DOE/ER/75700--T2

DE93 010303

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Nuclear power plant status diagnostics using a neural network with dynamic node architecture

by

Anujit Basu

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Mechanical Engineering
Major: Nuclear Engineering

Approved:

In Charge of Major Work

For the Major Department

For the Graduate College

Members of the Committee:

FG02-92ER75700

Iowa State University
Ames, Iowa
1992

Copyright © Anujit Basu, 1992. All rights reserved.

The Government reserves for itself and others acting on its behalf a royalty free, nonexclusive, irrevocable, world-wide license for governmental purposes to publish, distribute, translate, duplicate, exhibit, and perform any such data copyrighted by the contractor.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

TABLE OF CONTENTS

ACKNOWLEDGEMENTS viii

CHAPTER 1. INTRODUCTION 1

Nuclear Power Plant Safety 1

Neural Networks 2

Problem Statement 4

CHAPTER 2. ARTIFICIAL NEURAL NETWORKS 6

Introduction 6

Using a Neural Network: Training 9

Backpropagation 10

The Delta Rule 10

Backpropagation Neural Networks 17

CHAPTER 3. DYNAMIC NODE ARCHITECTURE 29

Introduction 29

Architecture Optimization : Existing Methods 30

Dynamic Node Architecture Theory 32

Importance of a Node 33

Demonstration of the DNA Algorithm 35

The Exclusive-Nor Problem 35

The 8-to-1 Decoder Problem 36

The Probability Density Function Separator Problem 37

Recall performance 40

Conclusions from the above experiments 41

CHAPTER 4. THE PROBLEM AND ITS SOLUTION 43

Introduction 43

Data Collection and Processing 44

Normalization of the Data 44

Final Data Set 47

Developing the Advisor 48

Training 48

Results 49

CHAPTER 5. CONCLUSIONS 59

Possible Future Work 60

BIBLIOGRAPHY 62

APPENDIX A. TRANSIENT DESCRIPTIONS 66

Malfunction RR15A 66

Malfunction FW17A 67

Malfunction MS14 68

Malfunction HP05 69

Malfunction FW02 70

Malfunction MS03 71

Malfunction MC01 72

APPENDIX B. COMPUTER CODES 73

BPK.FOR 73

DNA.FOR 82

TEST.FOR 95

APPENDIX C. SAMPLE DATA POINTS 98

LIST OF TABLES

Table 3.1: Exclusive-nor training data 35

Table 3.2: Dynamic node architecture training history for the exclusive-nor problem 36

Table 3.3: 8-to-1 decoder training data 37

Table 3.4: Dynamic node architecture training history for the 8-to-1 decoder problem 38

Table 3.5: Probability density function separator training data 40

Table 3.6: Dynamic node architecture training history for the probability density function separator problem 40

Table 3.7: Comparative recall performance of neural networks derived by DNA and FNA schemes 41

Table 4.1: The eighty-one plant variables used 45

Table 4.2: Training information of the advisor 49

Table 4.3: Network output, simulation time, time to scram, and time to diagnose for the seven transients 58

Figure 4.6: Recall error for rupture of main steam line 'A' inside primary containment (MS03A) 56

Figure 4.7: Recall error for trip of main circulation pump 'A' (MC01A) . 57

LIST OF FIGURES

Figure 2.1: A Three Layered Neural Network 8

Figure 2.2: Newton's Method 14

Figure 2.3: A Detailed Node 19

Figure 2.4: Feed Forward Activation Flow 20

Figure 2.5: Sigmoid Function 22

Figure 2.6: Delta Rule for Output Layer Nodes 25

Figure 2.7: Derivative of the Sigmoid Function 26

Figure 2.8: Generalized Delta Rule for Hidden Layer Nodes 28

Figure 3.1: The probability density functions used in the pdf separator problem 39

Figure 4.1: Recall error for recirculation loop 'A' rupture (RR15A) . . . 51

Figure 4.2: Recall error for main feedwater line 'A' break inside primary containment (FW17A) 52

Figure 4.3: Recall error for loss of feedwater heater (MS14) 53

Figure 4.4: Recall error for HPCI steam supply line break in HPCI room (HP05) 54

Figure 4.5: Recall error for trip of condensate pump 'A' (FW02A) 55

ACKNOWLEDGEMENTS

I wish to thank the Duane Arnold Energy Center for their cooperation in providing information and use of their control room simulator. Without their help, this thesis would not have been possible. I would also like to thank the faculty, staff and fellow students in the Nuclear Engineering program who made the past two years a very enjoyable learning experience. Their help is much appreciated. I am greatful to the Nuclear Engineering program for supporting me through my two years of graduate study. Special thanks to Dr. Eric Bartlett who has been both friend and guide, encouraging me through this endevour. I thank Dr. Danofsky and Dr. Udpa for being on my program committee. Last, but not least, I wish to thank my parents who inspired me to search out new avenues in the persuit of higher education.

CHAPTER 1. INTRODUCTION

Nuclear Power Plant Safety

The safe operation of a nuclear reactor in a power plant is of utmost importance to the nuclear engineering community and quite vital to creating a positive attitude towards nuclear energy among the rest of the society. This thesis is an attempt to demonstrate how Artificial Neural Networks (ANNs) can increase the operational safety of nuclear reactors by being the basis of a fault diagnostic system in a power plant. It is hoped that neurocomputing, as the science of neural networks is sometimes called, will provide a better approach to recognizing and classifying operational transients at a nuclear power plant.

Most power plants currently employ automatic safety systems that allow the plant to operate within a predefined normal operating parameter space. These systems check to see if the plant status conforms to the preassigned safety limits of the various plant variables. As the plant enters into an abnormal condition, indications of plant variables exceeding the normal range causes the safety systems to either trigger a scram that automatically shuts down the reactor or notify the operators through some alarms or indicators. The sequence of events leading to the plant shutdown are analyzed later by technical support teams located both on and off site. Use of the proposed advisor would be helpful in better understanding these events in real time.

Diagnostic systems in use at this time almost always rely on elaborate expert systems to evaluate the current plant status. Sometimes, these expert systems are interactive with the operator. Also, these expert systems go through a long, computation intensive, fault-tree type diagnosis routine. This may make them slow to respond in an emergency situation. The proposed advisor is expected to have quicker response, thus providing the operators with more time to rectify the problem, mitigate any possible damage, and save the plant from an unnecessary shutdown.

This thesis is part of an ongoing project at Iowa State University to develop ANN based fault diagnostic systems to detect and classify operational transients at nuclear power plants. The project envisages the deployment of such an advisor at Iowa Electric Light and Power Company's Duane Arnold Energy Center nuclear power plant located at Palo, IA. This advisor is expected to make status diagnosis in real time, thus providing the operators with more time for corrective measures.

Neural Networks

Neurai networks are a novel and fast-emerging branch of the science of artificial intelligence. Robert Hecht-Nielsen [23] defined a neural network as a "parallel, distributed information processing structure consisting of processing elements interconnected together with unidirectional signal channels called connections" (p. 593). Each processing element has a single output connection which "fans out" into as many collateral connections as desired. The processing element output signal can be of any mathematical type desired.

Neural networks draw interest because of the absence of a knowledge base which is the core of any expert system. Expert systems require that knowledge be specif-

ically inserted into them about every aspect of a system that needs to be analyzed. This knowledge is stored as numerous 'if-then' logic statements that assist the system in performing a fault-tree type analysis. In the case of a nuclear power plant, this requires that every possible scenario be investigated in as much detail as possible. It also requires that the personnel developing the system understand all the processes and systems in the plant and know the significance of each sensor reading in each of the scenarios being investigated. On the other hand, ANNs do not require knowledge to be explicitly inserted into them. In fact, the designer need not have a very intimate understanding of the importance of each sensor reading. All he need to know is that certain sensors are important indicators of the health of the plant [29]. ANNs learn the correct response from the training set during the training process, and generalize this information (For a more detailed discussion of neural networks and learning, see Chapter 2). The training set is the collection of input-output patterns that is used by the network to infer the functional relationship between the inputs and the outputs. Generalization is the ability to "quantitatively estimate certain characteristics or features of a phenomenon never before encountered based on similarities with things previously known" [5](p. 102). Neural networks, because of their parallel analog nature, are more noise tolerant than the conventional expert systems, and so manage to do a fairly credible job of classification under deteriorating sensor conditions. Faced with the fact that a majority of the anticipated transients at nuclear power plants have never actually occurred and data for such scenarios are obtained through computer simulation, the generalization capabilities of ANNs are especially useful for determining a solution for accident recognition [29]. If some physical aspects had been overlooked while constructing the models on which the simulations

and expert systems are based, an expert system would still function on the rules built into them. However, an ANN would be able to disregard this particular mistake and work on the basis of other information garnered through the training process.

Problem Statement

This work explores the use of ANNs to recognize operational transients at a nuclear power plant. Every transient is unique in the changes they cause in the values of plant variables. The plant status, as given by the values of these plant variables at any time, constitutes a pattern that is related to the transient that caused the same. So, the task of transient recognition can be thought of as a problem in pattern recognition, though on a rather massive scale. The pattern recognition capabilities of ANNs are quite well known. Their application in plant status diagnostics, therefore, is quite promising.

Though promising, neural networks have not witnessed the widespread use initially expected of them. One of the reasons is the problem of selecting an appropriate architecture for a desired task. Conventional ANNs require the architecture to be set before training is started. However, this choice of the architecture influences the training process as well as the post-training performance of the network. Thus "the viability of a specific architecture can only be evaluated after training" [5](p. 101). For a large and complex problem of the kind involving nuclear power plant status diagnostics, the search for an appropriate architecture would have proved to be an extremely time consuming affair involving a lot of guesswork. For the quick and efficient development of a trained ANN to perform the required task, it was thus imperative to develop a systematic method to arrive at a 'proper' network architecture

for any given problem. A network of the smallest size that can successfully classify the problem is said to have the 'proper' architecture for the problem. This led to the development of a derivative Dynamic Node Architecture (DNA) scheme for the backpropagation neural network algorithm.

The present work uses the DNA scheme to create an ANN that will be able to classify seven different transients and the normal conditions. The trained advisor would identify if the plant was in a normal operating condition, or if it was undergoing one of the seven transients. The database for training contains the values of eighty-one plant variables at one-second time intervals during the progression of the transients. The variables monitored are computer points of the full scale control room simulator at the Duane Arnold Energy Center (DAEC); these points correspond to meter readings in the actual control room at the plant. These variables were selected after intensive discussions between personnel at the plant and fellow researchers at Iowa State University [18].

CHAPTER 2. ARTIFICIAL NEURAL NETWORKS

Introduction

Artificial Neural Networks (ANNs) were developed as a result of the efforts of researchers to model the human brain. In fact, a neural network researcher is as likely to have a background in psychology as in electrical engineering. Interest in ANNs has been increasing for the past decade though the concept is over 30 years old [43]. Artificial neural networks, also called neural systems, connectionist systems, and neurocomputers, were brought into the spotlight in the 1987 announcement of the Japanese Sixth Generation computer project. The Japanese announcement coined the term "natural intelligence" to explain that the computers "would display behaviors based on biological rather than silicon models"[7](p. 46). The present age of neurocomputing started in 1960 with the publication of the Perceptron rule and the Least Mean Square (LMS) algorithm, two early rules for training adaptive elements [43]. In the years following these discoveries, many new techniques have been developed in this field, and the discipline is growing rapidly. One early development was Steinbuch's Learning Matrix [36], a pattern recognition machine based on linear discriminant functions. At the same time, Widrow and his students devised Madaline Rule I (MRI), the earliest popular learning rule for neural networks with multiple adaptive elements [44]. This led to the most famous neural network: Widrow's Adaptive Lin-

ear Element (ADALINE) developed in 1963 as a simple bin sorter [43]. Other early work included the "mode-seeking" technique of Stark, Okajima, and Whipple [35]. This was probably the first example of competitive learning wherein one particular solution is chosen over many possible competing solutions. However, until the 1980s artificial neural networks had been considered interesting but impractical. In 1982, J.J.Hopfield sparked a resurgence of interest in neural networks when he discussed how such interconnected neurons can have collective computational properties,with a distributed memory [25]. The most significant developments were the formulation of the backpropagation neural network algorithm independently by Werbos [41] and Rumelhart *et al.* [32]. Backpropagation has since become the most commonly used neural network paradigm [30].

Artificial neural network models attempt to achieve good performance via dense interconnection of simple computational elements or nodes. Instead of performing a program of instructions sequentially as in a von Neumann computer, neural networks "explore many competing hypotheses simultaneously using massively parallel nets composed of many computational elements connected by links with variable weights" [30](p. 4). The layered feed-forward ANN consists of nodes arranged in layers, with the nodes of any layer being connected to the nodes in an adjacent layer through variable weights (see Figure 2.1). The nodes of a layer are connected to every node of the layers immediately above and below them but not to any node in the same layer. In such feed-forward layered networks, the first layer is the input layer where the nodes are inactive, their outputs being equal to their inputs. The last layer is the output layer. The layers in between consist of "hidden" nodes, so called because they are isolated from the outside environment. The design of a network architecture is

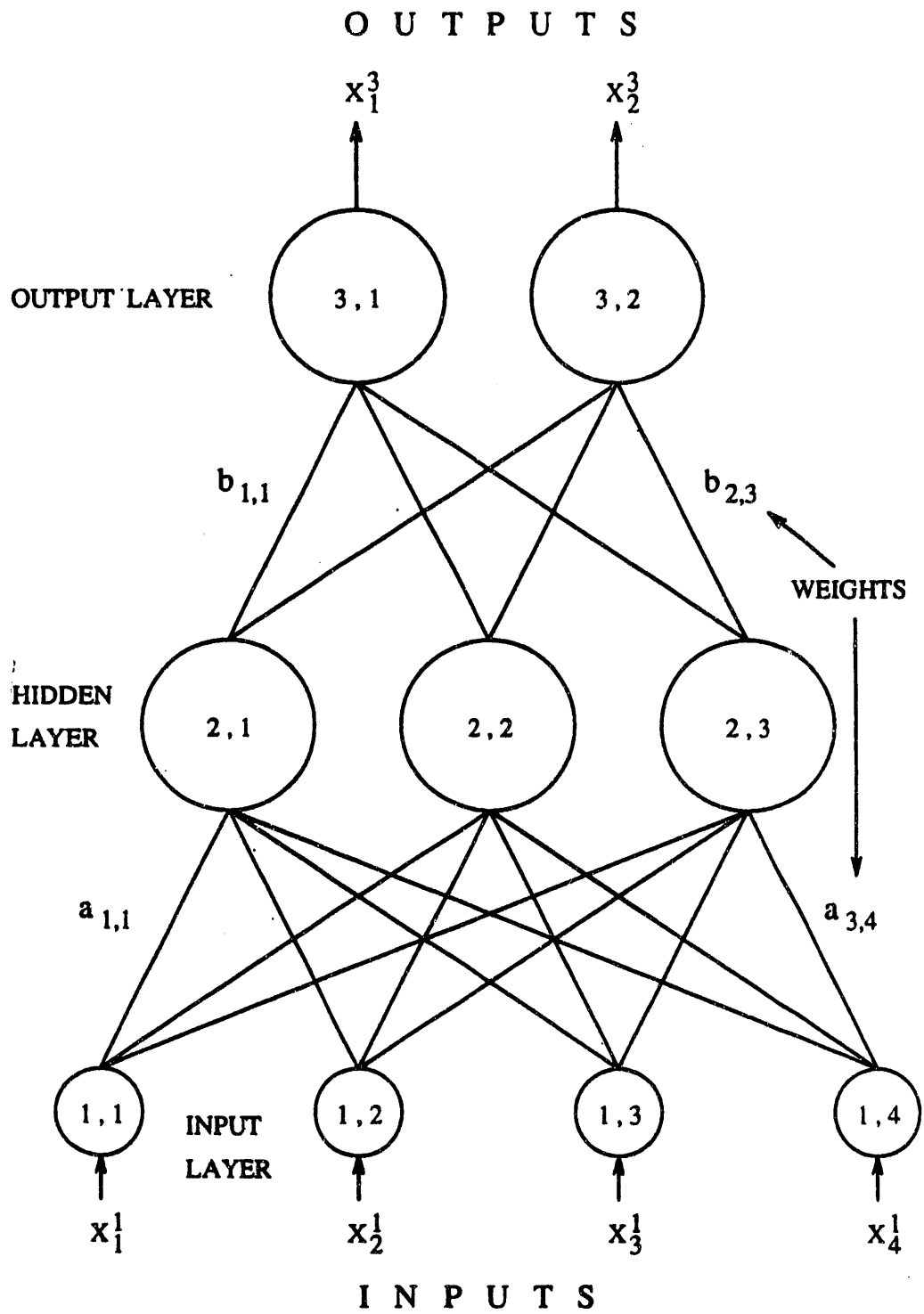

Figure 2.1: A Three Layered Neural Network

rather arbitrary, only the number of nodes in the input and output layers being fixed by the problem at hand. The nodes used in ANNs are nonlinear and typically analog. The simplest node sums weighted inputs and passes the result through a nonlinearity function, sometimes also called the transfer function. More complex nodes may include "temporal integration or other types of time dependencies" [30](p. 4) and more complex computations than summation.

"Neural network models are specified by the network topology, node characteristics, and training or learning rules" [30](p. 4). Given an initial set of weights, usually selected at random, these rules indicate how weights should be changed to improve performance. The nodes in a neural network carry out local computations, and thus various nodes can do their own calculations simultaneously. This massive parallelism results in high computation rates. But the potential benefits of neural networks extend beyond this. Neural networks typically provide a greater degree of robustness or fault tolerance than von Neumann sequential computers because there are many more processing nodes, each with primarily local connections [30]. Damage to a few nodes or links thus need not impair overall performance significantly.

Using a Neural Network: Training

A neural network can be used to solve a problem only after it has been trained on the problem. The training results in the network learning the solution to the problem. Learning in neural networks can be supervised or unsupervised. Supervised learning means the network has some omniscient input present during training to tell it what the correct answer should be [7]. The network then has a means to determine whether or not its output was correct and knows how to apply its particular learning law to

adjust its weights. Unsupervised learning means the network has no such knowledge of the correct answer and thus cannot know exactly what the correct response should be. Most training processes are accomplished by supervised learning. A training set consisting of sample datapoints is presented to the network and learning is based on this training set. The performance of the network is improved by changing the interconnecting weights according to the learning rule being used. A common measure of the performance of a network is the root mean square (RMS) error of the output nodes. The weights are randomized at the beginning and the process of presenting examples and changing weights is repeatedly carried out till the network performs to the user's satisfaction. The aim of the training process is to achieve as small an RMS error as possible, a global minimum. However, using the gradient slope descent methods of backpropagation, a neural network tends to become lodged in a local minima or on a plateau with a small gradient [1][4][24]. The training process becomes increasingly more complex as the size and scope of the training problem increases. The post-training performance of a network is determined by the closeness of the training set to the recall set. The most common training scheme for the feed-forward ANNs is the backpropagation method which is detailed in the next section. There are also many other good references on backpropagation including [23] [30] [32] etc.

Backpropagation

The Delta Rule

Backpropagation is a supervised learning method. It is without question the most commonly used learning algorithm in the world today [8]. Developed by Bernard

Widrow and Ted Hoff at Stanford University and first published in 1960, it used the Delta rule or Least Mean Squared (LMS) training law. The network itself was called ADALINE, derived from the term ADAptive LINear Element. Strictly speaking, Delta rule is the learning law which when applied to a feed-forward ANN results in the backpropagation network. The Delta rule changes the weights according to the following equation [9] :

$$W_{new} - W_{old} = \beta E \frac{X}{|X|^2} \tag{2.1}$$

where W represents the weight vector before and after the weights are adjusted (*old* and *new* respectively), β is the learning rate, X is the input pattern vector, $|X|$ is the length or magnitude of the input pattern vector, and E is the error for a node. As the name LMS training law suggests, this rule attempts to insure that the aggregate statistical LMS error over the entire training set is minimized in the network. In the case of backpropagation, it is assumed that there exists a unique set of weights that very nearly relates the inputs to the outputs for the given problem. The error in the weights of the processing element cannot be directly measured as the desired weight set is not known. The error in the weight vector manifests and is measured as an error in the output of the node. The current error, or how far away the network is from this ideal value, is calculated for the weights for the given input. The weight vector is then adjusted by calculating a Delta vector that is parallel to the input vector and has a magnitude as described in the previous equation (9). The weights are adjusted by adding this delta vector to the current weight vector.

The aggregate (over the training set) root mean squared error of the output of

a network is expressed as

$$RMS\ error = E(W) = \{\frac{1}{Nkmax}\sum_{n=1}^{N}[\sum_{k=1}^{kmax}(z_{k,n} - x_{k,n})^2]\}^{1/2} \qquad (2.2)$$

where N is the number of patterns in the training set, $kmax$ is the number of nodes in the output layer, $z_{k,n}$ is the expected output of the kth output node for the nth training pattern and $x_{k,n}$ is the actual output of the same node for the same pattern. This RMS error is a function of the existing set of weights. It can be shown mathematically that the aggregate mean squared error is a n-power function of the weight vector, where the weight vector has n components [8]. So a plot of the mean squared error vs. the possible weight vectors would give a hyperparaboloid in n-space. The Delta rule moves the weight vector from wherever it is on the surface of this hyperparaboloid towards the bottom. The bottom of the hyperparaboloid represents the nearest minimum mean squared error, and this is the point the weight vector needs to reach, assuming that this is the lowest that the error surface gets. As is known from vector analysis, the gradient always points along the direction of the steepest rise of a curve and the negative gradient is the steepest descent of the curve at any given point [8][9]. That means this learning algorithm always takes the most direct route from the current position of the weight vector to the nearest position where the hyperparaboloid bottoms out, based on the current pattern. It is to be noted that the minima reached by the Delta rule is almost always the nearest point of zero gradient. This might be either a local or the global minima. The goal is to reach the global minima, but this algorithm has a tendency to get lodged in local minima.

The mathematics of the Delta rule can be demonstrated in a rather simple manner starting with Newton's method. This method is used to find the roots of

an equation $f(x) = 0$. The basic assumption is that the function f is differentiable. This implies that the graph of f has a definite slope at each point and hence a unique tangent line. Suppose we take the value x_0 as an approximation of the root x^*. The tangent at the point $(x_0, f(x_0))$ on the graph of f is a rather good approximation to the curve in the vicinity of that point [17]. The point where this tangent meets the x-axis, x_1 is the next approximation of x^*. This value of x_1 is given by :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{2.3}$$

See Figure 2.2 for an illustration of this. Further iterations will provide a sequence of estimates as

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{2.4}$$

and so on. This interpretation of the Newton's method shows that this method, and so backpropagation, can get lodged in a local minima. So the initial guess x_0 is important. In fact in Figure 2.2, if our initial guess for the minima were x_0' instead of x_0, we would end up in the local minima. Usually some techniques like using a momentum term help take us beyond local minima. In backpropagation implementation this means that the initial selection of weights, usually done at random, might influence the network reaching the global minima or getting lodged in a local minima. Shown above was the steepest decent method wherein the second and higher order derivatives of the function f are ignored.

Typically, Newton's method utilizes a number of derivatives of the function $f(x)$ to arrive at the next estimate of the root x^*. The Taylor series expansion of $f(x_0 + h)$ is given by [6][17]

$$f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2!}f''(x_0) + \ldots + \frac{h^n}{n!}f^n(x_0) + \ldots \tag{2.5}$$

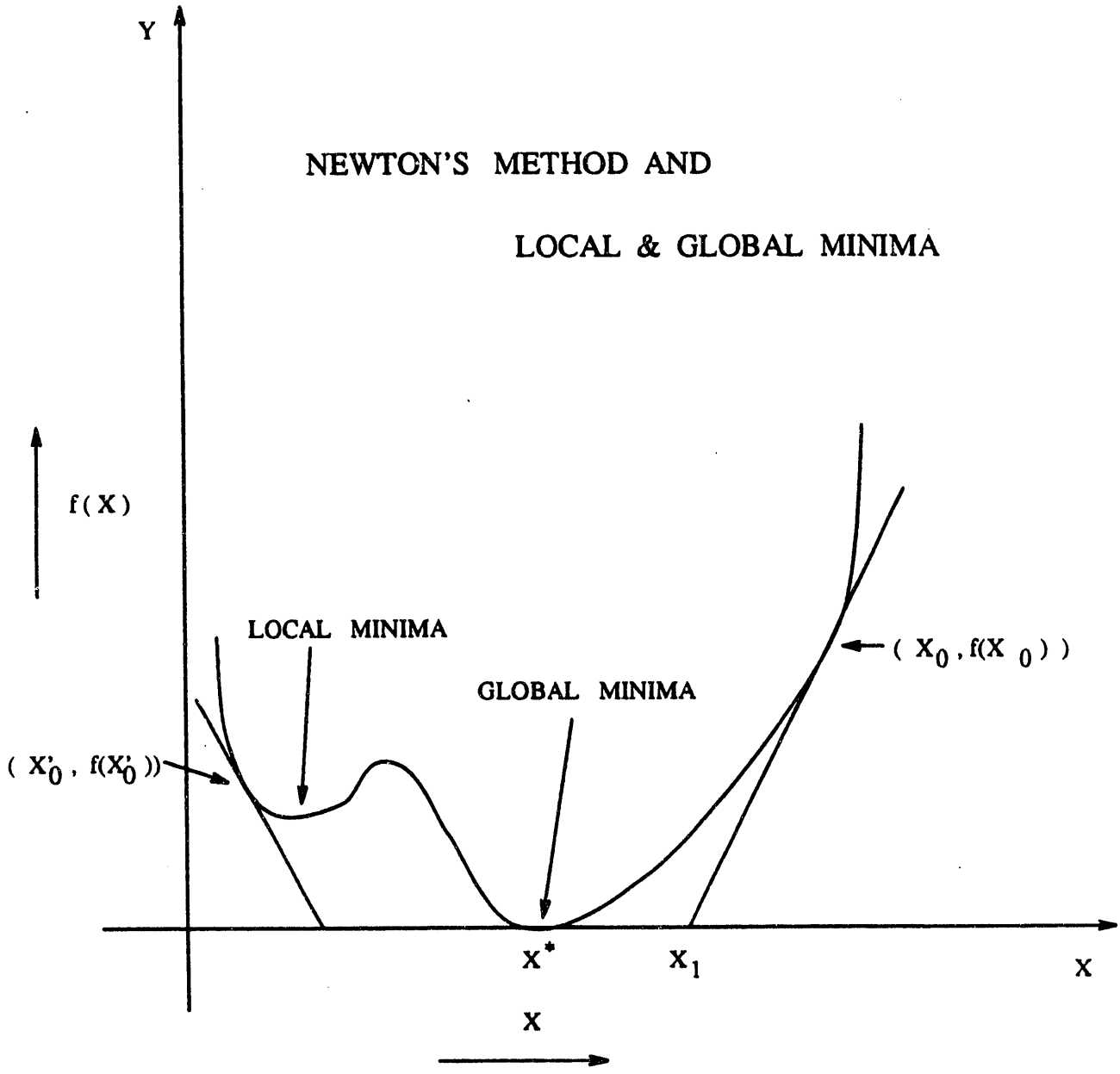

Figure 2.2: Newton's Method

This Taylor series expansion, when applied to the RMS error (Equation 2.2), shows the mechanics of the Delta rule. The RMS error is a function of the weights. Referring to this error function as $E(W)$, the purpose of the training is to minimize this error, i.e. to find the weight vector W^* that will minimize $E(W)$. The Taylor series expansion of this error function is [4][29] :

$$E(W + \Delta W) = E(W) + G^T(W)\Delta W + \frac{1}{2}\Delta W^T H(W)\Delta W + ... \tag{2.6}$$

$G(W)$ is the gradient vector given by [4][29] :

$$G(W) = \frac{\delta E}{\delta W} = [\frac{\delta E}{\delta W_1}, \frac{\delta E}{\delta W 2}, ..., \frac{\delta E}{\delta W_n}]^T \tag{2.7}$$

where W is an n dimensioned vector, i.e. there are n components of the weight vector. H is the Hessian matrix defined [4][29]

$$H = [H_{i,j}] \quad i = 1, ..., n \; and \; j = 1, ..., n \tag{2.8}$$

where [29]

$$H_{i,j} = \frac{\delta^2 E(W)}{\delta W_i \delta W_j} \tag{2.9}$$

Newton's method is used to try solve the above equations in an iterative manner. The estimate of the weight vector at iteration $i + 1$ is based on the weight vector at the previous iteration and the corresponding ΔW vector. Mathematically,

$$W_{i+1} = W_i + \Delta W_i. \tag{2.10}$$

The iterative process is stopped when the error function for the present set of weights reaches a value very near the value of the error function for the ideal weight set W^*, i.e. when $E(W_i) \approx E(W^*)$.

If the second and higher order derivatives in the Taylor series expansion of the error function (Equation 2.6) are ignored, we get the steepest decent method [17] given by

$$E(W + \Delta W) = E(W) + G^T(W)\Delta W. \tag{2.11}$$

Let us assume that the new estimate of W^*, i.e. $W + \Delta W$ is exact, then $W^* = W + \Delta W$. From the above equation we get

$$E(W^*) = E(W) + G^T(W)\Delta W. \tag{2.12}$$

If W^* defines a local minima in the weight space, then the gradient at the point is zero, i.e. $G(W^*) = 0$. The derivative of the error function at any point is the gradient at the point. So, taking derivative of the above equation we get [29][4]

$$G(W^*) = G(W) + H(W)\Delta W = 0 \tag{2.13}$$

which gives

$$\Delta W = -H^{-1}(W)G(W). \tag{2.14}$$

If we set the step size $\mu = H^{-1}$ we get the implementation of the steepest decent method. The iteration

$$W_{i+1} = W_i - \mu G(W_i) \tag{2.15}$$

will result in the vector W moving towards W^*, i.e. it will move towards the local minima. In the Delta rule, the learning rate β corresponds to the step size μ. This proves the earlier assertion that the Delta rule moves the weight vector towards the nearest minima, which may be either the global or a local minima, in the fastest manner. It also shows the importance of the initial guess of the weight vector.

Backpropagation Neural Networks

The publication of the backpropagation technique by Rumelhart *et al.* [32] has unquestionably been the most influential development in the field of neural networks in the past decade. The learning procedure they suggested involved the presentation of a set of pairs of input and output patterns. The ANN uses the input vector to produce its own output vector and then compares this with the expected or desired output vector. (This makes backpropagation a supervised learning method [7].) If there is no difference, no learning takes place. Otherwise the weights are changed to reduce the difference. This process utilizes the Delta and the Generalized Delta rules. For the output nodes the error is easily calculated as the difference between the actual output and the desired output. But for the nodes in the hidden layers, it is not possible to calculate the error in this way. The correct output of the hidden nodes are not known. To assign an error to the hidden nodes we backpropagate the error of the output nodes to the hidden nodes using the very same weights that were used to propagate the error to the output nodes in the first place[9]. The Delta rule modified for the hidden nodes is called the Generalized Delta rule [9][32].

Backpropagation networks are layered and feed-forward; they always consist of at least three layers of nodes. The number of inputs and outputs are fixed by the problem at hand. The only choice of network architecture is the number of hidden nodes. This choice needs to be carefully exercised. If the hidden layer is too large it will encourage the network to memorize the input patterns rather than generalize the input into features [9][23]. This is because the large number of nodes and weights give the network more ways to distinguish features, resulting in the specifics being learned better than the generalities. This reduces the network's ability to correctly classify

unfamiliar patterns after training is complete. On the other hand, a hidden layer too small will drastically increase the number of iterations, and thus the computer time, required to train the network and will most likely reduce the accuracy of recall [9]. There are no hard and fast rules for determining the optimum architecture; most rules in use at this time are empirical in nature, derived by heuristic methods.

The learning process in a backpropagation neural network consists of primarily two steps: the forward activation flow and the backward error backpropagation. The first gives an output in the output layer from which the error in that layer can be calculated. The second gives the error of the hidden nodes. Applying the Delta and the Generalized Delta rules, we can arrive at the new weight vector. This process is continued until the root mean squared (RMS) error of the output layer, also called the cost or energy function, is below a preselected value.

Forward Activation Flow When a pattern is presented to the network, the input layer passes the input values to the first hidden layer through the weights connecting the two layers. The input to the hidden nodes is a weighted sum of the outputs of the previous layer. These nodes then pass their inputs through their transfer function. The calculations within a node is shown in Figure 2.3. The output of one layer is used to generate the input of the next layer. This process is repeated until the output of the output layer is calculated. The forward activation flow is shown in Figure 2.4.

The transfer function, also called the activation function (see Figure 2.4), determines the activity or the excitation level of the node. The activation function used

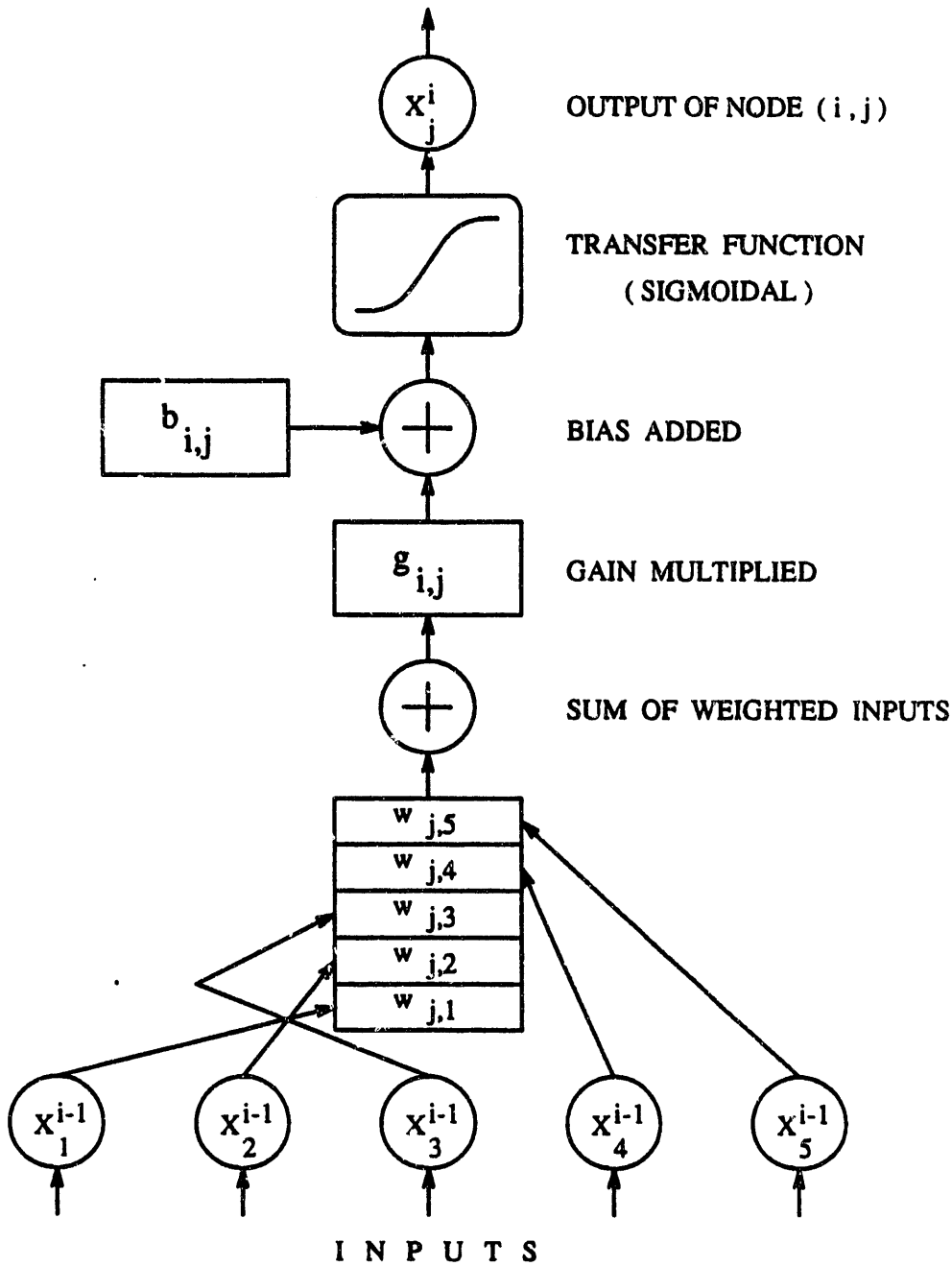

Figure 2.3: A Detailed Node

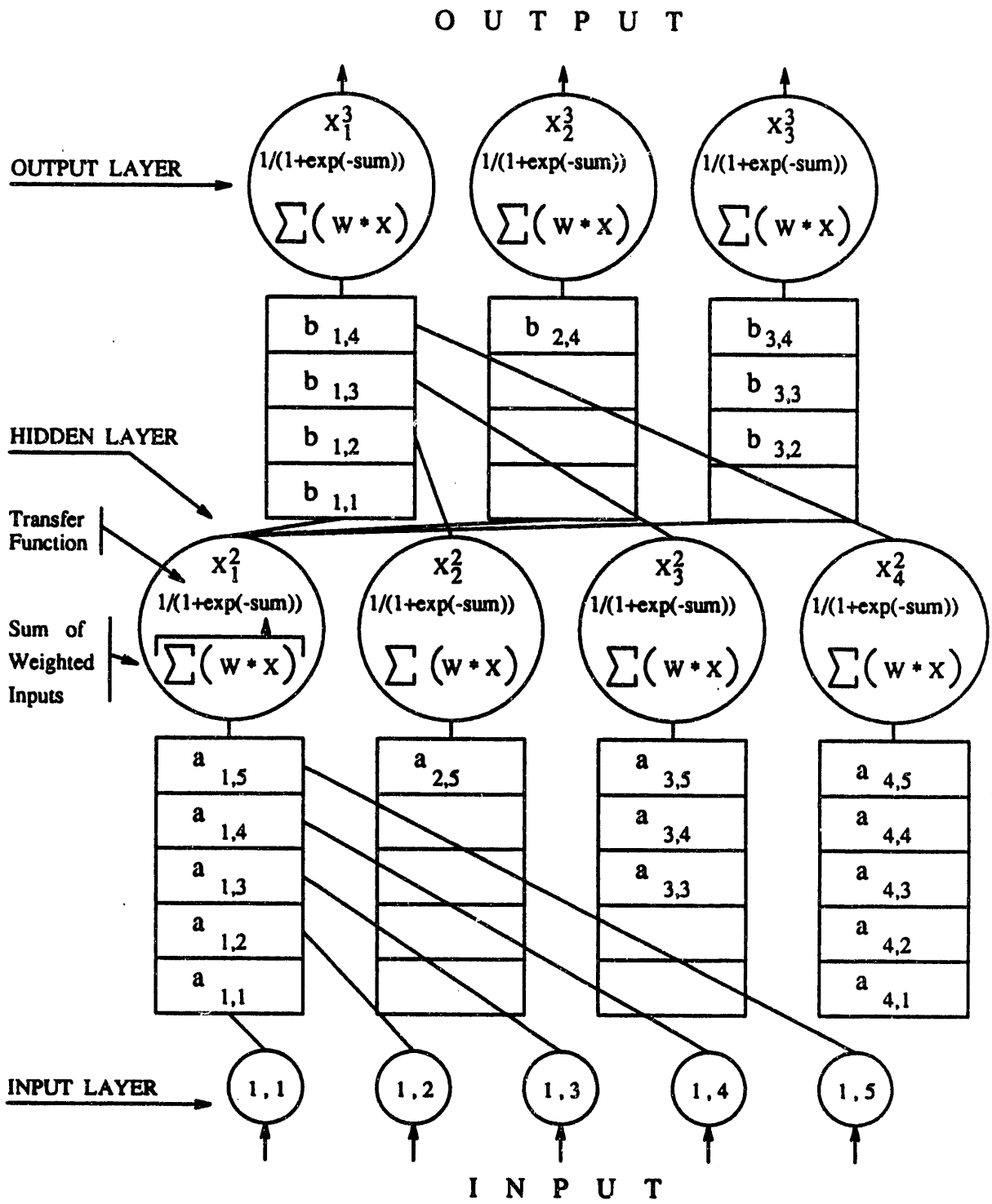

Figure 2.4: Feed Forward Activation Flow [19]

here is the sigmoid function given by

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

The above equation gives an output between 0.0 and 1.0 for any real value of x. (See Figure 2.5.) Networks with only three layers were used, i.e. there were an input, one hidden, and an output layers. In the terminology used from now on, subscripts i,j, and k relate to the input, hidden and output layers respectively and the subscript n refers to the nth training pattern.

The input of the jth node in the hidden layer is the weighted sum of the outputs of the input layer nodes. Mathematically,

$$sum_{j,n} = \sum_{i=1}^{imax} x_{i,n} a_{j,i} \tag{2.17}$$

where $x_{i,n}$ is the output of the ith input layer node for the nth pattern, $a_{j,i}$ is the weight connecting the jth hidden layer node to the ith input layer node and $imax$ is the number of nodes in the input layer. The output of the jth hidden layer node, $x_{j,n}$, after passing the input through the transfer function (Equation 2.16), is given by

$$x_{j,n} = \frac{1}{1 + e^{-sum_{j,n}}} \tag{2.18}$$

where $sum_{j,n}$ is given by Equation 2.17. Similarly, the input to the kth output node is

$$sum_{k,n} = \sum_{j=1}^{jmax} x_{j,n} b_{k,j} \tag{2.19}$$

where $b_{k,j}$ is the weight connecting the kth output layer node to the jth hidden layer node and $jmax$ is the number of nodes in the hidden layer. As with the hidden layer,

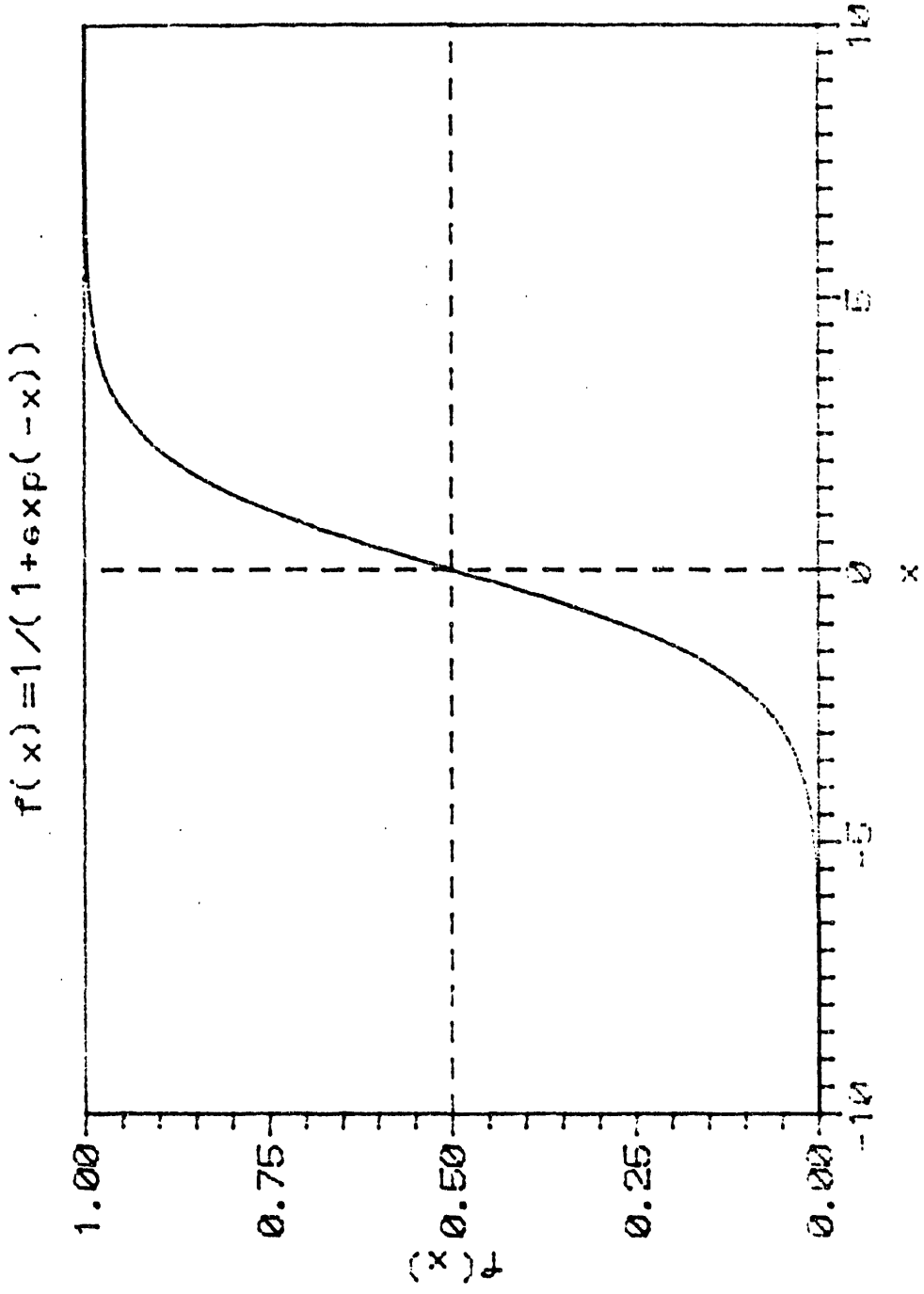

Figure 2.5: Sigmoid Function

the output of the kth output layer node, $x_{k,n}$, is given by

$$x_{k,n} = \frac{1}{1 + e^{-sum_{k,n}}} \tag{2.20}$$

By the above computations, the output of the network for the given weight set can be calculated for a given pattern.

Backward Error Flow Once the outputs of the network are calculated by the above process, the error in the nodes of the output layer can be found out by finding the difference between the actual outputs and the desired outputs. The desired or correct outputs are part of the data in the training set. The errors in the output nodes are propagated backward to the hidden nodes. This process is done using the very same weights that were used to propagate the errors from the hidden to the output layers. Let the error in the kth output node for the nth pattern be $err_{k,n}$.

$$err_{k,n} = |x_{k,n} - z_{k,n}| \tag{2.21}$$

where $z_{k,n}$ is the expected output of the kth output layer node for the nth training pattern. These errors are backpropagated to the hidden layer for each pattern. It is important to note that the weights are not changed after presenting each pattern but after all the patterns have been presented. This practice is called batch training. All work presented here utilized batch training. Batch training has been shown to increase the convergence rate and generalization capabilities of backpropagation neural networks [39].

The backpropagated error of the jth hidden layer node for the nth pattern is

$$err_{j,n} = \sum_{k=1}^{kmax} err_{k,n} b_{k,j} \tag{2.22}$$

where again $err_{k,n}$ is the error in the output of the kth output layer node for the nth pattern, $b_{k,j}$ is the weight connecting the kth output node to the jth hidden node, and $kmax$ is the number of nodes in the output layer.

Now the Delta rule is applied to the output nodes to change the weights $b_{k,j}$. Mathematically, this is written as (See Figure 2.6)

$$b_{k,j\,new} = b_{k,j\,old} + \beta \frac{1}{N} \sum_{n=1}^{N} err_{k,n} x_{j,n} \tag{2.23}$$

where N is the total number of patterns in the training set, β is the learning rate, and $err_{k,n}$ and $x_{j,n}$ are as given by equations 2.21 and 2.18 respectively.

To change the weights connecting the hidden and the input layers, i.e. the set of weights $a_{j,i}$, we need to utilize the Generalized Delta Rule [23]. For this we need to use the derivative of the activation function. For the particular sigmoid function used (Equation 2.16), the derivative is of the graceful form

$$f'(x) = f(x)[1 - f(x)]. \tag{2.24}$$

The Generalized Delta Rule is of the form [9][29]

$$W_{new} = W_{old} + E\beta \sum_{patterns} \sum_{nodes} f(x) f'(x) \tag{2.25}$$

where E is the error, β is the learning rate, and $f(x)$ is the activation function. For the sigmoidal function, the derivative looks like a bell (Figure 2.7), with relatively large values in the midrange of inputs and small values at either end. Applying the derivative serves two purposes. First, it ensures that as the outputs approach 0 or 1, only very small changes in weights can take place. This promotes stability in the network. Secondly, it prevents excessive blame to be attached to the hidden layer nodes [9]. When the connection between a hidden node and an output node is

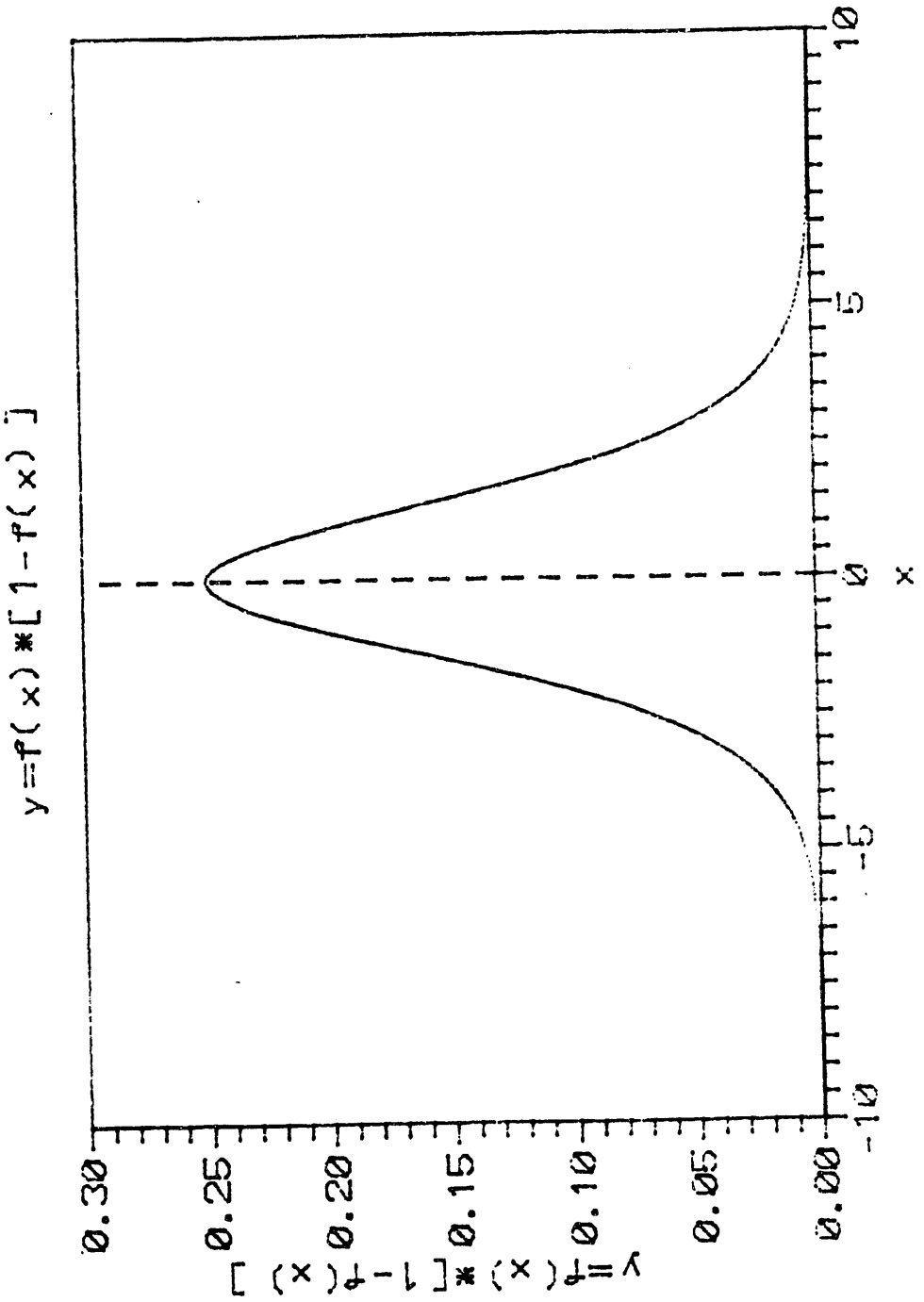

Figure 2.7: Derivative of the Sigmoid Function

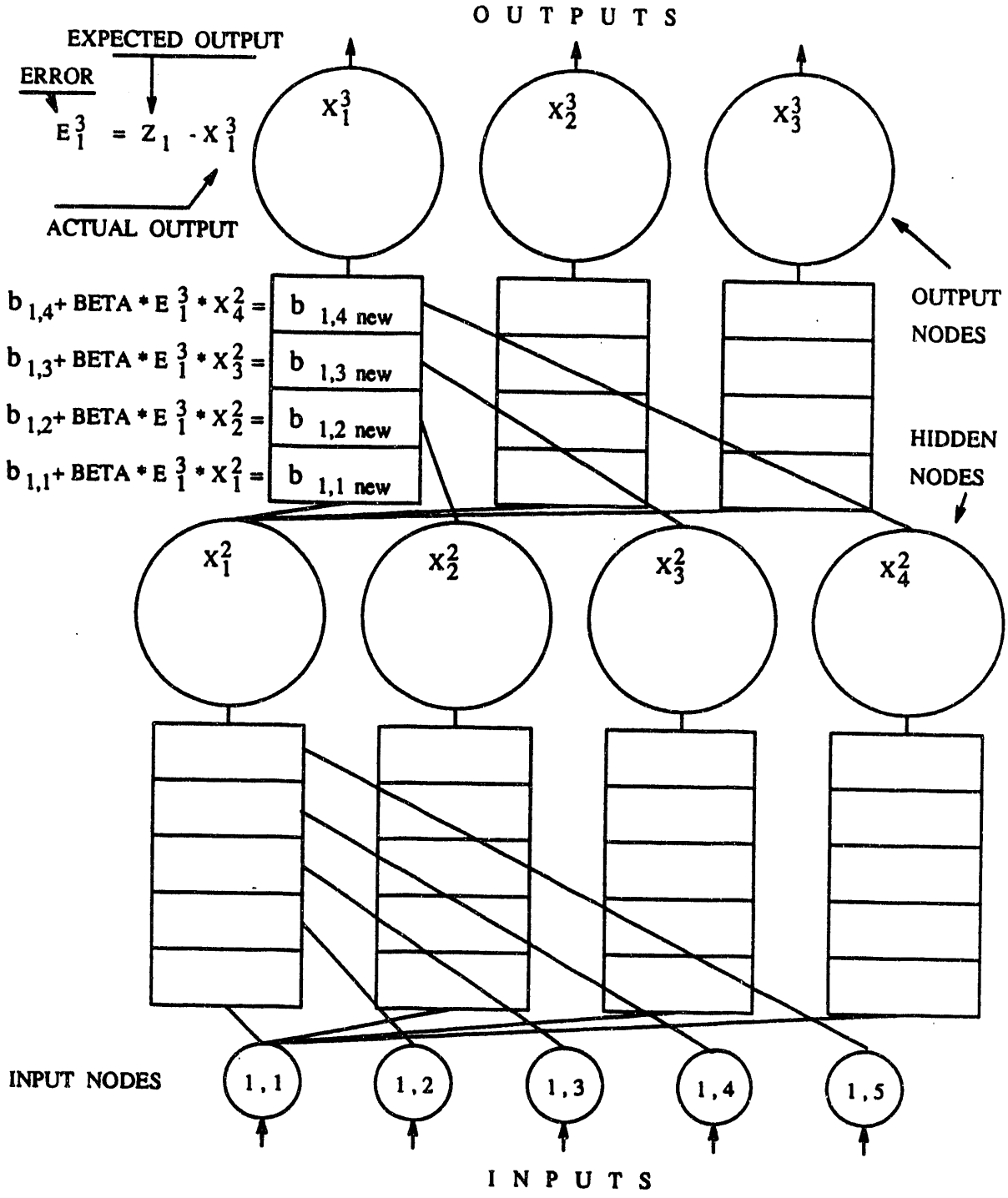

Figure 2.6: Delta Rule for Output Layer Nodes [19]

very strong, indicated by a very extreme value of the weight, and the output node has a very large error, the middle layer node may be assigned a very large error. If that hidden node had a very small output, then this would be inappropriate as it could not have singlehandedly caused the large error in the output node. The use of the derivative moderates the error attributed to the hidden nodes and only small to moderate changes are made to the hidden layer neurode's weights.

In the implementation for this work, the Generalized Delta Rule (See Figure 2.8) takes the form

$$a_{j,i_{new}} = a_{j,i_{old}} + \beta \frac{1}{N} \sum_{n=1}^{N} [err_{j,n} x_{j,n}(1 - x_{j,n})] x_{i,n} \tag{2.26}$$

where $a_{j,i}$ is the weight connecting the jth hidden layer node to the ith input layer node, and all other terms are as defined before.

The weights are changed after all the training patterns are presented. The RMS error of the network over the training set is calculated after the weights are changed. If this error is greater than the preselected target RMS error, then the training process is continued. Else, the training is terminated and the latest set of weights is saved. These weights are then used for the recall over 'unseen' patterns. It is of importance to note that there might be various functional relationships that might relate the input patterns to the output patterns in the training set. But only one of them will be able to do so for the problem as a whole. We seek to find this particular functional relationship.

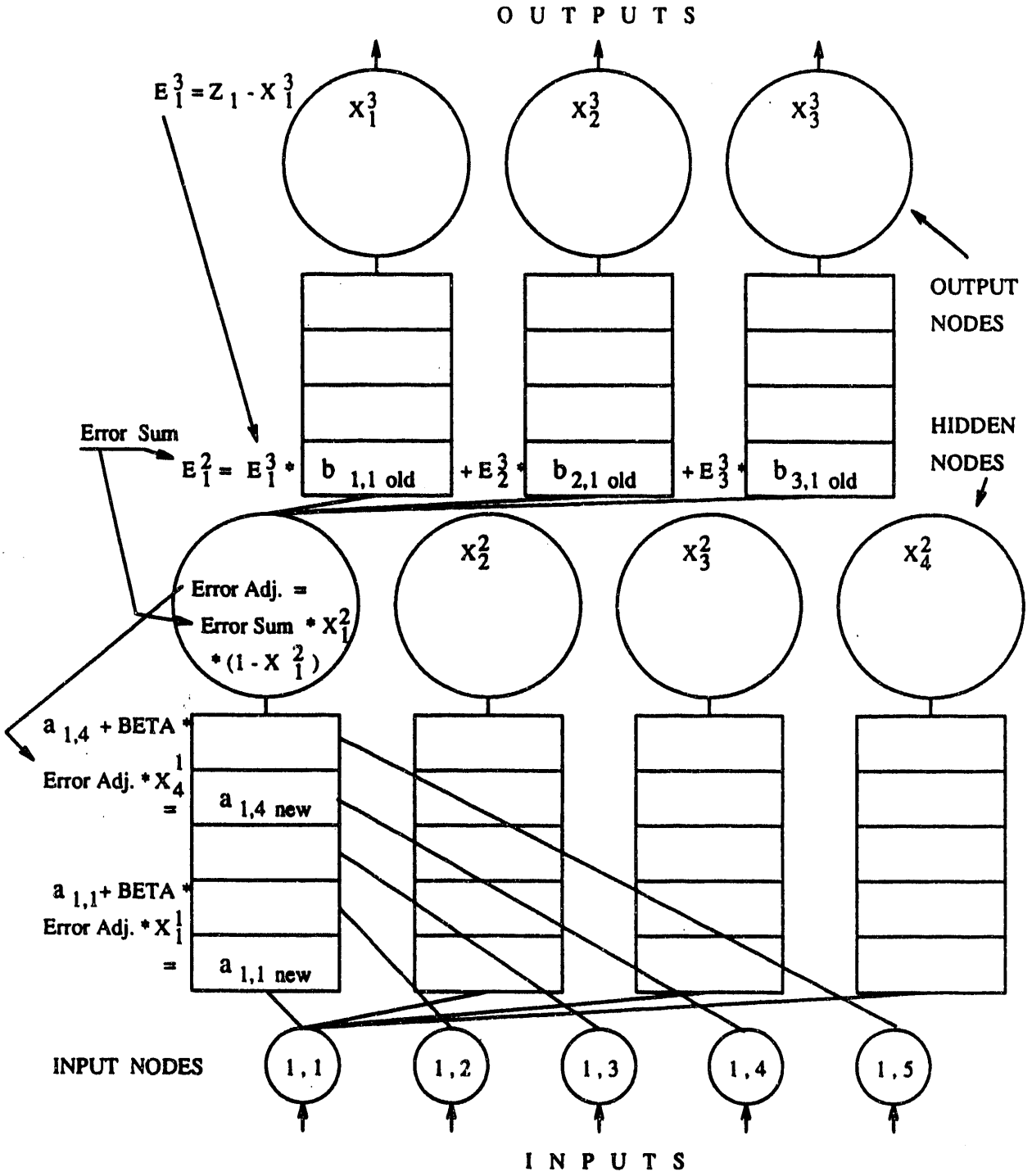

Figure 2.8: Generalized Delta Rule for Hidden Layer Nodes [19]

CHAPTER 3. DYNAMIC NODE ARCHITECTURE

Introduction

Backpropagation neural networks are layered feed-forward. The input and output layers are open to the environment. The middle layers, in between the input and the output layers, are isolated from the outside and are thus called hidden layers. The problem to be solved dictates the number of nodes in the input and output nodes. The number of nodes in the hidden layers is the only real choice the user has in deciding the architecture of the network. The architecture selected is of vital importance to the training characteristics and recall performance of the network. Most 'rules' currently used to decide network architectures are derived by heuristic methods and are empirical in nature. The usual approach to deciding network architecture is to start with various guesses, train all of them, and then retain the one with the best post-training characteristics [5]. This drastically increases the training time required before a network can be used to solve a problem. Problems of this kind have been instrumental in preventing the widespread use of neural networks as effective tools in solving many intractable problems [5].

The problem addressed in this thesis involves status diagnostics in a nuclear power plant. The input data consisted of eighty-one variables and the output was a combination of three booleans. The network needed to be trained a number of times

with different training sets. If an appropriate architecture was needed for each trial, the problem would have assumed mammoth proportions. A lot of guesswork would have been involved, the physical problem being not so well understood. So it was imperative to come up with some systematic method that would derive the optimum, or near optimum, architecture for any given problem.

It is of great importance to have the correct number of nodes in the hidden layer(s). If there are too many nodes, then the network will tend to memorize the relation between the inputs and the outputs in the training set. This is undesirable. The network should be able to learn only the generalities in the training exemplars and not the specifics of the patterns in the training set. The former will result in low training RMS as well as good generalization, i.e. good performance over recall data. On the other hand, memorization will result in a low training RMS but bad recall performance.

Architecture Optimization : Existing Methods

The recognized need for methods to optimize network architectures has stimulated researchers in neural networks to come up with various algorithms to do this. Approach to this problem has been two pronged: network pruning and weight elimination. In the first approach, training is started with a relatively large network. As this network learns the training set to the desired level of accuracy, nodes that are found to be unnecessary are removed. Various researchers have come up with different methods to determine which of the nodes need be removed [33]. As a node is lost, the network needs to be trained further until its RMS error over the training set is below the desired RMS error. This process is continued until the smallest network

that can successfully classify the training set is found. Any further reduction in the network size renders the network incapable of classifying the training set.

Some other researchers have attempted to prune unnecessary weights from a fully connected and trained network [28][40][45]. This results in a sparsely connected network devoid of redundant weights. The objection to pruning algorithms is that they start off by training a network too big for the problem. Thus computer time is wasted training unnecessary nodes and weights. Moreover, some amount of guesswork is still involved in deciding the network size to be trained. For very complex problems involving a lot of inputs, it is not always possible to make a very confident decision as to what network size is moderately larger than the optimum architecture.

Another noteworthy approach involves starting with small networks and building up until the network reaches a size where it can successfully learn the training set [1][24][38]. Most of these algorithms stop when the training set is first learned. But it is known that learning a functional relationship requires more nodes than recalling the same. Work by Hirose *et al.* [24] addresses this issue. When training results in the first network that can successfully learn the training set, the latest added node is eliminated. The smaller network is then further trained till it can learn the training set. The process is continued till the smallest network is found that can learn the problem. This process assumes that the last added node is the one that should be eliminated first. From a heuristic point of view it seems reasonable that the node that has spent the least amount of time in the network during training has the least contribution to the performance of the network. But there is no definite way to verify this assertion.

The derivative Dynamic Node Architecture (DNA) scheme was developed to skirt

all the above problems and arrive systematically at a near-optimum architecture for any problem. The scheme is detailed in the next section.

Dynamic Node Architecture Theory

When neural networks are used to solve a problem, it is important to utilize the network architecture appropriate for the problem. A network with too few nodes will be unable to learn the problem and will increase computation time. On the other hand, a network with too many nodes will memorize the problem. This will cause poor generalization. As previously defined, "generalization is the ability to quantitatively estimate the characteristics of a phenomenon never encountered before based on its similarities with things already known"[5](p. 102). A neural network, during its training, searches out general characteristics that classify the problem. If too many nodes, and weights, are used, the network will have too many ways to distinguish features [1] and will learn the specifics of the training set along with the generalities. This can be compared to the problem of finding the interpolation polynomial between various points [5]. An appropriate order polynomial gives a smooth curve. But as the order of the interpolating polynomial is increased, the interpolating curve tends to get chaotic, oscillating between the interpolating points.

The derivative Dynamic Node Architecture (DNA) scheme progresses in a systematic method to come up with the appropriate architecture for the problem. Training is commenced with a small network which won't be able to solve the problem. In all work presented in this thesis, training was started with just one hidden node. As training progresses, the network soon reaches a plateau and cannot reduce the RMS error beyond a certain point. Now a node is added to the hidden layer. The

weights connecting this node are assigned a very small value so that the addition of this node does not disrupt the network much. Training is now resumed until the network reaches another plateau. Now another node is added. This process is continued until the network reaches an RMS error value below a preselected value. This indicates that the network has learned the problem to the desired level of accuracy. Now, not all of these nodes may be necessary to recall the problem. So the hidden node with the least importance is removed from the network. The resultant network would require further training. If the deleted node had a very low level of importance, then the node had little contribution to the performance of the network. Upon continued training, the smaller network might be able to learn the problem. Then, another node (with the least importance) is deleted. This process is continued until the network is too small to learn the problem. Now nodes are added until the problem is relearnt. The process of deleting and adding nodes is continued until the algorithm starts oscillating about the optimum architecture. It is to be noted that the architecture given by this scheme may not be *the* optimum architecture but very close to it.

Importance of a Node

While deleting nodes, we get rid of the node with the least importance. The importance of a node is a function of the network outputs. If changes in the output of a hidden node is detrimental in deciding the output of the network more than a similar change in the output of another hidden node, it stands to reason that the former node is more important to the "dynamic functioning of the network" [5] than the later node. The importance of the jth hidden node with respect to the kth output

node is defined as [5]

$$I_{(x_j|x_k)} = E[|\delta x_{k,n}/\delta x_{j,n}|] * dx_j^{max} \tag{3.1}$$

where $E[...]$ is the expectation over the entire training set and dx_j^{max} is the maximum change in the output of the jth hidden layer node also over the entire training set. This importance function is called the derivative importance function. The derivative in the previous equation, which is the change in the output of the kth output node due to a change in the output of the jth hidden node, can be evaluated by partial differentiation of the transfer function. This gives

$$\frac{\delta x_{k,n}}{\delta x_{j,n}} = \frac{exp(-sum_{k,n}) * b_{k,j}}{[1 + exp(-sum_{k,n})]^2} \tag{3.2}$$

where $sum_{k,n}$ is of the form given by *Equation 2.19*, $b_{k,j}$ is the weight connecting the kth output layer node to the jth hidden layer node. *Equation 3.1* gives the partial importance of the jth hidden node with respect to the kth output node. If the network were to consist of more than one hidden layer, the partial importance of any of these hidden nodes with respect to any output node can be found out using the chain rule.

The total importance of the jth hidden node is the sum of the partial importances of that node with respect to all the output nodes. Mathematically,

$$I_{(x_j)} = \sum_{k=1}^{kmax} I_{(x_j|x_k)} \tag{3.3}$$

In the same way, the importance of a layer can be defined as the sum of the importances of the nodes in that layer.

Demonstration of the DNA Algorithm

This section demonstrates how the derivative Dynamic Node Architecture scheme works. For this purpose, the algorithm is used to learn three different problems. They are the Exclusive-Nor problem, the 8-to-1 decoder problem, and a probability density function separator problem.

The Exclusive-Nor Problem

The most simple network learning problem is the exclusive-nor problem. The training data is shown in Table 3.1. This is a simple two-input one-output problem.

Table 3.1: Exclusive-nor training data

Pattern	Input1	Input2	Output
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	1.0	0.0	0.0
4	1.0	1.0	1.0

The DNA algorithm training was started with one hidden node. Thus the starting architecture was 2 x 1 x 1. Table 3.2 shows the training history for this problem. The starting architecture is, as expected, unable to learn the problem. A second node is added. This architecture reaches a plateau and a third node is added. This 2 x 3 x 1 architecture too is unable to learn the problem fast enough and a fourth node is added. The 2 x 4 x 1 architecture manages to reaches an RMS error below the target of 0.01. Now, the node with the least importance is deleted. This leaves three nodes in the hidden layer. This architecture is now able to reach the target RMS error, and a further node is deleted. On further training, the 2 x 2 x 1 network

Table 3.2: Dynamic node architecture training history for the exclusive-nor problem

Arch.			RMS	Arch.			RMS	Arch.			RMS
In	Hid	Out	Error	In	Hid	Out	Error	In	Hid	Out	Error
2	1	1	0.50122	2	3	1	0.01793	2	4	1	0.02828
2	1	1	0.38725	2	4	1	0.01929	2	4	1	0.00997
2	2	1	0.39071	2	4	1	0.00993	2	3	1	0.02877
2	2	1	0.11059	2	3	1	0.02243	2	3	1	0.00982
2	3	1	0.11902	2	3	1	0.00972	2	2	1	0.06837
2	3	1	0.03376	2	2	1	0.04227	2	2	1	0.00999
2	4	1	0.04133	2	2	1	0.00999	2	1	1	0.49962
2	4	1	0.00997	2	1	1	0.49979	2	1	1	0.49224
2	3	1	0.02521	2	1	1	0.49286	2	2	1	0.42182
2	3	1	0.00981	2	2	1	0.32188	2	2	1	0.14522
2	2	1	0.06242	2	2	1	0.06229	2	3	1	0.18256
2	2	1	0.02109	2	3	1	0.06681	2	3	1	0.01644
2	3	1	0.03027	2	3	1	0.01483	2	4	1	0.02388

is also able to learn the problem classification. The further elimination of a node renders the network with only one hidden nodes. This is not sufficient to learn the problem, and a node is added. This process is continued and the algorithm oscillates around the optimum architecture as can be seen from Table 3.2. It is also evident that 2 x 2 x 1 is the appropriate architecture for the problem.

The 8-to-1 Decoder Problem

Eight distinct patterns can be made using three booleans. This problem involves firing one of eight output nodes for each of the eight patterns. So there are three inputs and eight outputs. The training data for this problem are in Table 3.3. As with the exclusive-nor problem, training is started with one hidden layer. So the starting architecture is 3 x 1 x 8. The training history for this problem can be found in Table 3.4. The training process is very similar to that for the exclusive-

Table 3.3: 8-to-1 decoder training data

Pattern									
1	Inputs	0.0	0.0	0.0					
	Outputs	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	Inputs	0.0	0.0	1.0					
	Outputs	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	Inputs	0.0	1.0	0.0					
	Outputs	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	Inputs	0.0	1.0	1.0					
	Outputs	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
5	Inputs	1.0	0.0	0.0					
	Outputs	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
6	Inputs	1.0	0.0	1.0					
	Outputs	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
7	Inputs	1.0	1.0	0.0					
	Outputs	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
8	Inputs	1.0	1.0	1.0					
	Outputs	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

nor problem. The target RMS error was 0.01. The network kept adding nodes till the RMS error finally fell below the target with six hidden nodes. Then nodes began to be deleted until at three hidden nodes, the network was unable to learn the problem. Following this, nodes began to be added and the oscillations noticed in the previous example is witnessed here too. As can be seen from Table 3.4, the optimum architecture arrived at by the DNA scheme is 3 x 4 x 8.

The Probability Density Function Separator Problem

In this problem, an artificial neural network with DNA is taught to recognize which of two probability density functions (pdf's) was used to sample a set of ten

Table 3.4: Dynamic node architecture training history for the 8-to-1 decoder problem

Arch.			RMS	Arch.			RMS	Arch.			RMS
In	Hid	Out	Error	In	Hid	Out	Error	In	Hid	Out	Error
3	1	8	0.51626	3	4	8	0.12675	3	5	8	0.06223
3	1	8	0.48722	3	4	8	0.00986	3	5	8	0.00988
3	2	8	0.42163	3	3	8	0.22187	3	4	8	0.05029
3	2	8	0.29358	3	3	8	0.08341	3	4	8	0.00906
3	3	8	0.33542	3	4	8	0.16794	3	3	8	0.11476
3	3	8	0.19744	3	4	8	0.02108	3	3	8	0.07338
3	4	8	0.21927	3	5	8	0.06048	3	4	8	0.14273
3	4	8	0.08952	3	5	8	0.00980	3	4	8	0.03227
3	5	8	0.12481	3	4	8	0.09443	3	5	8	0.09103
3	5	8	0.02275	3	4	8	0.00991	3	5	8	0.02466
3	6	8	0.05181	3	3	8	0.24218	3	6	8	0.04019
3	6	8	0.00996	3	3	8	0.08280	3	6	8	0.00980
3	5	8	0.08264	3	4	8	0.10042	3	5	8	0.08263
3	5	8	0.00992	3	4	8	0.03159	3	5	8	0.00957

numbers. The two functions were

$$f(x) = 0.5\pi cos(\pi x/2) \tag{3.4}$$

and

$$f(x) = 4x^3. \tag{3.5}$$

These equations and their graphs can be seen in Figure 3.1. If the pdf given by *Equation*3.4 (function A) is used then the expected output is 0.000; if the pdf given by *Equation*3.5 (function B) is used then the expected output is 1.000. The training data for this problem can be found in Table 3.5. Table 3.6 gives the training history for this problem. The training process is similar to the previous two examples. The target RMS error in this case was 0.05. The starting architecture was 10 x 1 x 1. The network expands to four hidden nodes before it reaches the target RMS error.

FUNCTION A : $f(X) = 0.5\pi \mathrm{Cos}(\pi X/2)$

OUTPUT : 0.00

FUNCTION B : $f(X) = 4X^3$

OUTPUT : 1.00

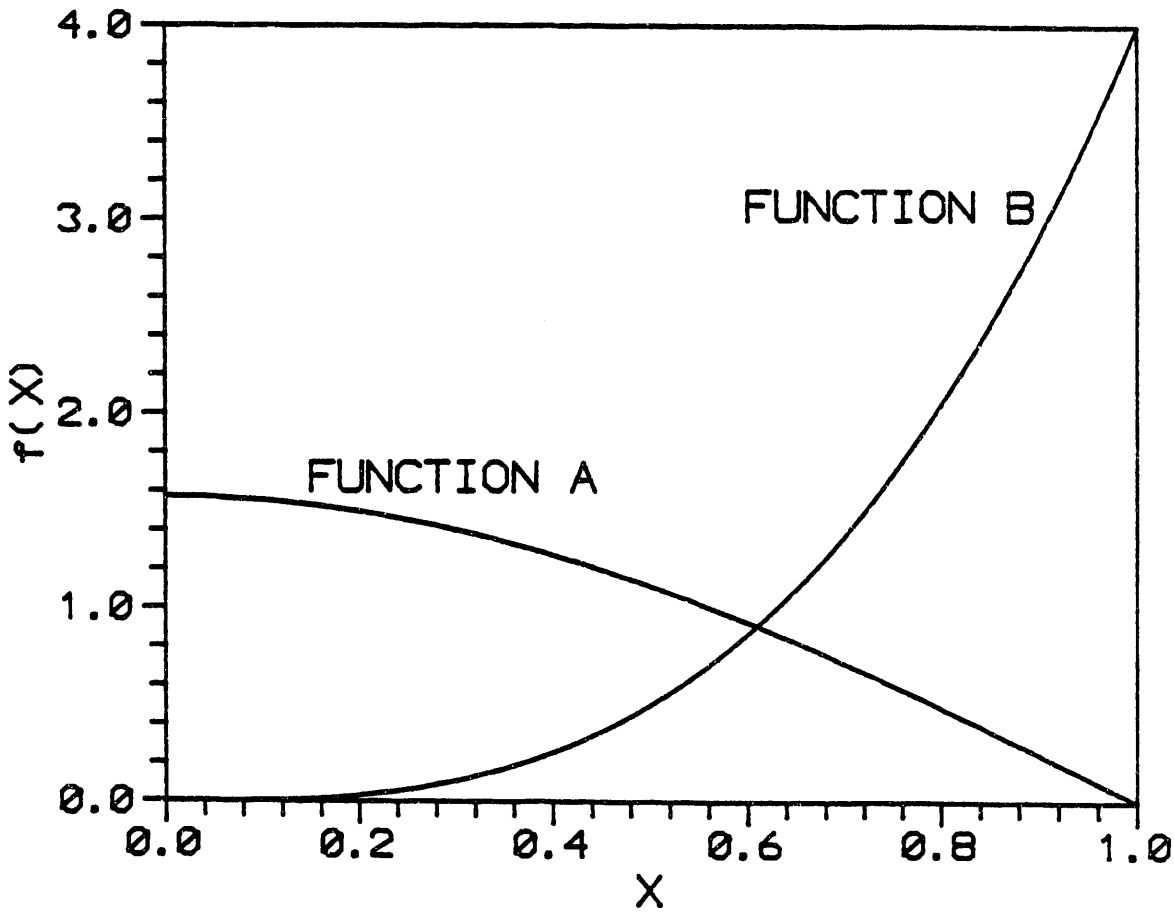

Figure 3.1: The probability density functions used in the pdf seperator problem

Table 3.5: Probability density function separator training data

		I	n	p	u	t	s			Output
0.388	0.099	0.327	0.892	0.394	0.036	0.505	0.571	0.383	0.454	0.000
0.628	0.996	0.487	0.628	0.953	0.873	0.365	0.917	0.899	0.666	1.000
0.892	0.099	0.394	0.034	0.695	0.619	0.223	0.011	0.126	0.020	0.000
0.996	0.628	0.873	0.480	0.970	0.953	0.766	0.365	0.666	0.425	1.000
0.099	0.034	0.619	0.091	0.350	0.695	0.316	0.223	0.020	0.120	0.000
0.628	0.480	0.953	0.614	0.850	0.970	0.830	0.766	0.425	0.658	1.000
0.034	0.091	0.695	0.227	0.580	0.350	0.375	0.316	0.120	0.503	0.000
0.480	0.614	0.970	0.769	0.943	0.850	0.863	0.830	0.658	0.918	1.000

Table 3.6: Dynamic node architecture training history for the probability density function separator problem

Arch.			RMS	Arch.			RMS	Arch.			RMS
In	Hid	Out	Error	In	Hid	Out	Error	In	Hid	Out	Error
10	1	1	0.4999	10	3	1	0.0662	10	3	1	0.0992
10	2	1	0.3454	10	4	1	0.5019	10	4	1	0.1492
10	2	1	0.2055	10	4	1	0.4404	10	4	1	0.0623
10	3	1	0.2819	10	5	1	0.3216	10	5	1	0.0812
10	3	1	0.1448	10	5	1	0.0422	10	5	1	0.0488
10	4	1	0.1403	10	4	1	0.0918	10	4	1	0.1008
10	4	1	0.0413	10	4	1	0.0488	10	4	1	0.0402
10	3	1	0.1304	10	3	1	0.1281	10	3	1	0.0961

From the training history, it is apparent that the DNA algorithm gives an optimum architecture of 10 x 4 x 1. The computer program saved the weights of the network of this architecture with the least RMS error.

Recall performance

Further computer simulations were carried out to compare the performance of DNA generated networks and networks trained by conventional Fixed Node Architecture (FNA) schemes. These comparisons were carried out on the exclusive-nor

and 8-to-1 decoder problems.

For the exclusive-nor problem the DNA algorithm gave an optimum network of size 2 x 2 x 1. Two more networks with 5 and 10 hidden nodes are trained on the exclusive-nor problem using an FNA scheme. These networks were trained to the same level of RMS error. Then all the three networks were used to recall on data corrupted by uniform noise. A similar experiment was done for the 8-to-1 decoder problem. The results of these experiments can be found in Table 3.7.

Table 3.7: Comparative recall performance of neural networks derived by DNA and FNA schemes

Problem	Architecture	Scheme	Train. RMS error	Recall RMS error
Exclus-	2 x 2 x 1	DNA	9.9956E-03	0.00999
ive-nor	2 x 5 x 1	FNA	9.9278E-03	0.01831
	2 x 10 x 1	FNA	9.7582E-03	0.01983
8-to-1	3 x 4 x 8	DNA	9.0636E-03	0.00906
decoder	3 x 8 x 8	FNA	9.3911E-03	0.03642
	3 x 10 x 8	FNA	9.6355E-03	0.03849

As can be seen from Table 3.7, the networks trained with the DNA scheme did a better job of generalization. This is reflected in their low recall RMS errors. The bigger networks, trained to a similar level of accuracy, did not generalize well and hence had a poor recall performance on noisy data. Also to be noted is that the performance deteriorated as the size of the network increased.

Conclusions from the above experiments

The above experiments prove that DNA works well with backpropagation. They also show that it is advantageous to use the optimum network architecture to solve

a problem. A large network may train on the training set, but is not a very good generalizer. This leads to poor recall performance on unseen patterns. The training history for the various problems suggest that it takes more nodes to learn a problem than to recall the same.

Having proved the viability of the DNA algorithm, it will now be applied to solve the nuclear power plant diagnostic problem.

CHAPTER 4. THE PROBLEM AND ITS SOLUTION

Introduction

The specific problem investigated in this thesis is the design of an Artificial Neural Network (ANN) based nuclear power plant status diagnostic advisor. This advisor is expected to correctly identify and classify seven distinct transients and the normal operating conditions. This work is expected to demonstrate the viability of ANNs to solve these types of problems. For a meaningful conclusion, the transients investigated needed to be much varied in nature. ANNs which can recognize such a diverse range of transients need to be able to draw information from a lot of plant variables. The choice of accidents and variables were thus very important for the project. Two major documents were consulted for the purpose. These were the *Updated Final Safety Analysis Report*,Chapter 15 : Accident Analysis [37] and the *Malfunction Cause and Effects Report* [21], both by the Duane Arnold Energy Center (DAEC). These documents describe most of the power plant transients of interest. Intensive discussions between personnel at DAEC and fellow researchers at ISU [18] resulted in a preliminary list of transients to be simulated and plant variables to be monitored [29]. From these transients, seven distinct ones were selected for this work. A description of each of these transients is given in Appendix A. These transients are the design basis loss of coolant accident (RR15A), main feedwater line

break inside primary containment (FW17A), loss of feedwater heater (MS14), High Pressure Coolant Injection (HPCI) steam supply line break inside the HPCI room (HP05), trip of condensate pump (FW02), main steam line break inside primary containment (MS03) and trip of main circulating water pump (MC01). The data were obtained from the DAEC operators training simulator.

Data Collection and Processing

Data were collected for eighty-one plant variables at intervals of one second as the simulated transients progressed. These variables were selected from the complete list of computer points available on the simulator [27]. They were decided to be sufficient to diagnose the transients currently being investigated [29]. A listing of these variables can be found in Table 4.1. The data consisted of the numerical values of these eighty-one variables and the time corresponding to each set of values. The data also consisted of a boolean that indicated the onset of the transient. The data before the transient started related to normal plant operating conditions. The raw data, off the simulator, was in a very unusable form. Codes written by Lanc [29] and myself were used to reformat the data in a form that could be used by neural networks.

Normalization of the Data

Neural networks require normalized data as input. One of the options available was to normalize the values of each variable based on the range of that variable over the seven transients. But this approach would be cumbersome for future work. As the scope of the project increases to investigate more transients, the data would need

Table 4.1: The eighty-one plant variables used

No.	Variable	Description
1)	A041	Longitudinal Power Range Monitor 16-25 Flux Level - Channel B
2)	A091	Source Range Monitor - Channel B
3)	B000	Axial Power Range Monitor - Flux Level - Channel A
4)	B012	Reactor Total Core Flow
5)	B013	Reactor Core Differential Pressure
6)	B014	Control Rod Drive System Flow
7)	B015	Rx Feed-Water Loop A Flow (Temp. Corrected)
8)	B016	Rx Feed-Water Loop B Flow (Temp. Corrected)
9)	B017	Clean-up System Flow
10)	B022	Total System Flow
11)	B023	Clean-up System Inlet Temperature
12)	B024	Clean-up System Outlet Temperature
13)	B026	Recirculation Loop A1 Drive Flow
14)	B028	Recirculation Loop B1 Drive Flow
15)	B030	Rx Feed-Water Channel A1 Temperature
16)	B032	Rx Feed-Water Channel B1 Temperature
17)	B034	Recirculation Loop A1 Inlet Temperature
18)	B036	Recirculation Loop B1 Inlet Temperature
19)	B038	Recirculation Loop A1 Wide Range Temperature
20)	B039	Recirculation Loop B1 Wide Range Temperature
21)	B061	Rx Jet Pumps 1-8 Flow (Channel B)
22)	B062	Rx Jet Pumps 9-16 Flow (Channel A)
23)	B063	Reactor Outlet Steam Flow - Channel A
24)	B064	Reactor Outlet Steam Flow - Channel B
25)	B065	Reactor Outlet Steam Flow - Channel C
26)	B066	Reactor Outlet Steam Flow - Channel D
27)	B079	RRP A MTR vibration
28)	B080	RRP B MTR vibration
29)	B083	Control Rod Drive Drive-Water Differential Pressure
30)	B084	Control Rod Drive Cooling-Water Differential Pressure
31)	B085	Torus Air Temperature Sensor #1
32)	B086	Torus Air Temperature Sensor #2
33)	B087	Torus Air Temperature Sensor #3
34)	B088	Torus Air Temperature Sensor #4
35)	B089	Drywell Temperature at AZ EL750
36)	B090	Drywell Temperature at AZ245 EL750
37)	B091	Drywell Temperature at AZ090 EL765
38)	B092	Drywell Temperature at AZ270 EL765
39)	B093	Drywell Temperature at AZ270 EL765
40)	B094	Drywell Temperature at AZ180 EL780
41)	B095	Drywell Temperature at AZ270 EL830

Table 4.1 (Continued)

No.	Variable	Description
42)	B096	Drywell Temperature at CENTER EL750
43)	B098	Torus Water Temperature
44)	B099	Torus Water Temperature
45)	B103	ILRT Drywell Pressure
46)	B104	ILRT Torus Temperature
47)	B105	Torus Water Level
48)	B120	Torus Radiation Monitor A
49)	B121	Torus Radiation Monitor B
50)	B122	Reactor Water Level
51)	B125	Reactor Water Level
52)	B126	Reactor Water Level
53)	B137	Torus Water Level
54)	B138	Torus Water Level
55)	B150	Core Spray A Flow
56)	B151	Core Spray B Flow
57)	B160	RCIC Flow
58)	B161	HPCI Flow
59)	B162	Reheater A Flow
60)	B163	Reheater B Flow
61)	B164	Drywell Radiation Monitor - A
62)	B165	Drywell Radiation Monitor - B
63)	B166	Post Treatment Activity
64)	B168	Pre Treatment Activity
65)	B171	Analyzer A - O_2 Concentration
66)	B172	Analyzer A - H_2 Concentration
67)	B173	Analyzer B - O_2 Concentration
68)	B174	Analyzer B - H_2 Concentration
69)	B180	Clean-up System Flow
70)	B196	Reactor Water Level in Fuel Zone A
71)	B197	Reactor Water Level in Fuel Zone B
72)	B247	Turbine Steam Bypass
73)	B248	Turbine Steam Bypass
74)	E000	4160V Switchgear Bus 1A1 A-B
75)	F004	Condensate Pump A&B Discharge Pressure
76)	F040	1P-1A Rx Feed-Water Pump Suction Pressure
77)	F041	1P-1B Rx Feed-Water Pump Suction Pressure
78)	F042	1P-1A Rx Feed-Water Pump Discharge Pressure
79)	F043	1P-1B Rx Feed-Water Pump Discharge Pressure
80)	F094	Feed-Water Final Pressure
81)	G001	Generator Gross Watts

to be renormalized. It was decided to normalize the values of each variable based on the maximum and minimum possible values of that variable [31]. In this way, there is no need to renormalize the data as more scenarios are investigated. Also, when the advisor is actually taken out to the plant (or simulator) at a later stage, the normalization of the input data will be needed to be done in real time, and this approach will make that possible. It is to be noted here that the normalization was done in the range 0.1 to 0.9. A look at the sigmoid function (Figure 2.5) shows that it is relatively easy to reach an output of 0.1 or 0.9, but it might be difficult to train the network to output in the extremities 0.0 and 1.0. In the same spirit, the input is also limited between these two values. A normalized value of 0.9 corresponds to a meter reading of 100% for the particular variable.

Final Data Set

The trained ANN would be classifying seven transients and normal operating conditions. The trained advisor would be expected to identify if the plant was in a normal operating condition, and if not, which of these seven transients it was going through. This requires eight distinct output patterns which can be formed by three booleans. One pattern was assigned to each of the seven transients and a pattern was assigned to all the normal conditions.

The final data set was put together for all the seven transients which consisted of the input patterns at each time-slice and the expected output for that pattern. Each time-slice input pattern consisted of the values of the 81 plant variables at the time the pattern was collected. This pattern had no temporal information in it. Some sample patterns are included in Appendix C. The final data set had 2566 patterns

corresponding to the entire length of simulation of the seven transients.

Developing the Advisor

Training

The values of the eighty-one variables at any given time was expected to contain enough information to make it possible to look at them at any instance of time and diagnose the plant status [2][3]. The problem dictated that there were 81 input nodes and 3 output nodes. The training set was chosen in an iterative manner. In the first trial, one pattern at the beginning and one at the end of each simulation were taken to form the training set. Training was initiated with one hidden node. As the training progressed, the DNA scheme added more nodes and finally gave an optimum architecture with 7 hidden nodes for the first trial. This first trial had 14 training patterns and was trained to an RMS error of 0.10. This trained network was now used to recall on the whole length of the seven simulations, and the RMS errors of the outputs for each of the patterns was plotted out. Obviously, the network did not do a very good job of classifying all the patterns. The patterns with the worst recall errors were added to the training data set and the network from the previous trial was trained further. This process was continued until the network could detect and classify all the transients within a reasonable amount of time. Care is taken so that patterns too close to the initiating events are not included in the training set. This is because these patterns correspond to a highly unstable plant condition. Inclusion of these in the training set might make the network grow in size in an effort to memorize them resulting in decreased generalization capabilities of the network.

It is to be noted here that this problem is slightly different from conventional

Table 4.2: Training information of the advisor

Trial	No. of training patterns	Starting architecture	Ending architecture
1	14	81 x 1 x 3	81 x 7 x 3
2	56	81 x 7 x 3	81 x 13 x 3
3	90	81 x 13 x 3	81 x 17 x 3
4	127	81 x 17 x 3	81 x 22 x 3
5	169	81 x 22 x 3	81 x 24 x 3
6	193	81 x 24 x 3	81 x 26 x 3
7	230	81 x 26 x 3	81 x 30 x 3
8	246	81 x 30 x 3	81 x 30 x 3
9	262	81 x 30 x 3	81 x 35 x 3
10	264	81 x 35 x 3	81 x 36 x 3

problems that are solved using neural networks. In most cases, a training set is given, and a network is trained based on that. The recall set is not known beforehand, and the trained network is then used to recall on unseen patterns. But in this case, the recall set is known from the simulations. We then try to define the training set so that it is a fairly accurate representation of the recall set. In fact, through the various trials, we force it to be so. Consequently, good generalization is extremely important in order to keep the number of patterns in the training set to a minimum. In the polynomial interpolation analogy (see page 29), this is comparable to finding the smoothest interpolation curve for the given points, and then adding the points that are off the curve and finding the interpolation curve all over again.

Results

Table 4.2 contains information about the various trials. As can be seen there, the final architecture of the advisor was 81 x 36 x 3. The advantages of the DNA algorithm can be appreciated here. Suppose a conventional fixed architecture scheme

were used to solve this problem. Then, for the first trial, a few architectures need to be guessed at, training done on all of them, and the architecture with the best performance used for later trials. But the training set changes, in fact increases, with each trial. So there is no guarantee that the best architecture for a given trial will also be good enough for the following trials. Table 4.2 shows that the network size needs to increase as the training size increases. A fixed node architecture scheme would have been extremely inconvenient in developing the advisor.

The recall RMS errors of the advisor over the seven simulations can be seen in Figures 4.1 through 4.7. Information about the seven transients can be seen in Table 4.3.

As can be seen, all the transients are detected within a reasonable period of time. The only exception is the trip of the main circulation water pump. This transient took over 75 seconds after the initiating event to be diagnosed by the advisor. A look at the transient description in Appendix A shows that this transient first manifests itself in the electrical components of the plant. An examination of the plant variables monitored (Table 4.1) reveals that not many variables pertaining to electrical components were monitored. This might be why the transient required so much time to be diagnosed.

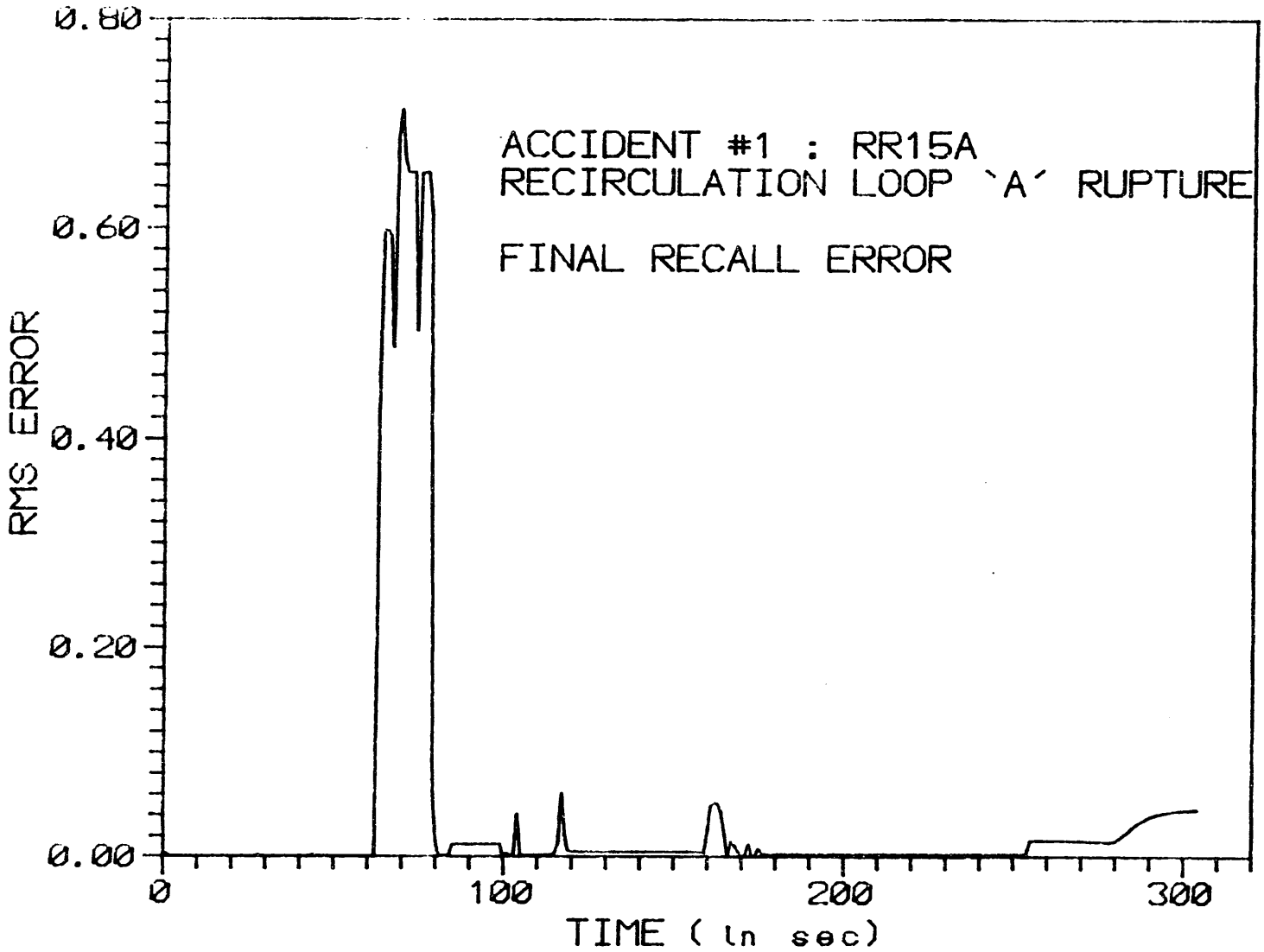

Figure 4.1: Recall error for recirculation loop 'A' rupture (RR15A)

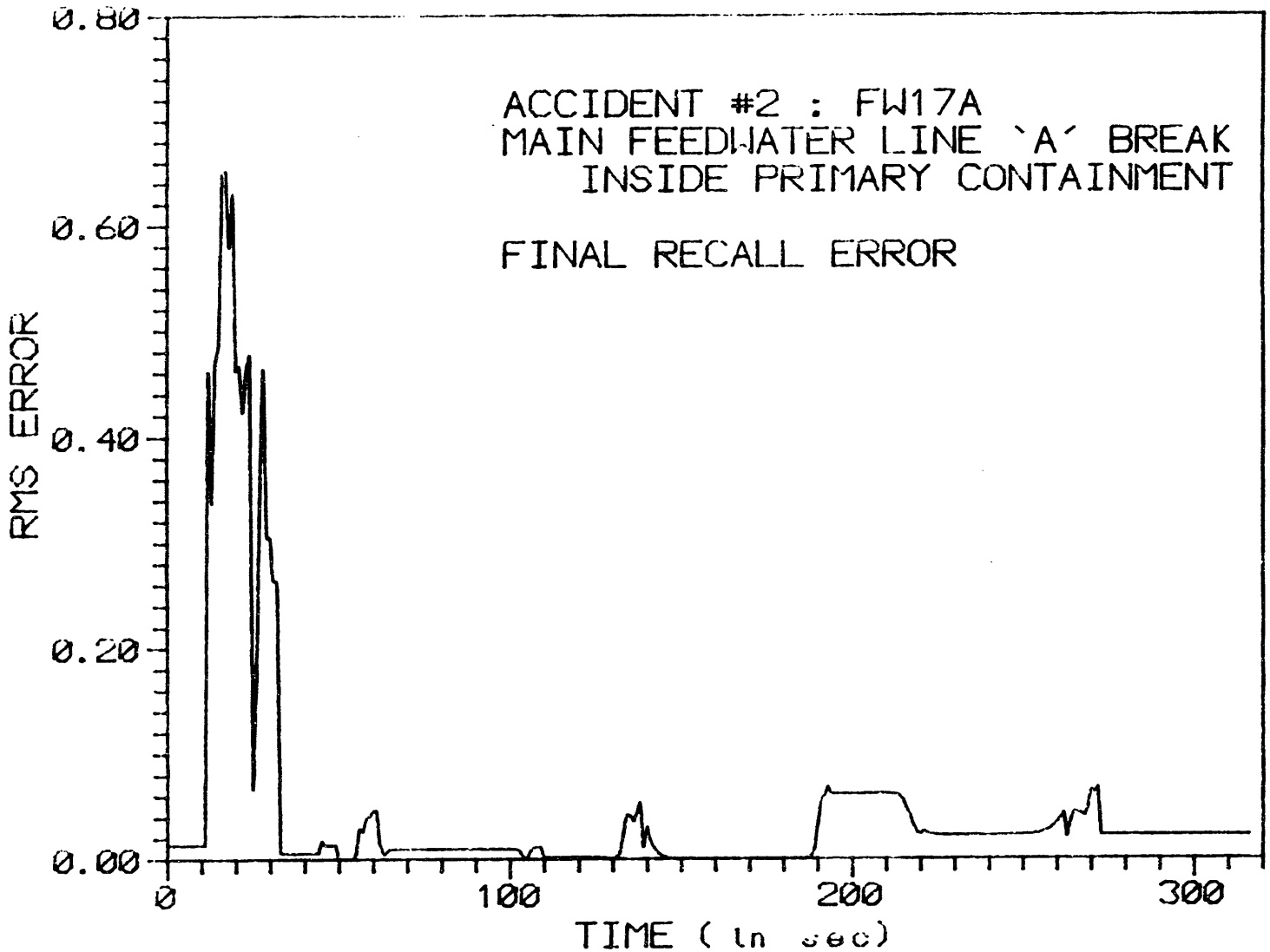

Figure 4.2: Recall error for main feedwater line 'A' break inside primary containment (FW17A)

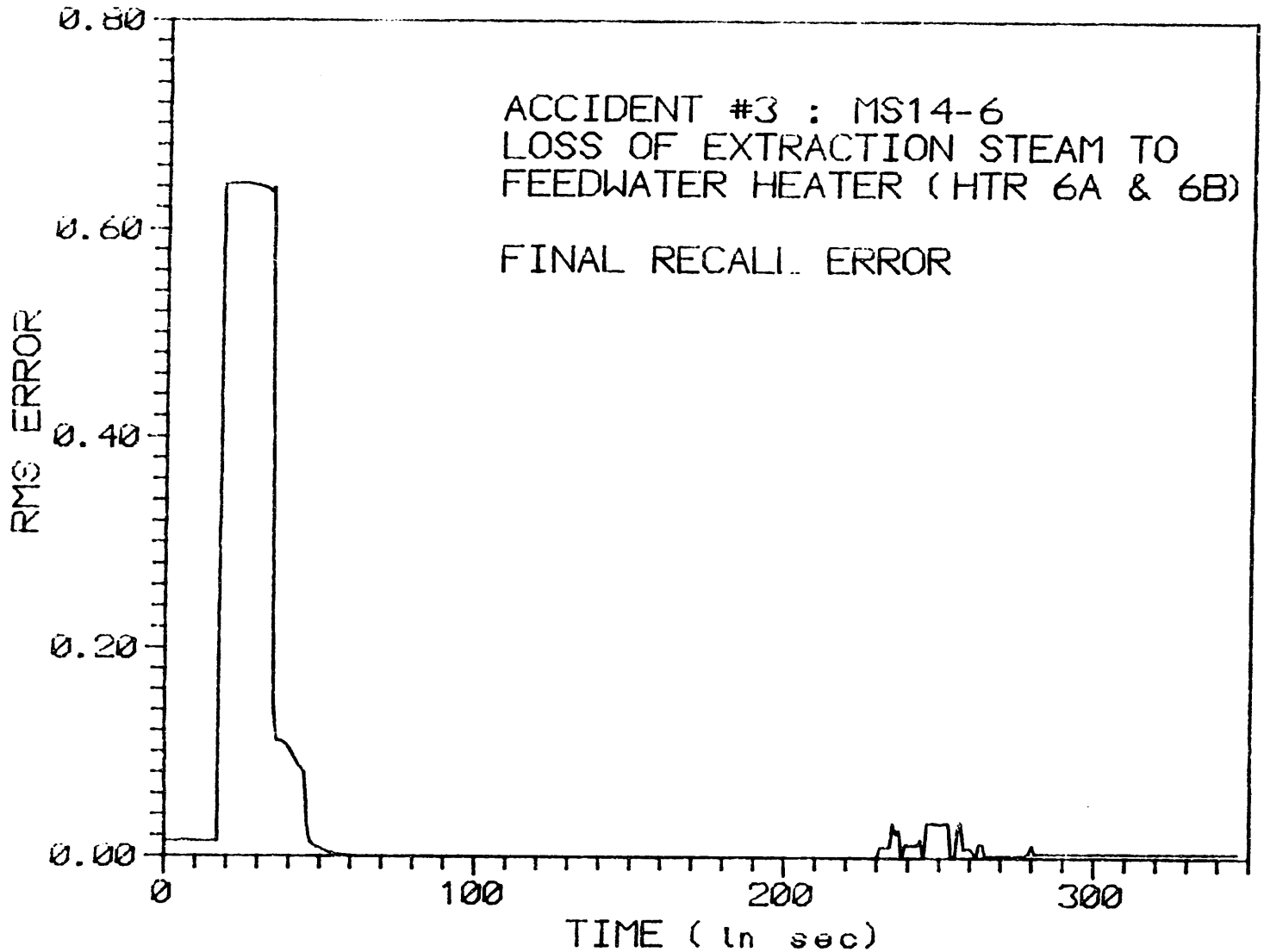

Figure 4.3: Recall error for loss of feedwater heater (MS14)

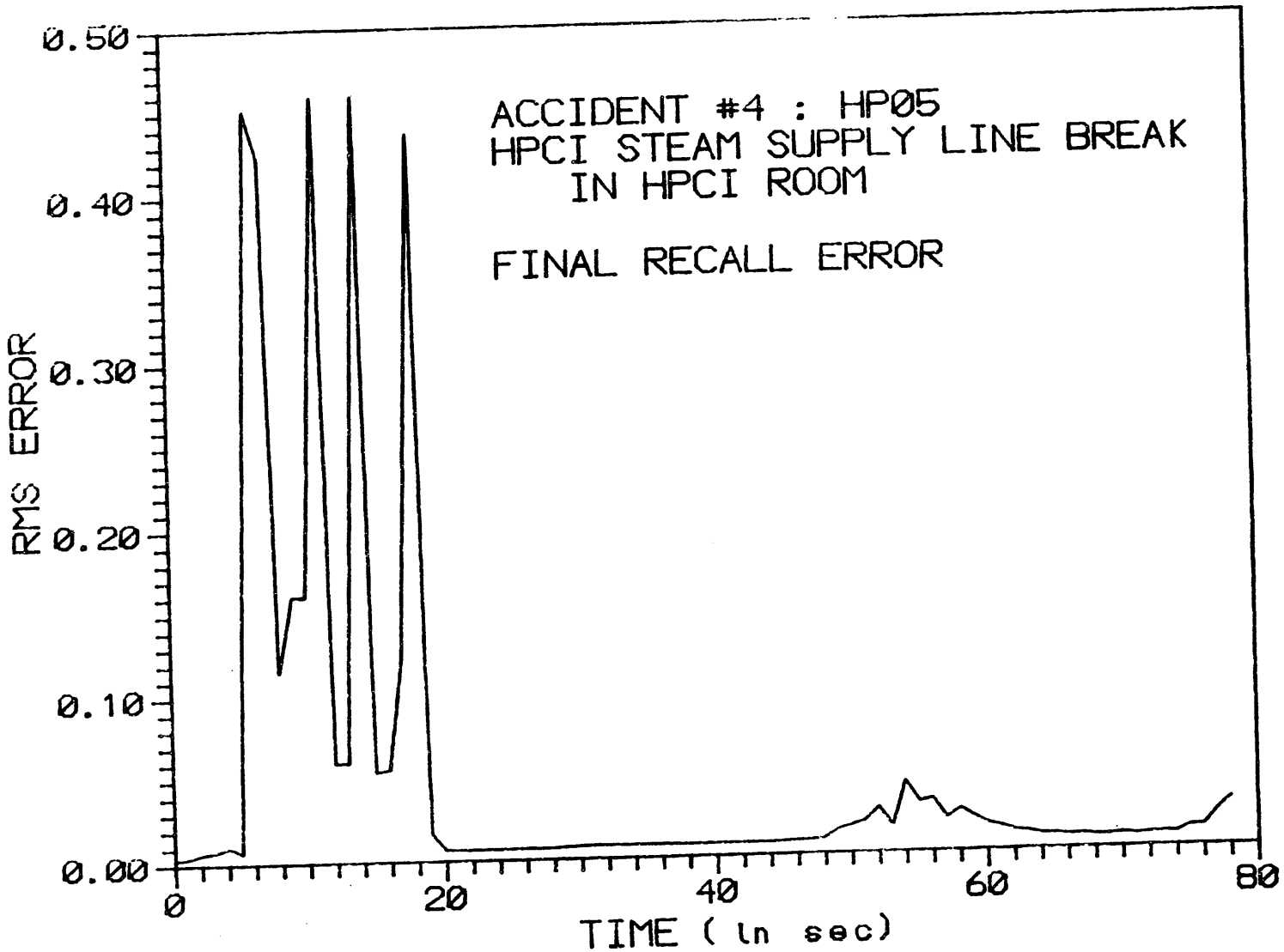

Figure 4.4: Recall error for HPCI steam supply line break in HPCI room (HP05)

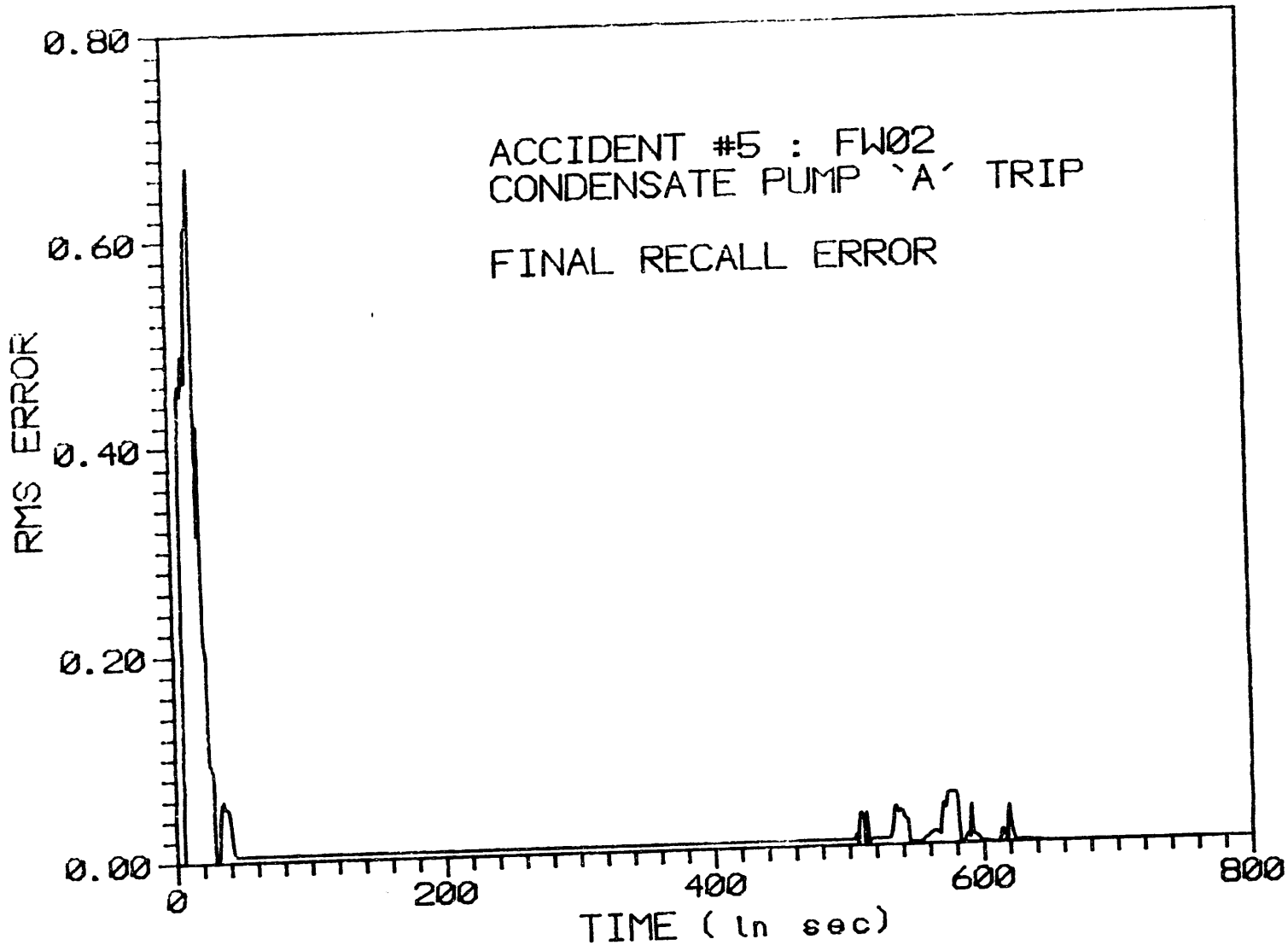

Figure 4.5: Recall error for trip of condensate pump 'A' (FW02A)

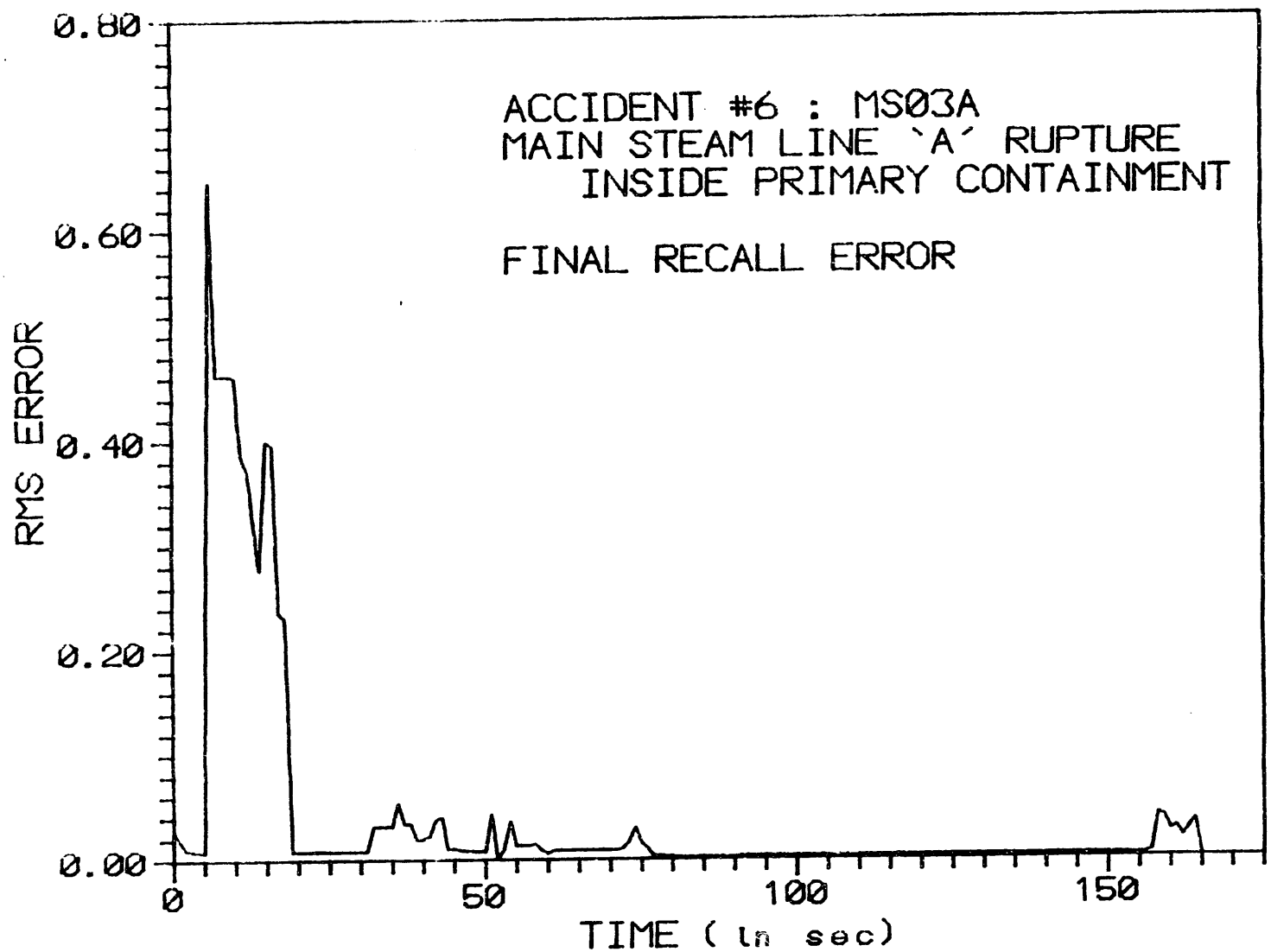

Figure 4.6: Recall error for rupture of main steam line 'A' inside primary containment (MS03A)

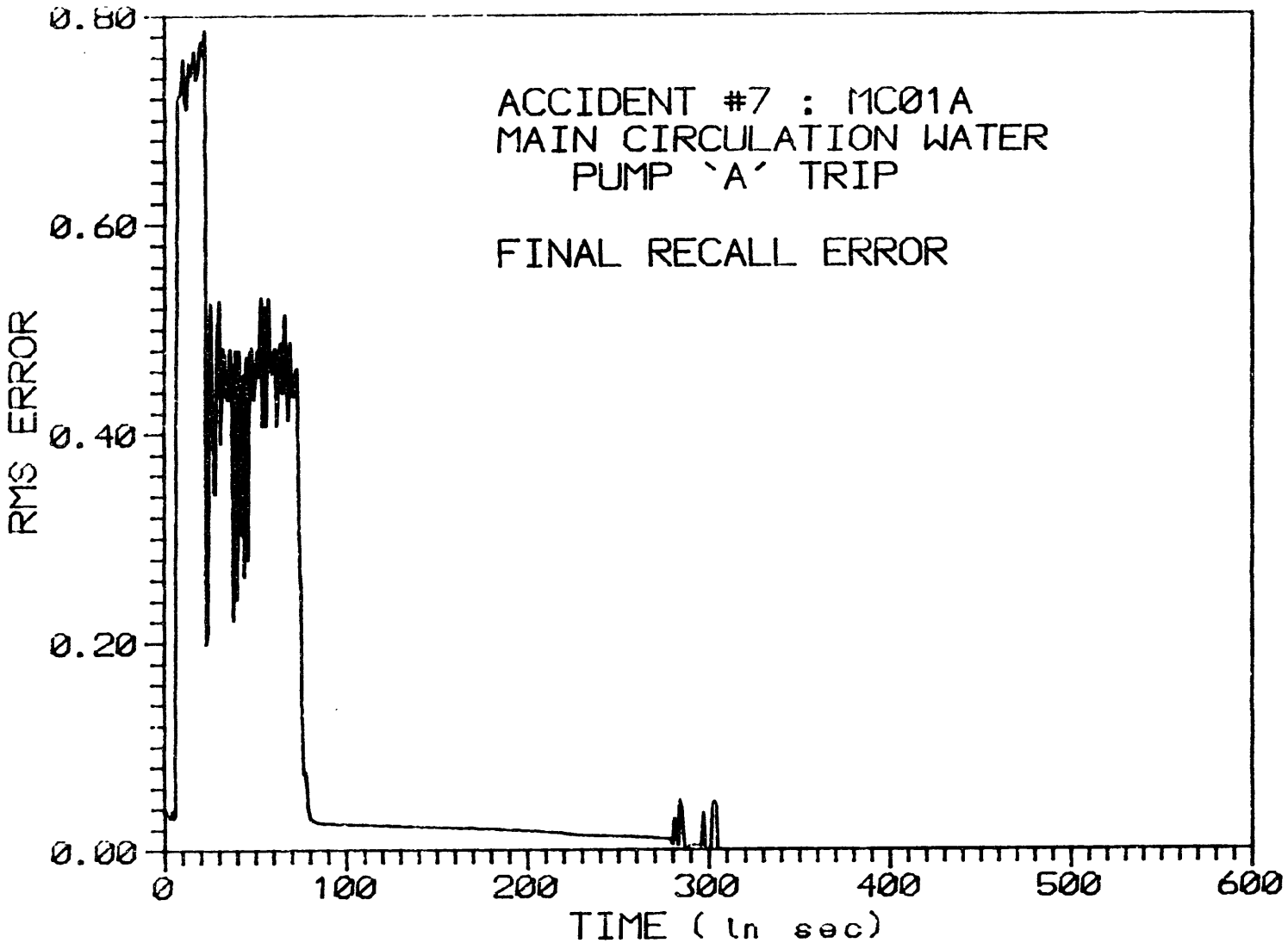

Figure 4.7: Recall error for trip of main circulation water pump 'A' (MC01A)

Table 4.3: Network output, simulation time, time to scram, and time to diagnose for the seven transients

Transient scinario	Desired output node activation			Trans. simulation time (sec)	Initiating event (sec)	Time to scram (sec)@	Time to diag. trans. (sec)@
	1	2	3				
Recirculation loop 'A' rupture inside primary containment (RR15)	0.1	0.1	0.9	304	63	1	17
Main feedwater line 'A' break inside of primary containment (FW17)	0.1	0.9	0.1	316	11	4	22
Loss of extraction steam to feedwater heater (MS14)	0.1	0.9	0.9	346	17	213	32
HPCI steam supply line break inside HPCI room (HP05)	0.9	0.1	0.1	78	5	--	15
Condensate pump 'A' trip (FW02)	0.9	0.1	0.9	764	6	--	38
Main steam line 'A' rupture inside primary containment (MS03)	0.9	0.9	0.1	165	5	8	14
Main circ water pump 'A' trip (MC01)	0.9	0.9	0.9	586	6	--	75
Normal condition	0.1	0.1	0.1	---	--	--	--

@ : Time since the initiating event.

CHAPTER 5. CONCLUSIONS

The first major conclusion from this thesis is the viability of a derivative importance function based Dynamic Node Architecture scheme to derive the optimum network architecture for any given problem. This work also demonstrates the viability of using neural networks to detect operational transients in boiling water reactor (BWR) nuclear power plants. An ANN was successfully trained to detect and classify seven distinct transients and the normal conditions. The purpose of this thesis was to develop the DNA scheme, and to apply it to the nuclear power plant status diagnostic problem. This was accomplished in its entirety.

The DNA scheme evolved a systematic method to arrive at the optimum architecture for a problem. It eliminated the guesswork associated with preset architectures used in conventional neural network training schemes. This helped in using neural networks in solving an intractable problem of the kind involving nuclear power plant diagnostics. The large amount of guesswork that would have been otherwise required to come up with the optimum architecture was eliminated.

The demonstrations with the DNA scheme showed that the networks with the optimum architectures had a better recall than larger networks trained to the same level of accuracy. It also showed the greater noise tolerance capabilities of the DNA derived networks compared to FNA derived networks.

The most important contribution of this work has been the demonstration of the viability of using ANNs to solve the nuclear power plant diagnostic problem. The advisor designed here was successfully able to recognize and classify seven distinct transients in reasonable amounts of time. The ANN utilized the values of all the eighty-one variables for which data were collected. The seven transients used for this study were much varied in nature. The network did a very decent job of detecting all of them, other than the one (MC01) for which the proper variables were not monitored.

Possible Future Work

This work opens the doors to a lot of possible future work. One of the first steps would be to widen the scope of the transients investigated. Along with an increase in the transient list, it will be imperative to look at more plant variables. One possible approach is to group the transients into lesser number of categories and use a base network to identify the category of a transient. Smaller networks can be trained to identify transients within a category once the category is decided by the base network.

Another avenue for research is to identify which plant variables are important to identify the transients, and which variables are redundant and confuse a network. Work done by Lanc [29] is important in this respect. The derivative importance function presented here and in [5] was used to cut this very same list of eighty-one variables down to twenty to recognize three of the transients. This indicates that all the eighty-one variables might not have been necessary for the present problem. If results presented in [29] are taken into consideration, the use of all the eighty-

one variables might have even made the training problem difficult. If transients are classified into different groups, then some variables might be important to identify transients in a certain group, while they might be totally unnecessary or misleading for another group. Thus research in this field will be important for the successful design of a broad-based nuclear power plant status diagnostic advisor.

BIBLIOGRAPHY

[1] T. Ash. "Dynamic Node Creation in Backpropagation Networks," IJCNN International Conference on Neural Networks, vol. 2. Washington D.C., June 1989. 623.

[2] E.B. Bartlett and R.E. Uhrig. "Nuclear Power Plant Status Diagnostics Using Artificial Neural Networks," Proceedings of the American Nuclear Society Meeting on Frontiers in Innovative Computing for the Nuclear Industry. September, 1991. 644-653.

[3] E.B. Bartlett and R.E. Uhrig. "Nuclear Power Plant Status Diagnostics Using Artificial Neural Networks." Nuclear Technology 97 (March, 1992): 272-281.

[4] E.B. Bartlett. "Nuclear Power Plant Status Diagnostics Using Simulated Condensation: An Auto-Adaptive Computer Learning Technique." Ph.D. Dissertation, University of Tennessee at Knoxville. 1990.

[5] E.B. Bartlett and Anujit Basu. "A Dynamic Node Architecture Scheme for Backpropagation Neural Networks." Intelligent Engineering Systems Through Artificial Neural Networks, Eds. C.H. Dagli, S.R.T. Kumara, and Y.C. Shin. New York: ASME Press, 1991. 101-106.

[6] W.H. Beyer. CRC Standard Mathematical Tables, 28th edition. Boca Raton, Florida: CRC Press, Inc., 1987. 347-348.

[7] M. Caudill. "Neural Networks Primer, Part 1." AI Expert 6 (Dec, 1987): 46-52.

[8] M. Caudill. "Neural Networks Primer, Part 2." AI Expert 7 (Feb, 1988): 55-61.

[9] M. Caudill. "Neural Networks Primer, Part 3." AI Expert 7 (June, 1988): 53-59.

[10] M. Caudill. "Neural Networks Primer, Part 4." AI Expert 7 (August, 1988): 61-67.

[11] M. Caudill. "Neural Networks Primer, Part 5." AI Expert 7 (Nov, 1988): 57-65.

[12] M. Caudill. "Using Neural Nets: Representing Knowledge, Part 1." AI Expert 8 (Dec, 1989): 34-41.

[13] M. Caudill. "Using Neural Nets: Fuzzy Decisions, Part 2." AI Expert 9 (April, 1990): 59-64.

[14] M. Caudill. "Using Neural Nets: Fuzzy Cognitive Maps, Part 3." AI Expert 9 (June, 1990): 49-53.

[15] M. Caudill. "Using Neural Nets: Making an Expert Network, Part 4." AI Expert 9 (July, 1990): 41-45.

[16] M. Caudill. "Using Neural Nets:Diagnostic Expert Nets, Part 5." AI Expert 9 (Sept, 1990): 43-47.

[17] Ward Cheney and David Kincaid. Numerical Mathematics and Computing . Montery, California: Brooks/Cole Publishing Company, 1985.

[18] Duane Arnold Energy Center simulator complex employees Don Vest, Criag Hunt and Dan Berchenbriter. Personal discussions and correspondence. Iowa Electric Power and Light Company, Palo, IA. 1991-1992.

[19] Adapted from a seminar given by Craig Harston of Computer Application Systems Inc., Signal Mts.,TN at the University of Tennessee, Knoxville, 1989.

[20] W. J. Freeman. "The Physiology of Perception." Scientific American (February, 1991): 78-85.

[21] J. Gould. Malfunction Cause and Effects Report, Task no. 06000004. Cedar Rapids, Iowa: Duane Arnold Energy Center, 1991.

[22] J. Gould. Updated Description of Initiating Events, Task no. 04000103. Cedar Rapids, Iowa: Duane Arnold Energy Center, 1991.

[23] R. Hecht-Nielsen. "Theory of the Backpropagation Neural Network." Proceedings of the International Joint Conference on Neural Networks, vol. 1. 1989. 593-605.

[24] Y. Hirose, K. Yamashita, and S. Hijiya. "Back Propagation Algorithm That Varies the Number of Hidden Nodes." Neural Networks 4 (1991): 61-66.

[25] J.J. Hopfield. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." Proceedings of the National Academy of Sciences 79 (1982): 460-464.

[26] S. Judd. "Learning in Networks is Hard.", IEEE First International Conference on Neural Networks, vol. 2, June 1987. 685-692.

[27] C. Kardos. DAEC Process Computer Points List. Cedar rapids, Iowa: Duane Arnold Energy Center, February 1991.

[28] Ehud D. Karnin. "A Simple Procedure for Pruning Back-Propagation Trained Neural Network." IEEE Transactions on Neural Networks 1.2 (June, 1990): 239-242.

[29] T.L. Lanc. "The Importance of Input Variables to a Neural Network Fault-Diagnostic System for Nuclear Power Plants." M.S. Thesis, Iowa State University, 1991.

[30] R.P. Lippmann. "An Introduction to Computing with Neural Nets." IEEE Acoustics Speech and Signal Processing Magazine 4 (April, 1987): 4-22.

[31] POINTMIN.MAX : The datafile for the minimum and maximum values of the computer points on the simulator at the Duane Arnold Energy Center, Iowa Electric Light and Power Company, Palo, IA. Provided by Don Vest, simulator instructor. 1991.

[32] E.D. Rumelhart, G.E. Hinton, and R.J. Williams. "Learning Internal Representations by Error Propagation." Parallel Distributed Processing: Explorations in the Microstructure of Cognition Vol. 1, Cambridge, Mass.: The MIT Press, 1986. 318-362.

[33] J. Sietsma and R.J.F. Dow. "Neural Net Pruning - Why and How." Proc. IEEE International Conference on Neural Networks, vol. 1, San Diego: July 1988. 325-333.

[34] J. Sietsma and R.J.F. Dow. "Creating Artificial Neural Networks That Generalize." Neural Networks 4 (1989): 67-79.

[35] L. Stark, M. Okajima, and G.H. Whipple. "Computer Pattern Recognition Techniques: Electrocardiographic Diagnosis." Commun. Ass. Comput. Mach. vol. 5 (Oct., 1962): 527-532.

[36] K. Steinbuch and V.A.W. Piske. "Learning Matrices and Their Applications." IEEE Trans. Electron. Comput. EC-12 (Dec, 1963): 846-862.

[37] "Accident Analysis" In Updated Final Safety Analysis Report. Cedar Rapids, Iowa: Duane Arnold Energy Center, 1984.

[38] J. Vaario and S. Ohsuga. "Adaptive Neural Architectures Through Growth Control." Intelligent Engineering Systems Through Artificial Neural Networks, Eds. C.H. Dagli, S.R.T. Kumara, and Y.C. Shin. New York: ASME Press, 1991. 11-16.

[39] T.P. Vogl, J.K. Mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon. "Accelerating the Convergence of the Back-propagation Method." Biological Cybernetics 59 (1988): 257-263.

[40] A.S. Weigend, D,E. Rumelhart, and B.A. Huberman. "Generalization by Weight Elimination Applied to Currency Exchange Rate Prediction." IJCNN International Conference on Neural Networks vol. 1. Seattle, Washington, 1991. 837-841.

[41] Paul J. Werbos. "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences." Ph.D. Thesis, Applied Mathematics, Harvard University, November 1974.

[42] Paul J. Werbos. "Building and Understanding Adaptive Systems: A Statistical/Numerical Approach to Factory Automation and Brain Research." IEEE Transactions on Systems, Man, and Cybernetics Vol. SMC-17, No. 1 (Jan/Feb 1987): 7-20.

[43] B. Widrow and M.A. Lehr. "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation." Proceedings of the IEEE 78.9 (September 1990): 1415-1441.

[44] B. Widrow. "Generalization and Information Storage in Networks of ADALINE 'neurons'." Self Organizing Systems 1962, Eds. M. Yovitz, G. Jacobi, and G. Goldstein. Washington DC: Spartan Books, 1962. 435-461.

[45] Y. Won and R. Pimmel. "A Comparison of Connection Pruning Algorithms with Back-Propagation." Intelligent Engineering Systems Through Artificial Neural Networks, Eds. C.H. Dagli, S.R.T. Kumara, and Y.C. Shin. New York: ASME Press, 1991. 113-119.

END

DATE FILMED

5/28/93