

1119

# **A NEURAL NETWORK BASED EXPERT SYSTEM FOR ANALOG CIRCUIT FAULT DIAGNOSIS**

*Dissertation submitted to The Jawaharlal Nehru University  
in partial fulfilment of the requirements  
for the award of the degree of*  
**MASTER OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE**

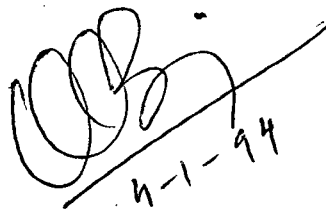
**V KRISHNA REDDY**

**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES  
JAWAHARLAL NEHRU UNIVERSITY  
NEW DELHI-110 067  
JANUARY 1994**

# CERTIFICATE

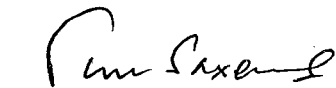
This is to certify that dissertation entitled "A NEURAL NETWORK BASED EXPERT SYSTEM FOR ANALOG CIRCUIT FAULT DIAGNOSIS", being submitted by V. KRISHNA REDDY to Jawaharlal Nehru University in the partial fulfillment of the requirement for the award of degree of **Master of Technology** in Computer science is a record of original work done by him under the supervision of Prof.P.C.SAXENA, Professor, School of computer and systems sciences, Jawaharlal Nehru University during the year 1993 monsoon semester.

The results reported in this dissertation have not been submitted in part or full to any other University or Institute for the award of any degree or diploma.



h-1-94

(Prof. K.K. BHARADWAJ)  
Dean, SCSS, J.N.U.,  
NEW DELHI.



(Prof. P.C. SAXENA)  
SCSS, J.N.U.,  
NEW DELHI.

## ACKNOWLEDGEMENTS

I feel pleasure to express my heartfelt gratitude to my guide **Prof P.C. SAXENA** for his uncompromising guidance, constant supervision and constructional help. This effort would not have succeeded without the valuable discussion, encouragement to pursue my thoughts in my own way, apt criticism and excellent guidance.

I extend my sincere thanks to Dean **Prof. K.K. BHARADWAJ**, for providing me the opportunity to undertake this project. I would like to thank the staff and authorities of our school for providing me the necessary facilities to complete my project.

I would like to thank some of my friends, Ramana, Govardhan, who helped me in completing this project successfully. Last but not the least, I would like to thank my classmates, for the encouragement they gave me in completing this project.

  
( **V. KRISHNA REDDY** )

# ABSTRACT

The Neural Network application to Analog Circuit Fault-Diagnosis is considered. The multilayer feedforward network with BackPropagation algorithm is chosen for this application. The circuit can be simulated for different faults using any simulator and nodal voltages under the different fault conditions can be obtained. These nodal voltages along with the fault numbers were formulated into a table, called as *Fault Dictionary*. The Neural Network is trained for all sets of the nodal voltages as input patterns and with the respective Binary coded fault numbers as output. Two approaches were followed in using the Neural Networks in Fault-Diagnosis. Software is developed for both the approaches. In the first approach the nodal voltages are directly applied as input patterns. In this all the accessible nodes of a circuit are to be considered as inputs for the neural network. In the second method using Lin & Elcheriff's approach, minimum number of test nodes are obtained for fault isolation and an unique integer code is obtained for all fault conditions. This approach isolated the faults from other faults to eliminate the ambiguities caused by tolerance limits of components of the circuit. Here the coded ambiguity values are given as the inputs and binary coded fault numbers as output values. Both the approaches were working satisfactorily. The Neural Network is converging to the specified error in reasonable number of iterations.

# INDEX

<b>Chapter</b>		<b>Page No</b>
1. Introduction	....	1
2. Neural Networks	....	12
3. Backpropagation Algorithm	....	21
4. Analog Circuit Fault-Diagnosis	....	31
5. Application of ANNs to Faultdiagnosis	....	35
6. Results and Conclusions	....	56
References	....	60
Appendix 1		
Boltzmann Machine	....	62
Appendix 2		
PARAS-PARAM	....	67
Appendix 3		
Taxonomy	....	72

# CHAPTER 1

## INTRODUCTION

Rapid advances in Neuroscience and in Computer Science are arousing renewed interest in Artificial Neural Networks as potentially new problem-solving architectures. The neural networks' derived its name from the network of *neurons*, which is the basic fundamental unit of nervous system, especially the brain. Human Brain has more than ten billion nerve cells or *neurons* interconnected in a massively parallel fashion. The brain's capabilities inspired many scientists to attempt computer modelling of its operation and the result has been the study of neural networks. It is common to refer to such networks as *Artificial Neural Networks* to distinguish them from the natural neural networks that are in human brain.

### **Analogy to the Brain**

The neuron is the fundamental unit of the nervous system and in particular the brain. Its nucleus receives and combines signals from many other neurons through input paths called *Dendrites*. If the signal is strong enough it activates the neuron which produces an output signal. The path of the output signal is called *Axon*. The axon splits up and connects to the dendrites of other neurons through a junction called *Synapse*. The amount of signal transferred depend on the *Synaptic Strength*. This simple transfer of information is chemical in nature but has

electrical side effects which can be measured. The structure of a neuron is as shown in Fig. 1.

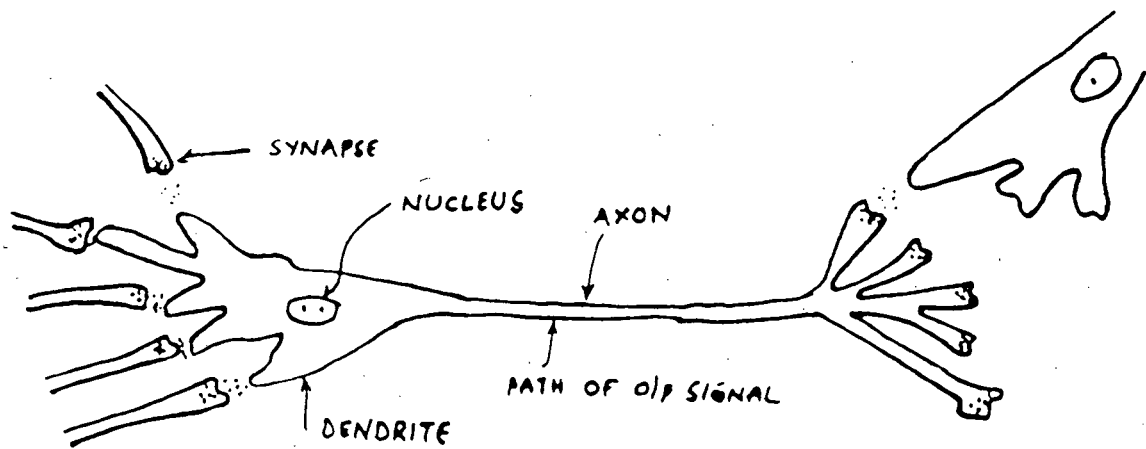


Figure 1 Structure of Biological Neuron

### Artificial Neuron

Artificial Neural Networks (ANNs) are made up of processing elements or nodes which resemble the neurons of human brain. A processing element (PE) has several input connections and a single output. The PE receives input signal through its input connections from other PEs. Each connection from one processing element to another processing element has some value called connection weight. The processing element calculates the weighted sum of its input signals according to a function to determine its output. The transfer function is generally a non-

linear function like hard limiters, threshold or sigmoid function. Positive weights which increase the strength of connection represent excitatory connection while negative weights which represent inhibitory connections decrease the strength of connection. The output is sent to other PEs through output terminal.

The structure of processing element is as shown in the figure 2. In this figure,  $x_1, x_2, x_3, x_4, \dots, x_n$  represent the input values from other neurons. And  $w_1, w_2, w_3, w_4, \dots, w_n$  represent the corresponding connection weights. The processing element calculates the weighted sum  $w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + \dots + w_n \cdot x_n$  of all these inputs, and then calculates its output using some transfer function  $F_n$ .

$$\text{output} = F_n(\sum_{i=1}^n w_i \cdot x_i)$$

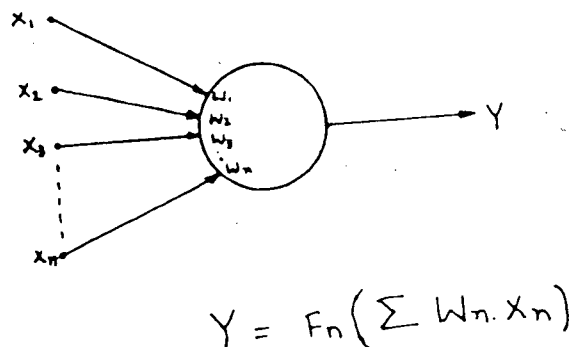


Figure 2 Structure of Artificial Neuron (Processing Element)

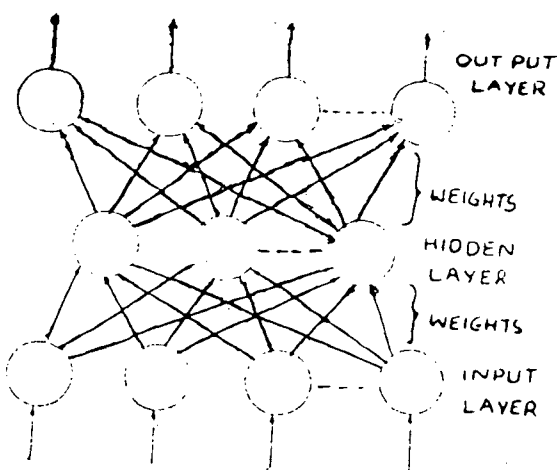
### Artificial Neural Networks

Artificial Neural Networks are dynamic systems composed of highly interconnected layers of simple neuron-like processing



elements as shown in figure 3. These layers are categorized as input layers where patterns are presented to the network and output layers which contain the response to a given input. Further more they may contain intermediate layers or hidden layers between input and output layers.

Neural network operations consists of a learning or training phase, recall phase, and generalization phase. During the learning phase the network is repeatedly presented with a set of input-output patterns. Learning is accomplished by a general rule which dynamically modifies the weights of all interconnections in an attempt to generate the desired output pattern for each presented input pattern. After the learning is completed the network operates in a recall phase where it generates response to input patterns used in the training. It also operates in a generalization phase where it generates response to similar or novel input patterns.



**Figure 3** Structure of Artificial Neural Network

Neural network computations are collectively performed by the entire network with knowledge represented in the connection weights between processing elements. Consequently, the collective operations result in a high degree of parallel computation which enables the network to solve complex problems rapidly. In addition, the distributed representation leads to greater fault tolerance and to graceful degradation when problems are encountered beyond its range of experience.

In the following table 1, comparison is made between human brain and artificial neural networks.

**Table 1 Comparison Between Brain and ANN**

Element	Brain	ANN
1. Organization	Network of neurons	Network of Processing elements
2. Components	Dendrites, Axons, Synapses, Summer, Threshold.	Inputs, outputs, weight Summation function, Threshold function.
3. Processing	Analog.	Analog or Digital.
4. Architecture	10 to 100 billion neurons	1 to 1,000,000 PEs
5. Hardware	Neuron	Switching Device.
6. Switching Speed	1 milli second	1 n.sec. to 1 m.sec.
7. Technology	Biological	Silicon, optical.

### **Comparison to AI :**

1. In AI knowledge is explicit in the form of rules. To create an AI an expert is needed to evaluate the problem and to define the rules. Where as in artificial neural networks we need not define any rules. ANNs can generate their own rules by learning from the examples.

### **Advantages of Neural Networks**

Because of its structure and the organization or processing of information, the neural networks offer some unique advantages over other conventional systems. The advantages are as follows:

**1) ADAPTIVE LEARNING:** An ability to learn how to do tasks based on the data given for training or for initial experience.

Adaptive learning is one of the most attractive features of neural networks; that is, they learn how to perform certain tasks by under going training with illustrative examples. Because the ANNs can learn to discriminate patterns based on examples and training, we do not need to elaborate training models, nor we need to specify the probability distribution functions. The designers sole concern is to select appropriate architecture, learning algorithm, training patterns, and network dynamics.

**2) SELF ORGANIZATION :** A neural network can create its own representation of information it receives during learning and operation.

Neural networks use their adaptive learning capabilities to self-organize the information received during learning. When the neural network self organizes, it creates representation of distinct features of the training data. Even when the ANNs are taught to learn certain classes of patterns, they self organize the information such that it is used for pattern recognition.

**3) FAULT TOLERANCE VIA REDUNDANT INFORMATION CODING:** Partial destruction of NN leads to degradation of performance. However, some network capabilities may be retained even after major damage.

NNs are the first computational methods available which are fault tolerant. There are two distinct aspects of fault tolerance. First, NNs can learn to recognize patterns which are noisy, distorted, even incomplete. This fault tolerance is with regard to data. Second, they continue to perform even after some part of the network itself is destroyed. This fault tolerance to damage within themselves.

This tolerance is due to that NNs have distributed (or redundant) information encoding. When neural network store information it is not localized. Instead it is shared by all interconnections and weights. Where as most computer algorithms and data retrieval systems store each piece of information in a localized addressable space.

4) **REAL TIME OPERATION** : NN computations can be carried out in parallel, and special hardware devices are being designed and manufactured, which can take advantage of this capability. NNS are well suited for parallel implementation. Their structure is such that only a few steps need to be performed per neuron. That is, only it has to perform weighted sum of inputs and applies some transfer function to the sum. Thus massive parallelism can be achieved through VLSI technology.

5) **EASE OF INSERTION INTO EXISTING TECHNOLOGY** : An individual network can be trained to perform a single, well defined task. Because a network can be rapidly prototyped, trained, tested, and verified, and translated into low cost hardware implementation, it is easy to insert neural networks for specific purposes into existing systems.

Because of these advantages, artificial neural networks are emerging computational technology which can significantly enhance a number of applications. ANNs are finding use in many real world applications such as speech processing, pattern recognition, image processing, diagnosis, natural language processing, combinatorial problems, noise filtering, medical diagnosis and control systems.

Software is developed for analog circuit fault diagnosis using fault dictionary approach. A new method of using Artificial neural network for the fault diagnosis is carried-out under this dissertation work. For this we have chosen D.C. fault dictionary method, which can be used for diagnosing hard failures in the circuits.

The essence of fault dictionary is, for each fault, its symptom or signature (usually voltages of some test nodes) is determined through a software simulation program and stored in a dictionary. Before simulating the network a set of test nodal voltages and also same set of faults are selected which are likely to occur in the actual usage of the circuit. Then the circuit is simulated for all the predetermined faults and the test nodal voltages corresponding to each fault are determined. Given a faulty circuit, one makes test measurements and then compares the result with those stored in the dictionary to identify the fault. Thus D.C. Fault dictionary approach is simulation before test (S.B.T.) method, since we are simulating the circuit before use. There is another method for fault diagnosis is simulation After Test (S.A.T.). But the main problem with direct comparison of nodal voltages, with those in the dictionary is, here we are assuming the values of components of the circuit as constant (absolute value) while simulating. But in practice, the values of components will vary within some tolerable band of values. Because of these tolerance values one set of nodal values may fall into another set of fault conditions. So it is difficult to determine the fault just by comparison to fault dictionary. So to distinguish different faults, these faults must be isolated from one another.

For this I have used two approaches. In the first approach I have used Back Propagation Neural Network for isolation. In this method first the circuit is simulated for different fault conditions and these nodal voltages are formed into a fault

dictionary. In the fault dictionary each nodal voltage will be given some address. The back propagation Neural Network is trained with these patterns, i.e., the nodal voltages are as input values and the corresponding address as output values. After training, the network will give the corresponding fault number for a given set of input node voltages in RUN mode.

In the second approach, the test nodal voltages are isolated from one another by using some logical procedure (Lin & Elcheriff's method). By using this procedure we can, not only isolate faults but also determine the minimum number of nodes required for isolation. The input to the isolation procedure is the test nodal voltages. The output is a unique integer code corresponding to different sets of nodal voltages. This integer code is formed into a fault dictionary. The addresses of faults are placed against the integer code. Then this table is applied as training patterns to the neural network. After training the neural network it gives the address of fault as output corresponding to the integer code as input.

It is found that both approaches were working satisfactorily. For the two approaches, I have developed software for BP algorithm and for isolation procedure in the second approach. The Neural Network is converging to the specified error in a reasonable number of iterations.

The remainder of the paper is organized in the following manner. The Basics of Neural Networks, its analogy to brain, and its advantages are discussed, and also its application to Analog

Circuit Fault-diagnosis is introduced in Chapter 1. In chapter 2 representation of neural networks and their applications are discussed. In chapter 3 the Back Propagation network is discussed. In the fourth chapter fault dictionary method of analog circuit fault diagnosis is described. In chapter 5 the concept of ambiguity sets and the applications of Neural Network to the analog circuit fault diagnosis is given. Both methods are discussed in detail in that chapter. Subsequently the results and conclusions of the neural network based approach is presented in chapter 6. In the appendix 1, Boltzmann machine algorithm is presented. In appendix 2, introduction on Parallel Programming on transputer based Parallel Super Computer PARAM is discussed. In appendix 3 Taxonomy of different neural networks is presented.



## CHAPTER 2

# NEURAL NETWORKS

Any neural network structure is represented in terms of three fundamental descriptors. They are

a) Interconnection architecture between processing elements.

b) Rules that determine whether or not a particular processing element will fire( or Transfer Function).

c) Training laws(or Learning laws).

**a) Architectures :** In this the physical organization and arrangement of the neurons is considered. Meso-structural (i.e., architectural) considerations are especially important in that they help us to distinguish different classes or types of network architectures. These considerations help us to make distinctions between different types or classes of networks which are currently in use. There are some basic distinctions which allow us to form categories of networks.

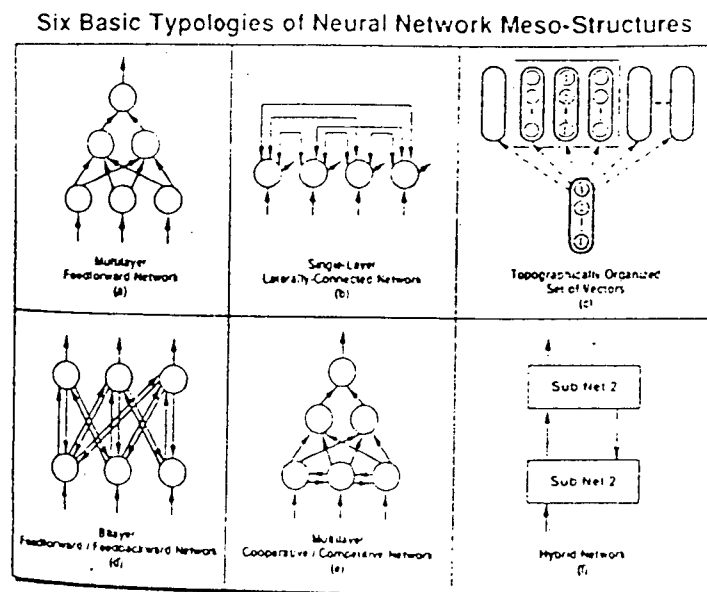
The first distinction is in the number of layers in a network. We classify networks as being single-layered, bi-layered or multilayered. Within this first distinction, we note that the type of connectivity allowed creates the possibility of different structures. We identify two major types of single-layered networks: those that are explicitly laterally connected, and those which have only implicit connectivity. Bi-layered networks typically have both feedforward and feedback connections. There is a

large class of multilayered networks which have strictly feed-forward connections, and there are a number of multilayered networks with complex connectivities ( feedforward, feedback, and lateral). Based on these distinctions, the networks are divided into different types, as following

- \* Multilayer, feedforward networks
- \* Single-layer, laterally-connected works
- \* Single-layer topologically ordered (vector matching) networks
- \* Bi-layer, feedforward / feedback networks & Multilayer,

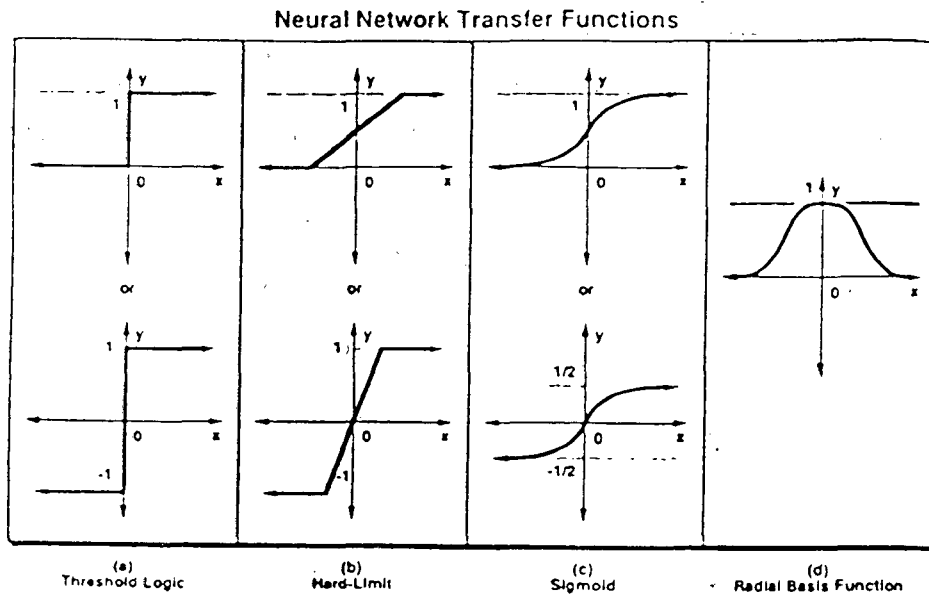
Cooperative networks

- \* Hybrid networks.



**b) Transfer functions :** The most common distinguishing factor among most of the artificial neurons being used today is their transfer function. This function specifies how the neuron will scale its response to incoming signal, and produces its activation. If the activation is strong enough, then the artificial neuron will output a signal to the neurons to which it is connected. Four typical transfer functions are

- i) Threshold logic nodes.
- ii) Hard-limit nodes
- iii) Continuous function(sigmoid) nodes.
- iv) Radial basis functions.



**Figure 5** Different Transfer functions

**i) THRESHOLD LOGIC:** Threshold logic nodes create binary state neurons by applying simple transfer function.

If Weighted sum > Threshold

Then Activation = 1.

Otherwise = 0.

- \* Easy to implement in hardware.
- \* Limited learning capability.
- \* Used in Hopfield/Tank network.

ii) **HARD LIMIT NODES:** In this both upper and lower limits are set on the summed input from other neurons, plus the thresholds.

If the total sum > upper limit, activation = 1.

< lower limit, activation = 0.

Between these two limits the output is linear function of weighted sum. At the transition points the function is not differentiable. This limit their use in applications which require sophisticated learning capabilities.

iii) **CONTINUOUS FUNCTION NODES:** Werbos suggested this Sigmoid transfer function for improved learning in Back propagation network. The transfer function is smoothly varying and is differentiable at all points. Back propagation network requires the differentiation output activation as a function of input.

The equation of sigmoid is

$$Y = 1/(1 + e^{(-kX)})$$

where X = weighted sum, Y = output activation of neuron,  
and k = constant.

Derivative of sigmoid is always positive, and is close to zero for either large positive or large negative values of X. And is at its maximum value when X is zero.

iv) **RADIAL BASIS FUNCTIONS:** This is typically a Gaussian function, which is useful when creating a neural network for continuous function mappings. The centers and widths of these functions may be adapted, which makes them a more adaptive function than sigmoid function. Mappings which may require two hidden layers of sigmoid function units can some times be accomplished by a single layer of neurons using radial basis functions.

c) **Learning :** Learning is the process of adapting the coefficient weights in response to stimuli being presented at the input buffer and optionally at the output buffer. An ANN can learn in supervised or unsupervised mode.

**SUPERVISED ANN:** ANN which requires a supervisor for learning is called supervised ANN. The ANN is provided with sets of inputs and corresponding desired output sets. After each trial the output obtained is compared with desired output and the difference corrected using some learning algorithm till actual output matches to an acceptable level.

**UNSUPERVISED ANN:** ANN's which do not require a supervisor for learning are called unsupervised ANN. Unsupervised ANN can be trained in two ways graded training and self organization training. In graded training inputs are given and occasionally grade is given according to performance of network. In self organiza-

tion learning network itself organizes into some useful configuration based on the inputs.

#### **PRACTICAL APPLICATIONS :**

Artificial Neural Networks are finding use in many real world applications such as speech processing, pattern recognition, image processing, diagnosis, finance, etc. Sometimes it may be necessary to combine ANN techniques with traditional techniques to solve the problems.

**1. SPEECH PROCESSING :** Speech processing problems include speech recognition, text to speech conversion. In speaker recognition problems, speech samples of some speakers are collected. The collection of features of speech like pitch period, zero crossing etc., forms a template which is stored in memory. Several such templates are stored in memory. When a test sample is given as input, it creates a template and recognizes the speaker if it finds a match in the memory. Kohonen devised a phonetic typewriter which could type from dictation. Sejnoviski and Rosenberg developed NETtalk which could change text into spoken language. NETtalk was used on Back Propagation algorithm.

**2. NATURAL LANGUAGE PROCESSING :** Natural language processing involves studying how we construct rules about language. Cognitive scientists Rumelhart and Mcckekabd devised a neural computing system which learns the past tense of English verbs.

**3. IMAGE COMPRESSION :** Image compression refers to transforming image data to a different representation which requires less

memory but from which original image can be reconstructed. Cottrell Munro and Zipser designed a three level neural network and achieved compression ratio of 8:1 with high fidelity.

**4. PATTERN RECOGNITION:** ANN's are finding applications in recognition of hand written characters. Nestor developed a system which accepts handwriting on a digitized pad. After being trained for interpreting a set of handwriting types, the neural system is able to interpret a type of handwriting it has never seen before. Some advanced pattern recognizing systems use neocognitron which is a multi-layer pattern recogniser that simulates the way visual information feeds forward in the cortex of the human brain.

**5. COMBINATORIAL PROBLEMS :** Neural computing system also solve certain combinatorial problems such as travelling salesman problem in which the goal is to find the shortest possible route the salesman can take to cover a certain number of cities in a specified area. Hopfield and David Tank have developed a system to solve this problem.

**6. PATTERN RECOGNITION IN IMAGES :** Groman and Sejnowski applied back propagation networks to classify sonar targets. David Gloer used back propagation in machine vision applications. This had two benefits a) minimal operators used for training the classifier and b) no assumptions were made.

**7. SIGNAL PROCESSING :** Lapedes and Farber used back propagation networks for doing prediction and system modeling. They showed that for chaotic time series back propagation exceeds convention-

al linear and polynomial predictive methods by many orders of magnitude.

**8. NOISE FILTERING :** Neural networks can also be used for noise filtering. They are able to preserve a greater depth of structure in detail than traditional filters while removing noise.

**9. SERVO CONTROL :** It is very difficult to control complex mechanical servo systems. Errors are introduced due to some physical variation. It is impossible to measure the variations accurately and solving is very complex. Neural networks have been trained to predict the error in the final position of a robot from the joint angles. The error is combined with desired output to provide correction and to improve accuracy.

**10. DIAGNOSIS :** Diagnosis is the recognition and identification of the cause of problem. Diagnosis may be made to identify medical conditions, machine fault or similar problems. The ability to deal with large data based, incomplete data and situations in which diagnostic rules are not known in advance makes ANNs suitable for diagnosis problems. Here the input is the data about the fault conditions and the output is the diagnosis of the problem. DESKNET which uses Back propagation Algorithm is able to diagnose different skin diseases.

**11. CONTROL SYSTEMS :** Broom stick balancing system is one of the proven applications of adaptive control. A broom stick with several sensors is pivoted of a cart upside down on its handle. The ANN is trained to move the cart back and forth so that broom



is balanced on its handle top. During trial and error learning, the system uses the feedback to control the movement of the balancing platform.

**12. OTHER APPLICATIONS :** Two other major areas in which ANNs are used are A) Financial and Economic Modeling and b) Functional Synthesis.

# CHAPTER 3

## BACKPROPAGATION ALGORITHM

The network gets its name from how it handles the errors. Actually the network is developed from the network Perceptron, which is a single layer network. But this is able to train the output units to learn to classify the patterns of inputs, provided they are linearly separable.

More complex and non-linearly separable classes can be separated with a multi-layer network. However, if there is any error in the output layer, the Back Propagation algorithm solves the problem by processing element or inter-connection to adjust to reduce the error, by assuming that all processing elements and connections are responsible for the erroneous result.

Responsibility for errors is affixed by propagating the output error backward through the connections to the previous layer. This process is repeated until input layer is reached. The name *BackPropagation* derives from this method of distributing the blame for errors.

The key distinguishing characteristic of the backpropagation is that it forms a mapping from a set of input stimuli to a set of output nodes using features extracted from the input pattern. This network can be designed and trained to accomplish a wide variety of mappings, some of which are very-complex. This is because the nodes in the hidden layer(s) of the network learn to respond to features found in the input.

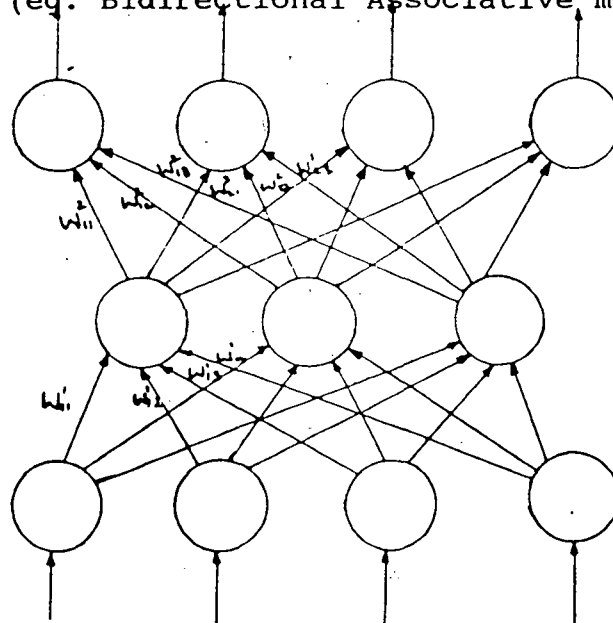
Because nodes in the back-propagation network learn to respond to features as the network is trained with different examples, the network develops the ability to generalize. For example, a back-propagation network is learned to distinguish between straight, concave, and convex curved lines. Even if the lines to be tested occur in different locations, or mixed with some noise or even some part of the line is missed, the network would be able to distinguish them. The network would probably respond correctly even if it is presented with a pattern which it has never seen before. The ability to make such complex distinctions, even when the presented pattern is different from those on which the network was trained, is due to the feature-detection and generalization abilities which are trained into the middle or hidden layer nodes.

In order for a back-propagation network to be successful for applications, the key issue is that the hidden layer nodes must be trained to recognize the right sets of features. These features must be sufficiently general, so that the network can respond correctly, even when its input is different from those it has previously encountered.

Backpropagation network can be represented in terms of three common descriptors as :

- i) Architecture.
- ii) Transfer function.
- iii) Learning laws.

1) **ARCHITECTURE** : The BackPropagation Network is a fully connected feedforward network as shown in following figure. **Fully connected Network** means each nodes of each layer is connected to each node of the next higher layer. In the feedforward network the output is calculated in the forward direction only. No part of the output is feed back to the input. There are some algorithms, in which output is fed back to the input in run mode (eg. Bidirectional Associative memory).



**Figure 6** Back Propagation Network

ii) **TRANSFER FUNCTION** : Backpropagation learning law requires that the transfer function for each nodes be defined by a continuous function. This function should be asymptotic for both infinitely large positive and negative values of independent variables (typically, the weighted sum of inputs). These conditions usually lead to a modified Sigmoid shape for transfer function. The use of this kind of transfer function is one of the major differences between BP and its predecessors, the Perceptron and ADALINE. Each of these earlier networks used

nodes with simpler transfer functions, and this limited their ability to be useful in the more complex pattern recognition problems.

One important factor about the sigmoid function is that its derivative is always positive, and is close to zero for large positive or negative values of  $X$ . The derivative has maximum when  $X$  is 0. This is important in helping the backpropagation learning law work effectively. This is because, the changes made to the weights is proportional to the derivative of the activation. If the derivative is near zero, then the changes are small. This is desirable, because the derivative is near 0 when the activation value is near 0 or 1, one of the two stable states. When the activation of the neuron is in the middle range, the activation must be changed such that it produces a value near one of the stable states (0 or 1). The derivative is large when the activation is in the middle range. So the changes in the weights is also fairly large. Thus the transfer function not only gives smooth and differentiable behavior, it also helps to give stability to the network.

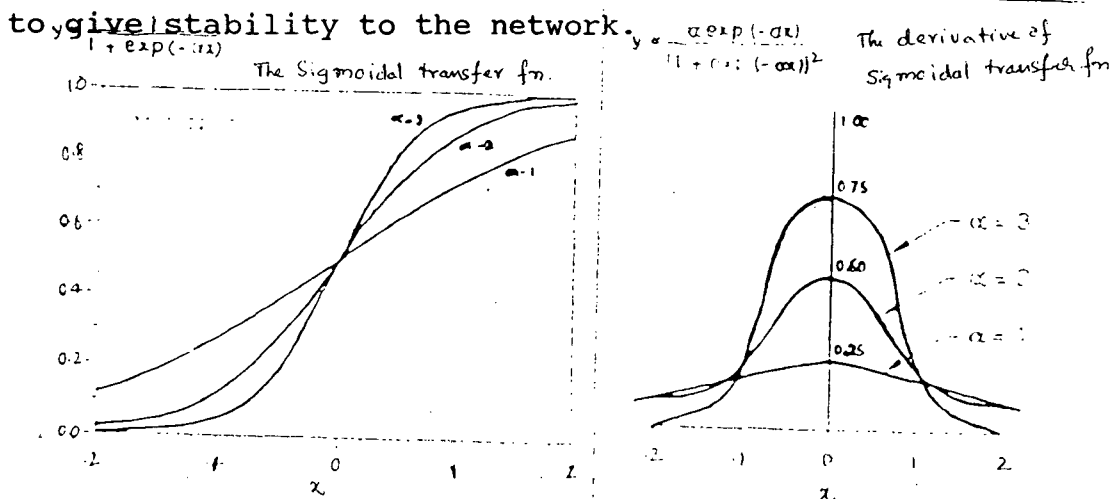


Figure 7 Sigmoid Transfer Function and its Derivative.

iii) **LEARNING LAW** : The backpropagation algorithm uses Generalized Delta Rule for its learning. In this procedure it uses Gradient Descent algorithm to adjust the weight values. The main advantage of gradient descent is that it makes changes to the weights such that the error drops most steeply. The idea of gradient descent is to make a change in the weight proportional to the negative of the derivative of the error, as measured on the current pattern, respect to each weight. Thus the learning rule becomes

$$W[i][j] = -k * dEp/dW[i][j]$$

If we change each weight according to this rule, each weight is moved towards its own minimum and we think of as the system is moving down hill in weight space until it reaches its minimum error value. When all the weights have reached their minimum points, the system has reached equilibrium. If the system is able to solve the problem entirely, the system will reach zero error and weights will no longer be modified. On the other hand, if the system couldn't solve the problem exactly, it will find a set of weights that produce as small an error as possible.

The basic idea of backpropagation method of learning is to combine a non-linear perceptron like system capable of making decisions with the error function as LMS and Gradient descent. To do this we must be able to readily compute the derivative of the error function with respect to any weight in the network and then change the weight according to the rule.

After calculating the derivative of an appropriate choice of non-linear function, the learning rule becomes

$$W_{ij} = e * \delta[i] * a[j]$$

Essentially, the term  $\delta[i]$  represents the effect of a change in the net input to unit  $j$  on the output of unit  $i$  in pattern  $p$ . The determination of  $\delta$  is a recursive process that starts with the output nodes. If the unit is an output unit, then the value of  $\delta$  becomes

$$\delta[i] = (T[i] - a[i]) f'_i(\text{net}_i)$$

where  $\text{net}[i] = \sum_j W_{ij} \cdot a_j + \text{Bias}_i$

The  $\delta$  term for hidden units for which there is no specified target is determined recursively in terms of the  $\delta$  terms of the units to which it directly connects and the weights of those connections. That is

$$\delta[i] = f'_i(\text{net}_i) \sum_k \delta_k \cdot W_{ki}$$

The application of the BP rule involves two phases: during the first phase the input is presented and propagated forward through the network to compute the output value  $a[j]$  for each output node. This is compared with the target values and calculates the values of  $\delta$  for all output nodes.

The second phase involves a backward pass through the network during which the delta term is computed for each unit in the network. Finally by using these  $\delta$ s we can easily compute the values that are to be applied to weights.

If the activation function used is Sigmoid,

$$a[i] = 1/(1+e^{-net[i]})$$

Then the above equation becomes

$$da[i]/dnet[i] = a[i] * (1-a[i])$$

For o/p unit  $\delta[i] = (T[i] - a[i]) * a[i] * (1-a[i])$

For hidden units  $\delta[i] = a[i] * (1-a[i]) \sum_k \delta[k] * W[j][k]$

#### ALGORITHM :

Step 1 : Initialize the weights and offsets.

Set all weights and node offsets to small random values.

Step 2 : Present input and desired output patterns used for the training of network.

Present a continuous valued input vector  $x_0, x_1, x_2, x_3, \dots, x_{(n-1)}$  and specify the desired outputs  $t_1, t_2, t_3, \dots, t_{(m-1)}$  for different sets of patterns. The input could be new on each trial from the training set that could be presented cyclically until weights stabilizes.

Step 3 : Calculate the outputs.

Use the sigmoid non-linearity to calculate the outputs.

The sigmoid or logistic function is given by

$$F(x) = 1/(1+e^{-x})$$

Net input to  $i$  th unit from the other  $j$  units is given by

$$I[i][s] = W[i][j][s] * X[j][s-1] + Bias[i]$$

After calculating the net input to any processing element, the activation of the P.E is calculated by using sigmoid

$$X[i][s] = F(I[i][s]) = 1/(1+e^{-(I[i][s])})$$



Like that the activation of the output layer (o/p of N.W) will be calculated.

Step 4 : Adaptive Weights.

Use recursive algorithm starting at the output nodes and work back to the first hidden layer. Adjust the weights by gradient descent algorithm. The LMS finds the values of all weights that minimizes this output error function is called gradient descent.

The idea of gradient is to make change in the weight proportional to the negative of the derivative of the error as measured on current pattern, with respect to each weight.

Thus 
$$W[i][j][s] = e * \delta[i][s] * x[j][s-1]$$

The determination of  $\delta$  is a recursive process that starts from the output units. The  $\delta$  for output units is given by 
$$\delta[i][s] = (t[i] - o[i]) * o[i] * (1 - o[i]).$$

For hidden units

$$\delta[i][s] = X[i][s] * (1 - X[i][s]) \sum_k \delta[k][s+1] * W[k][i][s+1]$$

And 
$$W[i][j][s] = e * \delta[i][s] * X[j][s]$$

Where  $e$  = Training rate parameter

This is called **Generalized Delta Learning Rule.**

Step 5 : Repeat by going to step2 until output error is below certain tolerable value.

**MOMENTUM** : In BP learning procedure the change in weight is proportional to weight error derivative. True gradient requires infinitesimal steps. The larger the leaning rate ( $\epsilon$ ), epsilon, the larger the changes in weights. For practical purposes one would like use learning as large as possible. In this way rapid learning rates can be achieved. But in practice when the learning rate is large the network goes into oscillations. One way to increase the learning rate without leading to oscillations is to modify the Backpropagation learning rate to include a **Momentum** ( $\alpha$ ) term. This can be accomplished by the following rule :

$$W[i][j](n+1) = \epsilon * (\delta[p][i] * a[p][j] + \alpha * W[i][j](n)).$$

The subscript  $n$  indexes the presentation number and momentum is a constant that determines the effect of past weight changes on the current direction of movement in weight space. This provides a kind of momentum in weight space that effectively filters out high-frequency variations of the error surface in the weight-space. In most of the simulations the value of momentum used is 0.9. With the larger values of momentum and training rate the system learns much faster.

**SYMMETRY BREAKING** : The BP learning has one more problem that can be readily overcome and this is the problem of symmetry breaking. If all weights start out at equal values and if the solution requires that unequal weights be developed, the system can never learn. This is because error is back propagated through the weights in proportion to the values of the weights. This means

all the hidden units connected directly to the output units will get identical error signals, and since the weight changes are dependent on error signals, the weights from those units to the output units must always be the same. The system is starting out at a kind of unstable equilibrium point that keeps the weights equal. But it is higher than some neighbouring points on the error surface, and once it moves away to one of these points. it will never return. This problem can be eliminated by starting the system with small random weights. Under these conditions the symmetry problem of this kind do not arise.

**LEARNING BY PATTERN or BY EPOCH :** In learning by Epoch method the derivative of an error function is summed over all patterns. In this case, I would present all patterns and sum the derivatives before changing the weights. Instead we can compute the derivatives on each pattern and make changes to the weights after each pattern rather than after each epoch.

## CHAPTER 4

# ANALOG CIRCUIT FAULT-DIAGNOSIS

Today the Analog Circuit Fault-Diagnosis field is still in its infancy, as there is not yet any user-oriented publicly available computer program for circuit diagnosis purposes such as those for circuit simulation purposes. Research is being carried out by many in this area. The importance of the diagnosis problem is now widely recognized. Traditionally circuit theory is centered on two aspects only - *analysis* and *synthesis*. To these we may add a third and equally important aspect, namely *diagnosis*.

Depending on whether the circuit simulation takes place before or after testing process, analog circuit diagnosis methods are classified into two main categories, The simulation before test (SBT) and the simulation after test (SAT) approach.

Most of the techniques used in SAT are based on *parameter identification*, and most of these techniques are applicable only to linear circuits. But in this the numerical difficulty is enormous when large circuits are considered.

The present trend in SAT is not to solve for parameter values, but to simulate the circuit under different conditions and use some decision algorithms to locate the faulty components (the parameter values are not found). This branch of SAT approach may be called fault verification techniques.

In SBT, the circuit is simulated by using some network simulator like SPICE. For each fault the network is simulated and its signature or symptom (usually voltages of some nodes) is determined and is stored in a dictionary. Given a fault circuit, one can make test measurement and compare the result with those stored in the dictionary to identify the fault. The technique of SBT is called **Fault dictionary approach**. Similar approach is followed for trouble shooting of electronic equipment manually as given in service manuals where a step by step testing procedure is described, together with a table of symptoms and possible causes. That is in essence a fault dictionary approach. In SAT approach a great amount of computing power is required after the testing process. In sharp contrast, the SBT approach requires negligible amount of computing power after the test process to locate the faults. However a comparable or even greater amount of computing power is needed before the test process in order to compile the fault dictionary.

Fault Dictionary is mainly meant for hard failures only. It is incapable of diagnosing soft failures, that is element value drifts. However, the fault dictionary will remain a very valuable part of the overall scheme for the following reasons:

i) Catastrophic failures usually cause numerical difficulty in the parameter identification techniques, for example, if a resistor characterized by its conductance  $G$ , becomes short circuited, then the solution should be  $G \rightarrow \infty$ . Now if an iterative algorithm is used to solve the nonlinear

equations, the computer program, seeing the ever increasing value of  $G$ , might interpret the phenomena as a divergence of the algorithm and terminate (possibly giving the result of last iteration).

ii) The fault dictionary approach is well suited for diagnosing short circuits and open circuits. Although the initial effort to compile the fault dictionary is quite demanding, that task is done once for all.

iii) Statistics show that short circuits and open circuits account for about 70-80% of the faults in analog equipment.

Various types of inputs have been proposed for the fault dictionary method. These include the DC, AC and piecewise constant inputs. Unfortunately, most of these methods either are limited to linear networks or have not progressed beyond the feasibility study stage. At present the DC fault dictionary is the only one that is used in practice with some degree of success. For these reasons we have chosen the DC fault dictionary approach.

The DC fault dictionary approach consists of two distinct stages.

Stage 1: Pre-test analysis to compile the fault dictionary.

In this stage the analog circuit is simulated by a digital computer program under nominal as well as all preselected catastrophic faults. Judiciously chosen DC input voltages are applied. The induced DC voltages at a selected set of test nodes

are calculated. These voltages are then stored in the automatic test equipment (ATE) and constitute the fault dictionary.

State 2: Post-test analysis to identify the fault.

In this stage, measurements of test node voltages have been made on the circuit, and the measured values are compared with those stored in the fault dictionary. First, a fault detection algorithm is applied to determine whether the circuit is faulty at all. If the answer is affirmative, then the fault is identified by the application of some fault isolation algorithm (e.g. minimum sum of squared errors).

## CHAPTER 5

# APPLICATION OF NEURAL NETWORK TO FAULT DICTIONARY METHOD

Two approaches are followed while using Neural Networks in the field of Analog Circuit Fault-diagnosis. In both the cases the output of the neural network directly gives whether the circuit is faulty or not and also which component of the circuit is faulty.

1) **DIRECT APPROACH:** In the first approach the test nodal voltages of the circuit are applied as input to the neural network. The output of the neural network directly gives the fault of the component. First, the neural network is trained with different sets of test nodal voltages corresponding to different fault conditions and its corresponding address as output to the Neural Network.

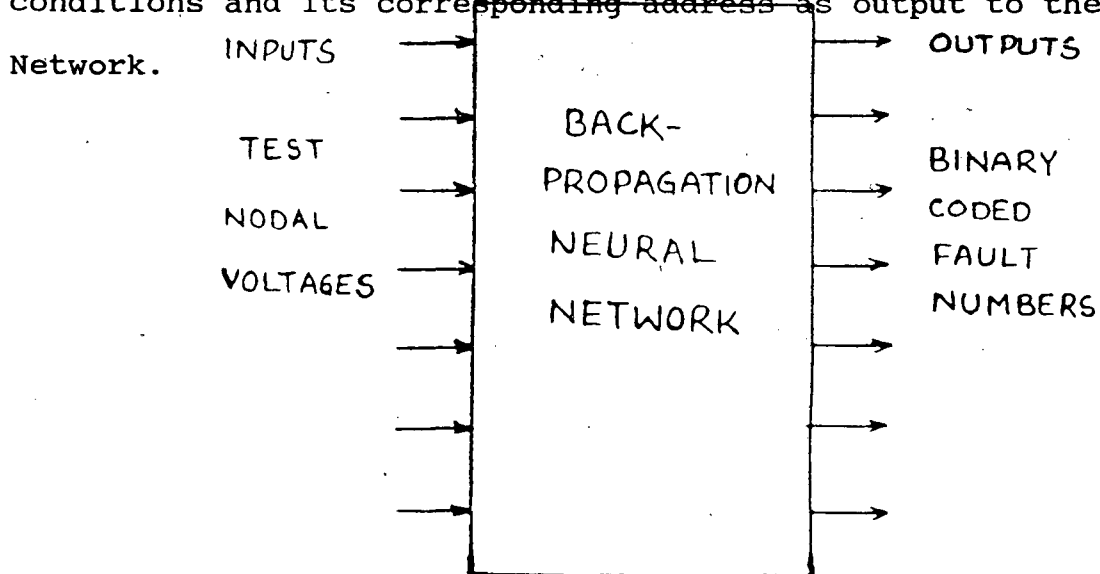


Figure 7 Block Diagram of Direct Approach.



For training of the network, the test nodal voltages are obtained through some network simulator CAD program. First the circuit is simulated in a network simulator, SPICE. Before we simulate the circuit for different sets of nodal voltages, we have to select some test nodes for this circuit such that those nodal voltages are obtained, when the circuit is in practical use. We also have to select some fault conditions which are likely to occur in practice. Then the circuit is simulated by using some network simulator for all those predetermined fault conditions and then the test nodal voltages are determined at these nodes for nominal and fault conditions. All these different sets of nodal voltages are formed into a fault dictionary. These must correspond to both nominal and fault conditions. The address of each fault is tabulated against its nodal voltages in the fault dictionary.

To test the network, we have taken simple example. It is having four test nodes and eight faults. The test nodal voltages for all the fault conditions (F1 to F8) along with the nominal condition(F0) are tabulated below.

**Table 2 Example test nodal voltages**

	F0	F1	F2	F3	F4	F5	F6	F7	F8
V1	5.0	7.0	7.4	7.3	7.2	9.6	9.7	9.8	5.2
V2	9.0	5.0	6.0	6.4	6.2	5.1	5.2	5.3	9.2
V3	9.5	6.0	6.1	6.2	8.0	6.3	6.4	5.3	4.0
V4	5.0	5.1	5.2	8.8	6.0	6.1	9.0	5.3	6.2

After getting the test nodal voltages these must be presented to some neural network which can produce the desired results. This is required because there are so many neural network paradigms available, and each neural network architecture and training system is better at some kinds of problems than others, and each network requires different types of data from the system for proper operation. For example, Back Propagation networks make wonderful mapping networks but may not be quite as good as counter propagation networks for some associative problems. So to select the suitable paradigm for the problem from the above paradigms the problem must be clearly defined.

There are so many paradigms available in the Neural Networks, suitable for different applications. To use any Neural Network it requires the information of the problem in terms of

1. What type of input, output are used. That is continuous values or binary values.
2. Type of learning. Whether it is Supervised or Unsupervised learning.
3. Type of application. That is Mapping, Associative Memory, Categorization, Temporal Mapping or Image Processing.

In this problem, Analog Circuit Fault Diagnosis, the input values used for training and recall are analog values, because the test nodal voltages of the circuit under test are (after normalization) to the network as input values. The output values are always binary values, which represent the address of various faults. The type of learning used is Supervised learning, be-

cause each set of input test nodal voltages are mapped to the corresponding binary coded addresses as output. The type of application is of Mapping type because the input voltages are mapped to output addresses.

By considering the characteristics of the problem, Analog Circuit Fault-Diagnosis, and the characteristics of different Neural Network algorithms it is found that Back Propagation algorithm can be used to solve the above problem effectively.

The Back Propagation algorithm is developed in C, in which various parameters can be changed. These parameters determine.

1. The rate at which the output error converges to the required error.
2. How frequently the network goes into local minima.

The main parameters that are to be considered for the above are Training rate parameter and Momentum term. As we have already seen, the rate of convergence increases with the increase of training rate parameter. but, as we increase the values of training rate parameter the network more frequently struck out in local minima. This can be overcome with the introduction of the Momentum term. But still there is threat from local minima. This local minima occurs when the output values of any node(s) reaches nearer to 1 or 0 instead of reaching 0 or 1. This can be explained as follows:-

In the output layer, the weight-change formula for the weight running from output unit  $j$  to a lower level unit  $i$  is:

$$W[i][j] = k * (t[j] - o[j]) * o[j] * (1 - o[j]) * o[i]$$

Where  $t[j]$  is target value,  $o[j]$  is the target value of the output unit,  $o[i]$  is the activation value of unit  $i$ , and  $k$  is the learning rate. The  $o[j] * (1 - o[j])$  term is the derivative of the activation function. A serious problem arises here in that when  $o[j]$  is close to 1 or 0 the term  $o[j] * (1 - o[j])$  is small and very little learning takes place. When some output is registering a 1 when it should be registering a 0 or registering a 0 when it should be a 1, it will take a very long time to undo this problem. To cope with this problem we have added the value 0.1 to output derivative term. Then it becomes

$$0.1 + o[j] * (1 - o[j])$$

Now the term never approaches zero. So the learning rate becomes very fast without any local minima.

We have trained the Back Propagation network directly with the normalized values of the above voltages as inputs and the binary coded fault numbers as out output numbers. The training patterns presented to the network are shown in the Table(3). The network could be able to map the input voltages to the output binary values in reasonable number of iterations.

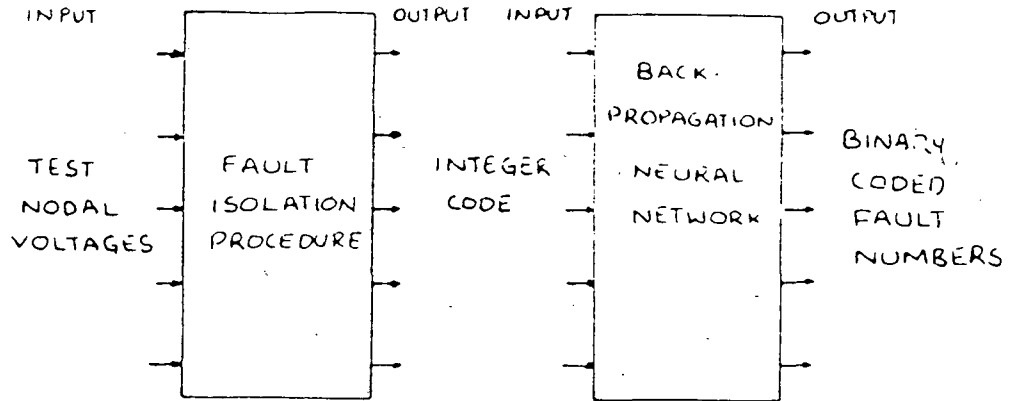
**Table 3** The training patterns

Input patterns	Output Patterns
.50 .90 .95 .50	0 0 0 0
.70 .50 .60 .51	0 0 0 1
.74 .60 .61 .52	0 0 1 0
.73 .64 .62 .88	0 0 1 1
.72 .62 .80 .60	0 1 0 0
.96 .51 .63 .61	0 1 0 1
.97 .52 .64 .90	0 1 1 0
.98 .53 .53 .53	0 1 1 1
.52 .92 .40 .62	1 0 0 0

After training of the network is completed, the network can be used recall mode. In this mode when we present network with the test nodal voltages, the network gives the fault number.

**2) INTEGER CODED APPROACH:** In this approach, instead of directly applying the test nodal voltages to the neural network for training as in the direct approach, we have used some isolation procedure (Lin and Elcherif) which not only isolates one fault from the other but also gives optionally minimum number of nodal voltages required for isolation. The isolation procedure produces as unique integer code for all the nodal voltages. Then this integer code is used as training patterns to the neural network as in the direct approach. The output of neural network

is the address of the fault. The block diagram is shown in the following figure.



**Figure 8** Block Diagram of Integer Code Technique

This isolation is required because of the ambiguities caused by the variations in values of the components. This variation in values of components is caused by the tolerance limits of the components. Because of the tolerance limits one set of nodal voltages may fall into another set of nodal voltages. This causes some ambiguity in recognizing or isolating one fault from other. So some form of isolation procedure must be used to diagnose these faults, i.e. to isolate or recognize one fault from other fault. For that we have used some logical procedure which not only isolates the faults but also determines how many number of minimum nodes which is not required. In this we can set our own tolerance limits according to the circuit components. Before we go into the details of how the isolation procedure works, what is ambiguity set and how it will be formed must be studied.

Consider a hypothetical case of a circuit with two test nodes and six faults. Suppose that the circuit of a CAD program yield the results shown in Table(4).

**Table 4** Voltages of two test nodes of hypothetical circuit

	Nom	F1	F2	F3	F4	F5	F6
V1	5.5	9.0	6.8	6.4	6.6	5.1	2.0
V2	4.0	8.0	5.0	5.2	7.8	7.6	5.0

These voltage values are obtained under the assumption of exact element values. In reality, the value of any element may vary within some tolerance range. If the measured value of V1 is 5.3 volts, we really cannot be certain whether the circuit is under nominal or fault 5 condition. Thus in this case, NOM and F5 form what is called an 'ambiguity set'. Suppose if we define the voltage of an ambiguity set to have a range of +/-7 volts about its center value and stipulate that different ambiguity set voltage ranges do not overlap in the present example the ambiguity sets obtained are tabulated as shown:

**Table 5** Tabulation of Ambiguity Sets

(node, ambiguity set)	Circuit Condition	Voltage Range
(1,1)	NOM,F5	4.6-6.0
(1,2)	F2,F3,F4	5.7-7.1
(1,3)	F1	8.3-9.7
(1,4)	F6	1.3-2.7
(2,1)	NOM,F6	2.8-4.2
(2,2)	F1,F4,F5	7.1-8.5
(2,3)	F2,F3	4.4-5.8

In this example test node 1 has 4 ambiguity sets and test node two has three ambiguity sets. Examination of the above table shows that fault F4 cannot be isolated from F2 and F3 if we use test node 1 only. Similarly, fault F4 cannot be isolated from F1 and F5 if we use test node 2 only. However, if both test nodes 1 and 2 are used then F4 can be isolated. This is because F4 is the only fault that occurs in both ambiguity sets (1,2) and (2,2). On the other hand the, faults F2 and F3 cannot be isolated even if both test nodes are used.

According to the above procedure, first ambiguity sets will be formed corresponding to the test nodal voltages. For example, take the test node 1 voltages used in the first procedure. One input vector and four test nodes (V1,V2,V3,V4) have been chosen. Test nodes have been determined by the use of circuit simulation



program. The origin of data is immaterial for the intended fault isolation. The voltages are given in Table (1).

For each test node we can define ambiguity sets, voltage ranges and circuit conditions. This process becomes very clear if we present the information of node 1 voltages in Table (1) in the form of plots as shown in Fig. 8 below. Circuit conditions that belong to the same ambiguity set correspond to the points in fig.8 that form a cluster. The ambiguity set table that is obtained is with the aid of the above figure is shown in Table(4)

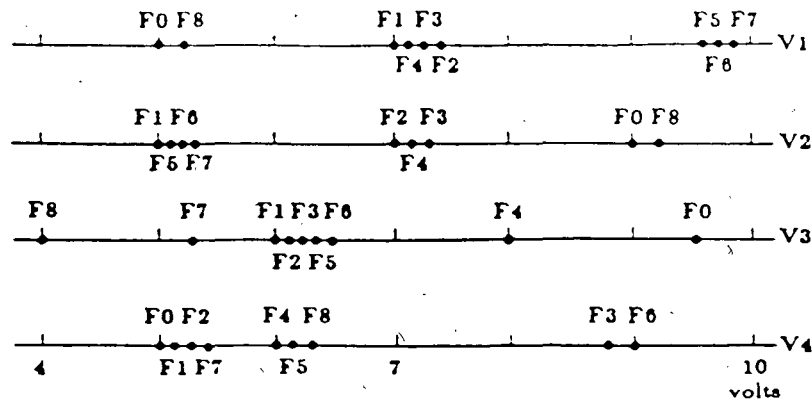


Figure 9. Formation of Ambiguity Sets.

**Table 6** Ambiguity sets' Contents and Voltage Ranges

(Node, Ambiguity Set)	Circuit Condition	Voltage Range
(1,1)	F0,F8	4.4-5.9
(1,2)	F1,F2, F3 F4	6.4-7.8
(1,3)	F5, F6,F7	9.0-10.4
(2,1)	F1,F5,F6,F7	4.5-5.6
(2,2)	F2,F3,F4	5.7-6.9
(2,3)	F0,F8	8.4-9.8
(3,1)	F8	3.3-4.4
(3,2)	F7	4.6-5.7
(3,3)	F1,F2,F3,F5,F6	5,8-6,9
(3,4)	F7	7.3-8.7
(3,5)	F0	8.7-10.2
(4,1)	F0,F1,F2,F7	4,5-5,5
(4,2)	F4,F5,F8	5.7-6.8
(4,3)	F3,F6	8.2-9.6

It is observed from the table that node 3 has five ambiguity sets, while nodes 1,2, and 4 each have three ambiguity sets. The range of each ambiguity set in Table (6) is determined in the following manner. First, the center of each cluster is taken to be the average of the two extreme values of the cluster. Next a range of +/-0.7 volt from the center is tentatively set. After the tentative ranges for all ambiguity sets have been calculated, a check is made to see whether the ranges of any two ambiguity sets (of the same test node) overlap. If so both ranges are

reduced by an equivalent, until a gap of 0.1 or 0.2 volt is obtained. After such revisions, the ranges are accepted for use. As an example, consider ambiguity sets (3,2) and (3,3) in Table (4). The second ambiguity set of node three has only one value 5.3 volts (correspond to F7), which is then also the centre value. Therefore, ambiguity set (3,2) has tentative range of 4.6 to 6.0 volts. The third ambiguity set of node three has two extreme values, 6.0 and 6.4 (corresponding F1 and F6, respectively), and hence a center value of 6.2 volts. Therefore, ambiguity set (3,3) has a tentative range from 5.5 to 6.9. These ranges overlap. So we decrease the upper boundary of set (3,2) from 6.0 to 5.7 and increase the lower boundary of set (3,3) from 5.5 to 5.8. Each range is reduced by 0.3 volts. And a gap of 0.1 volt has been created. This explains how the range from 5.8 to 6.9 is obtained for ambiguity set (3,3). Similar adjustments are made for all ambiguity set ranges.

#### **Fault Isolation By Intersection of Ambiguity Sets:**

Suppose some ambiguity set of some test node VJ contains only one circuit condition FK; then whether that circuit condition has occurred can be determined by measuring node voltage VJ only. In this case, we can say the fault FK has been isolated. For example, let the measured value of V3 is 8.3 volts. Then table 6 indicates that the value belongs to ambiguity set (3,2). Since it has only one element, namely F4, the circuit must be under the condition of fault 4.

On the other hand, if we measure one node voltage only and obtain  $V_3 = 6.5$  volts, we will not be able to isolate the fault. Table(6) indicates that 6.5 volts belongs to ambiguity set (3,3), and the circuit may be under any one of the following conditions: F1, F2, F3, F5, or F6.

Let us see how the use of additional test nodes can help in isolating the faults. First we have constructed an "ambiguity sets intersection table" as shown in Table(7).

**Table 7** Intersection of ambiguity sets of V3 and V4.

V3	V4		
	F0, F1, F2, F7	F4, F5, F8	F3, F6
F8	@	F8	@
F7	F7	@	@
F1, F2, F3, F5, F6	F1, F2	F5	F3, F6
F4	@	F4	@
F0	F0	@	@

In table 7 the row headings contain ambiguity sets of node 3, whereas the column headings contain those of node 4. Each (i,j) block of the matrix contains the result of intersection of ambiguity sets (3,i) and (4,j) with @ denoting a null set. For example, the content of the (3,1) block is determined as follows:

amb. set (3,3) amb. set (4,1)

$$=(F1, F2, F3, F5, F6) \quad (F0, F1, F2, F7) = (F1, F2)$$

We note that this intersection yields a total of seven ambiguity sets: (F8), (F7), (F,4), (F5), (F,1 ,F2) , (F3, F6). In particular, F5 has been isolated with the addition of test node 4. For example, if the measured values are  $V_3 = 6.5$  volts and  $V_4 = 6.2$  volts, then the circuit condition belongs to ambiguity sets (3,3) and (4,2), according to Table 7. But Table 7 shows that the only fault which occurs in both ambiguity set (3,3) and (4,2) is F5. Therefore we conclude that the circuit is under condition F5.

With two test nodes  $V_3$  and  $V_4$ , we have isolated faults F4, F5, F7 and F8 - a 50% isolation. We note that F1 has not been separated from F2, nor has F3 been separated from F6. More test nodes must be used to achieve a higher percentage of isolation.

Suppose that we decide to add test node  $V_1$  with the hope of resolving the ambiguity in (F1, F2) and also in (F3, F6). The effect of adding  $V_1$  can be seen from Table 8.

**Table 8** Intersection of ( $V_3, V_4$ ) with  $V_1$

(V3, V4)	VI		
	F0, F3	F1, F2, F3, F4	F5, F6, F7
F1, F2	@	F1, F2	@
F3, F6	@	F3	F6
5 Singletons		5 Singletons	

An ambiguity set with only one element is called a Singleton. Observe that in Table 8 F3 and F6 have been isolated, but F1 and F2 have not. Let us further add test node 2. The intersection operation is shown in Table 9.

**Table 9** Intersection of (V3,V4,VI) with V2

(V3,V4,VI)	V2		
	F1,F5,F6,F7	F2,F3,F4	F0,F8
F1,F2	F1	F2	@
7 Singletons	7 Singletons		

The faults F1 and F2 are now isolated. This indicates that if we use all of the four test nodes, we can have 100% fault isolation. but the result does to imply that one must use all four test nodes to achieve 100% isolation. For example, if the node to be added after (V3,V4) is V2 instead of VI we will have the intersection table shown in Table 10.

**Table 10** Intersection of (V3,V2) with V2

(V3,V4)	V2		
	F1,F5,F6,F7	F2,F3,F4	F0,F8
F1,F2	F1	F2	@
F3,F6	F6	F3	@
5 Singletons	5 Singletons		

The result indicates that 100% fault isolation can also be achieved without test node 1.

### **Reduction of the Number of Test Nodes**

In the general case, the determination of a minimum set of test nodes to achieve the highest percentage of isolation is a very time-consuming process for large circuits. Fortunately, for practical applications, we need not insist on getting the theoretical minimal number of test nodes. Any near-minimum solution will serve our purpose if the solution is simple. In other words, a heuristic method might be more useful for solving practical problems.

We present two heuristic procedures for reducing the number of test nodes.

#### **PROCEDURE 1**

**Step 1.** Select the node that has the largest number of ambiguity sets. If a tie occurs, arbitrarily select one among them.

**Step 2.** Select the next node whose intersection with previously selected nodes will result in the largest number of ambiguity sets. In case of a tie, arbitrarily select one.

**Step 3.** If the number of the resultant ambiguity sets is equal to the number of circuit conditions stop. Otherwise go to step 2.

Consider the previous example of Table 6. Node 3 is selected first, because it has five ambiguity sets, the largest among V1, V2, V3 and V4. Next, one node from V1, V2 and V4 is to be selected. The intersection of V3 with V4 yields a total of seven ambiguity sets (see Table 7). Similarly, V3 V1 yields six sets, and so does V3 V2. Therefore V4 is chosen as the second test node. The third test node is to be selected from V1 and V2. Now (V3, V4) V2 yields nine ambiguity sets (see Table 10), whereas (V3 V4) V1 yields only eight sets (see Table 8). Therefore V2 fault isolation is achieved. Test node V1 is seen to be redundant.

Procedure 1 will often lead to a near-optimum selection of test nodes. But even this procedure may be too time-consuming for large circuits. This is because that in step 2 every node has to be intersected with the previously selected group of nodes. Therefore, a further simplified procedure is given below.

## PROCEDURE 2

**Step 1.** Select the node that has the largest number of ambiguity sets. If a tie occurs, arbitrarily select one among them.

**Step 2.** In the remaining nodes, tentatively select one having the largest number of ambiguity sets. If a tie occurs,



pick any one among them. Now obtain the intersection of this node, VJ, with the previous selected group of nodes. If the intersection increases the total number of ambiguity sets, then select VJ as a test node. Otherwise, disregard VJ.

**Step 3.** If the number of resultant ambiguity sets is equal to the number of circuit conditions, stop. Otherwise, go to step 2.

Let us illustrate 2 with the previous example of Table 6. In step 1, we select V3 since it has five ambiguity sets, the largest among V1, V2, V3, V4. Each of the remaining nodes, V1, V2, and V4, has three ambiguity sets. According to step 2, we may arbitrarily pick one. Suppose that node 1 is picked. The intersection V3 V1 has six ambiguity sets, one more than that of V3 alone. Therefore, V1 is selected as the second test node. The third test node is to be arbitrarily selected from V2 and V4. Suppose that we tentatively pick V2.

The intersection V2 (V3 V1) has seven ambiguity sets, one more than (V3, V4). Therefore V2 is selected as the third test node. There is a total of nine circuit conditions (one nominal plus eight faulty conditions). Since  $7 < 9$ , we go through step 2 another time and include V4 as the fourth test node. As shown previously, with V1, V2, V3 and V4 all selected as test nodes, we achieve 100% fault isolation. But this clearly is not an optimum solution, since three test nodes V3, V4, and V2 will achieve the same goal.

In most cases procedure 2 will produce a satisfactory solution to the reduction of test nodes. Since its computational effort is much less than that of procedure 1, we have implemented procedure 2 in our present computer program.

Compilation of Fault-Dictionary: Once the test nodes have been selected using Procedure 1 or Procedure 2 described above, we can determine a unique integer code for each circuit condition (both fault & nominal). For this we need only the ambiguity set table as given in table 6. Each fault  $F_j$  belongs to exactly one ambiguity set of every test node.

This information is tabulated as shown below:-

**Table 11 Integer Codes for All Faults**

Circuit Condition	V3	V4	V2
F0	5	1	3
F1	3	1	1
F2	3	1	2
F3	3	3	2
F4	4	2	2
F5	3	2	1
F6	3	3	1
F7	2	1	1
F8	1	2	3

Here we assume that V3, V4, and V2 have been selected as the test nodes. As an illustration, consider the case 3,1,2 for F2.

This means that F2 is in third ambiguity set of V3, the first set of V4, and second set of V4. We can get other codes in the same way.

After obtaining the integer codes as above, Integer coded fault dictionary is formed to train the neural network. The fault dictionary or the training patterns contain the normalized values of integer code as input values. The integer coded fault dictionary that is used for training corresponding to the table 6 & Table 11 is shown in the following table 12.

**Table 12 Integer Coded Fault Dictionary**

Input Patterns			Output Patterns
5	1	3	0 0 0 0
3	1	1	0 0 0 1
3	1	2	0 0 1 0
3	3	2	0 0 1 1
4	2	2	0 1 0 0
4	2	1	0 1 0 1
3	3	1	0 1 1 0
2	1	1	0 1 1 1
1	2	3	1 0 0 0

The Backpropagation network is trained with the above patterns. After the training is completed the network can be used in run mode. In this mode first the test nodal voltages are applied to the isolation procedure, which gives a unique integer code. This integer code is applied as input to the BP network which gives the fault number as the output.

In our present approaches, there is no distinction between fault detection and fault isolation i,e, requires no additional computational effort. We include the nominal circuit (designated by F0 or NOM) in the ambiguity set manipulations. Separation of a fault FJ fro NOM amount to fault detection, while separation among faults is the usual fault isolation.

## CHAPTER 6

# RESULTS AND CONCLUSIONS

For the purpose of using BP network in analog circuit fault diagnosis, I have developed a program for BP algorithm in 'C'. This program will accept different neural network parameters, which gives the flexibility to the user to have his own network configuration and he can define his own network parameter values, which determine the rate of convergence of the network. We can vary the number of layers, no. of nodes in each layer. We can also vary the learning rate parameter, and the momentum term. As we increase the learning rate, the rate of convergence increases. Similarly as we increase the momentum term, the rate of convergence increases. Some of the results that are obtained by varying learning rate parameter and momentum term are presented in the tables 12 & 13 respectively. From the table we can observe that the rate of convergence is proportional to momentum term and learning rate parameter.

In the direct approach, in which we train the BP network directly with test nodal voltages; if the tolerance values are in the range  $\pm 0.1$  volt, we have trained the network with input values having  $\pm 0.1$  volt error. After training the network with error, we RUN the network for all possible patterns with  $\pm 0.1$  volt error. It is giving correct results for all patterns. If the tolerance value is .2 volts, we have used  $\pm 0.2$  volt

error. If the tolerance is 0.3 volts,  $\pm 0.3$  error is used in the training patterns. In all the above cases it is found that the network is working satisfactorily. By setting the values of error in the training patterns, we can set the tolerance limits of the components of the circuits. The main drawback of this procedure is all test nodal voltages are to be considered for isolation.

In the integer code approach, the isolation procedure not only isolate different faults, but also determine the minimum number of nodal voltages required for isolation. The results that are obtained for (4 nodes, 9 faults) network are given in chapter 5. In this procedure also we can set different values of tolerance limits. In this procedure, the neural network need not be trained with error values, as in first procedure. The isolation already takes place in the isolation procedure.

**Table 12** Results for various values of Momentum with Learning Rate = 0.8 and RMS Error = 0.1

---

MOMENTUM	NO. OF ITERATIONS
0.1	2945
0.2	2704
0.3	2500
0.4	2325
0.5	2171
0.6	2045
0.7	1949
0.8	1872
0.9	1825
1.0	1776

---

**Table 13** Results for various values of Training Rate with Moments  
= 0.9 RMS Error=0.1

---

LEARNING RATE	NO. OF ITERATIONS
0.3	4548
0.4	3419
0.5	2742
0.6	2316
0.7	2026
0.8	1825
0.9	1678
1.0	1588
1.5	1490
2.0	1039
2.5	872
3.0	767
3.5	727
4.0	711
4.5	617

---



## REFERENCES

1. Richard Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, April 1987.
2. James L. McClelland and David E. Rumelhart, "Explorations in parallel distributed processing A handbook of models, programs, and exercises", pub. The MIT Press.
3. Maureen Caudill, "Avoiding the great backpropagation trap", Don Tvetter, "Getting a fast break with backprop", Jessica Keyes, "Getting Caught in Neural Network" A.I. Expert, JULY 1991.
4. Alianna Maren, Craig Harston, Robert Pap, "Hand Book of Neural Computing Applications", Academic Press, INC.
5. Igor Aleksander, "Neural computing architectures", Pub. North Oxford Academic /
6. John J. Hopfield, "Artificial Neural Networks", IEEE Circuits and Devices Magazine, Sept., 1988
7. Maureen Caudill, "Neural networks Primer Part 1&2, AI Expert Dec., 1987
8. Tom J. Schwartz "Parables of Neural Networks", AI Expert, Dec., 1989.
9. Philip Treleaven, Macro Pacheco, Marley Vellasco, "VLSI Architectures of Neural Networks", IEEE Macro, Dec., 1989

10. P.M. Lin and Y.S. Elcheriff, "Analog circuits fault dictionary - New approaches and implementation" Int. Journal of circuit theory and applications vol. 13, pp149-172, 1985.
11. P. Duhamel and J.C. Rault, "Automatic test generation techniques for analog circuits and systems - A review", IEEE Transactions on Circuits and Systems, Vol. CAS-26, pp.410-440. July 1979.
12. Samuel N. Stevens and P.M. Lin, "Analysis of Piecewise-Linear Resistive Networks".

IEEE Transactions on Circuits and Systems.

13. Walter Hochwald and John D. Bastian, "A D.C. Approach for Analog fault dictionary determination", IEEE Transactions on circuits and systems", July 1979.
14. William J. McCalla, "Fundamentals of computer aided circuit simulation" pub. Kluwer Academic Publishers.
15. Heinz H. Schreiber, "fault Dictionary Based Upon Stimulus Design", IEEE Transactions on circuits and Systems, July 1979.

# APPENDIX I

## THE BOLTZMANN MACHINE

The Boltzmann machine is structurally and dynamically similar to the Backpropagation algorithm, and which can perform similar tasks. It differs from perceptron like networks in that they use an energy state optimization method derived from statistical considerations, rather than a Delta rule. This approach to learning allows them to perform optimization tasks as well as pattern recognition.

The Boltzmann machine has both conceptual and performance similarities to Back-propagation network. Both have hidden nodes, and both need to be trained to match input patterns to previously determined categories. Both networks were developed at about the same time, and were applied to the same type of problems. But the Boltzmann machine is not as popular as Backpropagation.

There are several reasons why the Boltzmann machine never achieved the popularity of the back-propagation network. First, the performance of the two networks was very similar, so there was no need to favour one over the other because of performance or capability considerations. Thus, the main criteria for selecting a network became ease of learning and ease of using it for a particular application. The back-propagation network was easier to both learn and to use.

The back-propagation concepts and learning rules come out of a fairly direct approach of minimizing an energy using differential calculus. In contrast, the Boltzmann machine concepts come out of statistical mechanics, expressed in either information theoretic and/or statistical thermodynamic formalisms. Not many people in the neural networks field have the background to understand either of these approaches. Its learning rule takes much longer to write down than the backpropagation learning rule, as it is more complex, and takes more time when training a network. As a result an explosion of applications of the back-propagation method has occurred, while interest in the Boltzmann machine has dwindled. But the problem with the back-propagation network is sometimes the connection weights take on values which trap the network in a local minimum. The goal of using the simulated annealing method is to avoid such local minima trapping.

The learning method for Boltzmann machines is called **Simulated Annealing**. The simulated annealing approach to learning is sometimes called stochastic or statistical, because it relies on generating random events and evaluating their effect in terms of desired goals and probability distributions.

The essence of the simulated annealing learning law is that we make an analogy between the energy state of the entire network and the energy state of a physical solid which is slowly cooled. We will pretend that each individual unit in the solid (atoms, molecules, etc.) can take on one of two possible states: a high-energy state(1) or a low energy state(0). The free energy ( a

term for thermodynamics) of the solid is a combination of two factors; the combined energies of the individual units and the negative of the entropy (the disorder among the units) times the temperature of the solid.

The key to this analogy is that whether a solid is in a high-energy or low-energy state, it is always in equilibrium. Equilibrium is found at the lowest point on the free energy curve. If we lower the temperature slowly, the solid will have an opportunity to find this lowest point, even if there are many other shallow minima. We want to accomplish the same thing with the Boltzmann machine network. We want to create some sort of *artificial temperature* so that as we slowly reduce this *temperature*, the connection weights take on values that put the network at the global minimum for the energy curve, and don't get trapped in one of the local shallow minima.

Let's make a physical analogy to this energy surface. Suppose that we hold in our hands a large tray of firm plastic that has some pockets or indentations in it. This would be like the bottom half of one of those cartons of eggs. Now, some of those indentations are very shallow, and some are very deep. This represents the energy surface (which is really multidimensional). Let's suppose that we have a single egg-sized ball in this tray. Although this is just a single, one, we want to think of it as corresponding to the entire collection of weights that we want to optimize. Our goal is to shake the tray so that the ball goes into the deepest of the indentations.

We could shake the tray continuously, and hope that just by keeping this up, the ball will fall into the hole. But how will we know when the ball has reached the deepest indentation? This can be tricky, especially if we don't know in advance which of the indentations is deepest. So we adopt a strategy. We shake the tray, pretty hard at first. Shake it some more. Shake some more, a bit more gently. Then more gently yet. Finally, just the barest of tremors. Now, let's look into the tray. Chances are, the ball is in the deepest pocket. That's because as we shook the tray more and more gently, the ball could be shaken into out of shallow indentations, but not the deepest ones. There's no absolute guarantee that it would have fallen into the very deepest one in the tray, but it is very likely to fall into in one of the deepest.

We want to take this idea and apply it to finding optimal connection weights in a neural network. To do this, we return to our original analogy, that of a solid which will undergo simulated annealing. We need to see how the *shaking of the tray* strategy fits in with annealing a solid, because this leads directly to the neural network.

Recall that our hypothetical solid was composed of identical units, each of which could be in one of two possible energy states, high(1) or low(0). The proportional number of units in each state is a function of temperature. At high temperatures, there are more high-energy units than there are at low temperatures. This proportion can be expressed as a

probability distribution. This temperature dependent probability distribution will be a key factor in the simulated annealing process. *Shaking the tray* corresponds to setting the temperature of the solid. *Shaking the tray hard* corresponds to high temperature, with high kinetic energy for the ball in the tray. *Shaking the tray gently* corresponds to low temperature. The ball has low kinetic energy, and cannot easily move out of a deep pocket. The pockets in the tray correspond to energy minima in the free energy surface. There are numerous local minima, but relatively few deep minima. These minima represent equilibrium states for a solid, and correspond to optimal connection weight values in the neural network.

## APPENDIX 2

### PARAS - PARAM

PARAM is a statically reconfigurable multicomputer with the transputer as the processing nodes. It has a switch and an exchange to provide reconfigurability of nodes depending upon specific communication demands of an application. Further provision for allocating the availability nodes to many users gives multi-user support. Optionally, PARAM can have PFS(Parallel File System), for fast and high capacity secondary storage.

These machines are best described as consisting of two parts: the front end host and back end compute engine. The back end compute engine is usually a network of transputers whereas the host can be either of PC, a SUN, or a VAX. All the development work is done on the host machine and the bootable image of the parallel program is downloaded to the back-end network for actual execution.

The T800 transputer incorporates a floating-point unit (FPU) together with a 30 MIPS (peak) CPU, 4 fast serial communication links and 4 Kbytes of fast SRAM, all on a single chip. The links can support 2.4 Mbytes/sec bidirectional communication on each of the four links, concurrently with the operation of the FPU and the CPU. The concurrent operation of the FPU and the CPU gives a sustained rating of 1.5 Mflops at a processor speed of 20 MHz. PARAS is a software development environment for message passing



machines built around transputers. These message passing machines are general purpose MIMD (Multiple Instruction Multiple Data) machine which employ a network of processing nodes to solve a single problem. Processing nodes execute sequential programs asynchronously and co-operate by sending data in the form of messages. The collection of interacting sequential programs which collectively perform a single job is called a *parallel program*. The development of such a program is greatly assisted by the PARAS environment.

PARAS has program development tools like compilers (for C and FORTRAN), linker, configurer, collector, librarian and decoder to convert the source code into executable code. Besides, it provides a rich and powerful runtime environment (Concurrent Runtime Environment) CORE. CORE includes facilities for message communication, process management, file and screen I/O, graphics and other miscellaneous services.

A typical (parallel) application program under PARAS consists of a set of processes working together to solve a particular problem. These processes may either be executing on different nodes of the network or on the same node. Some process, after doing a portion of the job, may need to give the result of its computation to other processes. Such a communication is accomplished by explicitly passing a message from one process to another.

The communication model of PARAS is based on abstract objects called *ports*. A port is basically a repository for

messages. The concept of a port is very similar to that of a letter-box attached to a house where the letters addressed to that particular house are delivered. Any person who knows the address of the house can post a letter. All the letters so received would be lying in the letter-box till cleared by the house-owner.

Similarly, a process owns a port that it creates. A sender process, which has to send to the receiver, knows the descriptor of a receiver's port. It sends its message with the descriptor as address. All messages to a particular port get queued up for the receiver to receive in first-in-first-out order. Any process which knows the receiver port's descriptor (address) can send messages to it.

The placement of tasks to processors has to be done at the configuration stage, before execution of the program. More than one task can be placed on the same processor if there is sufficient memory available. When the program starts execution, all the tasks start their execution simultaneously on their respective processors.

When a task starts execution, it has a single line of control and executes sequentially. This sequence of code in execution is called a *thread*. A thread is the actual active entity within a task. The starting thread of a task, is called the *main thread*. The user might wish to achieve more concurrency within a task on a particular processor. Since tasks have to be statically defined, a facility for dynamically spawning threads

within a task is provided. A processor is time-shared by the threads executing on it. Threads of the same task can share global resources like global variables (common block variables), file buffers, etc. consequently, threads of the same task can share the Global variables. This should be contrasted with two tasks placed on the same processor. Although they run on the same node they cannot share variables. Concurrency on a single processor can be achieved either by placing more than one task on that processor or by dynamically spawning more threads within the task.

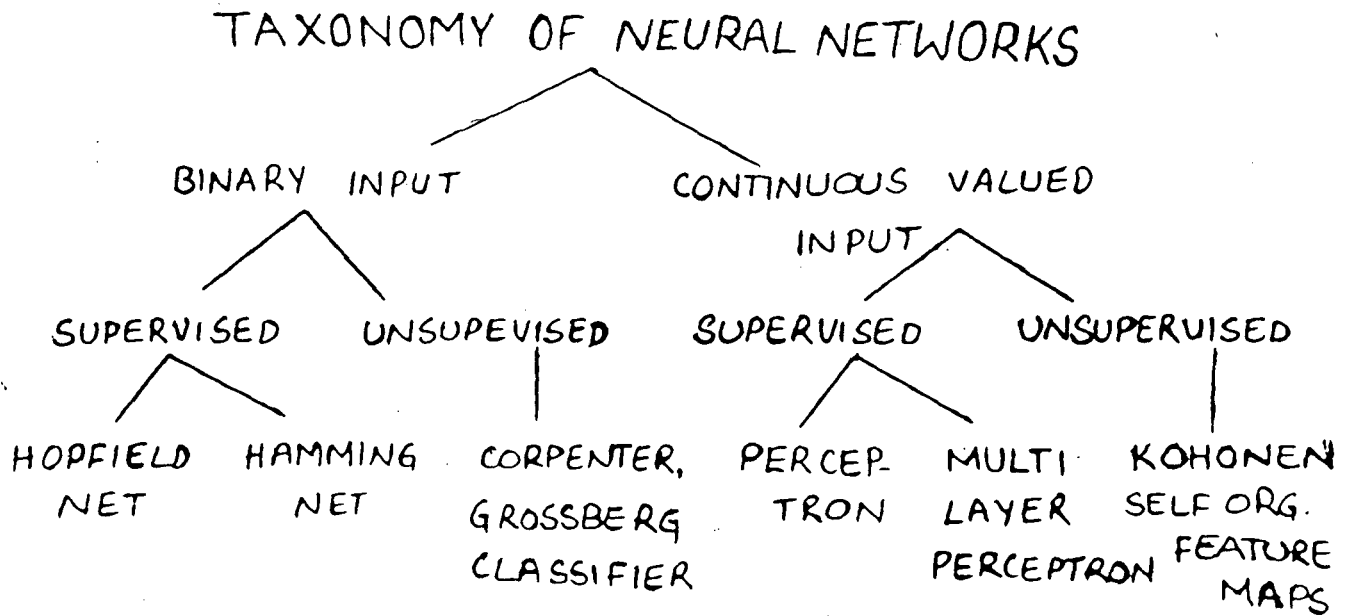
The Program Development Environment comprises a set of tools which include the compilers, the linkers, the configurer, the collector, the debugger, the decoder and the librarian. These tools run on the front-end host machine (with UNIX or DOS). Hence, the complete development can be done on the more familiar host and only for executing the program the user needs to go to the parallel machine.

Since PARAS allows writing parallel programs in a hardware-independent manner, when the program has to run on a specific hardware, it has to go through a state of configuration. Here, in the configuration file the actual details of the hardware in terms of number of processors and the way their communication links are connected are specified. This file and the linked modules for the tasks are given as input to the configurer. The different compiled and linked units of the parallel program are given together with a configuration specification file to the

configurer, which produces a binary file to be given as input to the collector. The collector, in turn, produces a bootable image of the parallel program for the specified machine. The bootable file has the code to be executed by the tasks alongwith the code to load them onto the different processors to the networks as per the user specification. This file is given to a server (running on the host) for loading the code onto the processors and executing it.

Thus, the application program is developed into an executable form (as a bootable file) using the following four basic tools. 1. Compiler 2. Linker 3. Configurer 4. Collector.

TAXONOMY



## Some Applications of Well-Known Networks: I

Network	Year Introduced	Inventors/ Developers	Primary Applications	Advantages	Disadvantages	Most Relevant Chapter
ADALINE/ MADALINE	1960	B. Widrow	Adaptive signal filtering, adaptive equalization	Fast, easy to implement, can be done using analog or VLSI circuitry.	Linear relationship between input & output assumed. Only linear separable classification spaces possible.	7
Adaptive Resonance Theory	1983	G. Carpenter & S. Grossberg	Pattern recognition	Able to learn new patterns, form new pattern categories, and retain learned categories.	Nature of categorical exemplars may change with learning.	11
Back-Propagating Perceptrons: Basic	1974-1986	P. J. Werbos, D. Parker, D. Rumelhart	Pattern recognition, signal filtering, noise removal, signal/image segmentation, classification, mapping, adaptive robotic control, data compression	Fast operation. Good at forming internal represent- ations of features in input data or classification and other tasks. Well studied. Many successful applications.	Long learning time.	7
Recurrent	1987	Almeida, Pineda	Robotic control, speech recognition, sequence element prediction	Best network so far for classifying, mapping time-varying information.	Complex network, may be difficult to train and optimize.	17
Time Delay	1987	D. W. Tank & J. J. Hopfield	Speech recognition	Performance equivalent to best conventional methods, faster operation.	Fixed window of temporal activity represented, responds awkwardly to differences in scale of input.	17
Functional-Link Network	1988	Y. H. Pao	Classification, mapping	Only two layers (input & output) needed; faster to train.	No clear way to identify functions for functional links.	15
Radial Basis Function Network	1987- 1988	Multiple Researchers	Classification, mapping	Network with single hidden layer of RBF neurons performs equivalent to basic BP network with two hidden layers.	Not yet known.	15
Back- Propagation of Utility Function Through Time	1974	P. J. Werbos	Maximize performance index or utility function over time, neurocontrol (e.g. robotics)	Most comprehensive neural approach for model-based prediction and/or control.	Can use only alter differentiable model identified, must adapt off-line if model is dynamic, and assumes model is exact.	22

## Some Applications of Well-Known Networks: II

Network	Year Introduced	Inventors/ Developers	Primary Applications	Advantages	Disadvantages	Most Relevant Chapter
Bidirectional Associative Memory	1987	B. Kosko	Heteroassociative (content-addressable) memory	Simple, clear learning rule, architecture, & dynamics. Clear proof of dynamic stability.	Poor storage capacity, poor retrieval accuracy.	11
Boltzmann Machine, Cauchy Machine	1984, 1986	G. Hinton, T. Sejnowski, D. Ackley, H. Szu	Pattern recognition (images, sonar, radar), optimization	Able to form optimal representation of pattern features. Follows energy surface to obtain optimization minima.	Boltzmann machine: very long learning time. Cauchy machine offers faster learning.	8
Boundary Contour System	1985	S. Grossberg, E. Mingolla	Low-level image processing	Biologically-based approach to excellent segmentation.	Complex, multilayered architecture.	12
Brain-State-in-a-Box	1977	J. Anderson	Autoassociative recall	Possibly better performance than Hopfield network.	Incompletely explored in terms of performance and applications potential.	9
Hopfield	1982	J. Hopfield	Autoassociate recall, optimization	Simple concept, proven dynamic stability, easy to implement in VLSI.	Unable to learn new states (fixed weights for discrete Hopfield), poor memory storage, many spurious states returned.	9
Learning Vector Quantization	1981	T. Kohonen	Autoassociative recall (pattern completion given partial pattern), data compression	Able to self-organize vector representations of probability distributions in data. Rapid execution after training is completed.	Unresolved issues in selecting numbers of vectors to use and length of time for appropriate training. Slow training.	10
Neocognitron	1975-1982	K. Fukushima	Recognition of hand-drawn characters and other linear-outline figures	Able to perform scale, translation and rotation invariant pattern recognition.	Requires many processing elements and layers, complex structures, scaling issues for real-world use still need to be resolved.	12
Self-Organizing Topology-Preserving Map	1981	T. Kohonen	Complex mapping (involving neighborhood relationships), data compression, optimization	Able to self-organize vector representations of data with a meaningful ordering among the representations.	Unresolved issues in selecting numbers of vectors to use and length of time for training. Slow training.	10