Learning Fuzzy Rule-Based Neural Networks for Function Approximation[1]

C. M. Higgins chuck@electra.caltech.edu
R. M. Goodman rogo@caltech.edu

Department of Electrical Engineering, 116-81
California Institute of Technology
Pasadena, CA 91125

Abstract

In this paper, we present a method for the induction of fuzzy logic rules to predict a numerical function from samples of the function and its dependent variables. This method uses an information-theoretic approach based on our previous work with discrete-valued data [3]. The rules learned can then be used in a neural network to predict the function value based upon its dependent variables. An example is shown of learning a control system function.

1 Introduction

The problem of estimating a function from a set of samples can be solved in a multitude of ways, including mathematical methods using an explicit model for the system to be learned, and model-free systems such as neural networks and fuzzy systems. The flexibility and wide applicability of model-free systems has led to wide interest in their use, particularly in learning control system functions. The ability of fuzzy systems to express arbitrary functions in terms of linguistic rules makes such systems an attractive alternative to neural network "black boxes," in which the function learned can only be observed through the input/output relationship.

While there are a large number of methods in existence for the estimation of functions using neural networks, methods for learning fuzzy rules from data are less well-developed [4,5]. Typically for an industrial fuzzy control system application, the rules are generated by hand. In this paper, we present a method for learning fuzzy rules from example data based upon information theory. This method of learning rules from data has been well-documented on discrete data (see [1,2,3]), and can be simply modified to be used for the learning of fuzzy rules.

2 Learning Rules from Examples

Given membership functions for the input (dependent) variables and output (function value) set up by the designer of the system, there are two necessary components of the rule-learning scheme. First, we need a way to tell which of two rules is the best. Second, we need a way to search the space of all possible rules in order to find the best without simply checking every rule in the search space.

2.1 Ranking Rules

Smyth and Goodman [2] have developed an information-theoretic measure of rule value with respect to a given discrete example set. This measure is known as the J-measure; defining a rule as *if y then x* where y is a conjunction of input variable values and x is a value of the output variable, the J-measure can be expressed as follows:

$$J = p(y)\{p(x|y)\log_2(\frac{p(x|y)}{p(x)}) + p(\bar{x}|y)\log_2(\frac{p(\bar{x}|y)}{p(\bar{x})})\}$$

[1]This work was supported in part by the Army Research Office under contract number DAAL03-89-K-0126, and in part by DARPA under contract number AFOSR-90-0199.

The probabilities are estimated from relative frequencies counted in the given discrete example set. Thus, given a rule we can go through an example set, count up matches with the left-hand side, right-hand side, both left- and right-hand sides, and so forth, to calculate a measure of the goodness of a rule.

In order to rank *fuzzy* rules, one must simply realize that in the discrete scheme an example either logically matched or did not match a rule whose value was being calculated. It is a simple extension to say that for numerical examples a fuzzy rule will match to some *degree* determined by the minimum membership in the conjunction being matched with the examples. In this case, the probabilities are calculated from *fuzzy* counts of rule matches, but the same counts are made and the calculation proceeds exactly as before.

2.2 Searching for the Best Rules

In [3], we presented an efficient method for searching the space of all possible rules to find the most representative ones for discrete data sets. The basic idea is that each example is a very specific (and quite perfect) rule. However, this rule is applicable to only one example. We wish to generalize this very specific rule to cover as many examples as possible, while at the same time keeping it as correct as possible. The J-measure is just the tool for doing this. If we calculate the J-measures of all the rules generated by ***removing a single input variable*** from the very specific rule, then we will be able to tell if any of the slightly more general rules generated from this rule are better. If so, we take the best and continue in this manner until no more general rule with a higher J-measure exists. When we have performed this procedure on the very specific rule generated from each example (and removed duplicates), we will have a set of rules which represents the data set. We have shown in previous work that a classifier can be generated from these rules to predict the output given the inputs.

In order to use this method for discovering fuzzy rules, we need only say how to generate the very specific rule from each numerical example. This is done by taking the ***greatest membership*** in each input variable. For example, if input variable one is more 'low' than 'med' or 'high', the rule for this example will say

```
        if input1=low and input2= ..... then output = ......
```

The rules are then found exactly as above using the extended J-measure, and a fuzzy system can be used to predict the output based upon the inputs.

3 Building a Neural Network

Once the rules are learned, the architecture for a neural network to calculate the output variable from the inputs can be constructed as shown in figure 1.

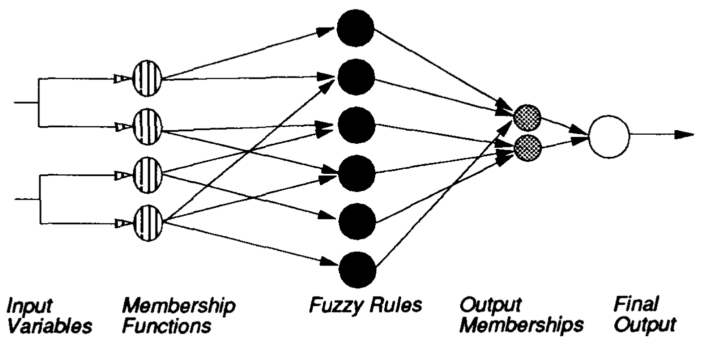


Figure 1: Network constructed from fuzzy rules

The input variables come into the first layer, which contains nodes which act as membership functions, responding only in a certain region of the input variable domain, with a degree between zero and one. The second layer is formed by the rules themselves. The nodes in this layer perform the fuzzy AND operation, computing a minimum of their inputs. The third layer contains a node for each output fuzzy set - these nodes gather the total weight from all the rules that the output is in that fuzzy set. Finally, the last layer contains a single node which performs a centroid defuzzification [4] of the outputs. This node normalizes the weights from the previous layer and weights them with the centers of the output membership functions to calculate the actual output.

4 Experimental Results

4.1 Function Approximation Experiments

In this section, we will show a simple function approximation experiment that demonstrates learning a two-dimensional function. The example function to be learned is a pyramid. This function depends on both the x and y coordinates. The membership functions are shown in figure 3. Samples of the function distributed evenly over the whole input space were presented to the learning system. The learned rules, also shown in figure 3, say that the output should be low except when both inputs are medium. This causes a central peak in the response which is quite similar to the desired response.

4.2 A Control System Experiment

While the above example is somewhat illustrative of how the system works, it is not of sufficient difficulty to be interesting to the practical reader. To address a more real-world problem, we shall describe the experiment of learning a system which keeps a radio-controlled car going at the same speed in a circle. That is, we shall describe the learning of a a "cruise control" system. A picture of the vehicle used can be found in figure 2.

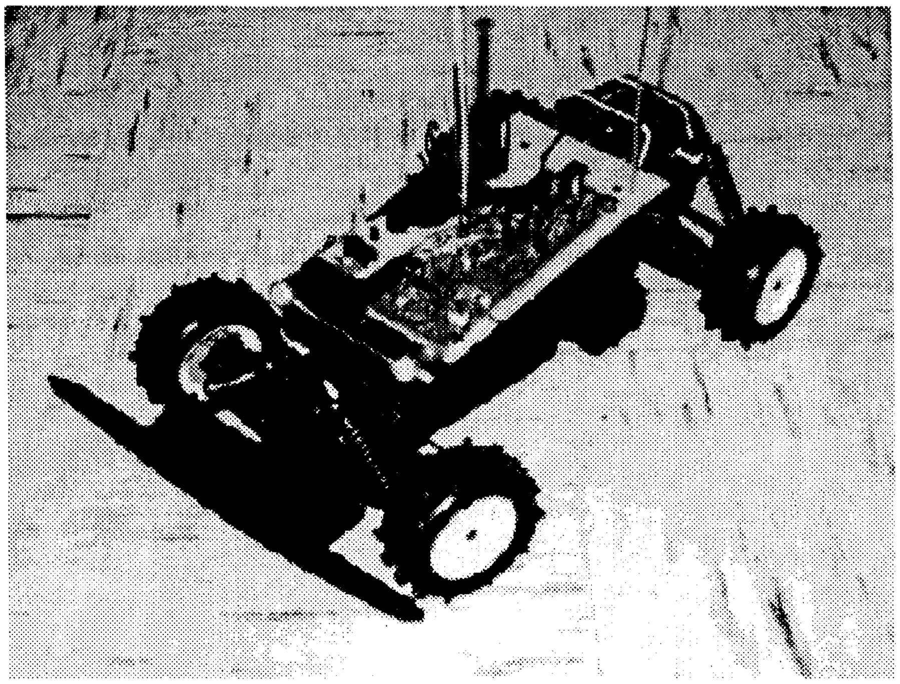


Figure 2: Experimental Vehicle

We began this experiment by designing a PD controller to solve this problem, and then setting up membership functions (shown in figure 4) for a fuzzy controller to be learned. The fuzzy controller was trained on the data shown in the graph in figure 4 at the bottom left. This data is a single run of the PD controller, starting at rest and maintaining the desired speed for a short time. The system was given samples of the velocity of the vehicle, the acceleration of the vehicle, and the resulting PD control output. The rules learned are shown in figure 4 at the middle. Also in that figure are shown the hand-crafted rules which would make the system perform exactly like the PD controller. We can see from the fuzzy system response (in figure 4 at the bottom right) that the system has learned the basic ideas necessary to control the vehicle. However, the learned controller has more oscillation in speed than desired. This is due to the fact that it was trained on data which did not show a large response to acceleration. We can see from the rules that its response to acceleration is less than in the desired rules. This problem could be overcome by more exhaustive training, covering more completely the space of the function to be approximated.

5 Summary

We have shown a method for learning fuzzy logic rules from numerical samples of a function in such a way that the function can be approximated by a fuzzy rule-based neural network using the rules. This has been demonstrated, not only for a 'toy' function approximation problem, but for a real control system.

In current research we are learning rule 'weights' to optimize the approximation of the learned function and learning the membership functions directly from the data, making it unnecessary for a system designer to intervene between the data and the learning system.

References

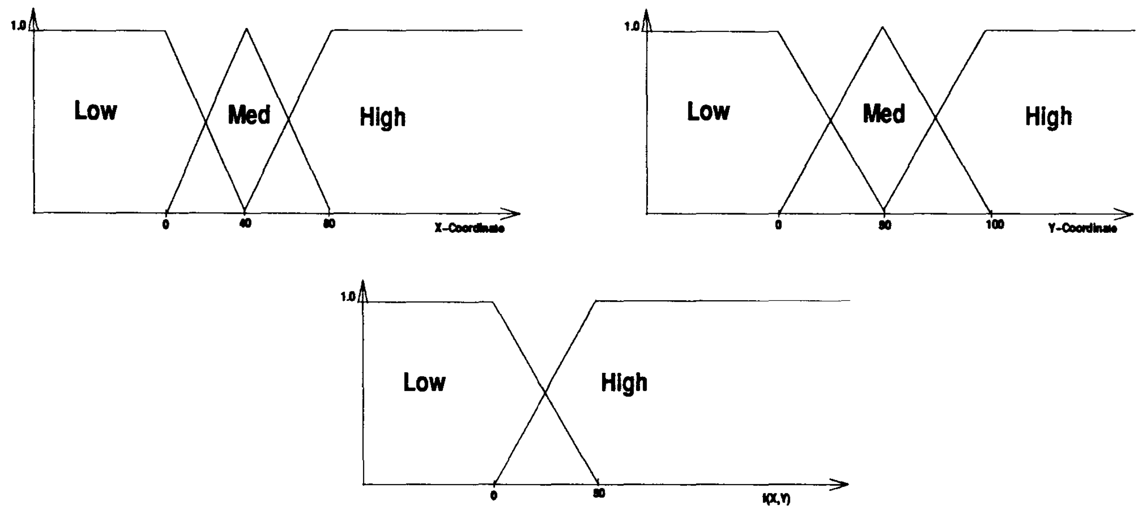
[1] R. Goodman, C. Higgins, J. Miller, P. Smyth, "Rule-Based Networks for Classification and Probability Estimation," to appear in *Neural Computation.*

[2] P. Smyth and R. Goodman, "An Information-Theoretic Approach to Rule Induction in Databases," to appear in *IEEE Transactions on Knowledge and Data Engineering.*

[3] C. M. Higgins and R. M. Goodman, "Incremental Learning using Rule-Based Neural Networks," *Proceedings of the International Joint Conference on Neural Networks*, vol. 1, 875-880, July 1991.

[4] B. Kosko, *Neural Networks and Fuzzy Systems,* Prentice Hall, Englewood Cliffs, NJ, 1992.

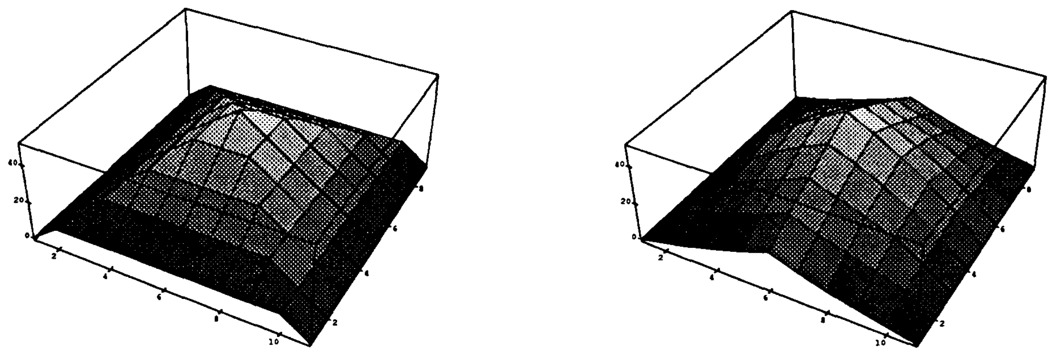
[5] C.-T. Lin and C. S. G. Lee, "Neural-Network-Based Fuzzy Logic Control and Decision System," *IEEE Transactions on Computers*, vol. 40, 1320-36, December 1991.



Membership Functions

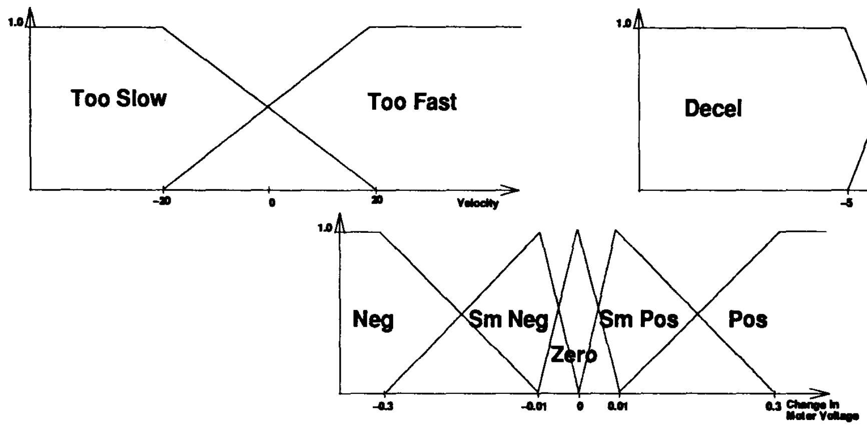
```
1 IF  y-coord=low                  THEN f(x,y) low
2 IF  x-coord=low                  THEN f(x,y) low
3 IF  x-coord=med AND y-coord=med THEN f(x,y) high
4 IF  y-coord=high                 THEN f(x,y) low
5 IF  x-coord=high                 THEN f(x,y) low
```

Learned rules



Left: Desired Response, Right: Learned Response

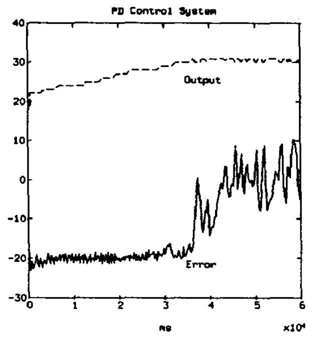
Figure 3: Pyramid Function Approximation

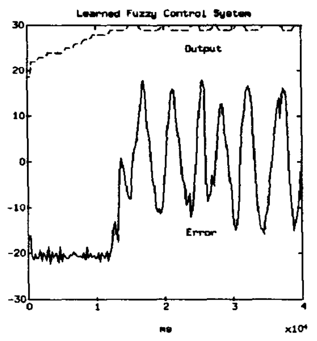


Membership Functions

```
DESIRED RULES:
          1.  IF velocity = too_slow THEN output = sm_pos
          2.  IF velocity = too_fast THEN output = sm_neg
          3.  IF acceleration     = accel    THEN output = neg
          4.  IF acceleration     = decel    THEN output = pos

LEARNED RULES:
     1. IF velocity=too_slow                          THEN output=sm_pos
     2. IF velocity=too_slow AND acceleration=decel THEN output=sm_pos

     3. IF velocity=too_fast AND acceleration=accel THEN output=sm_neg
     4. IF velocity=too_fast AND acceleration=decel THEN output=sm_neg

     5. IF acceleration=accel                         THEN output=sm_neg

     6. IF acceleration=decel                         THEN output=sm_pos
```

Desired and Learned Rules





Left: Hand-crafted PD Controller, Right: Learned Fuzzy Controller

Figure 4: Control Example – Radio-Controlled Car