_{dim})$
Stabilized KH	$O(N_{rule} * N_{dim})$

Table 7
Space complexities of learning methods

Method	Space complexity
BP	$O(N_{rule} * N_{dim})$
LM	$O(N_{sample} * N_{rule} * N_{dim})$
GA	$O(N_{ind} * N_{rule} * N_{dim})$
BEA	$O(N_{ind} * N_{rule} * N_{dim})$
MEMBP	$O(N_{ind} * N_{rule} * N_{dim})$
MEMLM	$O((N_{ind} + N_{sample}) * N_{rule} * N_{dim})$
BMABP	$O(N_{ind} * N_{rule} * N_{dim})$
BMALM	$O((N_{ind} + N_{sample}) * N_{rule} * N_{dim})$

5 Experimental Determination of Complexity and Errors

The results of the simulations will be presented in this section. The obtained quantities tell about different errors (MSE, MSRE, MREP) and computational times of the systems. Thus, these results allow the experimental determination of the errors and the experimental validation of time complexities.

The definitions of the errors:

- Mean Square of Error (MSE): $\frac{1}{m}\sum_{i=1}^{m}(d_i - y_i)^2$
- Mean Square of Relative Error (MSRE): $\frac{1}{m}\sum_{i=1}^{m}\frac{(d_i - y_i)^2}{y_i^2}$
- Mean Relative Error Percentage (MREP): $\frac{100}{m}\sum_{i=1}^{m}\left|\frac{d_i - y_i}{y_i}\right|$

The results of the runs and their short explanations follow in the next subsections. Every simulation will not appear though, because their huge number does not allow it, rather only some important cases will be presented.

In the simulations the parameters had the following values. The number of rules in the rule base was 4, the number of individuals in a generation as well as the number of clones was 10. 3 gene transfers and 6 gradient steps were carried out in each generation. The gradient vector and the Jacobian matrix computing functions were not given, hence pseudo-gradients and pseudo-Jacobians were computed where backpropagation or Levenberg-Marquardt gradient steps were applied.

In case of all parameter sets 10 runs were carried out. We did not take the mean of the obtained values, because so few runs were done that the outlyers affected the actual mean value very strongly. Instead of calculating the mean, we sorted the elements according to their values and took the 'median' (the middle one) of the elements. These medians were presented in the figures, to get a better overview.

Every figure has four graphs. The upper left one shows the MSE, the upper right one the MSRE, the lower left one the MREP error values and the lower right one the computational times. The broken lines denote the results given by Mamdani-inference, the continuous one the results by stabilized KH-interpolation.

5.1 Validation of Theoretical Complexities

The following example presents the maximum generation number dependence of BMALM learning. Although this is only an example, the other theoretical dependences can also be validated in practise by similar simulation series.

A one-dimensional problem was given and the stopping condition was active in the Levenberg-Marquardt method. During the runs the maximum numbers for generations were 10, 15, 20 and 25, respectively. So we could observe the dependence of the errors and the computational time on the maximum generation number (Figure 1).

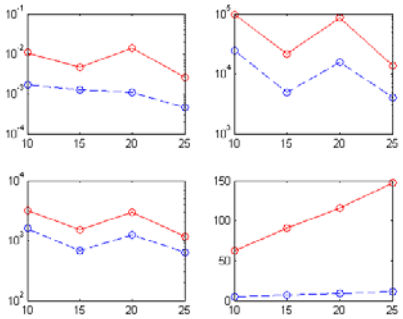

Figure 1
Maximum generation number dependence

Due to the few simulations it is not unambiguous that the errors are decreasing, but the decreasing tendecy can be noticed. However, the theoretical time complexity, namely, the linear dependency is unambiguously validated by the experiments.

5.2 Comparison of Learning Methods Based on Different Optimizations

The genetic algorithms used elitist strategy and the stopping condition was inactive in the Levenberg-Marquardt method.

During the runs the optimization methods were GA (1), MEMBP (2), MEMLM (3), BEA (4), BMABP (5) and BMALM (6), respectively. (The numbers in brackets are the numbers in the figures denoting the algorithms.) BP and LM were not used, since the use of only gradient techniques would be worthless, because they could improve the system only until they find the nearest local minimum on the error surface. By doing this series of experiments we could observe the dependence of the errors and the computational time on the applied optimizations.

At first, the system learned a one-dimensional function. In case of fixed generation numbers (20 generations) we can notice from the results (Figure 2 (left)) that the optimized MSE error values were lower when Mamdani-inference and when bacterial algorithms were used. The best results could be obtained by BMALM technique. On the other hand, it can also be observed that the computational demands of the methods are growing in the following order: GA, MEMBP, MEMLM, BEA, BMABP and BMALM.

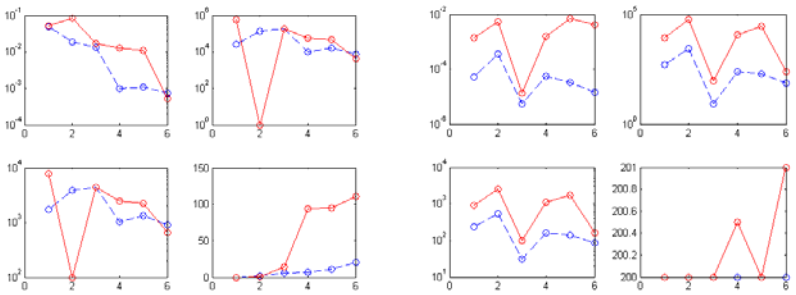

Figure 2
Comparison of learning methods based on different optimizations for a one-dimensional problem in case of fixed generation numbers (left) and in case of a 200 seconds time limit (right)

When a time limit (200 seconds) was given, learning methods applying Mamdani-inference gave better results (see Figure 2 (right)), because they had lower computational demands. (This time limit means that the learning stops at the end of the generation in which the limit has been exceeded.)

When the system learned a two-dimensional function with the previous fixed generation number (Figure 3 (left)), applying the interpolative inference gave lower errors. We obtained similar experiences about the optimization techniques like we did in the previous case.

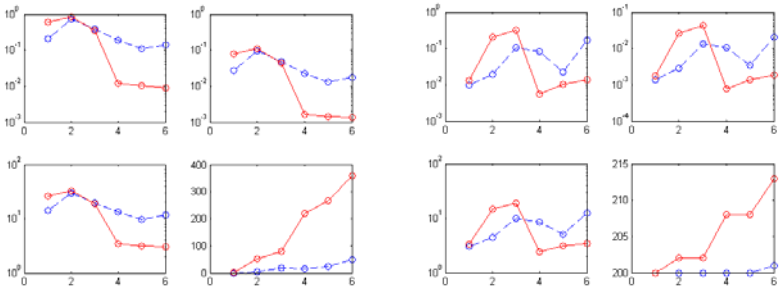

Figure 3
Comparison of learning based on different optimizations for a two-dimensional problem in case of fixed generation numbers (left) and in case of a 200 seconds time limit (right)

In case of a 200 seconds time limit, learning methods using stabilized KH-interpolation with bacterial algorithms gave the best results as well (see Figure 3 (right)).

As a summary we can say, according to low computation time, learning methods using Mamdani-inference had better performances, however their time complexity is exactly the same as the time complexity of the ones using stabilized KH-interpolation (see section 4.1) in case of the same system parameters. Although, for low dimensional problems the former ones gave lower errors than the latter ones, in higher dimensions (even in two dimensions during the simulations) the latter ones became better. The reason is that the number of rules was the same for each inference method and this became an advantage for the interpolation.

The situation is similar when we consider simple evolutionary algorithms against memetic (or for larger difference, bacterial memetic) algorithms. Learning methods using the former ones were faster than the latter ones, but the latter ones had lower errors in higher dimensions.

Conclusions

In our work we have compared various fuzzy rule-based learning and inference systems.

The starting point of the investigation was a modular system that we have implemented in C language. It contains several alternative versions of the two key elements of rule based learning, namely, the optimization algorithm and the inference method, which can be found in the literature. We obtained very different properties when combining these alternatives (changing the modules and connecting them) in all possible ways.

The investigations determined the values of the quality measures (complexity and accuracy) of the obtained alternatives both analitically and experimentally where it was possible.

In case of the same parameters in lower dimensions learning methods using Mamdani-inference had better performance in errors as well as in computational demands, however in higher dimensions learning methods applying stabilized KH-interpolation took the lead. In higher dimensions bacterial optimization techniques had advantages against the genetic ones.

To reinforce these tendencies, as a continuation of this work, carrying out more simulations is necessary.

Similarly like it has been proposed in this paper, comparison of systems that use other optimization algorithms and inference techniques can be performed.

Further research may aim to implement, integrate and compare other optimization algorithms and fuzzy inference methods that can be found in the literature.

Acknowledgements

This paper was supported by the Széchenyi University Main Research Direction Grant 2008, and a National Scientific Research Fund Grant OTKA T048832.

References

[1] L. A. Zadeh: Outline of a New Approach to the Analysis of Complex Systems and Decision Processes, IEEE Tr. Systems, Man and Cybernetics SMC 3, 1973, pp. 28-44

[2] E. H. Mamdani: Application of fuzzy algorithms for control of simple dynamic plant, IEEE Proc., Vol. 121, No. 12, 1974, pp. 1585-1588

[3] L. T. Kóczy and K. Hirota: Approximate Reasoning by Linear Rule Interpolation and General Approximation, Internat. J. Approx. Reason., 9, 1993, pp. 197-225

[4] D. Tikk, I. Joó, L. T. Kóczy, P. Várlaki, B. Moser and T. D. Gedeon: Stability of Interpolative Fuzzy KH-Controllers, Fuzzy Sets and Systems, 125, 2002, pp. 105-119

[5] K. Michels, F. Klawonn, R. Kruse, A. Nürnberger: Fuzzy Control Fundamentals, Stability and Design of Fuzzy Controllers Series: Studies in Fuzziness and Soft Computing, Springer, ISBN 978-3-540-31765-4, Vol. 200, 2006, 411 pp.

[6] D. Driankov, H. Hellendoorn, M. Reinfrank: An Introduction to Fuzzy Control, Springer, ISBN 978-3-540-60691-8, 316 pp.

[7] E. Alpaydin: Introduction to Machine Learning, The MIT Press, ISBN 978-0-262-01211-9, 2004, 445 pp.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams: Learning Representations by Backpropagating Errors, Nature, 1986, pp. 533-536

[9] K. Levenberg: A Method for the Solution of Certain Non-Linear Problems in Least Squares. Quart. Appl. Math., 1944, pp. 164-168

[10] D. Marquardt: An Algorithm for Least-Squares Estimation of Nonlinear Parameters. J. Soc. Indust. Appl. Math., 1963, pp. 431-441

[11] J. H. Holland: Adaption in Natural and Artificial Systems, The MIT Press, Cambridge, Massachusetts, 1992

[12] N. E. Nawa and T. Furuhashi: Fuzzy System Parameters Discovery by Bacterial Evolutionary Algorithm, IEEE Transactions on Fuzzy Systems, 1999, pp. 608-616

[13] P. Moscato: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989

[14] J. Botzheim, C. Cabrita, L. T. Kóczy, and A. E. Ruano: Fuzzy Rule Extraction by Bacterial Memetic Algorithms, Proceedings of the 11th World Congress of International Fuzzy Systems Association, IFSA 2005, Beijing, China, 2005, pp. 1563-1568