

UNCERTAINTY MANAGEMENT IN CONNECTIONIST  
EXPERT SYSTEMS

BY

AYMAN ALY RADWAN

Bachelor of Science

Ain Shams University

Cairo, Egypt

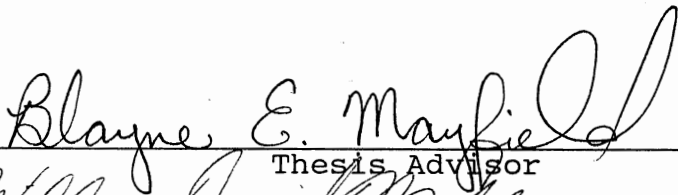
1984

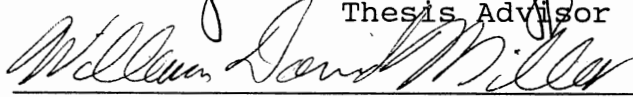
Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the degree of  
MASTER OF SCIENCE  
May, 1992

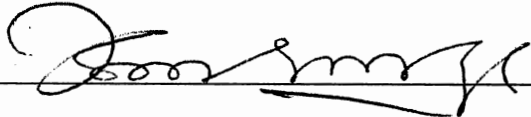
Thesis  
1992  
R1324

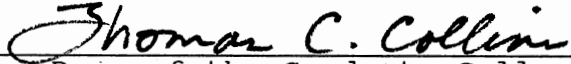
UNCERTAINTY MANAGEMENT IN CONNECTIONIST  
EXPERT SYSTEMS

Thesis Approved:

  
\_\_\_\_\_  
Thesis Advisor

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_  
Dean of the Graduate College

## PREFACE

A connectionist expert system is an expert system whose knowledge base is generated from training examples using an artificial neural network learning technique. Gallant [13] developed a model for a connectionist expert system in which a variable is represented by a node and accepts two values, true or false. This study adopts two approaches to help manage uncertainty in Gallant's model. The first approach is called the random cell method while the second one is the stairstep method.

I wish to express my sincere gratitude to my advisors Dr. Blayne Mayfield, Dr. David Miller, and Dr. K. M. George for their support and guidance during my study.

I would like to thank my parents, my brother Radwan, and my sister Hanan for their support and encouragement which helped me keep the end goal constantly in sight.

I wish to acknowledge USAID Project No. 263-0132, the Egypt Water Research Center, and Colorado State University for providing the financial assistance and the opportunity for conducting this research.

## TABLES OF CONTENTS

Chapter	Page
I. INTRODUCTION .....	1
Connectionist Expert Systems .....	1
Artificial Neural Network .....	1
Interaction with the environment .....	7
Topological Changes .....	8
Expert Systems .....	9
Expert Systems Advantages .....	10
Connectionist Expert Systems .....	10
Objective of the Study .....	15
II. LITERATURE REVIEW .....	19
Uncertainty in Expert System .....	19
Representation of Uncertain Information ...	19
Subjective Probability Theory .....	20
Dempster Shafer Theory .....	22
Weaknesses of DST .....	23
Bayesian Belief Network .....	24
Certainty Factor .....	24
The Use of Certainty Factor.....	25
Disadvantages of CF .....	26
Possibility Theory .....	27
Fuzzy Set Theory .....	27
Modifier Rules .....	29
Composition Rules .....	29
Truth Qualification Rules .....	29
Inference in Possibility Theory .....	29
Weaknesses of Possibility Theory .....	30
Training Algorithms for Neural Networks ...	32
Perceptron Convergence Procedure .....	32
Back-Propagation .....	33
Deficiencies of Back- Propagating .....	36
Generalization .....	36

Chapter	Page
III. APPROACHES TO HANDLE UNCERTAINTY IN CONNECTIONIST EXPERT SYSTEMS .....	38
The Random Cells Approach .....	38
The Stairstep Approach .....	43
Training algorithm for stairstep approach .....	48
IV. SIMULATION AND RESULTS .....	50
V. SUMMARY AND CONCLUSION .....	65
BIBLIOGRAPHY .....	68
APPENDIXES .....	72
APPENDIX A - RULES OF THE NETWORK NODES .....	72
APPENDIX B - TRAINING RESULTS CHARTS .....	82

## LIST OF TABLES

Table	Page
1. Results with training 10% of the hypothetical space of each node .....	55
2. Results with training 20% of the hypothetical space of each node .....	56
3. Results with training 30% of the hypothetical space of each node .....	57
4. Results with training 40% of the hypothetical space of each node .....	58
5. Results with training 50% of the hypothetical space of each node .....	59
6. Results of output nodes after testing the network by 10000 test cases (all nodes are trained on 10% of hypothetical space) .....	60
7. Results of output nodes after testing the network by 10000 test cases (all nodes are trained on 20% of hypothetical space) .....	60
8. Results of output nodes after testing the network by 10000 test cases (all nodes are trained on 30% of hypothetical space) .....	61
9. Results of output nodes after testing the network by 10000 test cases (all nodes are trained on 40% of hypothetical space) .....	61
10. Results of output nodes after testing the network by 10000 test cases (all nodes are trained on 50% of hypothetical space) .....	62
11. Comparison between random cell approach and stairstep approach .....	64

## LIST OF FIGURES

Figure	Page
1. Simplified drawing of a neuron .....	2
2. Artificial neural network .....	4
3. Activation calculation .....	5
4. Nonlinearity functions .....	6
5. Neural network classifications .....	8
6. Stairstep approach .....	18
7. Rosenblatt's idea (random functions) .....	41
8. Distributed method .....	42
9. Random cell approach .....	43
10. Internal structure of a node .....	44
11. Stairstep function .....	45
12. Connectionist model .....	51
13. Results of training node 22 .....	83
14. Results of training node 23 .....	83
15. Results of training node 24 .....	84
16. Results of training node 25 .....	84
17. Results of training node 26 .....	85
18. Results of training node 27 .....	85
19. Results of training node 28 .....	86
20. Results of training node 29 .....	86
21. Results of training node 30 .....	87



Figure	Page
22. Results of training node 31 .....	87
23. Results of training node 32 .....	88
24. Results of training node 33 .....	88
25. Results of training node 34 .....	89
26. Results of training node 35 .....	89
27. Results of training node 36 .....	90
28. Results of training node 37 .....	90
29. Results of training node 38 .....	91
30. Results of training node 39 .....	91
31. Results of training node 40 .....	92
33. Results of node 36 after testing the network with 10000 test cases .....	92
34. Results of node 37 after testing the network with 10000 test cases .....	93
35. Results of node 38 after testing the network with 10000 test cases .....	93
36. Results of node 39 after testing the network with 10000 test cases .....	94
37. Results of node 40 after testing the network with 10000 test cases .....	94

## NOMENCLATURE

AI	Artificial Intelligence
CF	Certainty Factor
DST	Dempster-Shafer Theory
FD	Frame of Discernment
FST	Fuzzy Set Theory

## CHAPTER I

### INTRODUCTION

#### Connectionist Expert Systems

##### Artificial Neural Network

Artificial neural networks are inspired by the way in which biological beings process information. A neuron is the lowest basic element of a mammalian brain. Actually, a neuron consists of three elements (Figure 1), which are:

- \* Cell body,
- \* Dendrites, and
- \* Axon.

A dendrite is the connection which transmits input signals, or impulses, from the axons of other neurons to the cell body of the neuron from which the dendrite originates. Signal transmission takes place through **a synapse** (Figure 1). Unlike electrical circuits, there is no physical or electrical connection at the synapse. Instead, there is a narrow gap called **a synaptic cleft** which separates a dendrite from an axon. The end bulbs of an axon release specialized chemicals, known as **neurotransmitters**

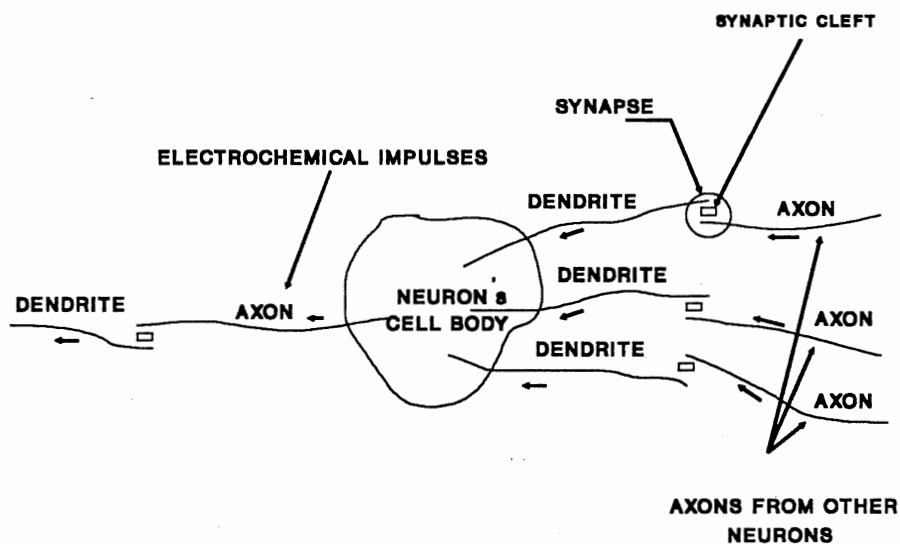


Figure 1. Simplified Drawing of a Neuron

which are produced by a neuron. These neurotransmitters are passed through a synaptic cleft to a dendrite, which transmits an electrochemical current to the cell body of another neuron.

Currents from all dendrites attached to a cell body are summed and averaged in the cell body. If the average over a short time interval is sufficiently large, the cell "fires" producing a pulse that passes down its axon to succeeding neurons. A neuron's capability to trigger, or activate, another neuron connected to it increases with the rate of those electrochemical impulses.

The origin of the artificial neural network goes back to McCulloch and Pitts [26] who developed a mathematical model to simulate a neuron. In an artificial neural

network, numerical weights among neurons are altered to simulate the changing of electrochemical responses in natural neurons. The artificial neural paradigm is sometimes called **connectionism** since it models solutions to problems by training simulated neurons in a highly connected network.

A connectionist model is specified by the following features:

- \* Network topology.
- \* Node characteristics.
- \* Training or learning rules.

The network topology describes the relationship among the network nodes and how they are organized. In some connectionist models a subset, or layer, of nodes is called the input layer, in which the node inputs are set externally. Another subset of nodes is called the output layer. Its outputs are used as the output of the network. A third kind of subset is called the intermediate, or hidden, nodes. These are necessary in many types of network computing functions known as nonseparable functions.

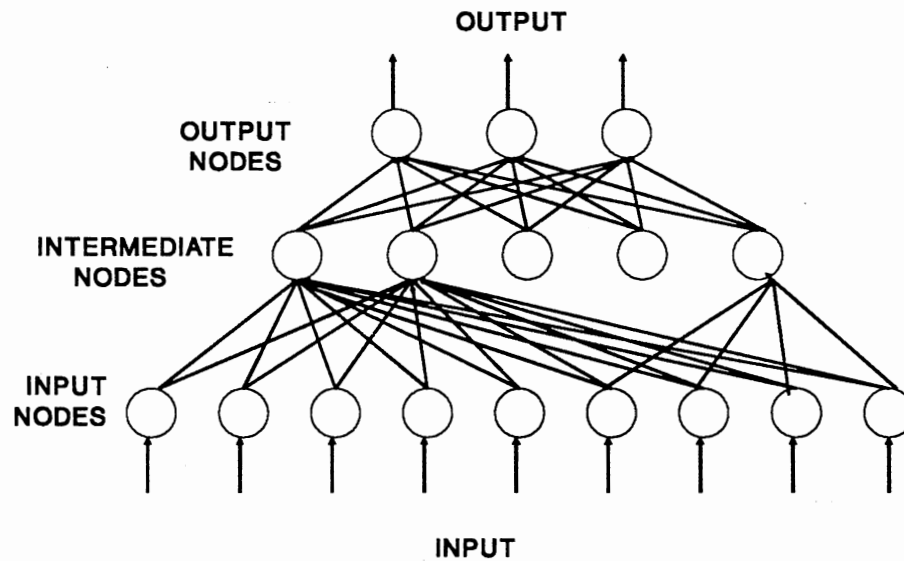


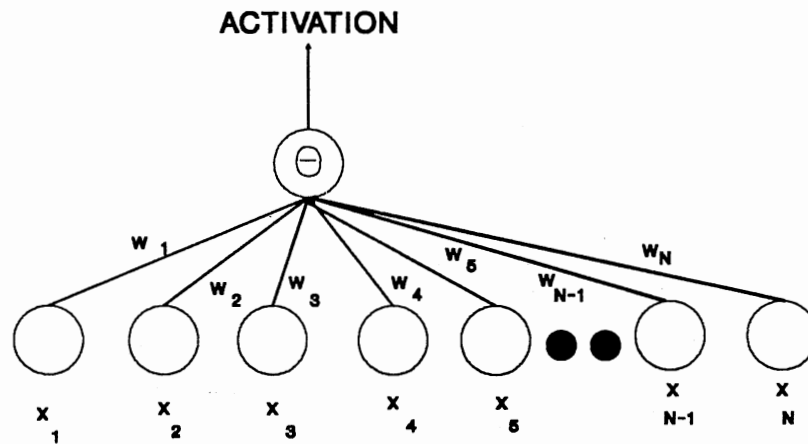
Figure 2. Artificial Neural Network

Connectionist models can be classified according to the direction of computation as follows:

- 1- **FEEDFORWARD** network, in which there are no directed cycles (i.e. the results of output or intermediate layers do not feed back to previous layers).
- 2- **FEEDBACK** network, in which there are directed cycles.

Node characteristics determine the way in which a node computes its activation, which is the output value computed by the following equation:

$$\text{activation} = f((\sum w_i * x_i) - \theta), \text{ for } i = 1 \text{ to } N,$$



$$\text{ACTIVATION} = F \left( \sum_{i=1}^N w_i x_i - \theta \right)$$

Figure 3. Activation Calculation

where  $w_i$  is a weight assigned to arc  $i$ ,  $x_1$  through  $x_n$  represent the activation levels of all nodes that feed impulses into the current node, and  $\theta$  is the threshold value for the current node. The threshold value is considered as a minimum limit that the sum  $\sum w_i * x_i$  must exceed in order to fire the cell. The activation function is usually nonlinear as it is shown in Figure 4.

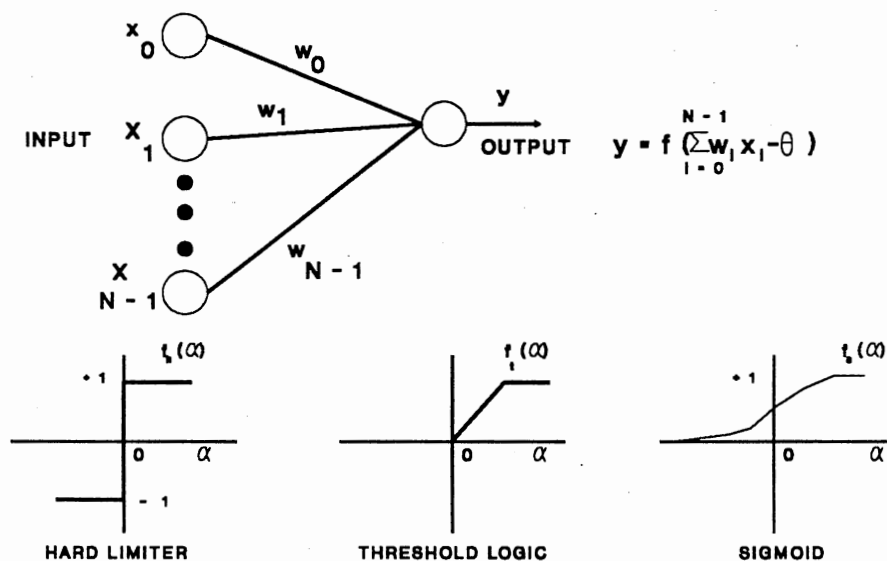


Figure 4. Nonlinearity Functions

Cell inputs and activations could be discrete, taking values in  $\{0,1\}$  or  $\{-1,0,1\}$ , or may be continuous, assuming values in the interval  $[0,1]$ . This model of a neuron, which is shown in Figure 3, is also known as a **perceptron** [24]. Perceptron attracted a great deal of attention especially after Rosenblatt [31] developed an algorithm to train it; this will be discussed in the next chapter.

In some connectionist models, nodes are visited in a fixed order, where activation is computed before visiting subsequent nodes. In other models, all nodes compute their activations simultaneously; while still other models pick a random node and compute its activation before computing the output of another randomly chosen node.

A neural network is taught to produce desired



outputs, or at least consistent ones, when a set of inputs are applied to the network. Learning is accomplished by sequentially applying input vectors to the network and adjusting the weights of arcs according to the obtained output. During training, the network weights gradually converge to values such that each input vector produces its desired output. Learning techniques are classified as follows:

**A. Interaction with the environment.**

- 1- Supervised learning.
- 2- Unsupervised learning.

**B. Network changes.**

- 1- Weight-change only.
- 2- Topology-change only.
- 3- Weight and topology-change.

Interaction with the Environment. In **supervised** learning, the desired outputs of the input data are known in advance. This type of learning adjusts the arcs' weights so that the resulting output is similar to the desired one. On the other hand, in **unsupervised** learning only the input data are known, and the learning algorithm classifies a set of input vectors into disjoint clusters in a way that elements in one cluster are similar to each other.

Topological Changes. Learning in which weights change is biologically plausible. Topology-change learning is a fairly recent technique, in which recruitment of uncommitted

nodes (i.e. adding new nodes to a network during the learning procedure) is taking place. Topology-change learning can be combined with weight-change as it has been used in back-propagation learning [23] for the allocation of new hidden nodes.

Most known neural networks can be classified according to their input type, binary or continuous, or their training method, supervised or unsupervised, as shown in figure 5.

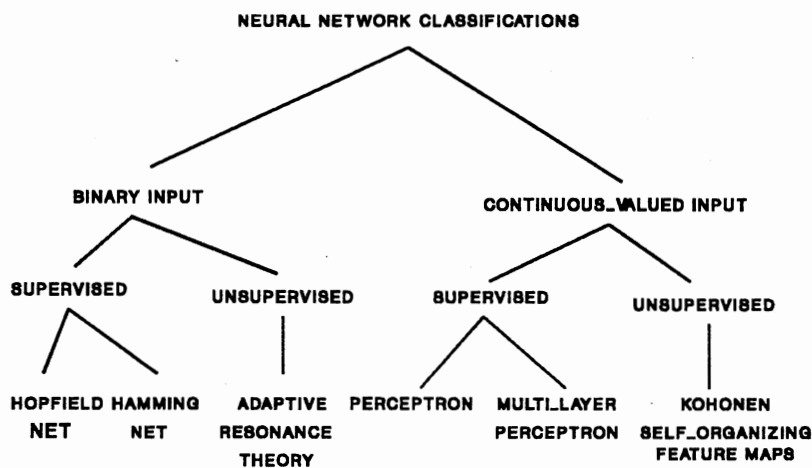


Figure 5. Neural Network Classifications

Learning can be classified as **easy**, in which the training examples specify the output of the intermediate cells, or it can be **hard**, in which the learning algorithm must find appropriate values for the intermediate cells.

## Expert Systems

Expert systems is a branch of AI that makes extensive use of specialized knowledge to solve problems at, near, or above the level of human expertise in some restricted problem domain. An ideal expert system includes the following features:

- \* Extensive specific knowledge from the domain of interest.
- \* Support for heuristic analysis.
- \* Application of search techniques.
- \* Capacity to infer new knowledge from existing knowledge.
- \* Ability to explain reasoning.

An expert system structure includes the following components:

- \* **User interface**, in which the user supplies facts and necessary information to the system and receives the required advice.
- \* **Explanation facility**, which is responsible for explaining the different decisions taken by the system.
- \* **Knowledge update facility**, which updates the knowledge base according to new facts or rules supplied by the user during the interface stage.
- \* **Knowledge base**, which contains the knowledge of the expert system domain represented in a specific structure that depends on the design provided by the knowledge engineer. Knowledge representation schemes in conventional expert systems vary from formal schemes such as formal logic

to non-formal schemes such as frames, scripts, semantic networks, or production systems.

- \* **Inference engine**, which is a driver program used by the system to infer conclusions from knowledge in the knowledge base.

Expert System Advantages. Expert systems have several advantages; some of them are:

- \* Making rare expertise widely available to many people.
- \* Reducing the cost of human experts by replacing them with expert systems, which presumably have the capability of mimicking their performance.
- \* Introducing intelligent tutoring, which could be considered as a good way of distributing the expertise among novices with the help of explanation facility.
- \* Combining the expertise of several experts in one expert system.

#### Connectionist Expert Systems

A connectionist expert system is an expert system whose knowledge base is generated from training examples using an artificial neural network learning technique. Gallant [13] introduces a model of a connectionist expert system in which control information within the knowledge base is represented by a matrix of integers. The general structure of the model is a network model similar to the network used in the PROSPECTOR program [13]. The nodes of

the network are variables, and each of them may assume one of three values:

1 ---> true  
-1 ---> false  
0 ---> unknown

Variables are grouped into 3 classes:

- \* Goal variables.
- \* Intermediate variables.
- \* Terminal variables.

Goal variables are the variables for which the network is designed to induce their values, such as treatments required for a certain disease. Intermediate variables, such as diseases, are the variables whose values are induced from the terminal variables, such as symptoms, and are used to induce goal variable values. Terminal nodes are set externally by the user either as initial input or as a response to questions from the system. The function of the expert system is to find out the activations, or the values, of goal variables given activations of terminal variables.

Generation of the network requires specifying the following information:

- \* The name of each node corresponding to a variable of interest.
- \* A question for each input variable, to elicit its value from the user.
- \* Dependency information about the variables in the system. This dependency information helps in the training

operation because it discards irrelevant variables from the training list for a node being trained.

- \* Sets of training examples for intermediate and output nodes. Each training example contains activation values for variables on which an output node depends along with the desired activation value for that output. The weights that are obtained after the training procedure will be arranged in the learning matrix, as it will be explained shortly.

Gallant specifies in his model that the variables in the system have hierarchical ordering (i.e. variables are indexed from 0, for terminal variables and the index is increased for intermediate and external output variables). Any variable will have a higher index than the indexes of the variables on which it depends. Also, he assumes that the network has no directed cycles.

A linear discriminant is used in Gallant's model to classify an intermediate or an output variable in the network. Each intermediate or output node in the network has nodes connected to it via arcs, which have their own weights set by the training algorithm. A linear discriminant is computed as follows:

$$D = \sum W_i V_i,$$

where  $W_i$  and  $V_i$  are the weight and the activation value for arc  $i$  and node  $i$  respectively. The activation for a node,  $V$ , will be:

$$1 \quad \text{if } D > 0.$$

-1 id  $D \leq 0$ .

Gallant defines the **Learning Matrix**,  $L$ , for a given set of variables and dependencies as a matrix of integers which has one row for each non-terminal variable and one column for each variable. Each row of  $L$  contains the weights of the arcs directed into the non-terminal variable corresponding to that row. Thus, the **knowledge base** for that system will consist of the **network**, **weights**, and **questions** for variables.

Inference in a connectionist expert system is performed by the **MATrix Controlled Inference Engine**, **MACIE**, which operates as follows:

- 1 Initially obtain values for some variables from the user.
- 2 Forward chain: make inference about non-terminal variables and compute likelihoods, which will be defined later.
- 3 If some goal variables are true, all goal variables are known to be false, or no more useful information is available, stop.
- 4 Pick an unknown goal variable, such as  $G$ , with the highest likelihood.
- 5 Backward chain: find a useful unknown terminal variable for determining  $G$  and obtain its value (by asking the user).
- 6 Goto 2.

In forward chaining, inference for a variable  $V_i$  on partial information, which happens when not all the values

of its dependent variables are known, could be done in the case that the activations of these unknown variables do not influence the sign of the discriminant calculation. So, if  $D$  is the current value for the discriminant calculation and  $U$  is the maximum effect of unknown variables, then

$$D = \sum_{j=0}^n L_{ij} V_j$$

$$U = \sum_{(j : V_j \text{ unknown})} |L_{ij}|$$

and if  $|D| > U$ , then we can conclude that

$$V_i = \begin{cases} 1 & \text{if } D > 0 \\ -1 & \text{if } D \leq 0. \end{cases}$$

Likelihood estimation is used to compare any two unknown variables to determine which one has more likelihood to eventually be deduced to be true or false. This comparison is useful when there are more unknown variables that have one bottom-up pass in the network. To compute the likelihood  $A_i$  for a variable  $V_i$ :

$$A_i = \begin{cases} V_i & \text{if } V_i \text{ is known,} \\ 0 & \text{if } V_i \text{ is unknown terminal variable,} \\ \frac{\sum_{j=0}^n L_{ij} A_j}{\sum_{j=0}^n |L_{ij}|} & \text{otherwise.} \end{cases}$$

In backward chaining, the system backtracks to find the values of unknown variables. Those variables are needed to compute the activation of output variables, whose current known dependent variables are not enough to do so. One of the heuristics to choose an input cell is:



- 1- Select the unknown output variable  $u_i$  such that  $|A_i|$  is maximum.
- 2- In pursuing node  $u_i$ , find the unknown cell  $u_j$  with the greatest absolute influence on  $u_i$ :

$$\max |L_{i,j}| : u_j \text{ unknown}$$

In this model an explanation facility is available to explain actions of the system in "if..then" format although input variables don't exist in that format.

#### Objective of the Study

The connectionist expert system, introduced by Gallant, has a major drawback because it deals with two-valued logic, true or false. Thus, the purpose of this study is to introduce uncertainty management into the connectionist expert system. To accomplish this goal, two approaches will be implemented and their performance will be compared. The first approach is to replace a node for each variable by several nodes to describe a certain state for that variable. Each variable state could represent a fuzzy term such as high, low, or medium. This fuzzy representation is suitable in domains where reasoning is performed on ill-defined boundary predicates such as in medical diagnosis. As another option, each variable state could specify a number to represent a certainty factor or the probability of that variable occurring. Each variable state ranges from 1 to 6, where a value of 1 will be represented as 1 0 0 0 0 0 while a value of 6 is represented as

1 1 1 1 1 1. Each training example describes the state of each input and the state of the correct output.

Usually training these types of inputs and outputs involves several difficult boolean functions that cannot be solved by a single perceptron, especially in the case of inconsistent training examples or even contradictory ones. To overcome this problem, the concept of random cells is adopted in the training algorithm. Gallant [15] proposed random cells as a modification to the random functions introduced by Rosenblatt [32] to enhance the learning capability of the perceptron.

In the approach, in this thesis, it is possible to combine several concepts of uncertainty interpretations in the system. For example, if we have a variable in the system for which our concern is the probability of its existence, then we may represent a probability number as the state of that variable. On the other hand, if we are concerned about the fuzziness of another variable, then the state will represent a fuzzy term. All these representations may be done in a single system to make it consistent with Chandrasekaran's [8] principle that **"resolution of uncertainty should be left to the human who is an expert in that domain."** Also, he mentioned that **"humans do not use a single method for resolving uncertainties of various types and a search for normative uncertainty calculi is pointless."**

The second approach is implementing the back-

propagation algorithm with the following modification. Each node in the network has its own internal structure as shown in Figure 6. The purpose of that structure is to obtain an output in the form of a uniform stairstep function. Each step in the output function represents a fuzzy term, such as low or high. The weights of arcs in the internal structure are developed independently of training examples so that the output function will take the form of a stairstep as it is shown in Figure 6.

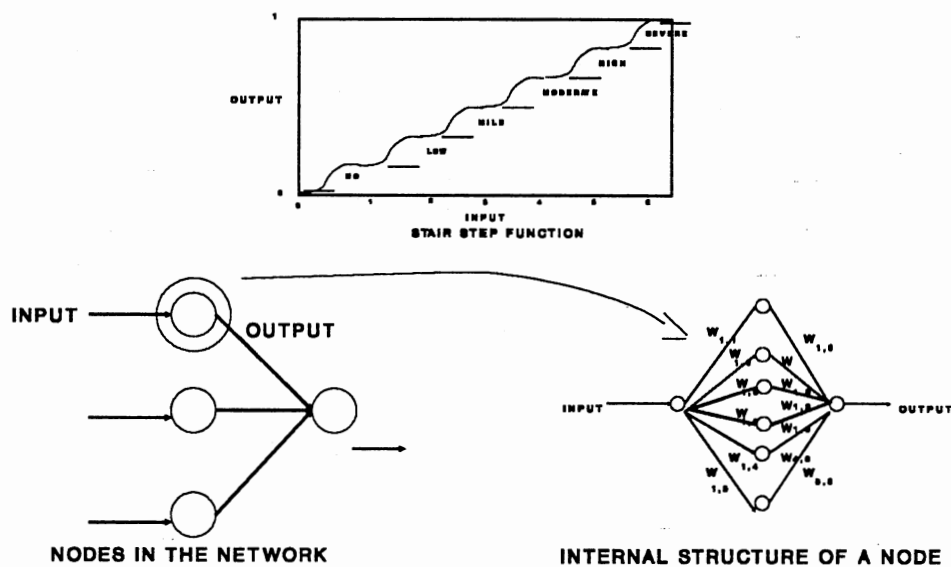


Figure 6. Stairstep Function approach

The simple output function in the back-propagation algorithm is replaced by a function in terms of the weights and biases of the internal structure of a node. As a result,

the delta value, which is used to adjust weights in the backward pass, is adjusted by the derivative of the modified output function.

A simulation of the connectionist expert system was developed to study the performance of each method and establish a comparison between them.

## CHAPTER II

### LITERATURE REVIEW

#### Uncertainty in Expert Systems and Neural Networks Training Process

##### Uncertainty in Expert System

Uncertainty in expert systems has the following sources [9]:

- \* Information can be unreliable, which usually happens due to ill-defined domain concepts or inaccurate data.
- \* Descriptive languages lack precision, which comes from expressing knowledge in natural language terms which have ill-defined boundaries.
- \* Inferences are sometimes drawn with incomplete information, so the system must accept unknown variables and perform approximate matching upon them.
- \* Experts sometimes disagree. In this case the expert system must attempt to resolve this conflict to obtain consistent decisions even when experts are contradicted.

##### Representation of Uncertain Information

Numerous paradigms have been used to represent uncertain information in expert systems; some of them are

quantitative, such as subjective probability theory, Dempster-Shafer theory, possibility theory, and certainty factors, and some are qualitative methods, such as Cohen's theory of endorsements.

### Subjective Probability Theory

Pascal and Fermat [28] defined an event's probability as **the proportion of cases in which the given event occurs**. This view of probability is known as **classical**, or **objective**, probability. Actually the interpretation of probability has been extended to include two other classes, which are [9]:

- \* **Personalistic or Subjective probability** which represents the degree of belief or confidence of an individual in a particular proposition's truth. The term **Bayesian** is often used as a synonym for subjective probability.
- \* **Logical probability** which measures the extent to which a set of propositions, isolated from human opinion or logical necessity, confirm the truth of another.

In most expert systems the subjective probability is preferred, due to the difficulty of implementing the objective probability that requires a large number of observations for events which may not be available in the real world.

The main advantage of the Bayesian approach is its having a strong mathematical background, unlike other methods, such as certainty factors or possibility theory

(which will be discussed later). Hence, Bayes' method gives accurate results, given that accurate input values to the system are available.

However, Bayesian approach has several drawbacks such as:

- \* It needs a huge number of probabilities that must be known to make the knowledge base applicable.
- \* It does not deal with unknowns. However, Cheeseman [6] argues that specifying two numbers, probability, and its standard deviation, solves the problem. Ignorance can be represented by a probability number along with a large standard deviation, while certain events will have a small standard deviation.
- \* Zadeh [42] claims that probability theory cannot pass the following tests, given that the predicates are fuzzy, while fuzzy logic does:
  - 1- It must provide a system for representing the meaning of various types of propositions related to uncertain events and uncertain dependencies.
  - 2- It must provide a system for inferring from a knowledge representation mentioned above.
- \* The assumption of conditional independence among pieces of evidence for the given hypothesis is not always true in real problems.

From the above discussion, it appears that the subjective probability approach is best suited for applications in which prior and conditional probabilities

are obtainable and when the assumption of conditional independence is true.

### Dempster-Shafer Theory

Dempster-Shafer Theory, DST, was developed by Arthur Dempster [12] and extended by Glen Shafer [33]. The motivation for this theory is to handle the deficiencies in the probability theory such as:

- \* The representation of ignorance.
- \* The idea that the subjective beliefs assigned to an event and its negation must sum to one.

In this theory, Dempster handled uncertainty by a range of probabilities rather than a single probability number. Dempster stated "in many situations, evidence that only partially favors a hypothesis should not be constructed as also partially supporting its negation."

Dempster theory succeeds in expressing ignorance explicitly. Probability theory expresses ignorance in two hypothesis, A and B, by assigning equal probability for both of them; i.e.,  $p(A) = p(B) = .5$ , while  $m(A) = m(B) = .5$  in Dempster theory means that the belief in A and B is the same. In other words, if all the focal elements, which are the subsets of the frame of discernment, are singletons, then no ignorance regarding their occurrence exists, and if there is a focal element which has more than one element, then some ignorance exists.



- Weaknesses of DST. 1- One of the major disadvantages of DST is its implementation complexity because nearly all the functions require exhaustively enumerating all possible subsets of FD, Frame of Discernment.
- 2- In the case where there are two expert sources, for example, one for whom the FD is {a, b, c, d} and other whose FD is {b, c, d}, then evidence combination will be impossible [2].
- 3- The assumption of the independence of evidence. When this is not the case, the result may be biased.
- 4- The assumption that the elements of FD are mutually exclusive is not always true as in the problem of multiple diseases in medical diagnosis. To overcome this problem, FD must be redefined to include all possible subsets of all the diseases. The redefinition of FD carries a problem of exponential growth of the computation, so if we have 100 diseases, then  $FD = 2^{100}$ .
- 5- DST lacks an effective decision making procedure to draw inferences from belief functions. Shafer and Logan have developed an algorithm which computes degree of belief for more hypotheses and implements Dempster's combination rule for hierarchical evidence. Until now no consensus existed on a formal scheme to be adopted in the belief propagation mechanism.
- 6- Zadeh [43] pointed out that the method of normalization ignores the belief that the object being considered does not exist.

### Bayesian Belief Network

Judea Pearl [29] has developed the Bayesian Belief Network, which is based on subjective probability theory, to show that the Bayesian approach is capable of providing results in a tree structured hierarchy of hypotheses.

A belief network is a network that consists of nodes and links. Each node represents a probabilistic variable. Each link between two nodes represents a relationship between them. The relationship is identified with a matrix that contains conditional probabilities. A knowledge engineer has to assign a relevant domain variable to each node in the net and determine the causal relationships among the nodes.

Pearl's updating scheme [28] added more flexibility for backward and forward inference in the network. The weak point in the Bayesian network is the assumption of conditional independence among variables, which cannot be generalized to real world problems. Also, the requirement of supplying the prior probabilities for the top nodes in the network may restrict the use of that network to the applications where obtaining these probabilities is possible.

### Certainty Factor

When Shortliffe [35] started developing his expert system MYCIN, he felt that the uncertainty management in

that system cannot be handled appropriately by the probability approach due to the following reasons:

- \* In medical problems, collecting good data to express prior probability and conditional probability is not an easy task and in many cases not all data is available.
- \* Probability is weak in representing medical knowledge and heuristics, which are required to solve a given problem.
- \* The probability approach does not offer a good explanation. Shortliffe derived his method from Carnap's theory of confirmation[4]. The certainty factor is defined as:

$$CF (H,E) = MB (H,E) - MD (H,E),$$

where

CF denotes hypothesis belief given observed evidence.

MB measures the degrees to which belief in hypothesis H would be increased if E were observed.

MD measures the degree to which the disbelief in H would be increased by observing the same evidence E.

The Use of the Certainty Factor. The main purpose of the certainty factor is to [22]:

- 1- Guide the program in its reasoning.
- 2- Cause the current goal to be deemed unpromising and pruned from the search space if its CF falls in the range [+0.2,-0.2].
- 3- Rank hypotheses after all the evidences have been

considered.

Disadvantages of CF. \* Adams [1] argues that the CF approach is deficient in ranking hypotheses properly as he explained in the following example:

Suppose we have two hypotheses **d1** and **d2** and evidence **e** which confirms the two hypotheses, and assume that

$$p(d1) = .8$$

$$p(d2) = .2$$

$$p(d1 | e) = .9$$

$$p(d2 | e) = .8$$

$$\text{then MB}(d1) = \frac{0.9 - 0.8}{0.2} = .5$$

$$\text{MB}(d2) = \frac{0.8 - 0.2}{0.8} = 0.75$$

So  $CF(d1,e) < CF(d2,e)$  which contradicts

$$p(d1 | e) > p(d2 | e).$$

\* Horvitz and Hecherman [21] have criticized the use of certainty factors as a measure of change in belief, given the fact that CFs were elicited from experts as a degree of absolute belief. As a consequence, the evidence combination function employed by the CF approach, which treats CFs as belief updates, results in values that are inconsistent with Bayes' theorem.

\* the major criticism for CF is that it is an ad-hoc approach even though it has some basis in probability and confirmation theories. The success of MYCIN, which adopts

the CF approach, cannot generalize to success of CFs in other fields of expert systems because MYCIN has short inference chains and simple hypotheses.

### Possibility Theory

As mentioned above, Lotfi Zadeh [42] criticized the probability approach because it is based on two-valued logic, true or false, which is not the case in most real-world applications. An expert system knowledge base has a great deal of human knowledge which, in most cases, has imprecise and qualitative information. Quite often experts express their knowledge in ill-defined boundary expressions such as low, high, or very likely. So, Zadeh [38] developed his possibility theory to handle the problem of fuzziness. Possibility theory is considered an extension to the fuzzy set approach, also developed by Zadeh [39].

Fuzzy Set Theory. In fuzzy set theory, an object may belong to several fuzzy sets with different degree of memberships. Membership is a real number associated with each element in the set and it ranges from 0 to 1. An object with membership of 1 means that the object belongs completely to the set while membership of 0 indicates that the object absolutely does not belong to the set. It is possible that an object has a degree of membership between 1 and 0 to indicate partial belonging. Thus, the difference between a fuzzy set and a crisp set, in which a member from

the universe of discourse either belongs to the set or not, is the range of possible values of membership, because in crisp sets it is 1 or 0 only. As an illustration for the concept of a fuzzy set, we consider the following example [9]:

Let  $U$  be the set of integers,  $U = \{1, 2, 3, \dots\}$ .

Let  $A$  be a fuzzy set of small numbers.

Then a subjective characterization of  $A$  could be:

$u$	1	2	3	4	5	6
$m_A(u)$	1.0	1.0	0.8	0.6	0.4	0.2

For a given frame of discernment, the possibility distribution has a qualitative difference from the probability distribution as shown in the following example [38].

If there is a statement 'Hans ate  $X$  eggs for breakfast' where  $X$  is taking values in  $\{1, 2, 3, 4, \dots\}$ , the possibility distribution expresses the degree of ease with which Hans can eat  $u$  eggs while the probability distribution expresses the probability of Hans eating  $u$  eggs. So the two distribution may look like as follows:

$u$	1	2	3	4	5	6	7	8
Poss( $u$ )	1	1	1	1	.8	.6	.4	.2
$P(u)$	.1	.8	.1	0	0	0	0	0

The main advantage of applying the possibility distribution concept in expert systems is its ability to represent linguistic variables together with quantifiers

and representing compound propositions by applying the following rules:

Modifier Rules. These rules define the impact of using modifiers, such as VERY, MORE OR LESS, and NOT, on the possibility distribution of a linguistic variable. Zadeh suggested squaring for VERY, square root for MORE OR LESS, and subtracting for NOT. However, more analysis could be done to determine appropriate functions for these modifiers in the case where Zadeh's suggested functions are not consistent.

Composition Rules. Composition rules are used in the case of compounding two or more propositions. The most commonly used composition modes are logical AND, logical OR, and logical IMPLICATION.

Truth Qualification Rules. These rules are analogous to the modifier rules in the sense that they define the possibility distribution of a fuzzy term quantified by a modifier. The modifier in this case is a truth modifier such as very true, more or less true, or quite true. A proposition "It is  $t$  that  $X$  is  $A$ ", where  $t$  is the truth quantifier of the proposition, can be expressed as ' $X$  is  $A$  is  $t$  --->  $\exists_X = A+$ '

$$\text{and } m_{A+}(u) = m_t(m_A(u))$$

Inference in Possibility Theory. Inference in crisp, first order, logic is usually performed by applying the

modus ponens rule, which states that if hypotheses 'A  $\rightarrow$  B' and A are true, then B is true. Modus ponens is generalized from this definition [37] to be applicable in fuzzy logic. Generalized modus ponens differs from the original version of modus ponens in two aspects which are:

- 1- Matching is not required to be exact.
- 2- Predicates are not required to be exact.

Weaknesses of Possibility Theory. The main strength of adopting possibility theory in expert systems is its power in representing the fuzziness inherent in real world problems. However, this approach received a great deal of criticism, especially from Bayesian approach advocates. Cheeseman [7] states that fuzzy logic fails as a general calculus of uncertainty because of the composition rules and he gives the following example:

$$\text{poss}(A \text{ and } B) = \min(\text{poss}(A), \text{poss}(B))$$

is true only when A and B have mutual dependence, i.e.,  $A \rightarrow B$  or  $B \rightarrow A$ . It is certainly not true in general. He [7] states that the claim of fuzzy logic advocates that vagueness is something different from uncertainty is not true. He argues that vagueness is uncertainty about intended meaning and can be represented by a probability distribution over possible meanings. He mentions several examples in his paper to support his idea. Also, in [6] Cheeseman states that the use of second order probability is considered an acceptable approach in solving



special problems for which fuzzy logic advocates claim that the fuzzy numbers approach is the most appropriate.

On the other hand, Masaharu Mizumoto, Satoru Fukami, and Kokichi Tanaka [25] mention in their paper that the proposed methods by L.A. Zadeh and E.H. Mamdani for fuzzy reasoning don't always fit our intuition. In this paper, the authors state that the reason for this drawback comes from the way in which Zadeh and Mamdani defined a fuzzy relation between an antecedent and a conclusion of a rule. Zadeh's definition of that relation is:

$$R = (A \times B) \cup (A' \times V),$$

$$\text{or } R = (A' \times V) \cap (U \times B),$$

where  $A$ ,  $A'$ ,  $B$ , and  $B'$  are fuzzy sets in the universes of discourse  $U$ ,  $U$ ,  $V$ , and  $V$  respectively. Mamdani defined  $R$  as follows:

$$R = A \times B.$$

The authors [25] give some examples of Zadeh and Mamdani's methods which do not give correct results. In the following examples, a conclusion is drawn from the following rule and fact:

If  $X$  is  $A$  then  $Y$  is  $B$ .

$X$  is  $A$ .

Relation	Antecedent	Conclusion
1	$X$ is $A$	$Y$ is $B$
2	$X$ is very $A$	$Y$ is very $B$
3	$X$ is very $A$	$Y$ is $B$
4	$X$ is more or less $A$	$Y$ is more or less $B$

5	X is not A	Y is unknown
6	X is not A	Y is not B
7	Y is not B	X is not A

The authors prove that Zadeh's methods do not satisfy the relations except relation 5, and Mamdani's method does not satisfy the relations except relations 1 and 3.

### Training Algorithms for Neural Networks

#### Perceptron Convergence Procedure

F. Rosenblatt [30] introduces a single layer perceptron learning algorithm which can accept continuous or binary input. This network model had an initial success in simple pattern recognition problems. A single node in the network computes a weighted sum of the input elements, subtracts a threshold ( $\theta$ ), and passes the result through a hard limit function, (Figure 4, p. 6), to output nodes. Rosenblatt develops an algorithm to train such networks in which weights of the connections and thresholds are initialized to small random values. A new input is applied to the network and the output is computed using a hard limit function; then it will be compared with the desired output. If there is some difference between the actual output and the desired one, connections weights will be adjusted, as explained in more detail in the next chapter. Later, Minsky and Papert [27] prove that the elementary perceptron

model has a limitation in being unable to represent some boolean functions such as EXCLUSIVE-OR. Also, they find that such type of training only works well with a separable training set; i.e., one for which hyperspace partitioning surface is linear.

### Back-Propagation

The back-propagation algorithm [33] is a generalization of the Least Mean Square, LMS, algorithm. It uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and obtained output. Training in back-propagation is performed in two modes as follows:

- \* **Forward Pass**, in which the activation for each node is computed through a function in the summation of input values multiplied by weights of arcs. The function which is usually used in back-propagation is the sigmoid function, as shown in Figure 4, p.6. The output of a node will be the input for another node in a subsequent layer in the network. These series of computations are performed until getting the output of the output layer.
- \* **Backward Pass**, in which the difference between the desired output and the actual output for output node  $j$  is computed and multiplied by the derivative of the nonlinear sigmoid function to obtain  $\delta$  value, the amount of change in the output from time  $t$  to time  $t+1$ .  
If node  $j$  is an output node, then

$$\delta_j = T_a(1 - T_a)(T_d - T_a),$$

where  $T_a$  is the actual output of node  $j$ ,  $T_d$  is the desired output of node  $j$ , and  $T_a(1 - T_a)$  is the derivative of the sigmoid function which is

$$T_a = 1 / (1 + e^{-X}),$$

where  $X$  is the net value of the summation mentioned above. If node  $j$  is a hidden layer node, then

$$\delta_j = T_a(1 - T_a) \sum \delta_k w_{jk},$$

where  $k$  ranges over all nodes in the layer above node  $j$ , and  $w_{jk}$  is the weight of the arc which connects  $j$  to a node in layer  $k$ . Thus, the change of weight which is required to minimize the error will be calculated as follows:

$$w_{ij}(n+1) = w_{ij}(n) + \mu \delta_j T_i + \alpha (w_{ij}(n) - w_{ij}(n - 1)),$$

where:

$w_{ij}(n+1)$  is the weight at time  $n+1$  from neuron  $i$  in the hidden layer to neuron  $j$  in the output layer.

$w_{ij}(n)$  is the weight at time  $n$ .

$\mu$  is the learning rate whose value affects the speed of training.  $\mu$  must neither be too small nor too large. If it is too small, it slows convergence. If it is too large, the weight changes will overshoot the current solutions and convergence will not occur.

$\alpha$  is the momentum factor which is used to improve the training time.  $\alpha$  is multiplied by the amount of the previous weight change. Using the previous change is a way of remembering the behavior of the adjusting

procedure over time. The effect of choosing the values of  $\mu$  and  $\alpha$  will be discussed shortly.

$T_i$  is the output of neuron  $i$ .

The network which uses this kind of learning has one or more intermediate, or hidden, layers to compute functions that cannot be computed properly with only two layers. The input could be continuous or binary. The desired output of all nodes is typically low,  $0 < \text{low} < 0.1$ , unless the node corresponds to the class the current input is from, in which it will be high,  $0.9 < \text{high} < 1.0$ . Weights in the network are set initially to small random numbers and training cases are applied to the network in consecutive iterations. In each iteration, weights are adjusted by a value proportioned to the difference between the desired output and actual output until the cost function is minimized to an acceptable value such as 0.1 or .05.

The number of hidden nodes determines the complexity of the partitioning surface of the hyperspace, which encompasses all the points in the training examples. Determining the number of nodes in a hidden layer must be done after studying the training examples well because if the underlying mapping is linear, the number of nodes must be small and vice versa; if the underlying mapping is highly nonlinear, the number must be increased.

The parameters in the back-propagation algorithm, learning rate  $\mu$  and momentum  $\alpha$ , affect the performance of the training procedure. Gradient descent algorithms like

back-propagation must respond to the hills and valleys of the objective function. Thus, with a high learning rate a valley or a minimum direction may be skipped; a high momentum may lead the training procedure into a local minimum if it skips a small hill which may lead to the global minimum.

Deficiencies of Back-Propagation. The back-propagation algorithm has proved to be effective in problems with small sizes. However, the algorithm has a major deficiency which is its slow rate of learning. Empirically, the learning time on a serial machine is very approximately  $O(N^3)$  [20], where  $N$  is the number of weights in the network. The time for one forward and one backward pass is  $O(N)$ . The number of training examples is typically  $O(N)$ . The number of times the weights must be updated is approximately  $O(N)$ . A second deficiency of back-propagation is its being biologically implausible because there is no evidence that a synapse can be used in the reverse direction. Also, it is possible that the algorithm may stick in a local minimum instead of a global minimum as a result of a bad choice of the learning parameters as mentioned above.

### Generalization

The main purpose of training an artificial neural network is to produce a network which generalizes correctly to new cases after training on an adequate number of typical

cases. Unfortunately, in much of the current research there is no formal method to conceptualize the meaning of generalization. Valiant [36] has introduced an approach to distinguish classes of boolean functions which can be induced from examples in polynomial time from classes that require exponential time. Valiant assumes that a hypothesis space is known in advance, and he allows the training cases to be selected according to any stationary distribution which must be used to generate the test cases.

The induced function is considered good enough if it differs from the true one on less than a small fraction,  $1/h$ , of the test cases. A class of boolean functions is considered to be learnable in polynomial time if, for any choice of  $h$ , there is a probability of at least  $(1 - 1/h)$  that the induced function is good enough after a polynomial number of training examples.

Maureen Caudill [5] mentions that the size of the middle layer in multi-layer models affects the performance of the network. If the size is too large, then the network tends to memorize the input patterns rather than generalize the input into features. On the other hand, if the size is small, it will increase the number of iterations required to train the network and will likely reduce the accuracy of recall.

## CHAPTER III

### APPROACHES TO HANDLE UNCERTAINTY IN CONNECTIONIST EXPERT SYSTEM

As mentioned in chapter 1, this study implemented two approaches to manage uncertainty in connectionist expert systems. These approaches are:

- 1- The random cell approach.
- 2- The stairstep approach.

#### The Random Cell Approach

In this approach, a variable will be represented by several nodes to express the uncertainty of that variable. The uncertainty could be the variable's degree of fuzziness or its probability of existence. Analysis of variables represented by 6 nodes was performed in this study. The maximum value that a variable can have occurs when all the nodes have true values, or 1. On the other hand, a minimum value is present when the first node is set to one and the rest of the nodes are set to false, or zero. Rosenblatt [30] introduces the perceptron training algorithm to train linear discriminants. This algorithm produces a weight vector  $L$  as follows:

- 1- Set  $L$  to the 0 vector.
- 2- Let  $L$  be the current weights. Randomly pick a training



example  $E_i$ , with corresponding classification  $C_i$ .

3a- If  $L$  correctly classifies  $E_i$ , do nothing. I.e., if

$L \cdot E_i > 0$  and  $C_i = +1$  } or

$L \cdot E_i < 0$  and  $C_i = -1$  }, then do nothing.

3b- Otherwise, form a new set of weights  $L'$  as follows:

$$L' = L + C_i E_i$$

4- Go to 2.

The perceptron learning algorithm has a limitation in dealing with non-separable training examples in which there is no set of weights that correctly classifies every example. To overcome this problem, Gallant [15] modifies the perceptron learning algorithm so that an optimal set of weights can be obtained in the case of nonseparable training examples. Gallant calls that algorithm the pocket algorithm, which refers to the process of saving the vector of weights with the longest consecutive run of correct classification trials in the perceptron learning algorithm. The pocket algorithm has the following steps:

1 - Set  $L$  to the 0 vector.

2 - Let  $L$  be the current weights. Randomly pick a training example  $E_i$  with corresponding classification  $C_i$ .

3a- If  $L$  correctly classifies  $E_i$  then:

3aa- If the current run of correct classifications with  $L$  is longer than the run of correct classifications for the weight vector in your pocket:

3aaa- Put  $L$  in your pocket and remember the length

of its correct run.

3b- Otherwise, form a new set of weights  $L'$  as follows:

$$L' = L + C_i E_i$$

4- Go to 2.

Gallant [15] shows that with an increase in the number of iterations, the probability that the pocket weights are optimal approaches 1. As expected, the main drawback for this algorithm is that there is no known limit to the number of iterations required to obtain the optimal set of weights. On the other hand, the pocket algorithm is suitable for noisy and redundant inputs, making it an ideal choice for training examples in connectionist expert systems.

Actually, a single linear discriminant is not able to compute all boolean functions of its input. However, adding an intermediate layer helps in computing such functions. For that reason, Rosenblatt [31] introduces the idea of using a layer of random functions prior to a single trainable cell as shown in Figure 7.

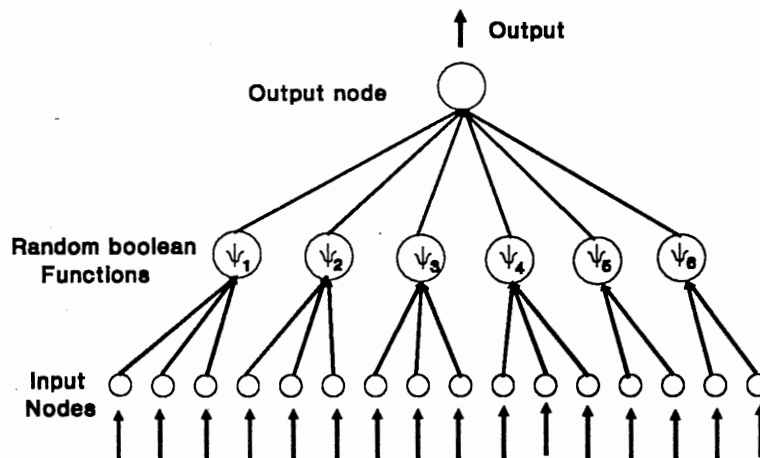


Figure 7. Rosenblatt's Idea (Random Functions)

Each random function has relatively few inputs and some of these functions will represent, by trial and error, features of interest. In the meantime, the perceptron learning algorithm will produce coefficients for the trainable cell; i.e., coefficients for the arcs which connect the intermediate layer to the output node. Minsky and Papert [27] show that some functions could never be recognized by a network if the random cells were limited as to which inputs each cell could see. To overcome this problem, Gallant [16] introduces random cells which have the following features:

- 1- Each cell in the random layer sees all inputs rather than a small subset.
- 2- A distributed representation of all crucial features in the activation patterns of all the random cells is desired rather than seeking individual cells that

recognize particular patterns.

- 3- Each cell in the random layer is a linear discriminant with fixed random coefficients rather than randomly selected boolean functions.
- 4- The learning algorithm is the pocket algorithm.

Gallant describes this modification as a distributed method in the sense that it has a sufficiently rich distributed random representation in the activations of the random discriminants, as represented in Figure 8, rather than having individual random cells compute individual features.

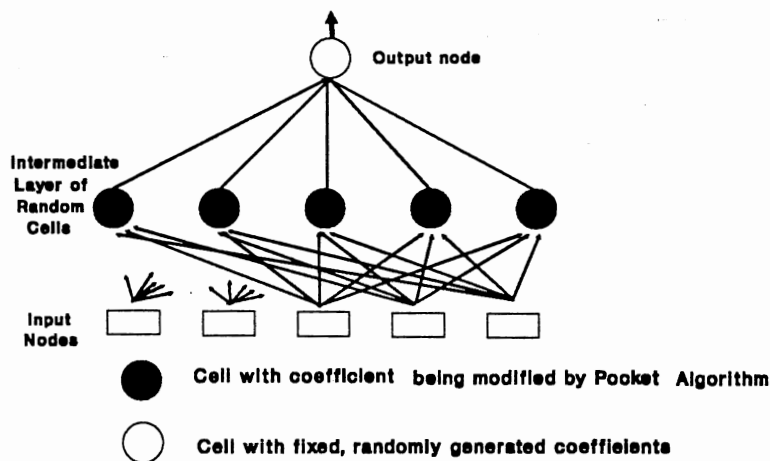


Figure 8. Distributed Method

Thus, in the first approach the input variables, each of which is represented by 6 nodes, will be connected to an intermediate layer of random cells by arcs that have randomly generated weights. In the meantime, the

intermediate layer is connected to the output layer by arcs with their weights obtained by applying the pocket weight learning procedure as it is shown in Figure 9.

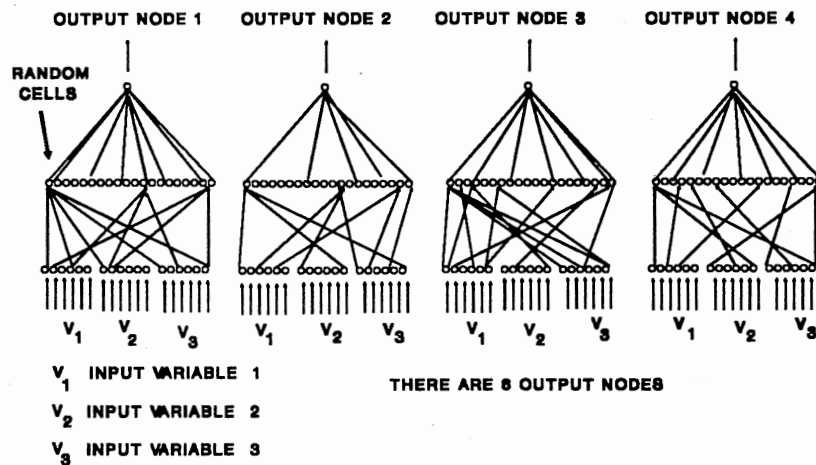


Figure 9. Random cells approach

### The Stairstep Approach

In this approach, a modification to the back-prorogation algorithm has been implemented. As mentioned in chapter 2, the output of a sigmoid function,

$$f(x) = \frac{1}{1 + e^{-x}},$$

ranges from 0 to 1 (not including the end points), in which an output value close to one is considered to be true while an output near to zero is regarded as false. A node has its internal structure as shown in Figure 10.

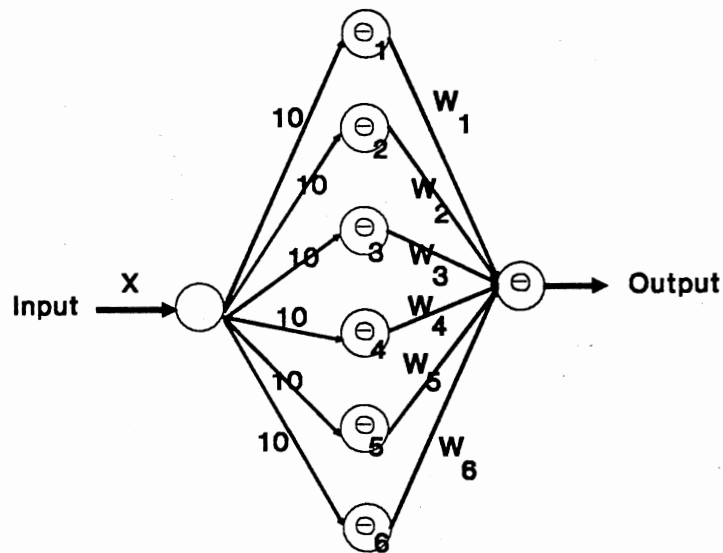


Figure 10. Internal structure of a node

The purpose of this representation is to obtain the output of any node in the network in the form of a uniform stairstep function. Each step in that output will represent a fuzzy value for the variable or its degree of severity. The weights and biases for this internal structure will be chosen without training [18] so that the stairstep function is as uniform as possible.

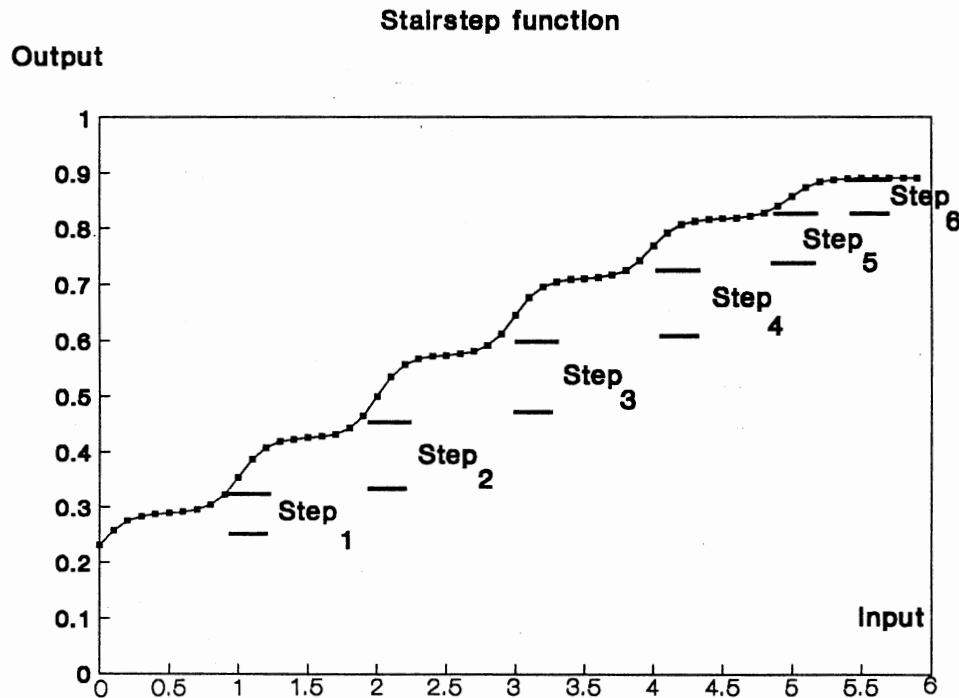


Figure 11. Output of stairstep function

As it is shown in Figure 11, the number of steps is equal to the number of internal nodes. As a consequence, we can increase or decrease the number of these internal nodes to accommodate the different kinds of problems in the real world. For the purpose of this study, 6 internal nodes are used, where step one represents a false result and steps 2 to 6 represent different truth values for a variable. As a result of adopting this representation, the output function will be modified to become:

$$f(x) = \frac{1}{1 + e^{-\left(\frac{w_1}{1 + e^{-10x + \theta_1}} + \frac{w_2}{1 + e^{-10x + \theta_2}} + \dots - \theta\right)}} \quad .(1)$$

Where:

$x$  is the input value for that structure of nodes.

$w_i$  is the weight of the arc which connects intermediate node  $i$  to the output node of the structure as it is shown in Figure 3.1.

$\theta_i$  is the bias of intermediate node  $i$ .

$\theta$  is the bias of the output node.

To obtain the uniform staircase function, which is represented in Figure 10, the following values are assigned to the above parameters:

$$\theta_1 = 0, \theta_2 = 10, \theta_3 = 20, \theta_4 = 30, \theta_5 = 40, \text{ and } \theta = 1.5.$$

$$w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = .6.$$

The modification of the output function of a node requires the derivative function, which is used in applying the delta rule as it is mentioned in chapter 2, to be modified to handle this sort of internal structure of a node.

To simplify the relation for this derivative let

$$Z = e^{-\left(\frac{w_1}{1 + e^{-10x + \theta_1}} + \frac{w_2}{1 + e^{-10x + \theta_2}} + \dots - \theta\right)}$$

$$Y_i = \frac{d}{dx} \left( \frac{w_i}{1 + e^{-10x + \theta_i}} \right) = \frac{10 * w_i * e^{-10x + \theta_i}}{(1 + e^{-10x + \theta_i})^2}$$

Thus:

$$\frac{d}{dx} f(x) = \frac{Z}{(1 + Z)^2} (Y_1 + Y_2 + Y_3 + Y_4 + Y_5 + Y_6). \quad (2)$$

A node with the internal structure that is shown in



Figure 10 will receive an input value from other nodes connected to it in case it is an intermediate or an output node. If the node is an input node, then it is set externally. A network node will produce an output which will be a one-digit value ranging from 1 to 6. This value will express the degree of fuzziness of that variable or the probability of its existence, depending on the interpretation of each step value in the output function.

To simplify the training process for that type of neural network, each step will have a corresponding network. In other words, step number  $n$  will have a network that classifies its inputs into  $n$  or  $n+1$ . The training examples for each net will be chosen as a subset of the total number of examples so that the training examples for net number  $n$  will be all the training examples that have desired outputs equal or greater than  $n$ . The output of any example in the mentioned subset that has desired output greater than  $n$  will be viewed as  $n+1$ .

In order to use these trained nets, an input will be presented to the network of step one and if the output is 2, the input will be presented to the network of step 2, and promotion to upper step nets will be continued as long as the output of each step's net is greater than the step number. This process will end if the output is equal to the step number or if we reach to the highest step, which equals six in the model.

### Training Algorithm of the Second Approach

The algorithm for training the different nets in a connectionist expert system is as follows:

- 1- Set step = 1.
- 2- for( $i=0; i < \text{number\_of\_examples}; ++i$ )
  - if( $\text{desired\_output}[i] < \text{step}$ )
    - discard that example from the training set.
- 2- set  $\text{number\_of\_curr\_examples} = \text{number\_of\_examples}$  with
  - desired output  $\geq \text{step}$ .
- 3- for( $i=0; i < \text{number\_of\_curr\_examples}; ++i$ )
  - if( $\text{desired\_output}[i] > \text{step} + 1$ )
    - $\text{desired\_output}[i] = \text{step} + 1$ ;
- 4- Forward Pass. Choose randomly one example and compute its actual output,  $T_a$ , by applying (1).
- 5- Backward Pass. If the actual output of the chosen example is not within the range of the boundaries of its desired step, then compute the error difference which equals  $T_d - T_a$ , where  $T_d$  is the desired output that equals the average value for the boundaries of the desired step. A recursive algorithm will be followed to adapt the weights of the network starting from the output node and working back to the first hidden layer. Adjust the weights by

$$w_{ij}(t+1) = w_{ij}(t) + \mu \sigma_j T_i + \alpha (w_{ij}(t) - w_{ij}(t-1)),$$

where:

$w_{ij}(t)$  is the weight from node  $i$  to node  $j$  at

time  $t$

$w_{ij}(t+1)$  is the weight at time  $t+1$ .

$\mu$  is the learning rate, see chapter 2.

$\sigma$  is the difference between the desired output and the actual one multiplied by a derivative calculated in equation (2).

if node is an output node, then

$$\sigma = \text{div}_j * (T_d - T_a),$$

otherwise,

$\sigma = \text{div}_j * \sum \sigma_k w_{jk}$ , where  $k$  ranges over all nodes in the layer  $k$  which is above node  $j$ .

$\alpha$  is the momentum factor, which is described in chapter 2.

$T_i$  is the output of node  $i$

6- Repeat by going to step 4 until obtaining the set of weights that correctly classifies most training examples for that step.

7- Set  $\text{step} = \text{step} + 1$

8- If  $\text{step} \leq 5$ , then go to 2, else stop.

## CHAPTER IV

### SIMULATION AND RESULTS

A simulation for the connectionist expert system has been developed to study the performance of the two approaches. The performance was measured in terms of the capability of the two approaches in learning input training examples and their robustness in predicting output results upon input data that have some difference from the trained ones. The model used in this study contains one input layer, which contains nodes that receive the input data of the system, two intermediate layers, and one output layer. As described in chapter 1, there is a hierarchical order of indexes of the nodes such that a node will have an index higher than the indexes of the nodes on which it depends as shown in Figure 12.

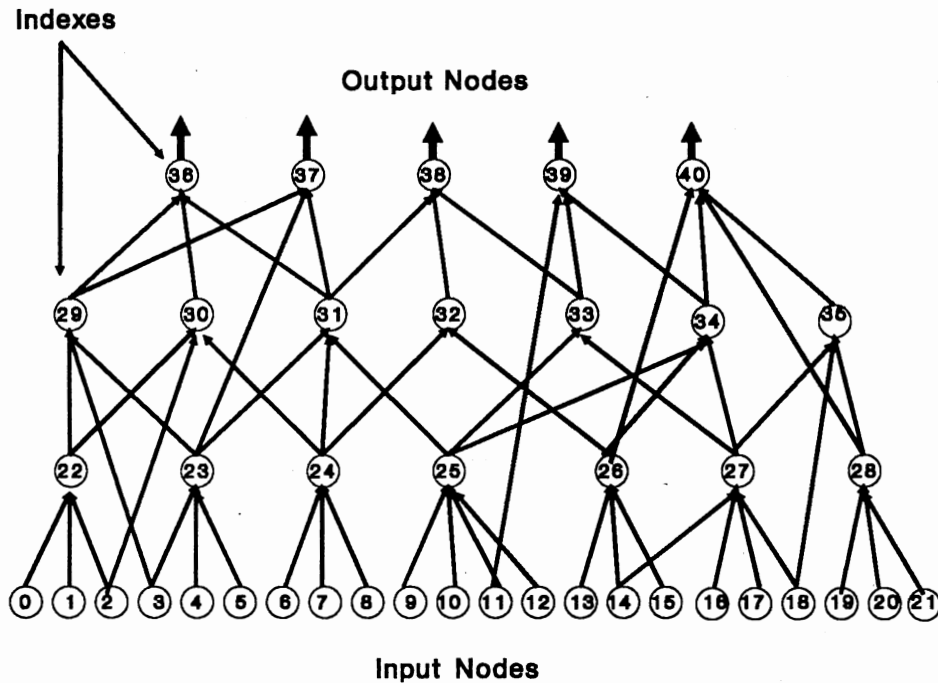


Figure 12. Connectionist model

This model could be regarded as a diagnostic expert system as follows.

- 1- The input layer receives the defects observed.
- 2- Each node in the first intermediate layer expresses the cause of defects.
- 3- A node in the third layer expresses the required treatment for the nodes connected to it from the previous layers.
- 4- The output layer describes extra treatments, which depend on the desired treatment in the third layer and, in some situations, on additional information about the kinds of defects or causes of these defects.

As mentioned in chapter 1, the training used in this kind of connectionism is easy learning, in which each intermediate or output node is trained separately. As it is shown in Figure 12, the number of input arcs for any output or intermediate node varies from 2 to 4 to judge the performance under different conditions of topologies.

A hypothetical space describing a hypothetical problem domain was constructed for every node in the network so that the training examples as well as the test data have the same source of judgement. To create this hypothetical space, a set of rules was used to compute the outputs of a node according to the values of inputs. These sets of rules were chosen so that underlying relationships among the input variables are nonlinear or nonuniform to simulate the situations in the real world where the data or point of views may come from several sources that may result in contradictions. A sample of these rules is as follows:

```
if(1 < x < 3) and (4 < y < 6) and (1 < z < 4))
  then output(u) = round((x * .2) + (y * .1) + (z * .5))
```

where

x, y, and z are the output of nodes x, y, and z respectively, which are connected to node u.

Appendix A has a complete set of rules used to develop the hypothetical spaces for all the nodes in the network.

Analysis of the performance is as follows:

1- Each intermediate or output node is trained

separately.

2- There are five training sets for each node. These sets are 10%, 20%, 30%, 40%, and 50% of the total hypothetical space. This means that if a node has 4 inputs, then the total hypothetical space is 1296 cases and the training sets are 130, 259, 390, 518, and 648 examples respectively.

3- For each training set, the number of cases that are correctly classified after the training is calculated to indicate the relationship between the increasing number of learning examples and the capability of the network in predicting unseen cases. Also, a comparison between the two approaches was performed to find out which one has a better generalization property in terms of the number of correctly classified cases and the standard deviations of these classifications.

4- The performance of the overall network was studied by applying several inputs to the input layer and performing the same sort of analysis as in step 3 on the nodes in the output layer. This analysis was carried on five times for each approach so that the first time has all the nodes trained on 10% of their hypothetical space and the second one has 20% and so on until 50%.

In the random cells approach, a limit of 1000 iterations was implemented to obtain the optimal set of weights. This 1000 iterations were repeated 100 times with different initial random weights in each time. On the

other hand, 10000 iterations were used in the stairstep approach. The learning rate for the stairstep approach is chosen to be 0.5 while the momentum factor is taken as 0.1. The following tables represent the results obtained for training the 19 nodes. The nodes have indexes starting with 22 to 40, as it is shown in Figure 12. The first five tables have the results of training each node separately. Tables number 6 to 10 represent the performance of the output nodes, which have indexes from 36 to 40, upon presenting 10000 unique test cases to the network. A complete set of charts that represent the results in the tables is provided in Appendix B.



TABLE 1.

RESULTS WITH TRAINING 10% OF THE  
HYPOTHETICAL SPACE OF EACH NODE

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
22	144	66	.600	6	112	52	1.0	12
23	145	67	.597	6	104	48	.976	12
24	157	72	.697	6	124	57	.857	12
25	1100	85	.388	17	957	74	1.26	65
26	149	69	.616	6	112	52	1.0	12
27	905	70	.601	17	774	60	1.62	65
28	151	70	.649	6	88	41	1.31	12
29	130	60	.713	6	111	51	.799	12
30	136	63	1.25	6	128	59	.969	12
31	169	78	.466	6	124	57	.757	12
32	23	64	1.04	2	17	47	.360	2
33	22	61	.799	2	18	50	.616	2
34	141	65	.716	6	104	48	.986	12
35	151	70	.656	6	124	57	.745	12
36	173	80	.461	6	124	57	.833	12
37	140	65	.940	6	110	51	1.13	12
38	136	63	.631	6	99	46	.962	12
39	142	65	.683	6	104	48	.972	12
40	898	69	.758	17	759	59	2.03	12

TABLE 2.

RESULTS WITH TRAINING 20% OF THE  
HYPOTHETICAL SPACE OF EACH NODE

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
22	151	70	.63	6	123	57	.940	22
23	165	76	.486	6	124	57	.868	22
24	170	79	.565	6	140	65	.822	22
25	1146	88	.340	22	1026	79	1.12	22
26	173	80	.446	6	129	60	1.02	22
27	939	72	.542	22	919	70	1.43	100
28	153	71	.690	6	117	54	.874	22
29	163	75	.495	6	135	63	.656	100
30	167	77	1.12	6	147	68	.716	22
31	171	79	.400	6	144	67	.634	22
32	29	81	.726	4	26	72	.471	3
33	22	61	.799	2	17	47	.589	3
34	163	75	.495	6	135	63	.751	22
35	157	73	.535	6	131	61	.760	22
36	179	83	.430	6	134	62	.739	22
37	168	78	.561	6	133	62	.964	22
38	139	64	.641	6	149	69	.656	22
39	153	71	.573	6	132	61	.742	22
40	937	72	.665	22	820	63	1.90	100

TABLE 3.

RESULTS WITH TRAINING 30% OF THE  
HYPOTHETICAL SPACE OF EACH NODE

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
22	176	81	.461	6	144	67	.713	32
23	169	78	.466	10	127	59	.853	32
24	180	83	.408	12	156	72	.745	32
25	1157	89	.327	22	1114	86	1.19	150
26	175	81	.481	6	137	63	.822	32
27	989	76	.500	22	1016	78	1.21	150
28	162	75	.627	10	131	61	1.04	32
29	175	81	.451	10	148	69	.642	32
30	175	81	.638	10	165	76	.656	32
31	187	87	.385	10	163	75	.548	32
32	31	86	.471	4	28	78	.304	5
33	24	67	.649	2	16	44	.533	6
34	184	85	.353	12	140	65	.739	32
35	170	79	.476	12	150	70	.723	32
36	182	84	.513	12	160	74	.604	32
37	177	82	.556	12	145	76	.855	32
38	166	77	.495	14	153	71	.585	32
39	169	78	.419	12	167	77	.49	32
40	977	75	.693	22	988	76	1.41	150

TABLE 4.

RESULTS WITH TRAINING 40% OF THE  
HYPOTHETICAL SPACE OF EACH NODE

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
22	178	82	.451	8	149	69	.656	42
23	177	82	.420	14	152	70	.632	42
24	182	84	.390	14	164	76	.649	42
25	1170	90	.312	25	1162	89	1.65	200
26	182	84	.430	16	155	72	.687	42
27	1001	77	.495	31	1087	84	1.03	200
28	177	82	.471	20	142	66	.868	42
29	182	84	.397	20	169	78	.495	42
30	178	82	.54	14	164	76	.680	42
31	182	84	.396	12	170	79	.504	42
32	33	92	.288	4	27	75	.204	6
33	28	78	.471	2	20	56	.446	9
34	190	88	.346	14	149	70	.742	42
35	182	84	.397	16	162	75	.604	42
36	186	86	.451	14	162	75	.707	42
37	180	83	.544	14	148	69	.703	42
38	170	78	.490	14	169	78	.504	42
39	188	87	.360	14	160	74	.522	42
40	1000	77	.583	37	1058	82	1.23	200

TABLE 5.

RESULTS WITH TRAINING 50% OF THE  
HYPOTHETICAL SPACE OF EACH NODE

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
22	180	83	.430	10	169	78	.581	52
23	182	84	.397	16	170	79	.577	52
24	184	85	.369	14	173	80	.540	52
25	1179	91	.380	31	1216	94	1.77	250
26	188	87	.397	18	174	81	.605	52
27	1018	79	.432	37	1116	86	.928	250
28	181	84	.451	22	163	75	.597	52
29	186	86	.376	22	170	79	.490	52
30	188	87	.360	16	176	81	.589	52
31	185	86	.376	14	186	86	.372	52
32	33	92	.408	4	32	89	.316	7
33	30	83	.408	2	26	72	.288	9
34	194	90	.321	16	172	80	.481	52
35	185	86	.378	16	166	77	.600	52
36	189	88	.432	14	179	83	.440	52
37	183	85	.446	14	161	75	.726	52
38	174	81	.467	16	182	84	.456	52
39	195	90	.310	14	177	82	.546	52
40	1020	79	.972	37	1123	87	1.11	250

TABLE 6.

RESULTS OF OUTPUT NODES AFTER TESTING THE NETWORK BY  
10000 TEST CASES (EVERY NODE IS TRAINED ON 10%  
OF ITS HYPOTHETICAL SPACE)

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
36	7608	76	.489	6	5698	57	.902	12
37	7732	77	.476	6	3170	32	.879	12
38	7726	77	.476	6	1520	15	1.56	12
39	6768	68	.569	6	6205	62	.875	12
40	7620	76	.488	17	7062	71	.553	65

TABLE 7.

RESULTS OF OUTPUT NODES AFTER TESTING THE NETWORK BY  
10000 TEST CASES (EVERY NODE IS TRAINED ON 20%  
OF ITS HYPOTHETICAL SPACE)

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
36	8048	80	.408	6	6234	62	.759	22
37	8291	83	.413	6	7431	74	.513	22
38	8111	81	.501	6	4387	44	.789	22
39	8403	84	.447	6	7282	73	.613	22
40	7946	79	.453	22	7562	76	.497	100

TABLE 8.

RESULTS OF OUTPUT NODES AFTER TESTING THE NETWORK BY  
10000 TEST CASES (EVERY NODE IS TRAINED ON 30%  
OF ITS HYPOTHETICAL SPACE)

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
36	8336	83	.408	12	6783	67	.788	32
37	9086	91	.304	12	7538	75	.498	32
38	9685	97	.177	14	5153	52	.769	32
39	8362	84	.421	12	7411	74	.459	32
40	8083	81	.437	31	7931	80	.524	150

TABLE 9.

RESULTS OF OUTPUT NODES AFTER TESTING THE NETWORK BY  
10000 TEST CASES (EVERY NODE IS TRAINED ON 40%  
OF ITS HYPOTHETICAL SPACE)

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
36	8556	86	.380	14	7338	73	.701	42
37	9244	92	.276	14	7763	78	.523	42
38	9722	97	.166	14	6601	66	.745	42
39	8531	85	.401	14	7803	78	.432	42
40	8183	82	.398	37	8262	82	.346	200

TABLE 10.

RESULTS OF OUTPUT NODES AFTER TESTING THE NETWORK BY  
10000 TEST CASES (EVERY NODE IS TRAINED ON 50%  
OF ITS HYPOTHETICAL SPACE)

Node	Stairstep Approach				Random Cell Approach			
	# of correctly classified cases	%	st.d	# of nodes in mid layer	# of correctly classified cases	%	st.d	# of nodes in mid layer
36	9706	97	.176	14	9600	96	.200	52
37	9461	95	.231	14	8921	89	.398	52
38	9744	97	.154	16	9810	98	.138	52
39	8891	89	.321	14	8673	87	.324	52
40	8268	83	.416	37	8418	84	.400	250

The experimental results give the stairstep approach an obvious edge over the random cell method in terms of generalization ability, which is the percentage of correctly classified test cases, and accuracy, which is better with smaller standard deviation. In fact, nothing comes without a price; the stairstep approach is a highly time consuming algorithm compared with the training time required for the random cell approach such that a node with 4 inputs may require 10 hours to train 648 examples while this time shrinks to 5 minutes in the random cell method.

One of the interesting properties that has been found with these experiments is that there is a distinct



difference in performance between the two methods. The considerable difference occurs when the number of training examples is a small subset of the hypothetical space of the input cases, which may be 10% to 40% of that space, while the difference becomes smaller, or negligible, with increasing the number of training examples. The comparison of performance for each method is better recognized by listing out the advantages and disadvantages of each approach as it is shown in the following table

TABLE 11.  
COMPARISON BETWEEN RANDOM CELL APPROACH  
AND STAIRSTEP APPROACH

	Stairstep method	Random cell method
Advantages	<ul style="list-style-type: none"> <li>* Has a better performance than the random cells method in terms of generalization and accuracy when the number of training examples is small, from 10% to 40% of hypothetical space.</li> <li>* Requires less storage for the sets of weights, which are obtained by training</li> </ul>	<ul style="list-style-type: none"> <li>* Much faster than stairstep approach</li> <li>* Has nearly equal or even higher, performance when the number of training examples increases to 50% of the hypothetical space.</li> </ul>
Disadvantages	<ul style="list-style-type: none"> <li>* Training process is very slow compared with random cell method.</li> </ul>	<ul style="list-style-type: none"> <li>* Requires a larger storage than stairstep method as a result of a large number of random cells required in case of nonlinear training examples</li> <li>* Modest performance with small number of training example compared with the stairstep approach</li> </ul>

## CHAPTER V

### SUMMARY AND CONCLUSION

The connectionist expert system model, introduced by Gallant has a limitation of handling the system variables on the basis of true or false values. To handle this limitation, two approaches are proposed to permit a variable to have several truth values such as low, medium, or high. In the first approach, a variable is represented by several nodes instead of one node. In this study, 6 nodes represent any intermediate or output variable in the system where the lowest value that a variable can have is 1 0 0 0 0 0 and the highest value is 1 1 1 1 1 1. The lowest value could be regarded as the false value or the lowest value of the probability or the certainty factor of the variable. The training algorithm implemented for this approach is the pocket algorithm, which is a modification to the perceptron learning procedure. A middle layer of random cells, which is proposed by Gallant to enhance the learning capability of a node with nonlinear training examples, is implemented in this approach.

While integer values are used to represent a variable in the first approach, floating point numbers are used in the second approach. In the second approach, a modification

to the back-propagation algorithm has been introduced to obtain a node output in the range of 1 to 6 instead of 0 to 1. The idea in this modification is introducing an internal structure to a node so that the output function will take a stairstep form. Each step in the output function represents a truth value for the variable. The number of steps is equal to the number of internal nodes so that we can increase the degrees of truth of a variable by increasing the number of these internal nodes. This modification requires the modification of the output function and the derivation calculation, which is used in applying the delta rule.

A simulation for a connectionist expert system is developed in this study to compare the performance of the two proposed approaches. To compute the output of a node, a set of rules is arranged so that the training examples and the test cases have the same source of judgement to measure the degree of generalization and accuracy of each method. Each node in the system is trained using five sets of examples equal to 10%, 20%, 30%, 40%, and 50% respectively of the hypothetical space of the cases for the node. A comparison of the performances between the two proposed approaches are carried out for each node upon training on the five training sets mentioned above. Also, 10000 test cases are presented to the system and the performances of the output nodes are compared between the two methods.

The experimental results show the following observations:

- 1- The random cell approach is much faster in the training process.
- 2- The stairstep approach has a better performance when the number of training examples is small relative to the total space of hypothetical cases.
- 3- The difference between the performance of the two approaches gets smaller with increasing the number of training examples.
- 4- The random cell approach requires more storage due to a large number of random cells used in the middle layer, especially when the underlying relationship among input variables is nonlinear and the number of training examples is relatively large.

## BIBLIOGRAPHY

1. Adams, J.B. A probability model of medical reasoning and the MYCIN model. Mathematical Biosciences, 32, (1976),177-186.
2. Bhatnagar, Raj K., and Kanal, Laveen N. Handling Uncertain information: A Review Of Numeric and Non\_Numeric methods. in Uncertainty in Artificial Intelligence, Kanal, L.N., and J.F. Lemmer, North\_Holland, 1986.
3. Buchanan, Bruce G., and Edward H. Shortliffe. Rule\_Based Expert Systems: The Mycin Experiments Of The Stanford Heuristic Programming Project (eds.). Addison\_Wesley, 1984.
4. Carnap, R. The Two Concepts Of Probability In Logical Foundations Of Probability. University of chicago press, 1950.
5. Caudill, Maureen. Neural Networks PRIMER Part III. AI EXPERT, 2, (Jun 1988), 53-59.
6. Cheeseman, Peter. In Defence of Probability. Proceedings of ninth International Joint Conf On Artificial Intelligence, Los Angeles, (1985), 1002-1009
7. Cheesman, Peter. Probability versus fuzzy reasoning. in Uncertainty in Artificial Intelligence, Kanal, L.N., and J.F. Lemmer, North\_Holland, 1986.
8. Chandrasekaran, B., and Tanner, Michael C. Uncertainty Handling in Expert Systems: Uniform vs. Task\_Specific Formalism. in Uncertainty in Artificial Intelligence, Kanal, L.N., and J.F. Lemmer, North\_Holland, 1986.
9. Chi, Keung., and Abramson, Bruce. Uncertainty management in expert systems. IEEE Expert, 3, 4 (1990), 29-46.

10. Colonnier, M. The electron microscopic analysis of the neuronal organization of the cerebral cortex. in The Organization of the Cerebral cortex, F. O. Schmitt, F. G. Wordon, G. Adelman, and S. G. Dennis, Eds, Cambridge, MA: MIT Press, 1981.
11. Cox, R. T. Of inference and inquiry - An Essay In Inductive Logic. Ed, Lovine and Tribus, M.I.T. Press, 1979.
12. Dempster, A.P. Upper and lower probabilities induced by a multivalued mapping. Annals of Mathematical Statistics, 38, (1967), 325-339.
13. Gallant, Stephen I. Automatic generation of expert system from examples. Proceedings of the 2nd international conference on Artificial intelligence Applications, IEEE press, (1985), 313-319.
14. Gallant, Stephen I., and Balachandra, R. Using automated techniques to generate an expert system for R/D project monitoring. Proceeding of International Conference on Economics and Artificial Intelligence, Paris, France, (1986), 87-92.
15. Gallant, Stephen I. Random Cells: An Idea Whose Time Has Come and Gone...And Come Again?. Proceedings of IEEE international Conference on Neural Networks, IEEE press, (1987), 21-24.
16. Gallant, Stephen I. Connectionist expert systems. Communication of the ACM, 31, 2 (Feb 1988), 152 - 169.
17. Giarratano, Joseph. Expert systems : Principles and programming, Chapter 3, Pws\_Kent, 1989.
18. Guimaraes, Gisele. Neural Network Approximation of Nonlinearities in Robotic Control. 4th Oklahoma symposium on artificial intelligence, Stillwater, ok, 1990.
19. Fisher, R. A. The use of multiple measurements in taxonomic problems. Ann. Eugenics, 7, (1936), 179-188.
20. Hinton, E.Geoffrey. Connectionist Learning Procedures. in ARTIFICIAL NEURAL NETWORKS Concept Learning, Diederich Joachim, IEEE press, 1990.
21. Horvitz, E. and Heckerman, D. The Inconsistent use of measures of certainty in artificial intelligence research. In Uncertainty In Artificial Intelligence, Kanal, L.N., and J.F. Lemmer, North\_Holland, 1986.

22. Jackson, Peter. Introduction to expert systems. Addison\_Wesley, 1990.
23. Kruschke, J. Creating Local and Distributed Bottlenecks in Hidden Layers of Backpropagation Networks. In: Touretzky, D., Hinton, G., and Sejnowski, T. (Eds.): Proc. of the connectionist models summer school 1988, Morgan Kaufman Publ., San Mateo, California, 1988.
24. Lippmann, Richard.P. An Introduction to Computing with Neural Nets. IEEE ASSP Magazine, (April 1987), 4 - 22.
25. Masaharu, M., Satory, F., and Kokichi, T. Fuzzy reasoning. in Advances In Fuzzy Set Theory And Application, 1979.
26. McCulloch, Warren S., and Pitts, Walter. A Logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5, (1943), 115 - 137.
27. Minsky, M., and Papert, S. Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, Mass, 1969.
28. Parrat, Lyman G. Probability and Experimental Errors in Science, John Wiley, 1986.
29. Pearl, Judea. On evidential reasoning in a hierarchy of hypotheses. Artificial Intelligence, 28, (1986), 9-15.
30. Rolston, David W. Principles of Artificial Intelligence and Expert systems Development, McGraw Hill, 1988.
31. Rosenblatt, F. The Perceptron, a probabilistic model for information storage and organization in the brain. Psychological Review, 62, (1958), 559-570.
32. Rosenblatt, F. (1961). Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan, Washington, D.C, 1961.
33. Rumelhart, D.E., Hinton, G.E. and Williams, R.J. Learning internal representations by back-propagation errors. Nature, 323, (1986), 533-536.
34. Shafer, G. A mathematical Theory Of Evidence. Princeton University Press, Princeton, N.J, 1976.
35. Shortliffe, E. H. Computer-Based Medical Consultations: MYCIN, New York: Elsevier, 1976.



36. Valiant, L.G. A theory of the learnable, commun, ACM, 27, (1984), 1134-1142.
37. Vogl, T.P., et al. Accelerating the Convergence of the Back Propagation Method, Biological Cybernetics, 59, (1988), 257-263.
38. Zadeh, Lotfi A. (1965). Fuzzy sets. Information and control, 8, (1965), 338-353
39. Zadeh, Lotfi A. Fuzzy sets as a basis for a theory of possibility. Fuzzy sets and systems, 1, (1978), 3\_28.
40. Zadeh, Lotfi A. The role of fuzzy logic in the management of uncertainty in expert systems. Fuzzy sets and systems, 11, (1983), 199-227.
41. Zadeh, Lotfi A. Review Of Books: A Mathematical Theory OF Evidence. The AI Magazine, 2, (Fall 1984), 81-83.
42. Zadeh, L. A. Is probability theory sufficient for dealing with uncertainty in AI : a negative view, in Uncertainty in Artificial Intelligence, Kanal, L.N., and J.F. Lemmer, North\_Holland, 1986.
43. Zadeh, Lotfi A. Fuzzy Logic, IEEE Computer, (April 1988), 83-93.

APPENDIX A  
RULES of THE NETWORK NODES

**Rules for node 22.**

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 4 and 1 =< z <= 3)
  output = round(.2 * y + .4 * z)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6)
  output = round(.4 * y + .2 * z)
* if(1 =< x <= 3 and y = 1 and 1 =< z <= 4)
  output = round(.2 * x + .5 * z)
* if(3 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.5 * x + .3 * z)
* if(1 =< x <= 4 and 1 =< y <= 4 and z = 1)
  output = round(.3 * x + .3 * y)
* if(4 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.2 * x + .4 * y)
* if(1 =< x <= 6 and 1 =< y <= 6 and 1 =< z <= 3)
  output = round(.3 * x + .2 * y + .2 * z)
* if(1 =< x <= 6 and 1 =< y <= 6 and 3 < z <= 6)
  output = round(.3 * x + .5 * y + .2 * z)

```

**Rules for node 23.**

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 3 and 1 =< z <= 4)
  output = round(.2 * y + .6 * z)
* if(x = 1 and 3 < y <= 6 and 4 < z <= 6)
  output = round(.6 * y + .1 * z)
* if(1 =< x <= 5 and y = 1 and 1 =< z <= 4)
  output = round(.2 * x + .5 * z)
* if(5 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.5 * x + .1 * z)
* if(1 =< x <= 4 and 1 =< y <= 4 and z = 1)
  output = round(.4 * x + .2 * y)
* if(4 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.1 * x + .5 * y)
* if(1 =< x <= 6 and 1 =< y <= 4 and 1 =< z <= 6)
  output = round(.2 * x + .1 * y + .4 * z)
* if(1 =< x <= 6 and 4 < y <= 6 and 1 =< z <= 6)
  output = round(.4 * x + .3 * y + .3 * z)

```

**Rules for node 24.**

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 3 and 1 =< z <= 4)
  output = round(.3 * y + .5 * z)
* if(x = 1 and 3 < y <= 6 and 4 < z <= 6)
  output = round(.5 * y + .5 * z)
* if(1 =< x <= 2 and y = 1 and 1 =< z <= 4)

```

```

output = round(.2 * x + .3 * z)
* if(2 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.1 * x + .3 * z)
* if(1 <= x <= 3 and 1 <= y <= 5 and z = 1)
  output = round(.1 * x + .4 * y)
* if(3 < x <= 6 and 5 <= y < 6 and z = 1)
  output = round(.1 * x + .3 * y)
* if(1 <= x <= 3 and 1 <= y <= 6 and 1 <= z <= 6)
  output = round(.4 * y + .2 * z)
* if(3 < x <= 6 and 1 <= y <= 6 and 1 <= z <= 6)
  output = round(.5 * y)

```

#### Rules for node 25

```

* if(x = 1 and y = 1 and z = 1 and u = 1)
  output = 1
* if(x = 1 and 1 <= y <= 4 and 1 <= y <= 3 and 1 <= u <= 3)
  output = round(.1 * y + .4 * z + .2 * u)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6 and 3 < u <= 6)
  output = round(.5 * z + .1 * u)
* if(1 <= x <= 4 and y = 1 and 1 <= z <= 4 and 1 <= u <= 3)
  output = round(.1 * x + .2 * z + .4 * u)
* if(4 < x <= 6 and y = 1 and 4 <= z <= 6 and 3 < u <= 6)
  output = round(.3 * z + .3 * u)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1 and 1 <= u <= 3)
  output = round(.1 * x + .1 * y + .3 * u)
* if(3 < x <= 6 and 4 < y <= 6 and z = 1 and 3 < u <= 6)
  output = round(.1 * x + .1 * y + .2 * u)
* if(1 <= x <= 3 and 1 <= y <= 4 and 1 <= z <= 4 and u = 1)
  output = round(.1 * x + .1 * y + .3 * z)
* if(3 < x <= 6 and 4 < y <= 6 and 4 < z <= 6 and u = 1)
  output = round(.1 * x + .1 * y + .2 * z)
* if(1 <= x <= 3 and 1 <= y <= 3 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.1 * x + .1 * y + .2 * z + .2 * u)
* if(1 <= x <= 3 and 3 < y <= 6 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.1 * x + .2 * z + .2 * u)
* if(3 < x <= 6 and 1 <= y <= 3 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.1 * y + .2 * z + .3 * u)
* if(3 < x <= 6 and 3 < y <= 6 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.25 * z + .25 * u)

```

#### Rules for node 26.

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 <= y <= 2 and 1 <= z <= 4)
  output = round(.2 * y + .3 * z)

```

```

* if(x = 1 and 2 < y <= 6 and 4 < z <= 6)
  output = round(.4 * y + .3 * z)
* if(1 < x <= 3 and y = 1 and 1 < z <= 4)
  output = round(.5 * x + .1 * z)
* if(3 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.2 * x + .4 * z)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1)
  output = round(.3 * x + .3 * y)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.4 * x + .1 * y)
* if(1 <= x <= 6 and 1 <= y <= 6 and 1 <= z <= 4)
  output = round(.3 * x + .2 * y + .1 * z)
* if(1 <= x <= 6 and 1 <= y <= 6 and 4 < z <= 6)
  output = round(.2 * x + .3 * y + .5 * z)

```

#### Rules for node 27

```

* if(x = 1 and y = 1 and z = 1 and u = 1)
  output = 1
* if(x = 1 and 1 <= y <= 4 and 1 <= y <= 3 and 1 <= u <= 3)
  output = round(.1 * y + .4 * z + .2 * u)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6 and 3 < u <= 6)
  output = round(.4 * y + .1 * z + .3 * u)
* if(1 <= x <= 4 and y = 1 and 1 <= z <= 4 and 1 <= u <= 3)
  output = round(.1 * x + .1 * z + .4 * u)
* if(4 < x <= 6 and y = 1 and 4 <= z <= 6 and 3 < u <= 6)
  output = round(.5 * x + .2 * z + .1 * u)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1 and 1 <= u <= 3)
  output = round(.1 * x + .1 * y + .3 * u)
* if(3 < x <= 6 and 4 < y <= 6 and z = 1 and 3 < u <= 6)
  output = round(.2 * x + .2 * y + .25 * u)
* if(1 <= x <= 3 and 1 <= y <= 4 and 1 <= z <= 4 and u = 1)
  output = round(.1 * x + .1 * y + .3 * z)
* if(3 < x <= 6 and 4 < y <= 6 and 4 < z <= 6 and u = 1)
  output = round(.3 * x + .1 * y + .2 * z)
* if(1 <= x <= 3 and 1 <= y <= 3 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.1 * x + .1 * y + .2 * z + .3 * u)
* if(1 <= x <= 3 and 3 < y <= 6 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.1 * x + .3 * z + .4 * u)
* if(3 < x <= 6 and 1 <= y <= 3 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.2 * x + .1 * y + .2 * z + .3 * u)
* if(3 < x <= 6 and 3 < y <= 6 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.3 * x + .25 * y + .25 * z + .2 * u)

```

#### Rules for node 28.

```

* if(x = 1 and y = 1 and z = 1)

```

```

output = 1
* if(x = 1 and 1 =< y <= 4 and 1 =< z <= 3)
  output = round(.1 * y + .6 * z)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6)
  output = round(.4 * y + .4 * z)
* if(1 < x <= 3 and y = 1 and 1 < z <= 2)
  output = round(.1 * x + .4 * z)
* if(3 < x <= 6 and y = 1 and 2 < z <= 6)
  output = round(.1 * x + .5 * z)
* if(1 =< x <= 4 and 1 =< y <= 3 and z = 1)
  output = round(.2 * x + .4 * y)
* if(4 < x <= 6 and 3 <= y < 6 and z = 1)
  output = round(.1 * x + .5 * y)
* if(1 =< x <= 6 and 1 =< y <= 6 and 1 =< z <= 6)
  output = round(.3 * x + .1 * y + .2 * z)
* if(1 =< x <= 6 and 1 =< y <= 6 and 3 < z <= 6)
  output = round(.2 * x + .3 * y + .5 * z)

```

#### Rules for node 29.

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 3 and 1 =< z <= 2)
  output = round(.4 * y + .4 * z)
* if(x = 1 and 3 < y <= 6 and 2 < z <= 6)
  output = round(.2 * y + .5 * z)
* if(1 =< x <= 5 and y = 1 and 1 < z <= 5)
  output = round(.1 * x + .4 * z)
* if(3 < x <= 6 and y = 1 and 5 < z <= 6)
  output = round(.3 * z)
* if(1 =< x <= 4 and 1 =< y <= 3 and z = 1)
  output = round(.4 * y)
* if(4 < x <= 6 and 3 <= y < 6 and z = 1)
  output = round(.5 * y)
* if(1 =< x <= 3 and 1 =< y <= 6 and 1 =< z <= 6)
  output = round(.2 * y + .4 * z)
* if(3 =< x <= 6 and 1 =< y <= 6 and 1 =< z <= 6)
  output = round(.4 * y + .1 * z)

```

#### Rules for node 30.

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 4 and 1 =< z <= 3)
  output = round(.1 * y + .4 * z)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6)
  output = round(.5 * y)
* if(1 =< x <= 3 and y = 1 and 1 < z <= 3)
  output = round(.5 * x + .3 * z)
* if(3 < x <= 6 and y = 1 and 3 < z <= 6)

```

```

    output = round(.4 * x + .6 * z)
* if(1 =< x <= 3 and 1 =< y <= 2 and z = 1)
    output = round(.4 * x + .1 * y)
* if(3 < x <= 6 and 2 <= y < 6 and z = 1)
    output = round(.6 * z)
* if(1 =< x <= 6 and 1 =< y <= 3 and 1 =< z <= 6)
    output = round(.2 * x + .4 * y)
* if(1 =< x <= 6 and 3 < y <= 6 and 1 < z <= 6)
    output = round(.5 * z)

```

#### Rules for node 31.

```

* if(x = 1 and y = 1 and z = 1)
    output = 1
* if(x = 1 and 1 =< y <= 4 and 1 =< z <= 3)
    output = round(.5 * y + .5 * z)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6)
    output = round(.6 * y + .3 * z)
* if(1 =< x <= 4 and y = 1 and 1 < z <= 3)
    output = round(.6 * z)
* if(4 < x <= 6 and y = 1 and 3 < z <= 6)
    output = round(.4 * z)
* if(1 =< x <= 3 and 1 =< y <= 2 and z = 1)
    output = round(.5 * y)
* if(3 < x <= 6 and 2 <= y < 6 and z = 1)
    output = round(.35 * y)
* if(1 =< x <= 2 and 1 =< y <= 6 and 1 =< z <= 6)
    output = round(.4 * y + .3 * z)
* if(2 < x <= 4 and 1 =< y <= 6 and 1 =< z <= 6)
    output = round(.2 * y + .3 * z)
* if(4 < x <= 6 and 1 =< y <= 6 and 1 =< z <= 6)
    output = round(.3 * y + .2 * z)

```

#### Rules for node 32.

```

* if(x = 1 and y = 1)
    output = 1
* if(x = 1 and 1 <= y <= 3)
    output = .5 * y
* if(x = 1 and 3 < y <= 6)
    output = .8 * y
* if(1 <= x <= 6 and y = 1)
    output = 1
* if(1 <= x <= 3 and 1 <= y <= 3)
    output = .3 * x + .2 * y
* if(3 < x <= 6 and 1 <= y <= 3)
    output = .3 * x
* if(1 <= x <= 3 and 3 < y <= 6)
    output = .2 * x + .4 * y
* if(3 < x <= 6 and 3 < y <= 6)

```

output = .5 \* y

**Rules for node 33.**

```
* if(x = 1 and y = 1)
  output = 1
* if(x = 1 and 1 <= y <= 2)
  output = .6 * y
* if(x = 1 and 2 < y <= 6)
  output = .4 * y
* if(1 <= x <= 6 and y = 1)
  output = .5 * x
* if(1 <= x <= 2 and 1 <= y <= 4)
  output = .25 * x + .25 * y
* if(2 < x <= 6 and 1 <= y <= 4)
  output = .2 * x + .4 * y
* if(1 <= x <= 2 and 4 < y <= 6)
  output = .1 * x + .6 * y
* if(2 < x <= 6 and 4 < y <= 6)
  output = .5 * x + .5 * y
```

**Rules for node 34.**

```
* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 <= y <= 3 and 1 <= z <= 3)
  output = round(.3 * y + .3 * z)
* if(x = 1 and 3 < y <= 6 and 3 < z <= 6)
  output = round(.4 * y + .2 * z)
* if(1 <= x <= 3 and y = 1 and 1 < z <= 3)
  output = round(.25 * x + .25 * z)
* if(3 < x <= 6 and y = 1 and 3 < z <= 6)
  output = round(.1 * x + .5 * z)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1)
  output = round(.4 * x + .3 * y)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.3 * x + .3 * y)
* if(1 <= x <= 6 and 1 <= y <= 4 and 1 <= z <= 6)
  output = round(.2 * x + .4 * y + .2 * z)
* if(1 <= x <= 6 and 4 < y <= 6 and 1 <= z <= 6)
  output = round(.2 * x + .4 * y + .4 * z)
```

**Rules for node 35.**

```
* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 <= y <= 3 and 1 <= z <= 4)
  output = round(.5 * z)
* if(x = 1 and 3 < y <= 6 and 3 < z <= 6)
  output = round(.4 * y)
```



```

* if(1 <= x <= 3 and y = 1 and 1 < z <= 4)
  output = round(.5 * x)
* if(3 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.3 * x)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1)
  output = round(.4 * x + .5 * y)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.3 * x + .7 * y)
* if(1 <= x <= 6 and 1 <= y <= 6 and 1 <= z <= 4)
  output = round(.3 * x + .4 * y)
* if(1 <= x <= 6 and 1 <= y <= 6 and 4 < z <= 6)
  output = round(.2 * x + .3 * y)

```

#### Rules for node 36.

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 <= y <= 3 and 1 <= z <= 4)
  output = round(.3 * y + .3 * z)
* if(x = 1 and 3 < y <= 6 and 4 < z <= 6)
  output = round(.1 * y + .5 * z)
* if(1 < x <= 3 and y = 1 and 1 <= z <= 4)
  output = round(.5 * x + .1 * z)
* if(3 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.3 * x + .4 * z)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1)
  output = round(.3 * x + .2 * y)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.5 * x)
* if(1 <= x <= 6 and 1 <= y <= 4 and 1 <= z <= 6)
  output = round(.2 * x + .3 * y + .2 * z)
* if(1 <= x <= 6 and 4 <= y <= 6 and 1 <= z <= 6)
  output = round(.3 * x + .1 * y + .5 * z)

```

#### Rules for node 37.

```

* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 <= y <= 3 and 1 <= z <= 3)
  output = round(.1 * y + .5 * z)
* if(x = 1 and 3 < y <= 6 and 3 < z <= 6)
  output = round(.5 * y + .2 * z)
* if(1 <= x <= 4 and y = 1 and 1 <= z <= 4)
  output = round(.2 * x + .6 * z)
* if(4 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.5 * x + .3 * z)
* if(1 <= x <= 3 and 1 <= y <= 4 and z = 1)
  output = round(.3 * x + .3 * y)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.6 * x + .1 * y)

```

```
* if(1 =< x <= 6 and 1 =< y <= 4 and 1 =< z <= 6)
  output = round(.2 * x + .2 * y + .2 * z)
* if(1 =< x <= 6 and 4 =< y <= 6 and 1 =< z <= 6)
  output = round(.3 * x + .5 * y + .2 * z)
```

#### Rules for node 38.

```
* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 4 and 1 =< z <= 3)
  output = round(.3 * y + .4 * z)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6)
  output = round(.5 * y + .3 * z)
* if(1 =< x <= 3 and y = 1 and 1 =< z <= 4)
  output = round(.5 * z)
* if(3 < x <= 6 and y = 1 and 4 < z <= 6)
  output = round(.4 * x)
* if(1 =< x <= 3 and 1 =< y <= 4 and z = 1)
  output = round(.6 * x)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.2 * x + .2 * y)
* if(1 =< x <= 3 and 1 =< y <= 6 and 1 =< z <= 6)
  output = round(.2 * x + .5 * z)
* if(3 < x <= 6 and 1 =< y <= 6 and 1 < z <= 6)
  output = round(.4 * x + .1 * z)
```

#### Rules for node 39.

```
* if(x = 1 and y = 1 and z = 1)
  output = 1
* if(x = 1 and 1 =< y <= 4 and 1 =< z <= 3)
  output = round(.2 * y + .7 * z)
* if(x = 1 and 4 < y <= 6 and 3 < z <= 6)
  output = round(.8 * y + .1 * z)
* if(1 =< x <= 3 and y = 1 and 1 =< z <= 3)
  output = round(.2 * x + .3 * z)
* if(3 < x <= 6 and y = 1 and 3 < z <= 6)
  output = round(.1 * x + .5 * z)
* if(1 =< x <= 3 and 1 =< y <= 4 and z = 1)
  output = round(.2 * x + .4 * y)
* if(3 < x <= 6 and 4 <= y < 6 and z = 1)
  output = round(.25 * x + .25 * y)
* if(1 =< x <= 4 and 1 =< y <= 6 and 1 =< z <= 6)
  output = round(.1 * x + .3 * y + .3 * z)
* if(4 < x <= 6 and 1 =< y <= 6 and 1 =< z <= 6)
  output = round(.1 * x + .3 * y + .1 * z)
```

#### Rules for node 40

```
* if(x = 1 and y = 1 and z = 1 and u = 1)
  output = 1
```

```
* if(x = 1 and 1 <= y <= 3 and 1 <= y <= 2 and 1 <= u <= 4)
  output = round(.3 * y + .1 * z + .1 * u)
* if(x = 1 and 3 < y <= 6 and 2 < z <= 6 and 4 < u <= 6)
  output = round(.2 * y + .2 * z + .3 * u)
* if(1 <= x <= 3 and y = 1 and 1 <= z <= 2 and 1 <= u <= 5)
  output = round(.2 * x + .4 * z + .1 * u)
* if(3 < x <= 6 and y = 1 and 2 <= z <= 6 and 5 < u <= 6)
  output = round(.2 * x + .1 * z + .6 * u)
* if(1 <= x <= 3 and 1 <= y <= 3 and z = 1 and 1 <= u <= 3)
  output = round(.2 * x + .4 * y + .1 * u)
* if(3 < x <= 6 and 3 < y <= 6 and z = 1 and 3 < u <= 6)
  output = round(.4 * x + .2 * y + .3 * u)
* if(1 <= x <= 3 and 1 <= y <= 2 and 1 <= z <= 4 and u = 1)
  output = round(.2 * x + .1 * y + .3 * z)
* if(3 < x <= 6 and 2 < y <= 6 and 4 < z <= 6 and u = 1)
  output = round(.1 * x + .1 * y + .6 * z)
* if(1 <= x <= 3 and 1 <= y <= 3 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.2 * x + .3 * y + .1 * z + .1 * u)
* if(1 <= x <= 3 and 3 < y <= 6 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.2 * x + .7 * z + .1 * u)
* if(3 < x <= 6 and 1 <= y <= 4 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.2 * x + .2 * y + .1 * z + .2 * u)
* if(3 < x <= 6 and 4 < y <= 6 and 1 <= z <= 6 and
  1 <= u <= 6)
  output = round(.1 * x + .4 * y + .1 * z + .4 * u)
```

APPENDIX B  
TRAINING RESULTS CHARTS

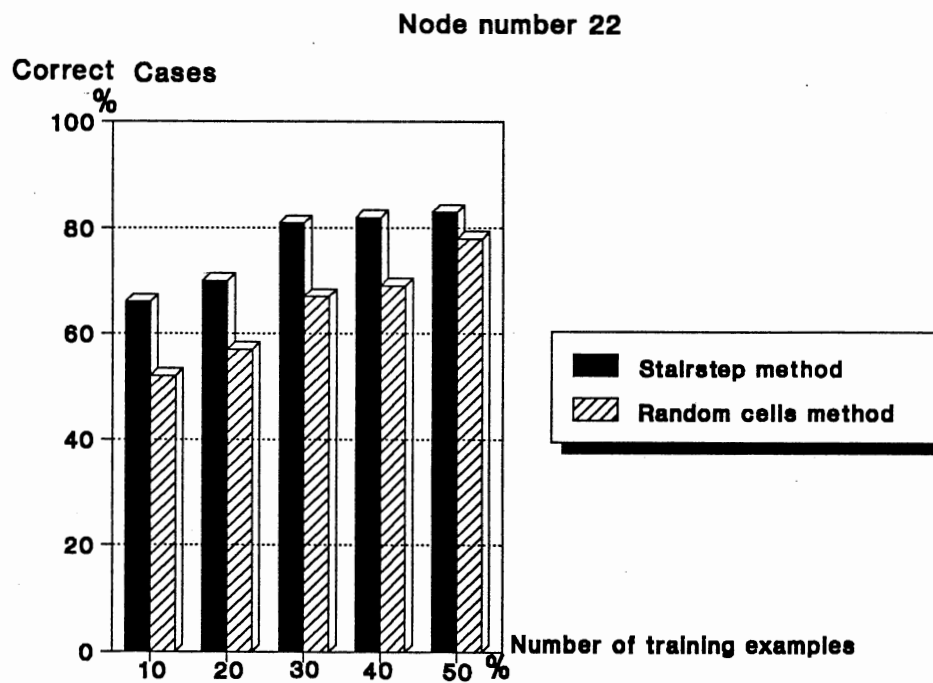


Figure 13. Results of Training Node 22

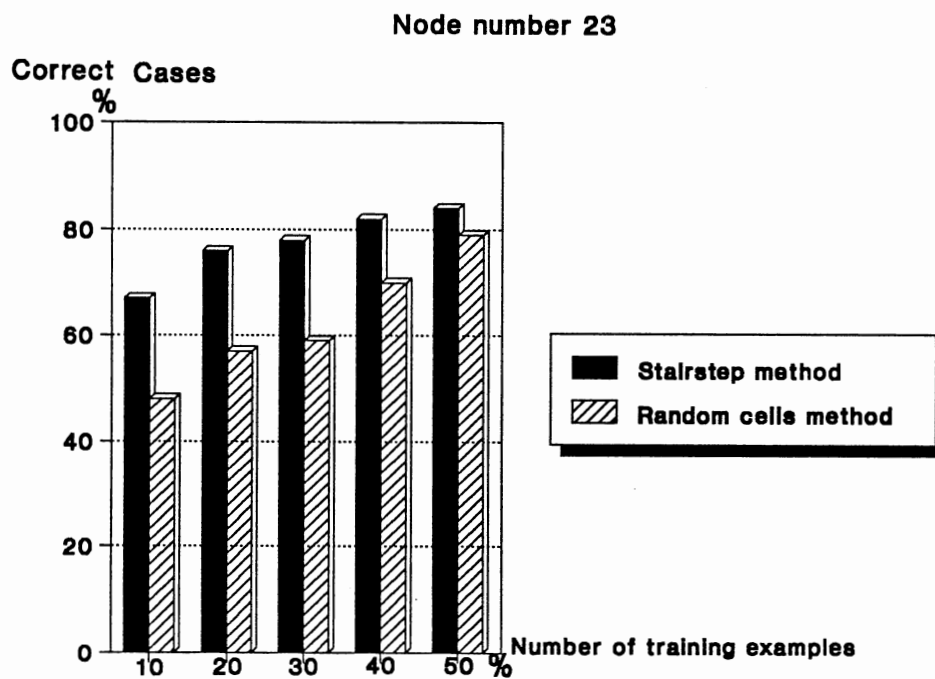


Figure 14. Results of Training Node 23

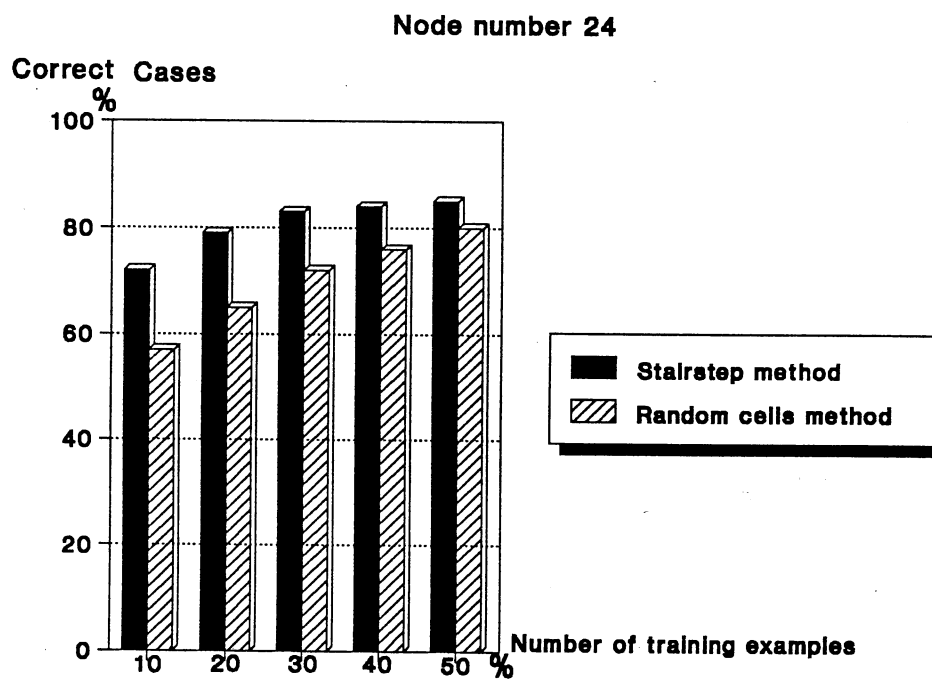


Figure 15. Results of Training Node 24

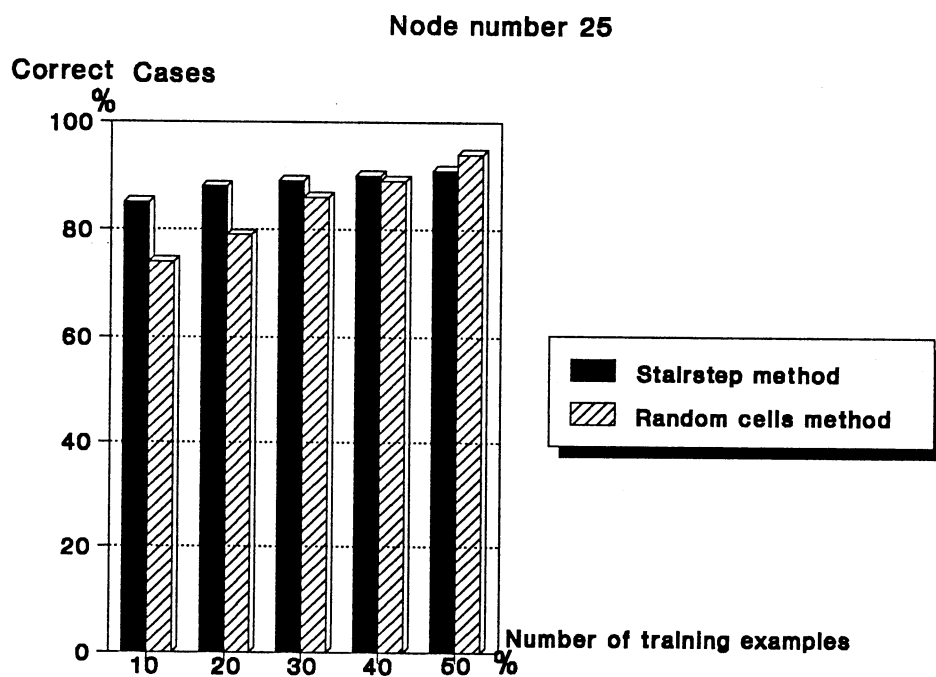


Figure 16. Results of Training Node 25

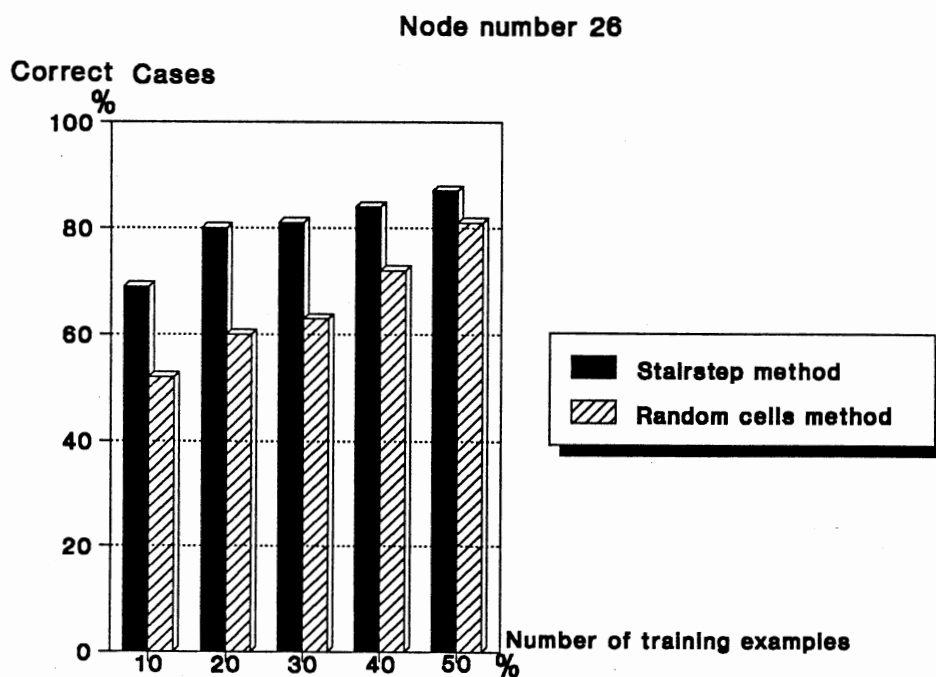


Figure 17. Results of Training Node 26

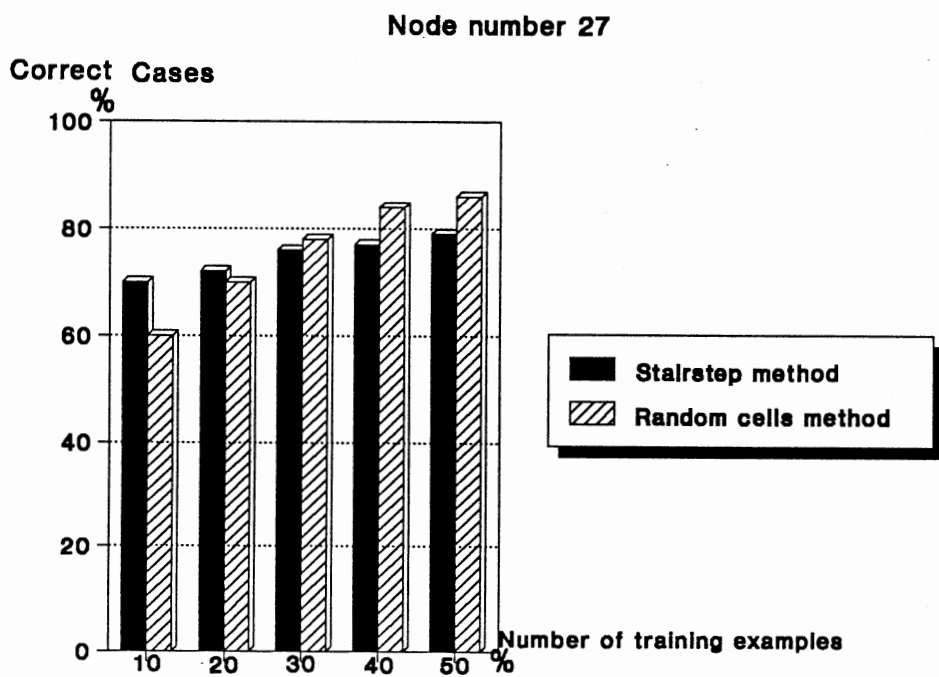


Figure 18. Results of Training Node 27

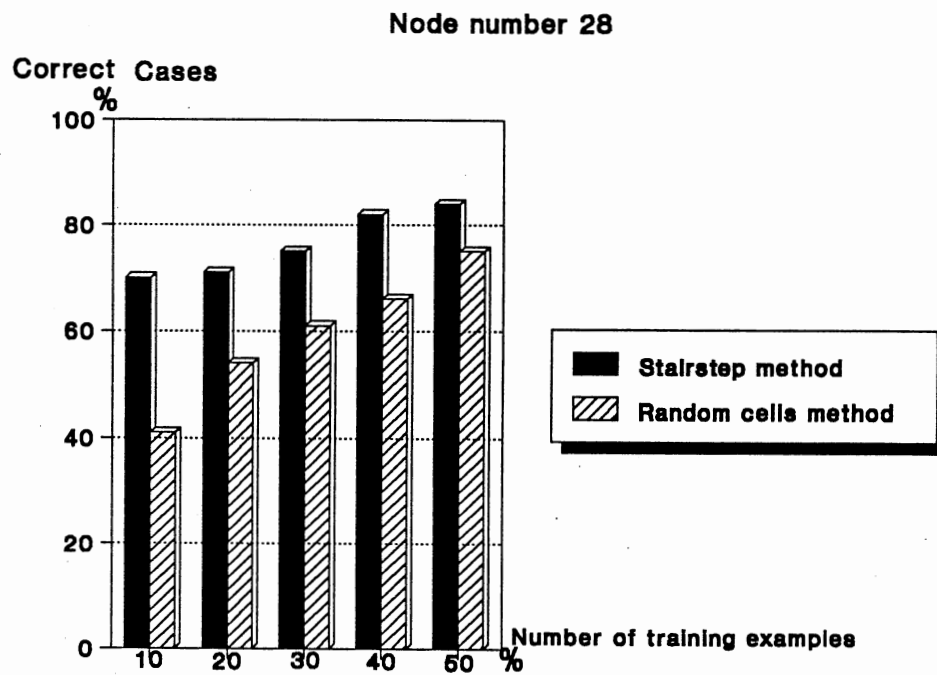


Figure 19. Results of Training Node 28

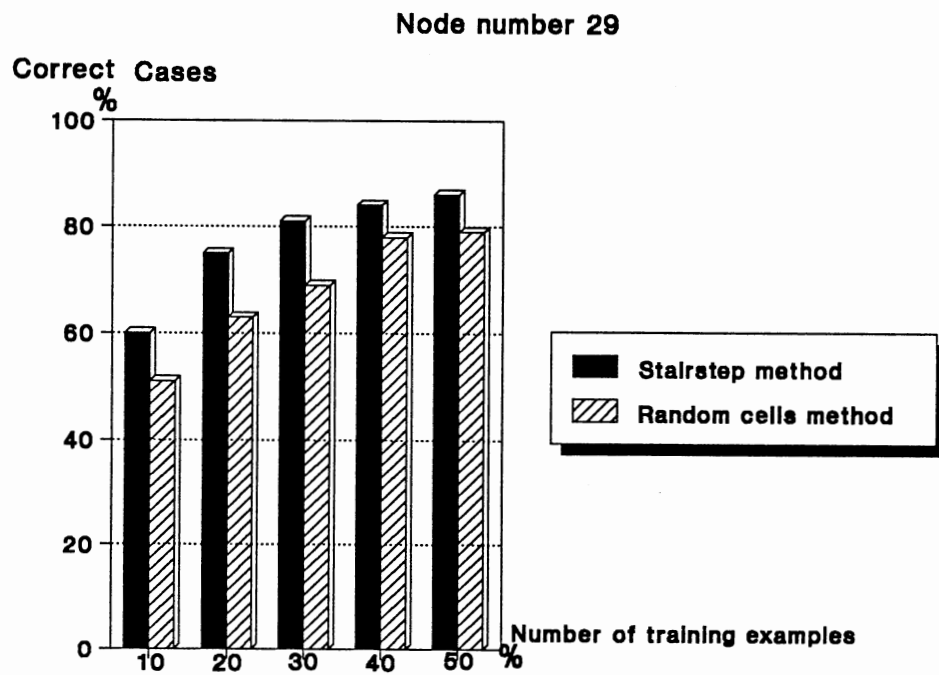


Figure 20. Results of Training Node 29



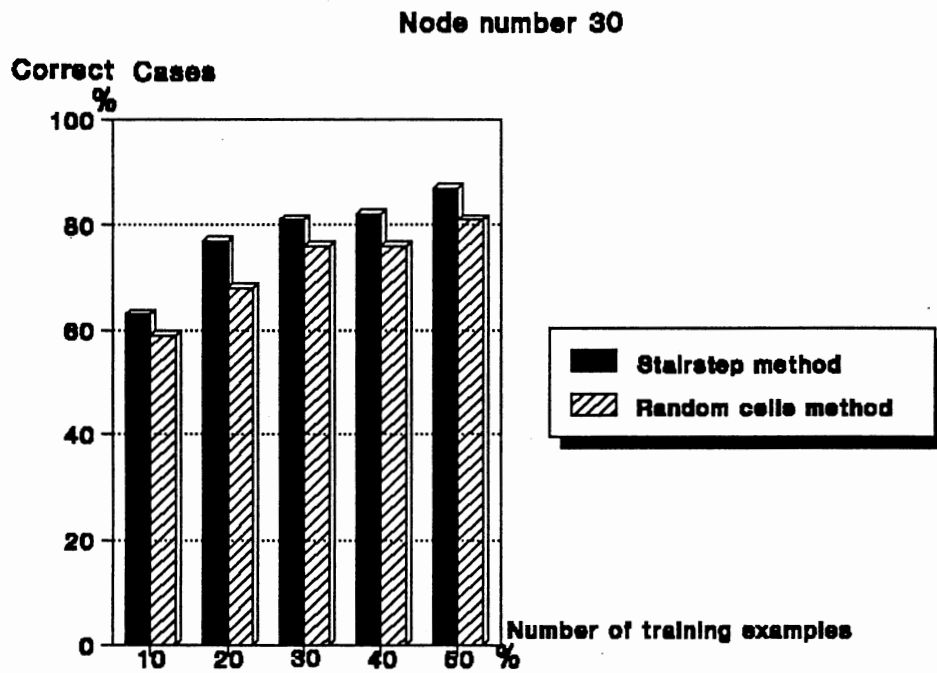


Figure 21. Results of Training Node 30

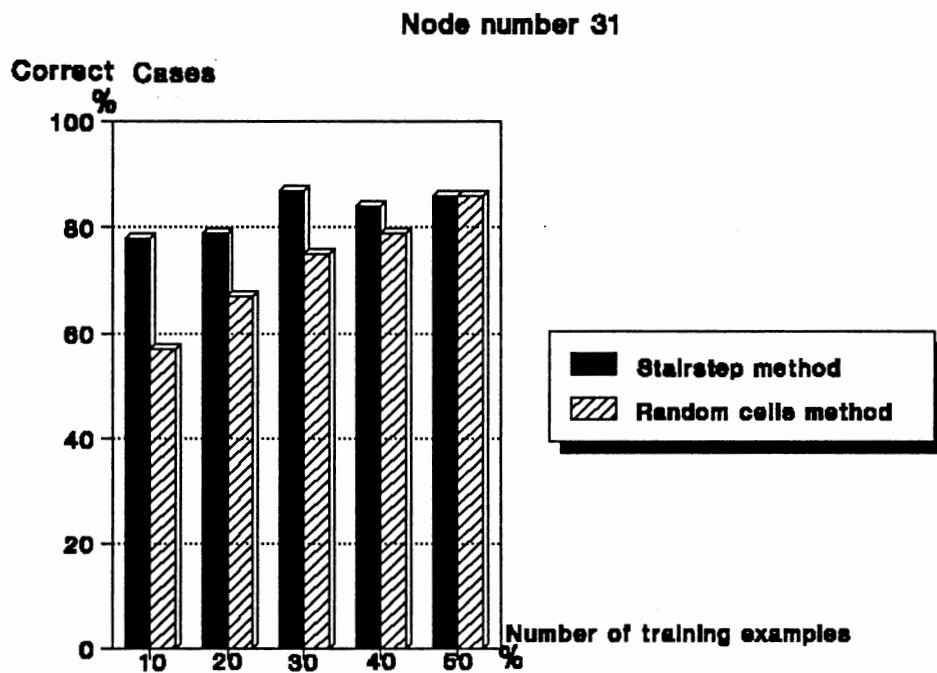


Figure 22. Results of Training Node 31

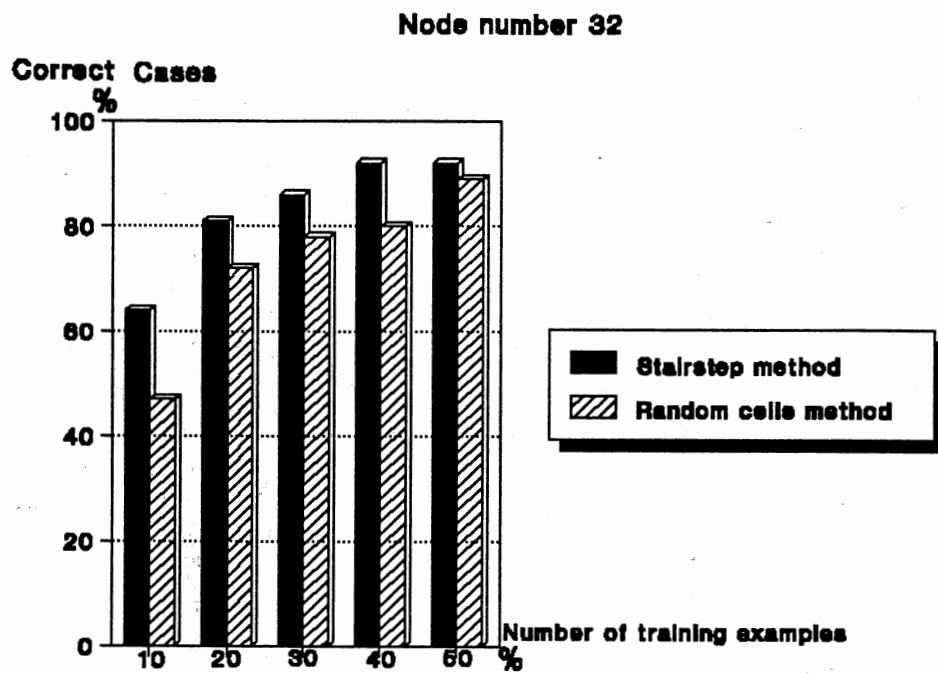


Figure 23. Results of Training Node 32

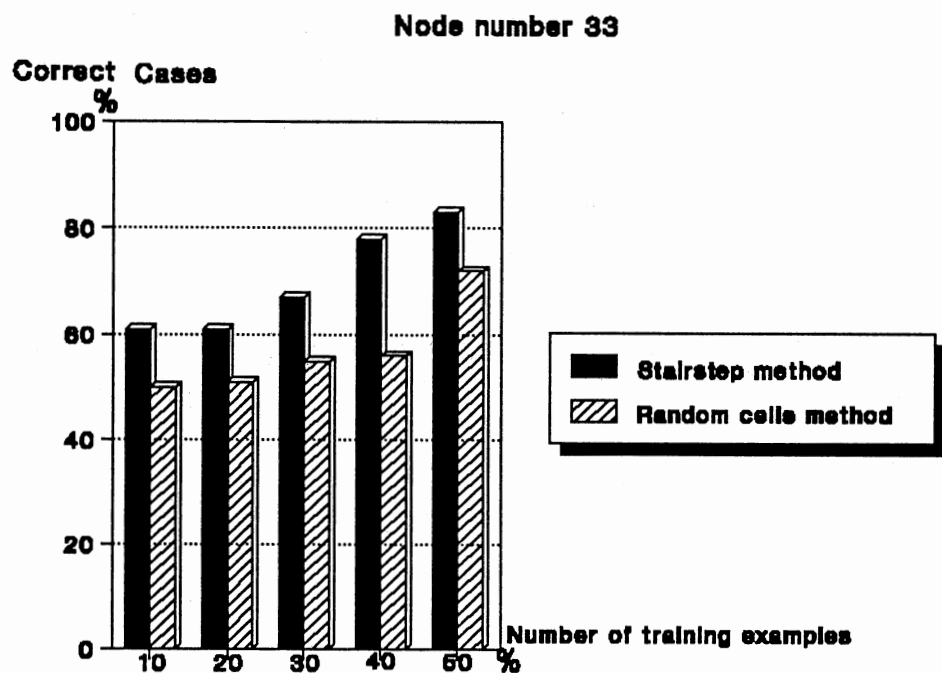


Figure 24. Results of Training Node 33

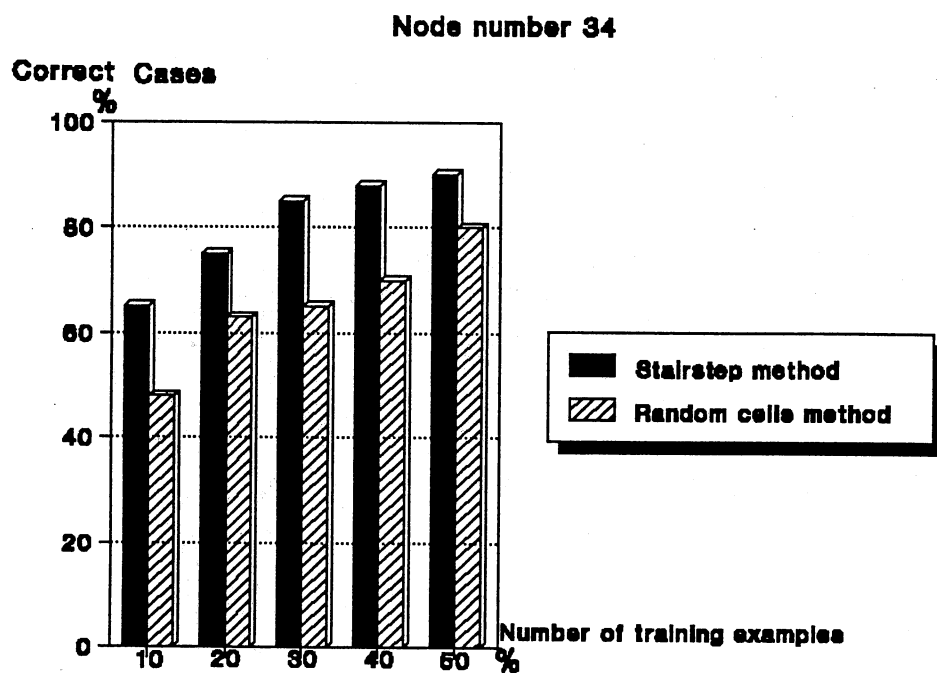


Figure 25. Results of Training Node 34

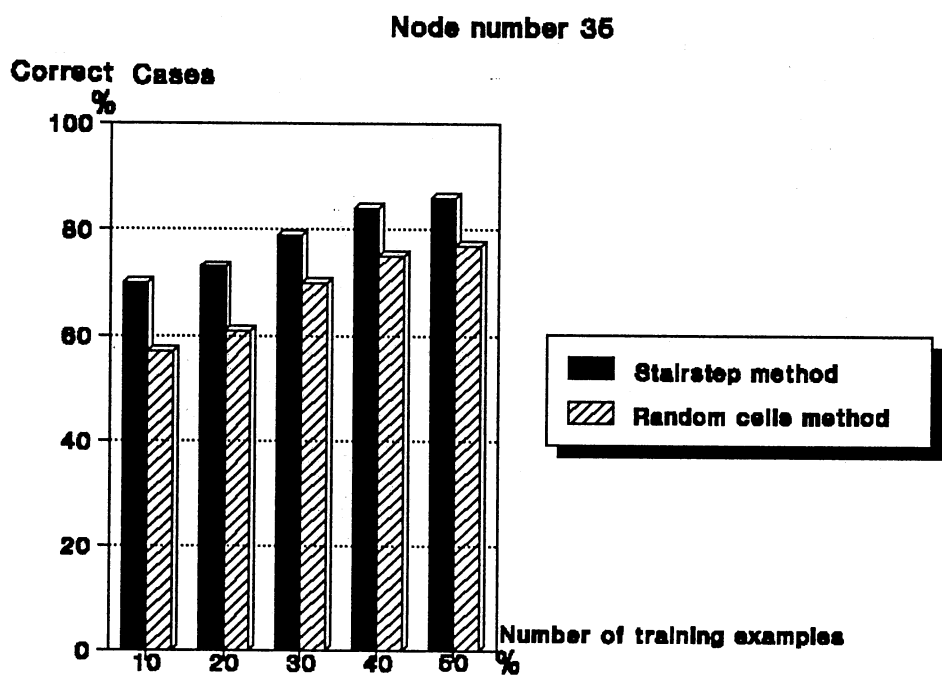


Figure 26. Results of Training Node 35

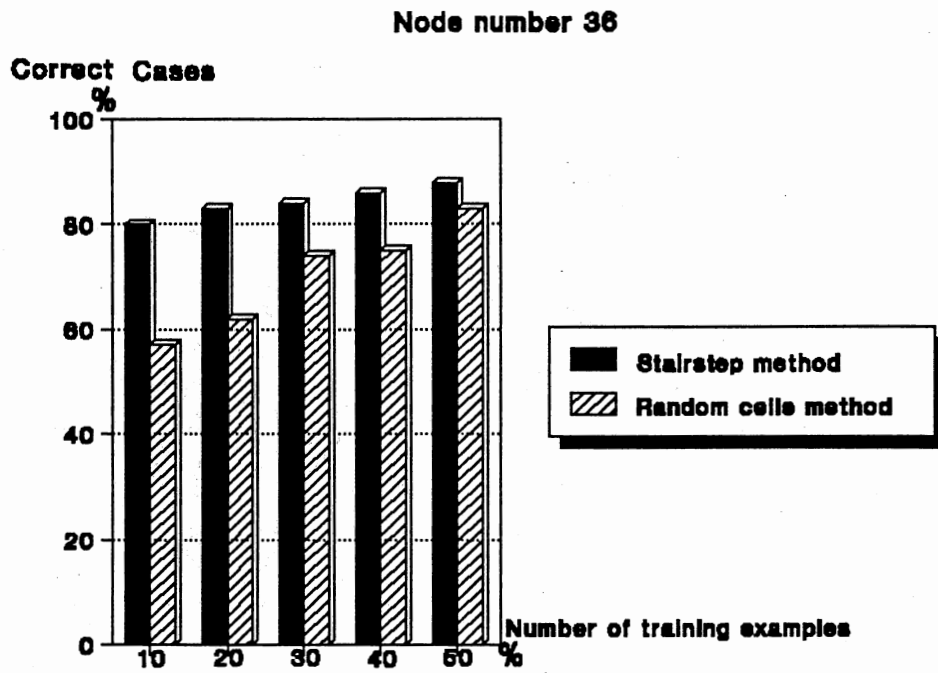


Figure 27. Results of Training Node 36

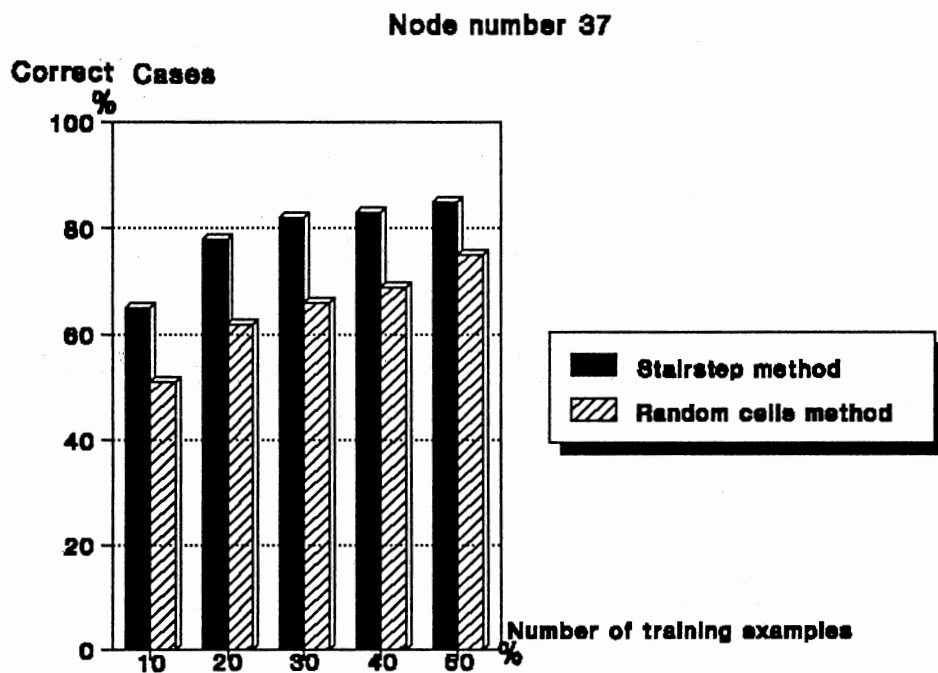


Figure 28. Results of Training Node 37

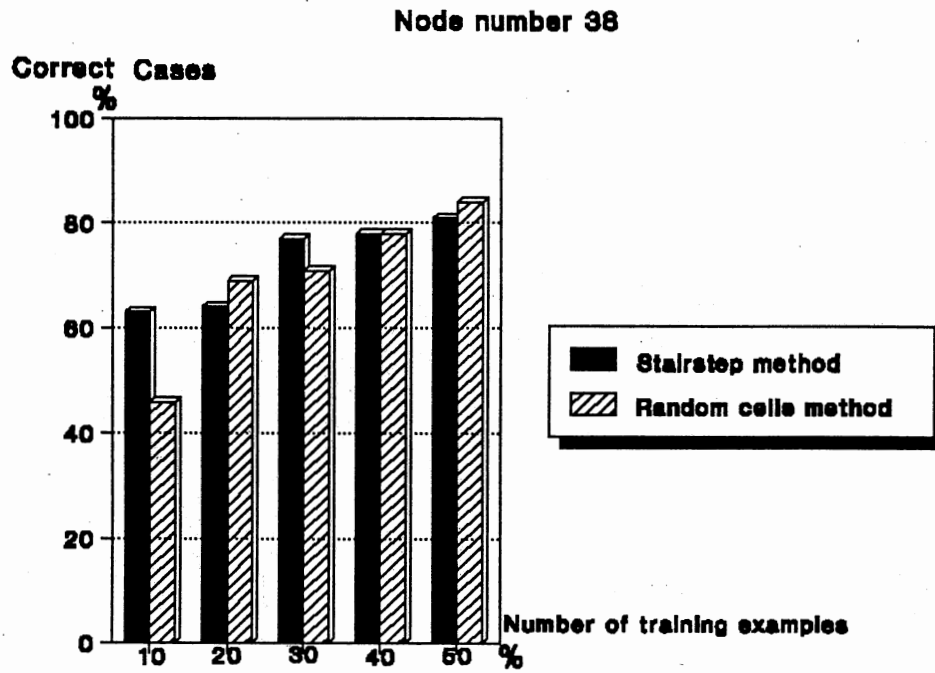


Figure 29. Results of Training Node 38

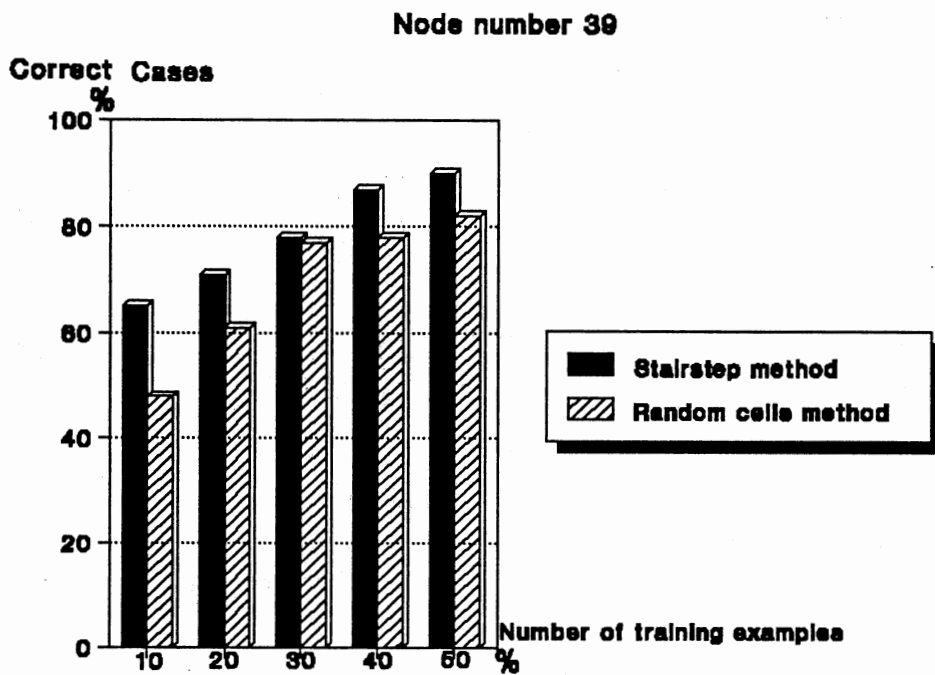


Figure 30. Results of Training Node 39

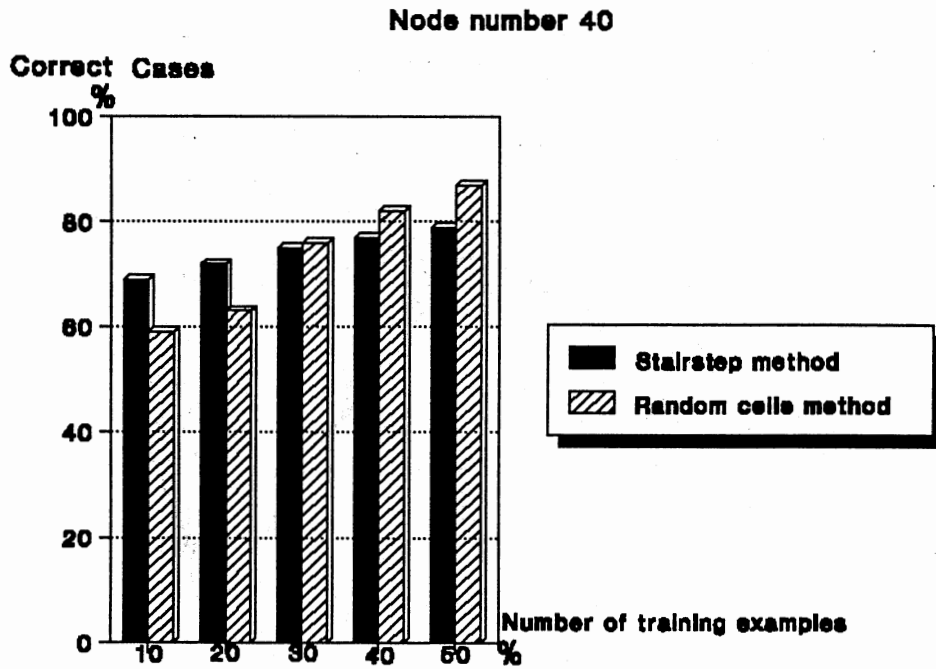


Figure 31. Results of Training Node 40

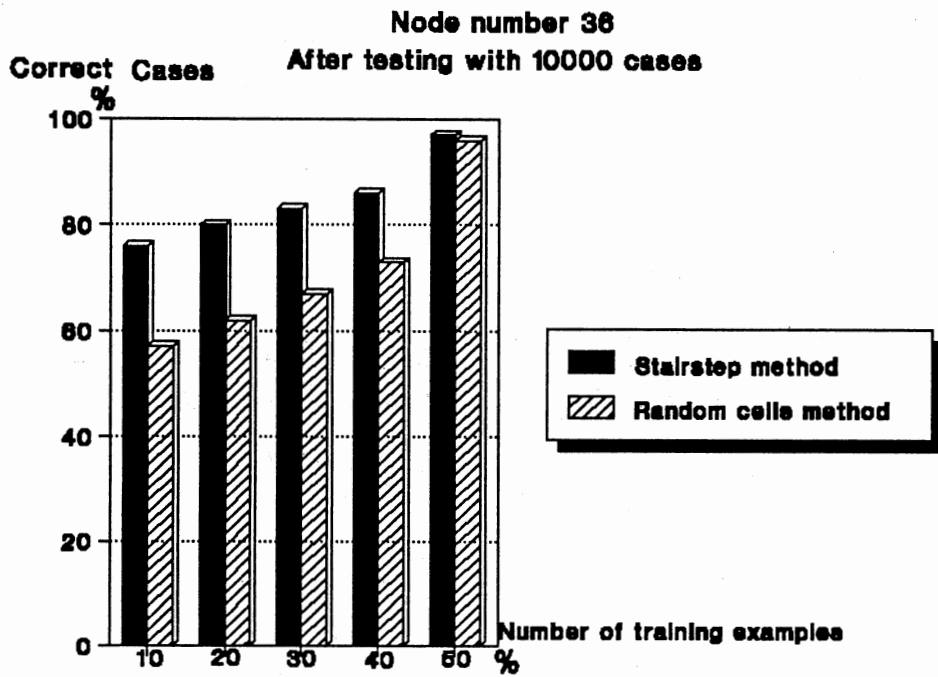


Figure 32. Results of Node 36 After Testing the Network by 10000 Cases

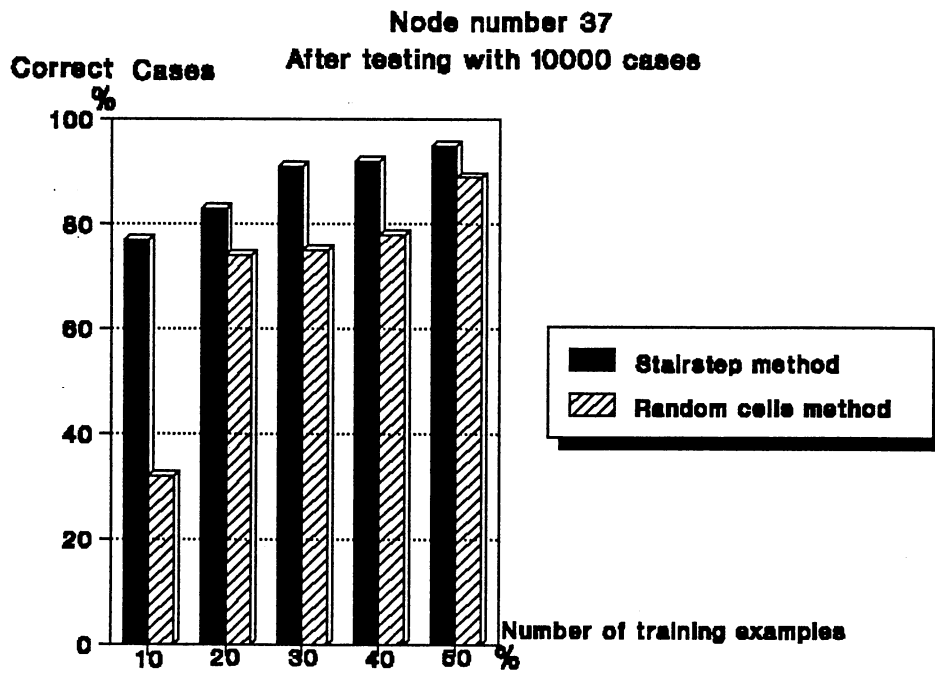


Figure 33. Results of Node 37 After Testing the Network by 10000 Cases

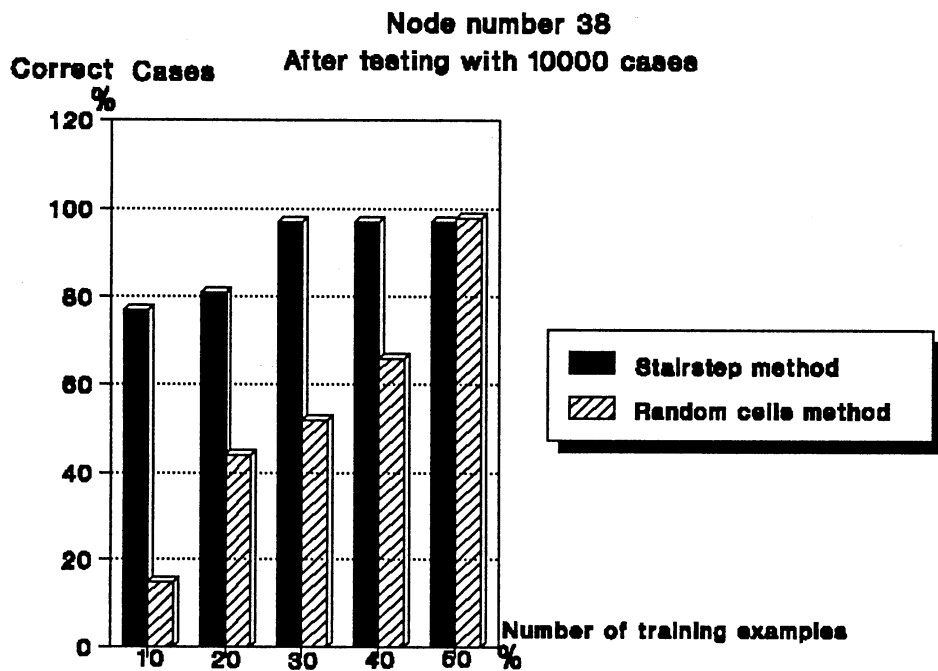


Figure 34. Results of Node 38 After Testing the Network by 10000 Cases

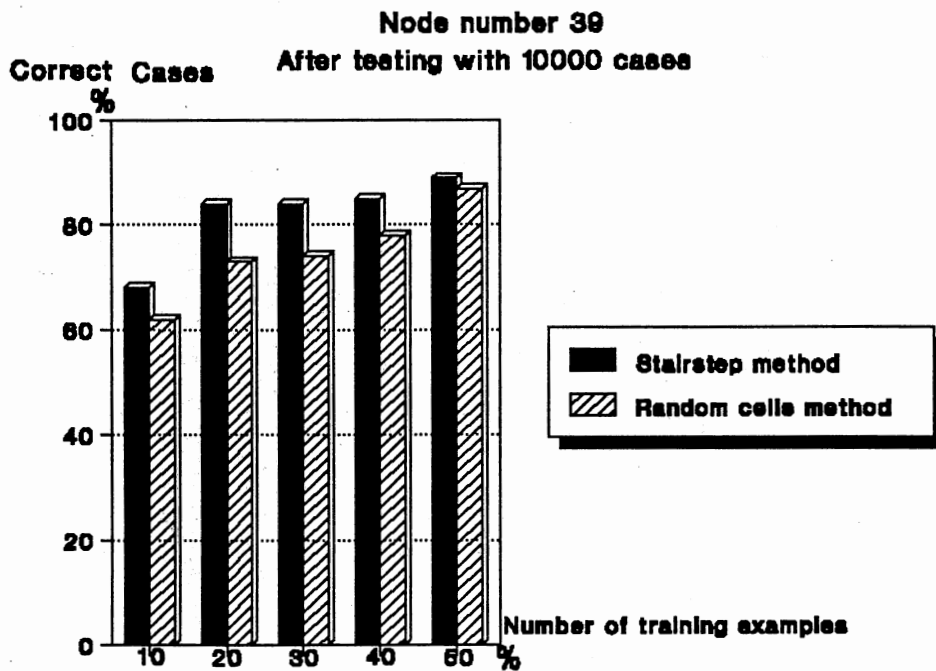


Figure 35. Results of Node 39 After Testing the Network by 10000 Cases

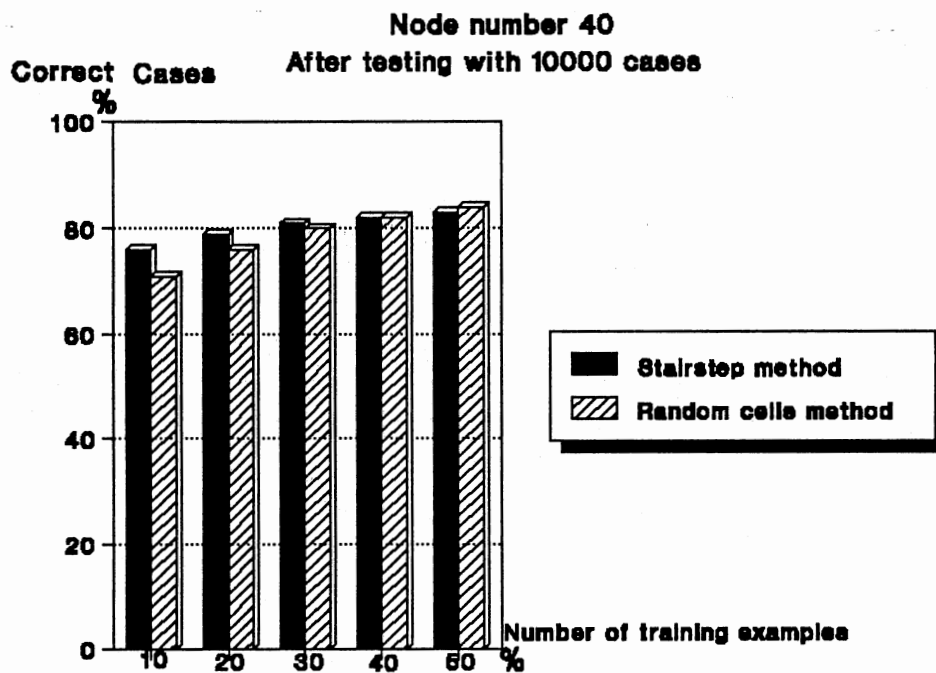


Figure 36. Results of Node 40 After Testing the Network by 10000 Cases



VITA 2

Ayman Aly Radwan

Candidate for the degree of

Master of Science

Thesis: UNCERTAINTY MANAGEMENT IN CONNECTIONIST EXPERT  
SYSTEMS

Major Field: Computer Science

Biographical:

Personal Data: Born in Cairo, Egypt, May 8, 1961.

Education: Graduated from Ismeal El Kabany high school, Cairo, Egypt in June 1979; received Bachelor of Science Degree in Civil Engineering from The Ain Shams University in June 1984; completed requirements for the Master of Science Degree at Oklahoma State University in May, 1992.

Professional Experience: Construction Engineer, Ministry of Housing, Cairo, Egypt, June 1984 to October 1988. Research Assistant, Water Research Center, Cairo, Egypt, October 1988 to present.