

# A Hybrid Connectionist, Symbolic Learning System

Lawrence O. Hall and Steve G. Romaniuk  
Department of Computer Science and Engineering  
University of South Florida  
Tampa, Fl. 33620  
email:hall@sol.usf.edu romaniuk@sol.usf.edu

## Abstract

This paper describes the learning part of a system which has been developed to provide expert systems capability augmented with learning. The learning scheme is a hybrid connectionist, symbolic one. A network representation is used. Learning may be done incrementally and requires only one pass through the data set to be learned. Attribute, value pairs are supported as a variable implementation. Variables are represented by groups of connected cells in the network. The learning algorithm is described and an example given. Current results are discussed, which include learning the well-known Iris data set. The results show that the system has promise.

## Introduction

This paper describes a hybrid connectionist, symbolic approach to learning the type of information that would be used in rule-based expert systems. The system does its learning from examples. They are encoded in much the same way that examples to connectionist systems would be presented. The exceptions are due to our variable representation. The system can learn concepts where imprecision is involved. The network representation allows for variables in the form of attribute, value pairs to be used. Relational comparators are supported. The system has been used to learn knowledge bases from some small examples originally done in EMYCIN [Van Melle, et. al. 1984]. It has also been used in learning defects in CMOS semiconductor chips [Romaniuk and Hall 1989] and the Iris data set [Weiss, et. al 1989]. In this paper, the emphasis is on the learning algorithm. It uses a network structure, which is configured based on the distinct examples presented to the system. For examples which resemble others previously seen, bias values of cells in the network are adjusted. The system can learn incrementally. Rules may also be directly encoded in the network [Romaniuk and Hall 1989], but this is not shown here.

## SC-net – A Fuzzy Connectionist Expert System

A connectionist model is a network, which in its simplest format has no feedback loops. It consists of three types of cells (input, output, and hidden cells). Every cell has a bias associated with it, which lies on the real number scale. Cells are connected through links which have weights associated with them. In the SC-net model of a connectionist network, each cell can take on an activation value within the range [0..1]. This corresponds to the fuzzy membership values of fuzzy sets. The uncertainty handling constructs come from fuzzy set theory [Kandel 1986].

In fuzzy logic one may define disjunction (fOR) as the maximum operation, conjunction (fAND) as the minimum operation and complement (fNOT) as strong negation. Since fOR and fAND are defined as maximum and minimum operations, we let certain cells act as max and min functions, in order to provide for the above operators. In order to be able to distinguish cells as modeling the min (fAND) or the max (fOR) function we use the sign of the bias of a cell to determine which of the two functions is to be modeled. Furthermore, we denote a bias value of zero to indicate when a cell should operate as an inverter (fNOT).

## The Network Structure

We can think of every cell in a network accommodating  $n$  inputs  $I_n$  with associated weights  $CW_n$ . Every cell contains a bias value, which indicates what type of fuzzy function a cell models, and its absolute value represents the rule-range. Every cell  $C_i$  with a cell activation of  $CA_i$  (except for input cells) computes its new cell activation according to the formula given in Figure 1. If cell  $C_i$  (with  $CA_i$ ) and cell  $C_j$  (with  $CA_j$ ) are connected then the weight of the connecting link is given as  $CW_{i,j}$ , otherwise  $CW_{i,j} = 0$ . Note, an activation value outside the given range is truncated. An activation of 0 indicates no presence, 0.5 indicates unknown and 1 indicates true. In the initial topology, an extra layer of two cells (denoted as the positive and the negative cell) is placed before every output cell. These two cells will be collecting information for

$CA_i$  – cell activation for cell  $C_i$ ,  $CA_i$  in  $[0..1]$ .  
 $CW_{i,j}$  – weight for connection between cell  $C_i$  and  $C_j$ ,  $CW_{i,j}$  in  $\mathbb{R}$ .  
 $CB_i$  – cell bias for cell  $C_i$ ,  $CB_i$  in  $[-1..+1]$ .

$$CA'_i = \begin{cases} \min_{j=0,\dots,i-1,i+1,\dots,n}(CA_j * CW_{i,j}) * |CB_i| & CB_i < 0 \\ \max_{j=0,\dots,i-1,i+1,\dots,n}(CA_j * CW_{i,j}) * |CB_i| & CB_i > 0 \\ 1 - (CA_j * CW_{i,j}) & CB_i = 0 \text{ and } CW_{i,j} \neq 0 \end{cases}$$

Figure 1: Cell activation formula

(positive cell) and against the presence of a conclusion (negative cell). These collecting cells are connected to every output cell, and every concluding intermediate cell (these are cells defined by the user in the SC-net program specification). The final cell activation for the concluding cell is given as:

$CA_{\text{output}} = CA_{\text{positive\_cell}} + CA_{\text{negative\_cell}} - 0.5$ .

Note, the use of the cell labeled UK (unknown cell) in Figure 2. This cell always propagates a fixed activation of 0.5 and, therefore, acts on the positive and the negative cells as a threshold.

The positive cell will only propagate an activation  $\geq 0.5$ , whereas the negative cell will propagate an activation of  $\leq 0.5$ . Whenever there is a contradiction in the derivation of a conclusion, this fact will be represented in a final cell activation close to 0.5. For example, if  $CA_{\text{positive\_cell}} = 0.9$  and  $CA_{\text{negative\_cell}} = 0.1$ , then  $CA_{\text{output}} = 0.5$ , which means it is unknown. If either  $CA_{\text{positive\_cell}}$  or  $CA_{\text{negative\_cell}}$  is equal to 0.5, then  $CA_{\text{output}}$  will be equal to the others cell activation (indicating that no contradiction is present).

## Learning and Modifying Information

The system can be used (after learning) as an expert system. Alternatively rules can be generated for use in another system, which is discussed in [Romaniuk and Hall 1989]. We will next give a formal description of the learning algorithm used in the SC-net network model. This description is then followed by an example.

### Recruitment of Cells Learning Algorithm:

Let  $V_c$  be given as a learn vector with the following format:

$$V_c = (\delta_{v_1}, \delta_{v_2}, \dots, \delta_{v_h}, \delta_{v_{h+1}}, \delta_{v_{h+2}}, \dots, \delta_{v_{h+i}})$$

With  $v_k \in [1, N]$ , where  $N \geq h + 1$  is the number of input, intermediate and output nodes. The number  $v_k$  represents a component of the learn vector. Each of the  $\delta_m$  are thresholds for the individual components of the learn vector. Here the  $\delta_{v_k}$ ,  $k = 1, \dots, h$  are either facts or intermediate results. The  $\delta_{v_l}$ ,  $l = h + 1, \dots, h + i$  are either intermediate or final results. The  $v_k$  and the  $v_l$  components can both be intermediate results, since one can be an input to the other. For every learn vector  $V_c$  do

(1) Apply learn vector  $V_c$  to the current network, by assigning the thresholds of all facts and intermediate

results to the appropriate cells as their new activation.

For all  $k = 1, \dots, h$  do

let  $CA_{v_k} = \delta_{v_k}$ , that is assign the threshold  $\delta_{v_k}$  of every component  $v_k$  to its corresponding cell  $C_{v_k}$  as an activation. The above is called the initialization phase.

(2) Simulate the network in one pass by starting at level 1 and ending at level  $n$  ( $n$  is the maximum level of the network). In this simulation phase the new cell activation of every cell is calculated using the earlier listed formula for cell activation calculation. The only cells that are excluded from the simulation are input cells, or intermediate cells which correspond to any  $C_{v_k}$ ,  $k = 1, \dots, h$  of the learn vector. Their cell activation is provided through the use of the  $\delta_{v_k}$ .

(3) For all the  $l = h + 1, \dots, h + i$  component values of the learn vector  $V_c$  compare these (expected outputs  $\delta_{v_l}$ ) with the actual outputs  $CA_{v_l}$ . These are given through the cell activation of cell  $C_{v_l}$  after the simulation phase.

(3.1) Let  $\delta_{v_l}$  denote the expected (i.e. desired) output of cell  $C_{v_l}$  and  $CA_{v_l}$  the actual (activation) output of this cell. Let  $\epsilon$  be the error threshold (currently  $\epsilon = 0.01$ ). If  $(CA_{v_l} - \epsilon) \leq \delta_{v_l}$  and  $(CA_{v_l} + \epsilon) \geq \delta_{v_l}$  then consider the vector  $V_c$  learned for the output component  $C_{v_l}$ .

return to (3) until done with the  $h+i$  component.

else go to (3.2)

(3.2) In the case that there is a contradiction in expected output and calculated activation, it will be noted and the next  $l$  value processed at step 3. Otherwise,

if  $(|\delta_{v_l} - CA_{v_l}| > 5 * \epsilon)$  then go to (3.4)

if  $(CA_{v_l} > 0.5)$  then

for all cells  $C_j$  connected to the positive collector cell of cell  $C_{v_l}$  and  $\neq C_{v_l}$  do

calculate the new cell bias  $CB'_j$  as follows:

(3.2.1) if  $|CA_j - \delta_{v_l}| \leq 5 * \epsilon$  then

if  $CA_j > \delta_{v_l}$

$$CB'_j = \text{sign}(CB_j) * \left( |CB_j| - \frac{(|CB_j| - \delta_{v_l})}{\alpha} \right)$$

else

$$CB'_j = \text{sign}(CB_j) * \left( |CB_j| + \frac{(|CB_j| - \delta_{v_l})}{\alpha} \right)$$

end for

where  $\alpha$  indicates the degree in change of the cell bias

based on the old cell bias and the given threshold (currently  $\alpha$  is set to 6). The more  $\alpha$  is increased the less impact the newly learned information has.

return to (3) until done with the h+i component.

else go to (3.3)

(3.3) If  $CA_{v_i} < 0.5$  then

for all  $C_j$  connected to the negative collector cell of cell  $C_{v_i}$  and  $\neq C_{v_i}$  do

(3.3.1) if  $|CA_j - \delta_{v_i}| \leq 5 * \epsilon$  then

if  $CA_j > \delta_{v_i}$

$$CB'_j = \text{sign}(CB_j) * \left( |CB_j| - \frac{|(|CB_j| - \delta_{v_i})|}{\alpha} \right)$$

else

$$CB'_j = \text{sign}(CB_j) * \left( |CB_j| + \frac{|(|CB_j| - \delta_{v_i})|}{\alpha} \right)$$

end for

return to (3) until done with the h+i component.

(3.4) Consider the vector  $V_c$  as unknown to the current network. Therefore, recruit a new cell, call it  $C_m$ , from a conceptual pool of cells. Make appropriate connections to  $C_m$  as follows:

For all  $C_{v_k}$  in  $V_c$  do for  $k = 1, \dots, h$

(3.4.1) if the threshold  $\delta_{v_k} \geq 0.5$  then connect  $C_{v_k}$  to  $C_m$  with weight  $CW_{v_k,m} = \frac{1}{\delta_{v_k}}$ .

(3.4.2) if the threshold  $\delta_{v_k} < 0.5$  then recruit a negation cell, call it  $C_n$ , from the conceptual pool of cells (a negation cell has a bias of zero). Let  $CB_n = 0$ . Next, connect cell  $C_{v_k}$  to cell  $C_n$  and assign the weight  $CW_{v_k,n} = 1$ . Then connect cell  $C_n$  to cell  $C_m$  and assign as a weight  $CW_{n,m} = \frac{1}{1-\delta_{v_k}}$ .

(3.5) If the threshold  $\delta_{v_i} \geq 0.5$  then connect  $C_m$  to the positive collector cell  $C_{v_i, \text{positive}}$  and let  $CW_{m,v_i, \text{positive}} = 1$ . Additionally, let  $CB_m = -\delta_{v_i}$ .

Go to step (3) until the h+i component has been processed.

(3.6) if the threshold  $\delta_{v_i} < 0.5$  then recruit a new cell  $C_n$  from the pool of cells. Assign bias of 0 to  $CB_n$ . Connect  $C_m$  to  $C_n$  and make  $CW_{m,n} = 1$ . Let  $CB_m = -(1 - \delta_{v_i})$ . Now connect cell  $C_n$  to the negative collector cell  $C_{v_i, \text{negative}}$ . Let  $CW_{n,v_i, \text{negative}} = 1$ .

Go to step (3) until the h+i component has been processed.

In step (3), from 3.1-3.3, the algorithm is checking an error tolerance. If the example is within the tolerance, nothing is done. Otherwise, the biases of the appropriate cells connected to an information collector cell are modified to reduce the error, if the concept is close to an old one. The parameter alpha is used for tuning the algorithm. If the value is 1 the just learned vector takes precedence over everything previous. If it was set to infinity, no change would occur as a result of the new learn vector.

The steps 3.4, 3.5 and 3.6 of the algorithm are used when a new concept (or version of) is encountered. A

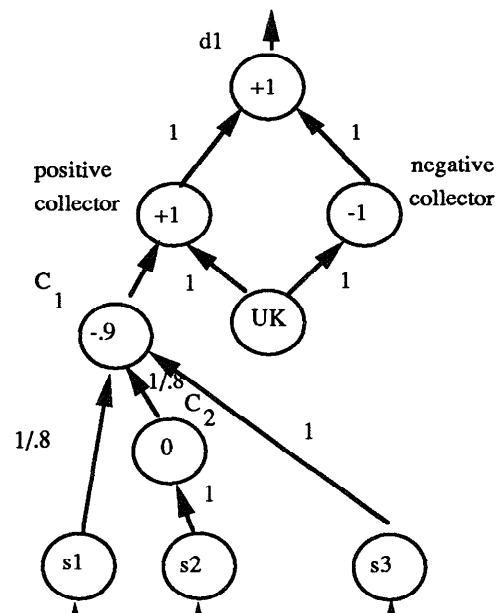


Figure 2: Network after learning  $V_1$

totally unfamiliar concept is indicated by an activation value of 0.5. The algorithm essentially creates a new reasoning path to the desired output in a manner somewhat analogous to an induced decision tree [Michalski, et.al. 1983].

We will now exemplify the above listed algorithm on a small example. Assume our initial network contains no information. The only thing we initially know about are its input, intermediate and output cells. Let these be given as follows: input cells  $s\{3\}$ ; output cells  $d\{1\}$ ;

The above represents a fragment of the SC-net programming language for setting up the initial network configuration (Note, that the si represent inputs and the di outputs). In our example we will not consider intermediate results. Let us further assume, we want the uncertainty values of the inputs to be defined over the range [0,1]. This means we use a value of 1 to represent complete presence of a fact/conclusion, a value of 0 complete absence, and finally let 0.5 denote the fact that nothing is known. We also allow uncertainty in any of the responses. Our initially empty network is given in Figure 2, if you subtract cells  $C_1, C_2$  and all connections associated with them. Let us next attempt to learn the following vector:  $V_1 = \{s1 = .8, s2 = .2, s3 = 1, d1 = 0.9\}$

Simulation of cell recruitment learning algorithm:

Since we have 3 facts and 1 conclusion,  $h=3$  and  $i=1$ . Step (1): In the initialization phase we assign the thresholds of every fact to the corresponding input cell as its new activation. This results in the following assignments:

$$CA_{s1} = .8, CA_{s2} = .2, CA_{s3} = 1.$$

Step (2): Next, the network is simulated. Since it is initially empty the resulting cell activation of d1 will be 0.5.

Step (3): Here l runs from 4 to 4. In the chosen example, there is only one conclusion (d1). Therefore,  $CA_{d1} = 0.5$  and  $\delta_{d1} = 0.9$ .

Step (3.1): The condition is false. We can conclude that  $V_1$  is an unknown concept to the network.

Step (3.2): Since  $\delta_{d1} = 0.9$  and  $CA_{d1} = 0.5$  the condition is also false.

Step (3.4): Having reached this point in the algorithm, we can conclude, that the concept represented by  $V_1$  has no representation within the network. This leads to recruiting a new cell  $C_1$  from the conceptual pool. We then connect the cells of all facts to the newly recruited cell  $C_1$ .

k=1. Step (3.4.1): Since  $\delta_{s1} = 0.8$  we connect cell  $C_{s1}$  to cell  $C_1$  and assign it a weight of  $\frac{1}{0.8} = 1.25$ .

k=2. Step (3.4.2):  $\delta_{s2} = 0.2$ , therefore a new cell  $C_2$  is recruited.  $CB_2 = 0$ . Connect  $C_{s2}$  to cell  $C_2$  and assign a weight of 1. Connect  $C_2$  to  $C_1$  and let the weight be  $\frac{1}{(1-0.2)} = \frac{1}{0.8}$ .

k=3. Step (3.4.1): Since  $\delta_{s3} = 1$ , it is connected to  $C_1$  and the connection assigned a weight of 1.

Step (3.5): Since we connect to the positive collector cell for d1, a weight of 1 is assigned. Finally  $CB_1$  is set to -0.9.

Fig. 2 shows the network after learning the first learn vector. Let  $V_2 = \{s1 = 0.9, s2 = 0.1, s3 = 1, d1 = 0.85\}$  be a second learn vector. Then step 3.2.1 of the algorithm will cause the bias for cell  $C_1$  to change. The bias will become -0.8916 after learning  $V_2$ .

In the case of contradictory information (over time or whatever) we will actually let the network learn the complement of what we want to forget. Therefore, if any fact fires in the network, so will its complement and the anding of the two will result in an unknown conclusion. It may surprise the reader that we are not deleting information from the network by changing the appropriate weights, which seems to have the same effect. This is done because some rules may be given to (encoded in) the system a priori. When the network is constructed solely through learning, changing connection weights serves the same purpose.

## Implementation of variables

### Overview

Variables are of great value in implementing powerful constructs in conventional symbolic expert systems. MYCIN [Waterman 1986] is a good example of such an expert system. It uses <Object, Attribute, Value> triplets for the implementation of variables. Connectionist type expert systems hardly make use of variables at all, since it seems that their implementation is far from being simple, if even possible to realize. As Samad [Samad 1988] pointed out, variable bind-

ings can be handled more elegantly using microfeatures, which will be associated with every cell (slots in the cell could be used to hold certain information, like variable values, binding information, separation of attributes and values etc.), or one could think of every cell representing a microfeature by itself. It is obvious that one can associate microfeatures with every cell in a neural network. The amount of microfeatures have to be fixed in the design phase and once they have been chosen they cannot be changed.

In another implementation one can think of every cell having just one or two microfeatures (actually indexes into some memory module) which point to some memory area, containing all the information for the implementation of variables and their values. Since one of the main purposes for using connectionist networks is speed (parallel processing), making use of the information in some central storage place seems to contradict the notion of parallel processing.

### Variables in SC-net

The approach here differs from the above in an attempt to avoid the problems discussed. The SC-net system distinguishes between two different types of variables. The first category consists of fuzzy variables. They allow the user to divide a numerical range of a variable into its fuzzy equivalent. Consider for example the age of a person. Let us refer to the variable as **age**. Age can now take on values in the closed interval [0..100] (this is accomplished by defining age as a variable in the declaration part of a SC-net program). Let us further assume that the following attributes are associated with this variable:

```
child : 0..12
teenager : 13..19(5, 25)
```

If age is assigned the value 15 then we might like the associated attribute teenager to be completely true and the attribute child somewhat true, too. The reason for this is that the age of 15 is still reasonably close to the child interval, causing it to be activated with some belief. The numbers in parentheses after the main range for the teenager declaration indicate how far on either side of the range some belief (less than total and going to 0 at the age of 5 and 25) will persist. The representation is accomplished by the use of a group of cells to represent the variable and attribute. This is true of both the fuzzy variables discussed above and the scalar variables, such as color [Romaniuk and Hall 1989]. Due to space considerations, only an example of age[teenager] is shown below. The age for our example has been compressed into [0, 1], such that an age of 17 would map to 0.17.

In general we will have the following format (pi-shaped function definition):

```
attribute_value : lower_bound..upper_bound
(lower_plateau, upper_plateau).
```

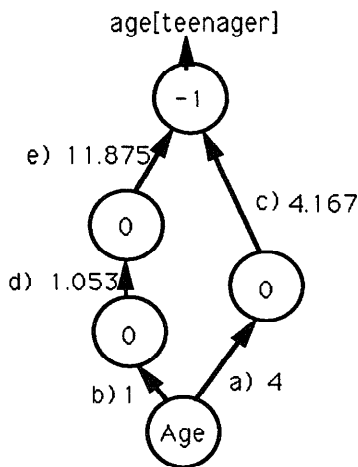


Figure 3: age[teenager] network

In case of the value teenager, we guarantee that a value  $< 5$  and  $> 25$  is certainly no longer a teenager (membership value equal 0), and a value of  $\geq 13$  and  $\leq 19$  returns a membership value of 1. Figure 3 shows the resulting network. The weights have been calculated as follows:

- a) weight:  $\frac{1}{\text{upper\_plateau}} = \frac{1}{0.25} = 4.$
- b) weight: 1.
- c) weight:  $\frac{\text{upper\_plateau}}{(\text{upper\_plateau} - \text{upper\_bound})} = \frac{0.25}{(0.25 - 0.19)} = 4.167$
- d) weight:  $\frac{1}{(1 - \text{lower\_plateau})} = \frac{1}{(1 - 0.05)} = 1.053.$
- e) weight:  $\frac{(1 - \text{lower\_plateau})}{(\text{lower\_bound} - \text{lower\_plateau})} = \frac{(1 - 0.05)}{(0.13 - 0.05)} = 11.875.$

### Current results

Our current research has been dealing with medium sized data sets. The results obtained have been quite promising. After allowing the system to learn the knowledge bases of GEMS and FEVERS, comparisons were made with an EMYCIN [Van Melle, et. al. 1984] version of these two knowledge bases running under MultiLisp [Halstead 1985]. All our tests were conducted by randomly sampling the rule bases, and determining possible assignments to the inputs (facts), that would fire the premise of some rule. In all cases the results were almost identical and only differed by a few percent in the certainty factor of the final conclusion. In the following, a couple of examples are given.

Fig. 4 gives a set of selected rules which were taken from the FEVERS knowledge base for the purpose of

```
(putprops rule032
subject patientrules
action (conclude patient comb-ab abdom-pain tally
1000)
premise ($and (same patient symp1 abdom-symp) ($or
(same patient abdominal rt-upper-pain) (same patient
abdominal lft-upper-pain) (same patient abdominal
lower-pain))))
```

```
(putprops rule033
subject patientrules
action (conclude patient disease fabry's tally 750)
premise ($and (same patient sex male)(same patient
comb-ab abdom-pain) ($or ($and (same patient comp-
host dis) (same patient dis-hist1 stroke)) ($and (same
patient symp1 skin-symp) (same patient skin punctate-
lesions)) (same patient symp4 edema))))
```

```
(putprops rule 054
subject patientrules
action (conclude patient disease rheumatic-fever tally
800)
premise ($and (between* (val1 patient age) 4 16)
($or ($and (same patient comp-host dis) ($or (same
patient dis-hist3 strep-hist) (same patient dis-hist3
rheumatic-fever-hist))) (same patient symp1 cardiac-
symp)(same patient symp5 chorea) ($and (same pa-
tient symp1 skin-symp) (same patient skin erythema-
marginatum))))))
```

Figure 4: Sample of randomly selected rules of the FEVERS knowledge base

this simulation. An approximate translation of rule 032 is the following.

rule032: if the patient has abdominal symptoms and the patient has right upper abdominal pain or left upper abdominal pain or lower abdominal pain then there is conclusive evidence (1000) that the patient's abdominal symptom is abdominal pain.

Tables 1 and 2 demonstrate two examples of actual simulations after learning. Each table indicates what rule(s) were fired in the EMYCIN version, lists the facts that were supplied and their certainty values. The system provided certainties are given under the goals derived with the EMYCIN certainty given first and our systems' second. The accuracy of the certainty values of the results are in all cases, within several percent, almost identical. The range of certainty is given as [-1000,+1000]. A straightforward mapping was done.

In comparison with the Iris data of 150 examples, we followed the example of Weiss [Weiss, et. al 1989]. After learning all cases, there was no apparent error. In our system this means that the solution presented with the most certainty was correct. It was occasionally the case that one or more other conclusions would have some certainty. In some cases the certainties were

rule	facts	goals
054	dis (999) strep-hist (999) cardiac-symp (555) age is 15 (999)	rheumatic-fever 799/799

Table 1: Example of crisp and uncertain input

rule	facts	goals
032	abdom-symp (700) rt-upper-pain (750)	abdom-pain 700/700
033	male (800) edema (750)	cholecystitis 560/530 fabry's 525/488

Table 2: Example with uncertainty and chaining

close to the chosen value. When the system was tested by learning all the cases except one and then being presented the left out case, the error rate was 0.04. These results compare favorably with the results shown by Weiss.

### Summary

In this paper we have presented a hybrid connectionist, symbolic learning system. This algorithm is based mainly on changing the network topology, then changing weights, as can be found in most conventional connectionist learning paradigms. An interesting fact one may have noticed about the simulation phase for learning is that it will not make a guess on an unseen example, if all the data is crisp and there are discrete ranges for all variables. The non-learning simulation is slightly modified to allow for partial pattern matches.

This system has some important expert systems features, which normally have been neglected in other approaches to constructing connectionist expert systems. These include the representation of variables, variable binding, and relational and fuzzy comparators. Other features such as consultation and explanation facilities have been added and tested favorably in several different applications. Finally, tests with several domains show good performance. On the one widely tested data set used, the system compares favorably with other methods.

### Acknowledgements

This research partially supported by a grant from the Florida High Technology and Industry Council, Computer Integrated Engineering and Manufacturing Division.

### References

- [Halstead 1985] Halstead, R. H. 1985. Multilisp: A Language for Concurrent Symbolic Computation. *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 4, October.
- [Kandel 1986] Kandel, A. (1986), *Fuzzy Mathematical Techniques with Applications*, Reading Ma., Addison-Wesley.
- [Keller, et. al. 1990] Keller, J. M., and Tahani, H. 1989. Backpropagation Neural Networks for Fuzzy Logic. To appear *Information Sciences*.
- [Van Melle, et. al. 1984] van Melle, W. , Shortliffe, E. H., and Buchanan, B. G. 1984. EMYCIN: A Knowledge Engineer's tool for constructing rule-based expert systems. In *Rule-based Expert Systems*, B. Buchanan and E. Shortliffe (eds)., Reading Ma. Addison-Wesley, 302-328.
- [Michalski, et.al. 1983] Michalski, R. S., Carbonell, J. G., Mitchell, T. M. 1983. *Machine Learning: An Artificial Intelligence Approach*. Palo Alto, Ca., Tioga Publishing.
- [Minsky and Papert 1988] Minsky, Marvin L., Papert, Seymour A. (1988). *Perceptrons*. Second Edition. Cambridge, Ma. The MIT Press.
- [Rummelhart, et. al. 1986] Rummelhart, D. E. , McClelland, J. L., (Eds.) 1986. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol I, Cambridge, Ma. MIT Press.
- [Waterman 1986] Waterman, Donald A. 1986, *A Guide to Expert Systems*. Reading, Mass: Addison-Wesley.
- [Weiss, et. al 1989] Weiss, S.M. and Kapouleas, I. 1989, An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods, Proceedings of IJCAI '89, Detroit, Mi.
- [Romaniuk and Hall 1989] Romaniuk, S. and Hall, L.O. 1989, FUZZNET, A Fuzzy Connectionist Expert System, Technical Report CSE-89-07, Dept. of Computer Science and Engineering, Univ. of South Florida, Tampa, Fl.
- [Romaniuk and Hall 1990] Romaniuk, S. G., Hall, L. O. 1990, FUZZNET: Towards a Fuzzy Connectionist Expert System Development Tool, In Proceedings of IJCNN, Washington D.C., January.
- [Samad 1988] Samad, T. 1988, Towards Connectionist Rule-Based Systems, In. Vol. II, Proceedings of the International Conference on Neural Networks.