FuNNy

A self learning fuzzy system

Madsen, Per Printz

Published in:
Proceedings of the 2005 International Conference on Machine Learning; Models, Technologies and Applications

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

FuNNy: A self learning fuzzy system

Per Printz Madsen
Ph.D, M.Sc.E.E.
Dept. of Control Engineering, Inst. of Electronic Systems
Aalborg University, Fredrik Bajers vej 7 DK-9220 Aalborg , Denmark
E-mail: ppm@control.aau.dk

Abstract— **The purpose of this paper is to describe a tool that is easy to use for implementing self learning fuzzy systems. This tool which is called FuNNy generates fuzzy systems. The tool consists of a compiler and a C learning library. The compiler translates a fuzzy system (written in a dedicated language, called FuNNy language) to C. The C learning library contains the learning algorithm. The generated C code is simple standard C and therefore it can be applied to all computers with a C-compiler. The learning algorithm is either a gradient descend method based on a numerical calculation of the gradient or a random search method. The input fuzzyfication can be described by four different kinds of membership functions. The output fuzzyfication is based on singletons. The rule base can be written in a natural language. The result of the learning is a new version of the fuzzy system described in the FuNNy language.**

Index Terms— **Fuzzy system, Fuzzy control, Natural language, Self learning, C-target.**

I. Introduction

FUZZY SETS and Fuzzy Control have been used in many applications since Zadeh, in 1965 [7], described the Fuzzy sets and Mamdani et all, in 1974 [1], described and used Fuzzy sets for control purpose. Fuzzy sets for control is often called Fuzzy Control. Another way of doing non-linear MIMO (Multi-Input, Multi-Output) control is the use of ANN (Artificial Neural Networks). The main difference between ANN and Fuzzy systems is that ANN are based on data driven learning and that Fuzzy systems are based on previous knowledge described in fuzzy logic terms. Many scientists around the world have combined Fuzzy systems with the data driven learning methods known from ANN. This combination is often called NeuroFuzzy systems. One of the best known of these combinations is the NeuroFuzzy system called ANFIS [3]. These combined systems, ex ANFIS, have been used in many different cases. For instance [5] [2] or [6]. One of the main advantages of combining fuzzy systems and learning methods is that it becomes possible to use all the previous knowledge to build a good start guess, or in other words a Fuzzy system that describes what the programmer knows about the system, and after that uses measured input/output relations to optimize the system. By combining previous knowledges with measured input/output data it is often possible to: make a system that inter- and extrapolate better than a pure ANN system. Another advantage of these combined systems is that they often are more precise than pure Fuzzy systems because the learning algorithm often finds a better solution.

The propose of this paper is to describe an easy to use tool and method to implement a NeuroFuzzy systems called the FuNNy system. The FuNNy system can be pre-programmed in a natural fuzzy language and optimized by learning. FuNNy is a public domain tool, that consists of a compiler and a C module of learning algorithms. The compiler translates a fuzzy system, written in the language FuNNy to a C module. The learning algorithms can easily be used to adjust the fuzzy system, given by the C module and thereby find the optimal value for the parameters in the fuzzy system. The result of the learning is a new fuzzy system expressed in the FuNNy language.

II. Method

THE FuNNy tool can be divided into three different components. The language FuNNy, the compiler and the learning algorithms.

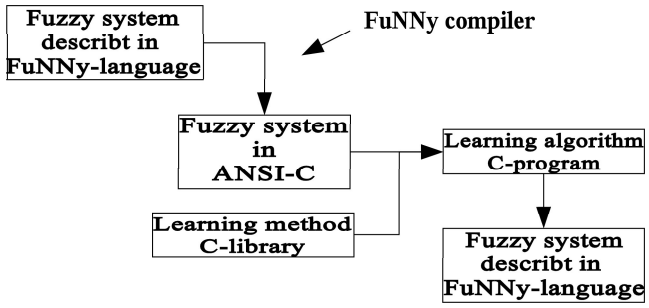

Fig. 1. The structure of the FuNNy system.

Figure: 1 shows the software modules and the steeps from a Fuzzy system, given by a description in the FuNNy language to a new Fuzzy system, adjusted by the learning algorithm. The first Fuzzy system could be the pre-description, written by the programmer. The last Fuzzy system is the same Fuzzy system as the first one, but now, adjusted be the learning algorithm. The result of the learning is expressed in the FuNNy language. This means that it is easy to inspect the learning adjustment by comparing the two FuNNy programs.

The steeps from the pre-knowledges to a final Fuzzy system.

1) Write a fuzzy system in the FuNNy language, based on the pre-knowledges.
2) Compile the FuNNy program with the FuNNy compiler to a C-program.
3) Use this C-program and a learning library to build a learning algorithm.

4) Use this learning algorithm to generate a new FuNNy program, that describes the adjusted fuzzy system.
5) Compile this adjusted fuzzy system with the FuNNy compiler.
6) Use the Fuzzy system, given by the new C-program. If the Fuzzy system does not perform as expected, then go to step 4.

A. The FuNNy language

The FuNNy language consists of two parts. A definition part and a rule part. The definition part specifies the fuzzyfication of input and output.

```
temp: input;
flow: input;

cold   temp: sigmoid(25.0,a, 33.0,a, 1.0,c);
middle temp: bell(33.0,a, 2.0,a, 1.0,c);
hot    temp: sigmoid(40.0,a, 33.0,a, 1.0,c);

to_little flow: sigmoid(0.0,a, 0.5,a, 1.0,c);
to_much   flow: sigmoid(1.0,a, 0.5,a, 1.0,c);

down       cold_water: output(-0.02,a);
no_change  cold_water: output( 0,a);
up         cold_water: output( 0.02,a);

down       hot_water: output(-0.02,a);
no_change  hot_water: output( 0,a);
up         hot_water: output( 0.02,a);
```

The definition part starts with a declaration of the input linguistic variables. In this example: `temp` and `flow`. Then the definition of the linguistic values follows: `cold`, `middle` and `hot` for the temperature and `to_little` and `to_much` for the flow. The last part is the definition of the output linguistic variables and values. There is two output linguistic variables: `cold_water` and `hot_water` and three values for each of them: `down`, `no_change` and `up`. Each linguistic value is given by a membership function. The input can be defined by four different types of membership functions and the output can be defined by one type of membership function.

The input membership functions are:

The Trapeze function: `trapez(a,A,b,A,c,A,d,A,e,A)` where `A` is the adaption flag. This flag can be set to `a` or `c`. If the flag is set to `a`, the preceding parameter will be adjusted by the learning algorithm and if it is set to `c`, the parameter will not be affected by the learning algorithm. For the meaning of the parameters `a,b,c,d,e` see figure: 2.

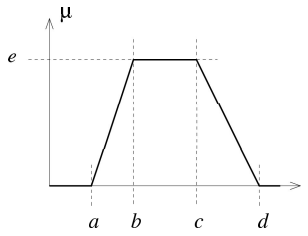

Fig. 2. Trapeze-membership function

The Triangle function: `triangle(a,A,b,A,c,A,d,A)`. See figure: 3.

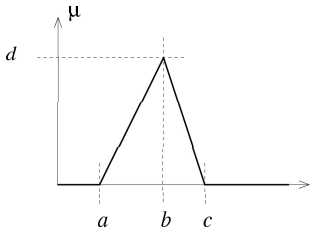

Fig. 3. Triangel-membership function

The Bell function: `bell(a,A,b,A,c,A)`. See figure: 4.

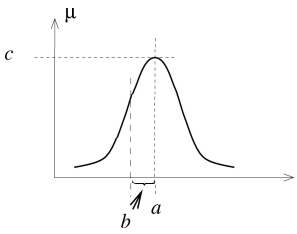

Fig. 4. Bell-membership function

The Sigmoid function: `sigmoid(a,A,b,A,c,A)`. See figure: 5.

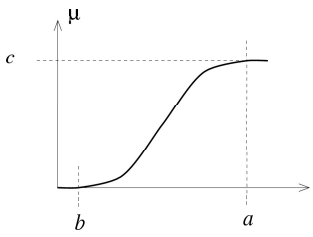

Fig. 5. Sigmoid-membership function

The output can only be specified as a singleton membership function. like: `down cold_water: output(-0.02,a);` where `-0.02` is the placement of the singleton and the `a` specifies that the placement can be adjusted by the learning algorithm.

The adaption flag gives the opportunity to make a much more understandable result of the learning. Ex you can ask the Fuzzy system "if `cold` is given by ... what is then the optimal choice for `hot`". Another advantage of the flexible selection of adjusted parameters is, that it is often possible to investigate the information in the learning data with respect to the different parts of the Fuzzy model and by that means extract some informations about, where in the working area or, how to collect more learning data.

After the definition part comes the rule part. Each rule consists of an if part (the antecedent) and a then part (the consequent). The antecedent consists of one or more logical expressions. It is possible to use the logical operator `and` `or` to combine logical expressions. The brackets `( )` can be used to group the logical expressions. The `and` operator is

based on the min operation and the `or` operator is based on the max operation. The linguistic hedge `not`, `very`, `extremely` and `more_or_less` can be used to manipulate the linguistic values. If μ is the membership function, defining the linguistic value then the linguistic hedge is defined by: `not` $\mu_{\text{not}} = 1 - \mu$, `very` $\mu_{\text{very}} = \mu^2$, `extremely` $\mu_{\text{extremely}} = \mu^4$ and `more_or_less` $\mu_{\text{more or less}} = \sqrt{\mu}$

The two following rules are a part of a shower controller:

```
if   temp is hot and flow is not to_little
then hot_water is down

if   temp is cold and flow is not to_much
then hot_water is up
```

The implication is done by the min operation. The rule `if x is A then y is B` where A is given by μ_A and B is given by μ_B then the implication $\mu_{\text{rule}}(y)$ is given by $\mu_{\text{rule}}(y) = \min(\mu_A(x), \mu_B(y))$. x is a known value, because it is the input. $\mu_B(y)$ is a singleton so $\mu_{\text{rule}}(y)$ is also a singleton.

The rule inference is done by height defuzzyfication.

$$y = \frac{\sum_{i=1}^{N} b_i \cdot a_i}{\sum_{i=1}^{N} b_i}$$

Where: N: is the number of rules, b_i: is the height of the output singleton given by $\mu_{\text{rule}}(y)$, and a_i: is the parameter from the output membership function used in the rules and defined in the definition part.

B. Learning

The parameters of the FuNNy system can be adapted by a learning algorithm. This, of course, requires that the learning algorithm is stable and able to adjust the parameters with respect to a performance function. The performance function, $P(w)$, is the squared prediction error between the desired output and predicted output. The Fuzzy system described by the FuNNy language consists of some strong non-linearities and some dis-continuities in the derivative of the output. This means that it is very complex, if not impossible, to use at traditional gradient-descent method based on the mathematical derivation of the output. If it is possible to find the gradient it will have some dis-continuities and therefore, in some cases, it will make the learning algorithm unstable. It is therefore necessary to find another way. One method is to use a numerical calculation of the gradient and another is to use learning algorithm that is not based on the gradient ex Genetic algorithms or other kind of random search methods.

1) Gradient-descent based on numerical calculation of the gradient. (NG-learning): This learning algorithm for the FuNNy system is based on two point numerical derivation of the gradient given by equ: 1. This numerical derivation method is fast to calculate and gives a good estimation near the minimum.

$$\frac{\partial P_i(w_j)}{\partial w_j} \approx \frac{\Delta P_i(w_j)}{\Delta w_j} = \frac{P_i(w_j + \Delta w_j) - P_i(w_j - \Delta w_j)}{2\Delta w_j} \tag{1}$$

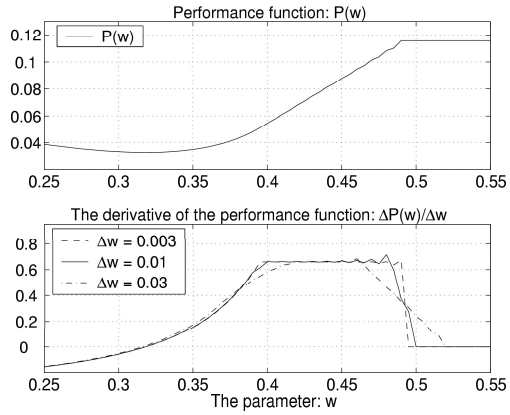

Fig. 6. The numerical calculation of the gradient for three different values of Δw.

where $P_i(w)$ is the squared prediction error for the i'th sample in the learning data. w_j is the j'th parameter.

Figure: 6 shows the estimation of the gradient for different values of Δw. There is a dis-continuity of the real gradient $\frac{\partial P(w)}{\partial w_j}$ at the point $w = 0.49$, but, when using a the numerical method, the gradient is continuous. This means, that it is easier to make the learning algorithm stable. There is a large difference between the three curves, but it does not matter, because it is not close to the minimum of the performance function $P(w)$. Close to the minimum of $P(w)$ the three curves it is almost equal. Therefore the minimum is found quite good in all three cases. If Δw is not chosen too large, the learning algorithm will find the correct minimum. Another important observation from figure: 6 is the area: $w \approx 0.49$ and upwards. In this area the gradient is zero but it is not the minimum. This means, if the parameter is initialized to a value in this area the learning algorithm will not be able to adjust the parameter. This illustrates the importance of using the previous knowledge to build a good start guess for the system.

The adjustment of the j'th parameter, given the i'th input/desired-output data point, is done by equ: 2.

$$w_j = w_j - \alpha \frac{\Delta P_i(w_j)}{\Delta w_j} \tag{2}$$

Where α is chosen to a small value ex 0.01.

α can be adjusted during the learning, like this:

- Initiation: $\alpha = 0.01$
- Increase α with 1 % if $\sum_{\forall \text{data}} P$ decreases over two epochs.
- Decrease α with 0.5 % if $\sum_{\forall \text{data}} P$ increase.

2) The random search learning method: An other way of adjusting the parameters is the random search method. This method is not based on a gradient, but only on a random adjustment of the parameters and on the performance function.

1) Build a fuzzy model based om pre-knowledge.
2) Calculate the output for all inputs: $\bar{y}_i = \bar{F}(\bar{\Theta}, \bar{x}_i)$. Where $\bar{F}$ is the vector function from input to output, $\bar{x}_i$ is the i'th input and $\bar{\Theta}$ is the parameter vector.

3) Calculate the performance function: $P_p = \sum_{\forall \bar{x}_i}(\bar{d}_i - \bar{y}_i)^2$. Where $\bar{d}_i$ is the i'th desired output.
4) Make a copy of the parameters $\bar{\Theta}$: $\bar{\Theta}_p = \bar{\Theta}$.
5) Adjust the parameters: $\bar{\Theta} = \bar{\Theta} + \bar{r}$. where $\bar{r}$ is a vector with normal distributed random elements with a zero mean and a small variance.
6) Calculate the output for all inputs: $\bar{y}_i = \bar{F}(\bar{\Theta}, \bar{x}_i)$.
7) Calculate the performance function: $P = \sum_{\forall \bar{x}_i}(\bar{d}_i - \bar{y}_i)^2$
8) if $P < P_p$ then $P_p = P$ and $\bar{\Theta}_p = \bar{\Theta}$. else $\bar{\Theta} = \bar{\Theta}_p$.
9) Go to 5 as long as you wish.

III. Examples

As an illustration of the capabilities of the FuNNy system figure: 7 shows the result of an adaption to the function: $F_y = 0.6\sin(\pi x) + 0.3\sin(3\pi x) + 0.1\sin(5\pi x)$. This function is the exact same as in [4]. In this case the adaption is done by the ANFIS system: [3]. The FuNNy model contains 24 adaptive parameters. In [4] is used an ANFIS model with 25 parameters.

The FuNNy system, before learning:

```
x : input;
  mf1   x: bell(-1.05,a, 0.15,a, 1,c);
  mf2   x: bell(-0.75,a, 0.15,a, 1,c);
  mf3   x: bell(-0.45,a, 0.15,a, 1,c);
  mf4   x: bell(-0.15,a, 0.15,a, 1,c);
  mf5   x: bell( 0.15,a, 0.15,a, 1,c);
  mf6   x: bell( 0.45,a, 0.15,a, 1,c);
  mf7   x: bell( 0.75,a, 0.15,a, 1,c);
  mf8   x: bell( 1.05,a, 0.15,a, 1,c);
  o1    y: output( 0.0,a);
  o2    y: output(-0.5,a);
  o3    y: output(-0.5,a);
  o4    y: output(-0.5,a);
  o5    y: output( 0.5,a);
  o6    y: output( 0.5,a);
  o7    y: output( 0.5,a);
  o8    y: output( 0.0,a);
rules
  if x is mf1 then y is o1
  if x is mf2 then y is o2
  if x is mf3 then y is o3
  if x is mf4 then y is o4
  if x is mf5 then y is o5
  if x is mf6 then y is o6
  if x is mf7 then y is o7
  if x is mf8 then y is o8
```

As shown in figure: 7 the result of the learning gives a good fitting to the learning data. The learning algorithms converges smoothly toward a minimum. Figure: 8 illustrates the difference of the input membership functions before and after the learning.

The next example illustrates what happens when no pre-knowledges is used. In this case the input fuzzyfication is set like in the previous example, but the output singletons is all set to one.

As shown in figure: 9 the learning result is not nearly as good as in figure: 7, and the membership functions are not distributed optimally. Figure 10. The learning algorithm has found a local minimum of the performance function.

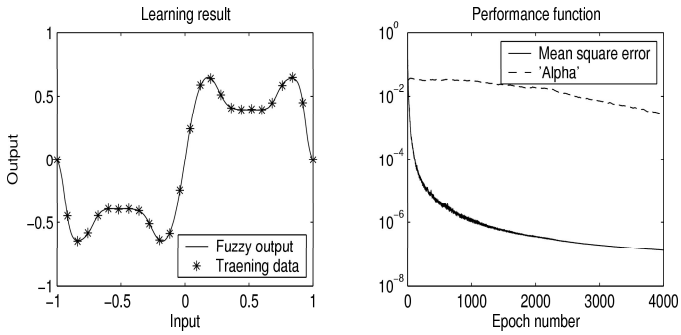

Fig. 7. The result of the learning.

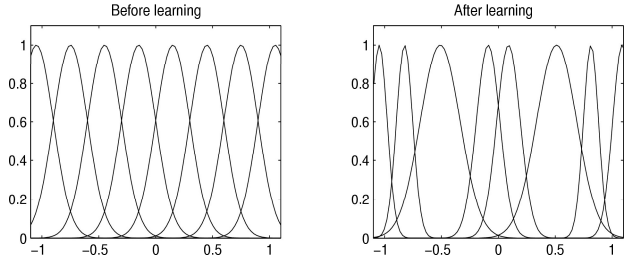

Fig. 8. The input membership functions, before and after the learning.

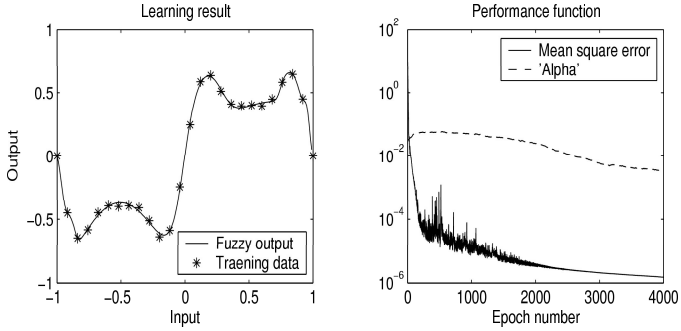

Fig. 9. The result of the learning.

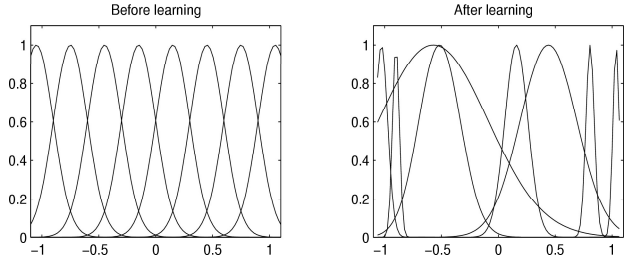

Fig. 10. The input membership functions, before and after the learning.

There is two ways the FuNNy system can deal with local minimums. One way is to use pre-knowledges and thereby select a good start point for the learning. If lucky the system is near the global minimum which means that the gradient based learning will find a good solution. An other way is to use the Random search method. The Random search method is much slower than the gradient based learning method.

IV. Conclusion

THE fuNNy system is a self learning Fuzzy system. The programming and the learning of the FuNNy system are done by the FuNNy compiler and the C-library. It is possible to select witch of the parameters that are going to be adjusted by the learning algorithm. There is two learning algorithms in the library: A gradient decent method and a Random search method. The gradient decent method requires that the pre-programmed FuNNy system is close to a good minimum. When using the gradient decent method it is often a good idea to mix the learning data in order to present the input/output data, to the FuNNy system, in a random order. This randomizing of the learning data speeds up the learning and minimizes the risk of ending up in a local minimum. The compiler translates the Fuzzy system written in the FuNNy language to a C program. This C program contains only standard C code, and therefore it can be applied to all ordinarily micro computers. The FuNNy system can be found on the webpage: www.control.aau.dk/ ppm/FuNNy.

References

[1] S. Assilian and E.H. Mamdani. An experiment in linguistic synthesis with a fuzzy logic controller. In *International Joint Man Machine-Studies*, volume 7(1), pages 1–13, 1974.

[2] Jettrey T. Drake and Nadipuram R. Prasad. ANFIS for parameter estimation in coherent communications phase synchronization. *IEEE Signal processing Society Workshop*, pages 433–442, 2001.

[3] J.-S. R. Jang. ANFIS: Adaptive-Network-based Fuzzy Interence Systems. *IEEE Transactions on Systems, Man, and Cybernetice*, pages 665–685, 1993.

[4] J.-S. Roger Jang and Ned Gulley. *Fuzzy Logic Toolbox*. The MathWorks, Inc., Januar 1995.

[5] S. Khanmohammadi L. Hassanzadeh and J. Jiang Gh. Alizadeh. Implementation of a Functional Link Net_ANFIS Controller for a Robot Manipulator. *Third International Workshop on Robot Motion and Control*, pages 399–404, 2002.

[6] Chia-Chi Chen Wen-Liang and Shing-Chia Chen. ANFIS based PRML system for read-out RF signal. *IFSA World congress and 20th NAFIPS International conference*, pages 912–917, 2001.

[7] L. A Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.