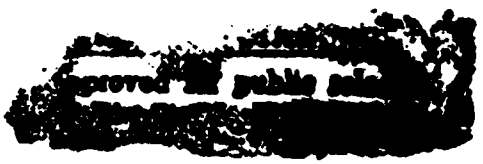

# A FORTRAN BASED LEARNING SYSTEM USING MULTILAYER BACK-PROPAGATION NEURAL NETWORK TECHNIQUES

THESIS

Gregory L. Reinhart
Captain, USAF

AFIT/GOR/ENS/94M-11





DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/94M-11



# A FORTRAN BASED LEARNING SYSTEM USING MULTILAYER BACK-PROPAGATION NEURAL NETWORK TECHNIQUES

THESIS

Gregory L. Reinhart
Captain, USAF


AFIT/GOR/ENS/94M-11



Approved for public release; distribution unlimited

AFIT/GOR/ENS/94M-11

# A FORTRAN BASED LEARNING SYSTEM USING MULTILAYER BACK-PROPAGATION NEURAL NETWORK TECHNIQUES

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Operations Research

Gregory L. Reinhart, B.S.

Captain, USAF

March, 1994

## THESIS APPROVAL

STUDENT: Captain Gregory L. Reinhart CLASS: GOR 94-M

THESIS TITLE: A FORTRAN Based Learning System Using Multilayer Back-Propagation Neural Network Techniques

DEFENSE DATE: 3 MARCH 1994

| COMMITTEE: | NAME/DEPARTMENT | SIGNATURE |
|---|---|---|
| Advisor | Ltc Kenneth W. Bauer/ENS | Kenneth W Bauer |
| Reader | Prof. Daniel E. Reynolds/ENC | Daniel E Reynolds |

## *Preface*

The purpose of this research effort was to develop an interactive computer system which would allow the researcher to build an "optimal" neural network structure as quickly as possible, given a specific problem. The objective was to develop software to assist the researcher in building an appropriate back-propagation neural network. The software enables the researcher to quickly define a neural network structure, run the neural network, interrupt training at any point to analyze the status of the current network, re-start training at the interrupted point if desired, and analyze the final network using two-dimensional graphs, three-dimensional graphs and confusion matrices.

Two, classical, classification problems are used to verify and validate the system:

1. The XOR problem
2. The four class MESH problem

The analysis conducted on these two problems involved finding the "optimal" network architecture for a multilayer perceptron. This optimal architecture was found by varying network parameters such as number of middle nodes, learning rates, and momentum rates. Two and three-dimensional graphs, automatically produced by the system, were analyzed at various stages to see how the activation and saliency surfaces were changing as network parameters changed.

By validating this system using the XOR and MESH problems, the hope is that other practitioners will use this interactive system to build "optimal" network structures for real-world problems.

While performing the analysis, developing the computer code, and writing the thesis, I had a great deal of help from several individuals. First, I am very greatful to Ltc K.W. Bauer, my faculty advisor, for allowing me to develop the computer code first and the thesis narrative second. This sequence of events gave me a thorough understanding of the problem, and ultimately produced a superior final product. His insights and pointed suggestions kept me on track. I am also indebted to Professor D.E. Reynolds for his sound advice of keeping my eye on the big picture, and not allowing me to get buried by the

details. Finally, special thanks goes to my wife Kathy, and my sons Justin and Adam. This project could not have been completed without their sacrifice and faith in me.

Gregory L. Reinhart

*Table of Contents*

*List of Figures*

*Abstract*

An interactive computer system which allows the researcher to build an "optimal" neural network structure quickly, is developed and validated. This system assumes a single hidden layer perceptron structure and uses the back-propagation training technique. The software enables the researcher to quickly define a neural network structure, train the neural network, interrupt training at any point to analyze the status of the current network, re-start training at the interrupted point if desired, and analyze the final network using two-dimensional graphs, three-dimensional graphs, confusion matrices and saliency metrics. A technique for training, testing, and validating various network structures and parameters, using the interactive computer system, is demonstrated. Outputs automatically produced by the system are analyzed in an iterative fashion, resulting in an "optimal" neural network structure tailored for the specific problem. To validate the system, the technique is applied to two, classic, classification problems. The first is the two-class XOR problem. The second is the four-class MESH problem. Noise variables are introduced to determine if weight monitoring graphs, saliency metrics and saliency grids can detect them. Three dimensional class activation grids and saliency grids are analyzed to determine class borders of the two problems. Results of the validation process showed that this interactive computer system is a valuable tool in determining an optimal network structure, given a specific problem.

# A FORTRAN BASED LEARNING SYSTEM USING MULTILAYER BACK-PROPAGATION NEURAL NETWORK TECHNIQUES

## *I.* Introduction

The multilayer back-propagation training procedure for neural networks holds great potential. However, in practice, this training procedure can be a researcher's nightmare. In contrast to most other training procedures, there are many parameters that may be adjusted and may have a *major* effect on the results. This myriad of parameters provides the motivation to develop an interactive tool which will allow the researcher to develop and fine-tune a customized neural network, given a specific problem. The purpose of this thesis is to build and test such an interactive tool.

### *1.1* Background

The back-propagation neural network is the latest contender for "champion" learning system. Sometimes simplistically compared to human biological systems, neural networks were long thought by many to be an impractical representation for learning. However, recent developments have proven this view incorrect, and the back-propagation learning system has created much excitement because of strong theoretical and applied results.

In 1969, Minsky and Papert wrote a book titled *Perceptrons* which had a significant influence in discouraging research on neural networks [12:249-252]. At that time, no procedure had been developed for learning with multilayer neural networks. The great potential for classification and prediction by multilayer neural networks had been discussed since the first generation of perceptrons made their appearance in the late '50s and early '60s. However, it is only in the mid-'80s that a practical multilayer neural network training procedure, known as back-propagation, has emerged. We now know the strong theoretical potential of the multilayer neural network for learning. According to Hornik, a single

hidden layer with sufficient hidden units is capable of approximating any response surface [8:360].

A neural network is a computer program that makes decisions based on the accumulated experience contained in successfully *solved* cases. It extracts decision criteria from samples of solved cases stored in a computer. In many professional fields expertise is scarce, and the codification of knowledge can be quite limited in practice. Expertise in the form of records of solved cases, may be the sole source of knowledge. The argument in favor of neural networks is that they have the potential to exceed the performance of experts and the potential to discover new relationships among concepts and hypotheses by examining the record of succesfully solved cases. In this thesis we examine one of the most prominent techniques for training a neural network: the *Multilayer Back-Propagation* technique. We will confine our attention to the most prominent and basic learning task, that of *classification.*

For classification problems, a neural network can be viewed as a higher-level system that helps build the decision-making system itself, called the *classifier*. The simplest way of representing a classifier is as a black box which produces a decision for every admissible pattern of data that is presented to it. Figure 1 illustrates the simple structure of a classification system. It accepts a pattern of data as input, and produces a decision as output.

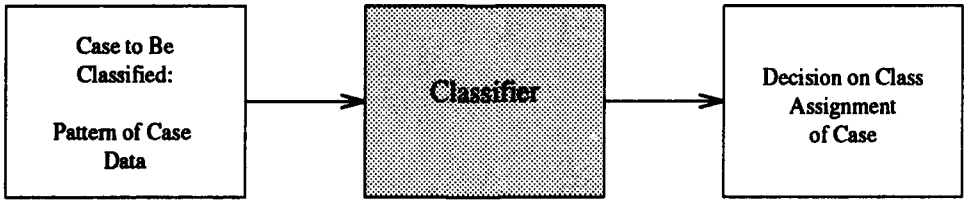



Figure 1. Classification System

The neural network has available to it a finite set of samples of solved cases. The data for each case consists of a pattern of features and the corresponding *correct* classification. *Features* also go by a host of other names, including *attributes* and *independent variables.*

The goal of a neural network is two-fold. The first goal is to extract decision rules from sample data. Samples are organized as cases, with each case consisting of measurements or

feature values, and a simple indicator of the correct class. The second goal is prediction on new cases, not discrimination between the existing sample cases. It is usually quite easy to find rules to discriminate, or separate, the sample cases from each other. It is much harder to develop decision criteria that hold up on new cases. Thus, the learning task becomes one of finding some solution that identifies essential patterns in the samples that are not overly specific to the sample data.

A final but significant point is made by Weiss and Kulikowski. The multilayer back-propagation neural network technique falls into the class of *nonparametric* methods. That is, it makes no assumptions about the functional form of the underlying population density distribution, such as that of a normal (bell-shaped) curve [20:12].

## *1.2* Research Objectives

The primary motivation for this research can be found in Figure 2. This flowchart, developed by Belue, shows the general process a researcher must go through to develop an "optimally" trained neural network [1:44]. The task is to start with a standard set of network parameters and analyze your way to an "optimally structured" neural network classifier.

The purpose of this research effort was to develop an interactive computer system which would allow the researcher to move from the box labeled, "Set Parameters to Standards" to the box labeled, "Optimal Structure Obtained", as quickly and painlessly as possible. To achieve this goal, three objectives were defined. The first objective was to revise and extend the capabilites of existing FORTRAN software for analyzing multilayer back-propagation neural networks. Where possible, more efficient algorithms were incorporated into the current software. The second objective was to interface the revised FORTRAN software with an interactive 2D and 3D graphics package. The graphics package chosen for this interface was MATLAB 4.1. The final objective was to validate the software using the classical "exclusive or" and "mesh" problems.

It was envisioned that the enhanced computer system would attain the following research objectives:

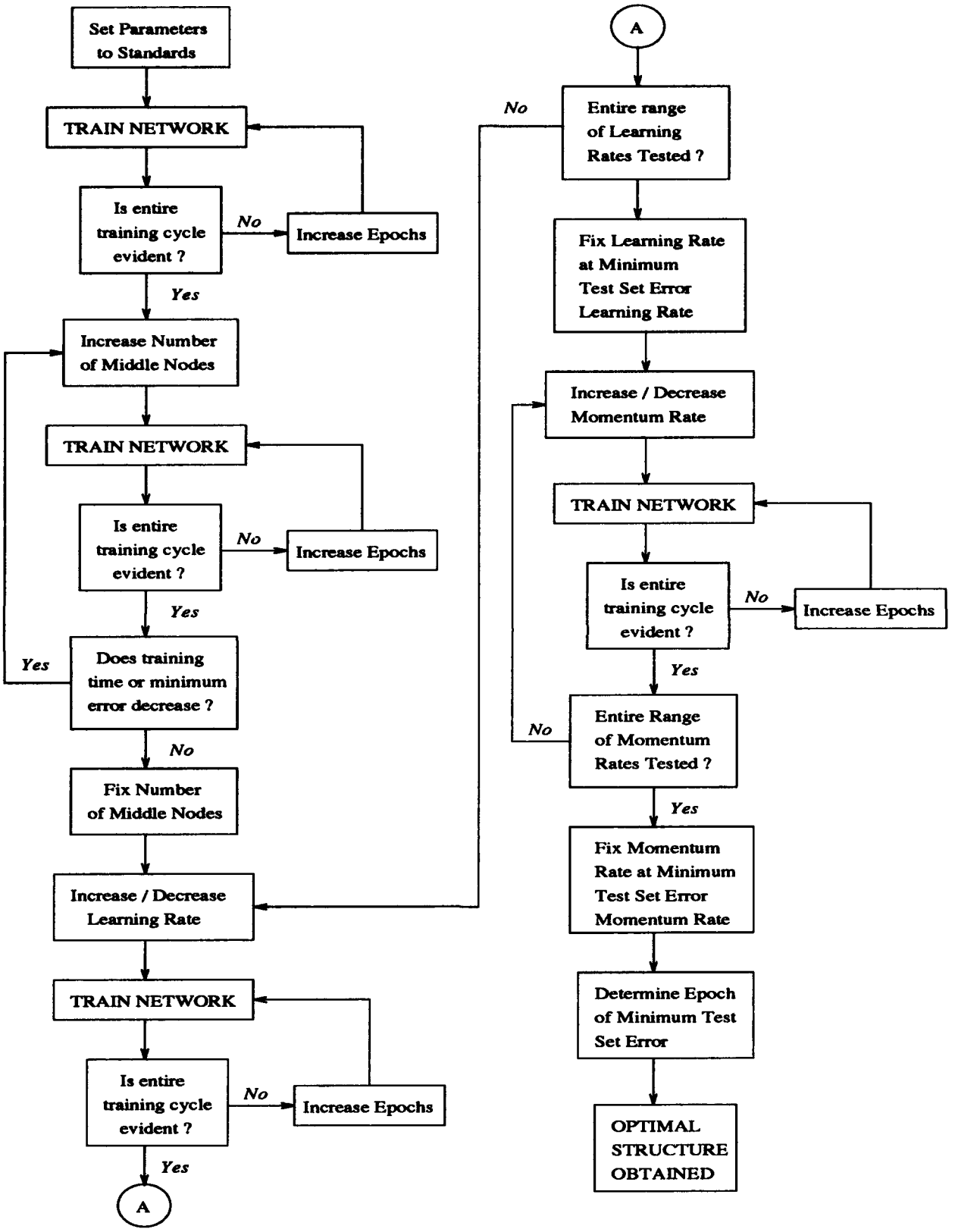



Figure 2. Building An Optimal Network

- Ability to monitor change in weights between network nodes *while* the network was training. The specific weights to monitor would be user defined and would be reported after each epoch.
- Ability to interrupt network training to see the "status" of the neural network.
- After analyzing the "status" of the network, decide whether to continue training the network or to quit training and report on final results.
- After network is trained, produce 2D graphs of historical output errors, classification errors, and user requested weights. Also produce 3D graphs of network activations and saliencies as specified by the user. All 3D graphs will have the ability to rotate to any view in an interactive mode and then be printed.
- Reduce the number of random numbers required to run the network.
- Increase computer storage efficiency in order to handle larger networks and data sets.
- Interface a sort algorithm into the current FORTRAN program to improve the efficiency of reshuffling the training and test sets after each epoch.

### *1.3* Scope

FORTRAN software developed by Belue for her thesis titled *An Investigation of Multilayer Perceptrons for Classification* provided an excellent program shell from which to begin the revisions and extensions [1]. In addition, subroutines developed by Steppe for her dissertation titled *Feature Selection in Feedforward Neural Networks* were incorporated into the main body of the software [18].

In order to revise and incorporate new procedures into the existing software, a thorough study of the underlying concepts of multilayer back-propagation and feature saliency was required. In addition, the logic and program flow of the existing software had to be analyzed and thoroughly understood. The next step was to develop an interface between FORTRAN and MATLAB that would allow the user to "jump" between the two languages as often and whenever the user desired, while still maintaining network training integrity. All throughout this process, a constant eye was kept on ways to improve computational efficiency and ways to shrink data storage requirements. Finally, graphs, output reports,

and raw data files had to be designed to display the volumes of data produced by the program.

# *II.* Literature Review

This chapter provides a review of the literature concerning multilayer back-propagation neural networks, the saliency metric, and sort algorithms. Specifically, it will define terms peculiar to the neural network field, describe the multilayer back-propagation algorithm, define the concept of saliency and its calculation, describe high-order inputs and their relation to correlation matrices, and finish with a discussion on the Shell-Mezgar sort algorithm. The intent is to give the reader a feel for why things were calculated the way they were in the FORTRAN code and why the parameter file is designed the way it is. See Appendix A for an example of the parameter file.

## *2.1* Terms Defined

As in many other fields of science, neural networks have their own brand of terminology. Several basic terms related to this field are defined below.

- **Back-propagation** A learning algorithm for updating weights in a multilayer, feedforward, mapping neural network that minimizes mean squared mapping error [4].
- **Classifier** The decision making system built by the neural network. In a sense, the final set of weights.
- **Epoch** A complete presentation of the data set being used to train the multilayer perceptron, also called a training cycle.
- **Exemplar** The input data to a neural network is a finite set of solved cases. Each case is known as an exemplar or input vector.
- **Feature** The individual measurements found in exemplars which contain information useful for distinguishing the various classes. In other fields, features are known as attributes or independent variables.
- **Feedforward** Characterized by multilayer neural networks whose connections exclusively feed inputs from lower to higher layers; in contrast to a feedback network, a feedforward network operates only until its inputs propagate to its output layer. An example of a feedforward neural network is the multilayer perceptron [4].

- **Hidden Units** Those processing elements in multilayer neural network architectures which are neither the input layer nor the output layer, but are located in between these and allow the network to undertake more complex problem solving [4].
- **Learning Algoithms** In neural networks, the equations which modify some of the weights of processing elements in response to input and output values [4].
- **Multilayer Perceptron** A multilayer feedforward network that is fully connected and which is typically trained by the back-propagation learning algorithm [4].
- **Neural Network** An information processing system which operates on inputs to extract information and produces outputs corresponding to the extracted information [4].
- **Single-layer Perceptron** A type of neural network algorithm used in pattern classification problems and trained with supervision. Connection weights and thresholds in a perceptron can be fixed or adapted using a number of different algorithms [4].
- **Supervised Training** A means of training adaptive neural networks which requires labeled training data and an external teacher. The teacher knows the correct response and provides an error signal when an error is made by the network [4].
- **Weight** A processing element (or neuron or unit) need not treat all inputs uniformly. Processing elements receive inputs by means of interconnects (also called 'connections' or 'links'); each of these connections has an associated weight which signifies its strength. The weights are combined to calculate the activations [4].

## *2.2* Error Rates

The overall objective of building a classifier is to learn from samples and to generalize to new, as yet unseen cases. Performance is most easily and directly measured in terms of the error rate, which is the ratio of the number of errors to the number of samples or cases.

$$error\ rate = \frac{number\ of\ errors}{number\ of\ cases} \tag{1}$$

| | | NETWORK CLASS | | | |
|---|---|---|---|---|---|
| | | Class 1 | Class 2 | Class 3 | Total |
| TRUE CLASS | Class 1 | 166 | 8 | 8 | 182 |
| | Class 2 | 3 | 180 | 3 | 186 |
| | Class 3 | 4 | 3 | 147 | 154 |
| | Total | 173 | 191 | 158 | 522 |

Figure 3. Confusion Matrix for Three Classes

Weiss states that the *true* error rate is statistically defined as the error rate of the classifier on an asymptotically large number of *new* cases that converge in the limit to the actual population distribution [20:17]. The requirement for estimating the true error rate is that the sample data are a *random* sample.

An error is simply a misclassification: the classifier is presented a case, and it classifies the case incorrectly. If all errors are of equal importance, a single error rate, as calculated in Equation 1, summarizes the overall performance of a classifier. However, for many applications, distinctions among different types of errors turn out to be important. For example, the error committed in tentatively diagnosing someone as healthy when one has a life-threatening illness (known as a false negative) is usually considered far more serious than the opposite type of error of diagnosing someone as ill when one is in fact healthy (known as a false positive).

If distinguishing among error types is important, then a *confusion matrix* can be used to lay out the different errors. Figure 3 is an example of such a matrix for three classes. The confusion matrix lists the correct classification against the predicted classification for each class. The number of correct predictions for each class falls along the diagonal of the matrix. All other numbers are the number of errors for a particular type of misclassification error.

The *apparent* error rate of a classifier is the error rate of the classifier on the sample cases that were used to design or build the classifier. Since we are trying to extrapolate performance from a finite sample of cases, the apparent error rate is the obvious starting point in estimating the performance of a classifier on new cases. For most types of classifiers, the apparent error rate is a poor estimator of future performance. Lippmann believes that in general, apparent error rates tend to be biased optimistically. The true error rate is almost invariably higher than the apparent error rate. This happens when the classifier has been overfitted (or overspecialized) to the particular characteristics of the sample data [10].

It is useless to design a classifier that does well on the design sample data, but does poorly on new cases. And unfortunately, using solely the apparent error to estimate future performance can often lead to disastrous results on new data. Many a learning system designer has been lulled into a false sense of security by the mirage of favorably low apparent error rates [20:25].

### *2.3* Training vs Test vs Validation Set

Instead of using all the cases to estimate the true error rate, the cases can be partitioned into three sets [7:115-117]. The first set is used to design the classifier, the second to test the classifier, and the third to validate the classifier.

- **The Training Set:** This set is used to design or train the weights in the multilayer perceptron. Foley's Rule [5] provides some guidelines as to the minimum number of training vectors required for accurate classification as a function of the number of input features. Foley showed empirically that the *number of training samples per class* should be greater than three times the number of features.
- **The Test Set:** This set is used to test the accuracy of training *while* training is ongoing. After each epoch, the test set acts as a barometer for determining when the accuracy of the perceptron is at an acceptable level.

- **The Validation Set:** After the multilayer perceptron is considered optimally trained, the validation set is presented to the classifier. It verifies the performance of the classifier since its exemplars are never seen by the classifier during its development.

## *2.4* Multilayer Perceptrons

This section will use a building block approach to define the algorithm used in multilayer back-propagation. It starts out with the idea of a linear discriminant, then graduates to the concepts of single output perceptron, learning rate, multilayer perceptron, back-propagation, and finally, the concept of momentum.

*2.4.1 Linear Discriminants.* Linear discriminants are the most common form of classifier, and are quite simple in structure. The name linear discriminant simply indicates that a linear combination of the evidence will be used to separate or discriminate among the classes and to select the class assignment for a new case. For a problem involving $n$ features, this means geometrically that the separating surface between the samples will be an $(n-1)$ dimensional hyperplane [10]. In most situations, classes can overlap and therefore cannot be completely separated by a plane (or line in two dimensions). The classic example of this is the logical "exclusive or (XOR)" problem.

Equation 2 gives the general form for any linear discriminant,

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - w_0 \tag{2}$$

where $(x_1, x_2, \ldots, x_n)$ are the usual list or vector of $n$ features, and $w_i$ are constants that must be estimated. A linear discriminant simply implements a weighted sum of the values of the observations. Intuitively, we can think of the linear discriminant as a scoring function that adds to or subtracts from each observation, weighing some observations more than others and yielding a final total score. The class selected, $C_i$, is the one with the *highest* score.

*2.4.2 Single-Output Perceptron.* The simplest neural net device is the single-output perceptron. More complex neural networks can be described as combinations of

many single-output perceptrons in a network. The simplest perceptron is a device that decides whether an input pattern belongs to one of two classes. The perceptron is strictly the equivalent of a linear discriminant. Recall from Equation 2 that a linear discriminant is simply a weighted scoring function. The weights, $w_i$, can assume real values, both positive and negative, so we can rewrite Equation 2 as:

$$\sum_i w_i x_i + \theta \tag{3}$$

where the evidence (features or independent variables) are described as *inputs*, $x_i$, and $\theta$ is the constant or bias. Geometrically speaking, in two dimensions, the constant $\theta$ indicates (the intercept) where the line crosses the y-axis [20:82].

Figure 4 illustrates the general form of a single-output perceptron, also known as an adaline. Data feeds into the perceptron's input nodes numbered $x_1$ to $x_n$ and the $w_i$ on each branch of the perceptron weights the inputs. The bias, or threshold is an additional node whose input is one. The procedure sums across the weighted inputs, adds a bias term, and transforms the sum so that the activation $z$ of the perceptron is:

$$z = f[(\sum_{i=1}^{N} w_i x_i) + \theta] \tag{4}$$

The single-output perceptron produces an output indicating membership in class 1 or class 0 as indicated by Equation 5. The constant $\theta$ is referred to as the *threshold* or *bias*, because Equation 5 can be read to indicate that the sum of the weighted products must exceed -$\theta$.

$$f[\cdot] = \begin{cases} 1 & \text{if } \sum_i w_i x_i + \theta > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The weights of the perceptron are constants, and the variables are the inputs. As with other linear discriminants, the main task is to learn the weights. The perceptron is trained on sample cases found in the training set, and uses a sequential learning procedure to determine the weights. Sample cases are presented sequentially, and errors are corrected

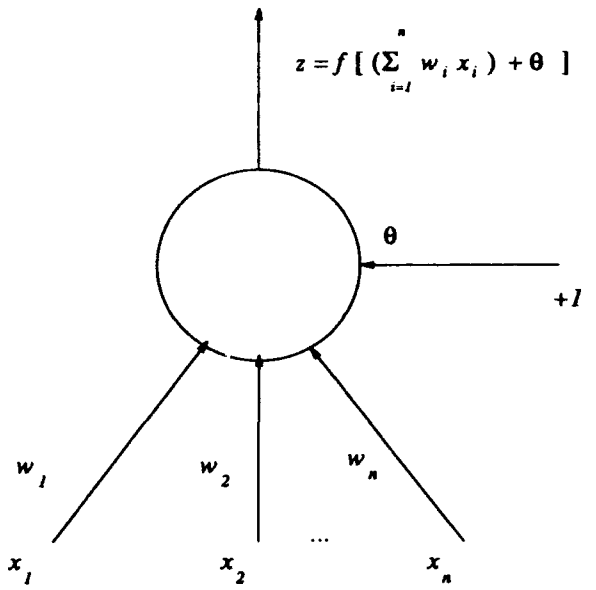



Figure 4. Single-Output Perceptron

by adjusting the weights after each erroneous output. If the perceptron output matches the *desired* or *true* output, the weights are not adjusted.

Equation 6 describes the general form of an iterative procedure that adjusts the weights. A sample is presented to the perceptron. Each new weight is computed by adding a correction to the old weight. We describe the current weight as $w_i(t)$, the weight at time $t$. The new weight is $w_i(t+1)$, which will be the current weight at time $t+1$. The new weight $w_i(t+1)$ is computed by adding an adjustment factor, $\Delta w_i(t)$ to the current weight $w_i(t)$. The threshold $\theta(t)$ is also revised.

$$\begin{aligned} w_i(t+1) &= w_i(t) + \Delta w_i(t) \\ \theta(t+1) &= \theta(t) + \Delta\theta(t) \end{aligned} \tag{6}$$

The task of the training procedure is to find $\Delta w_i$, the adjustment to any weight $w_i$. The training procedure for perceptrons is given in Equation 7, where $d$ is the desired or true answer, $f[\cdot]$ is the perceptron output, and $x_i$ is the perceptron input.

$$\begin{aligned} \Delta w_i(t) &= (d - f[\cdot])x_i \\ \Delta\theta(t) &= (d - f[\cdot]) \end{aligned} \tag{7}$$

When a case is presented to the perceptron and the output is correct, no change is made to any of the weights. If the output is incorrect, each weight is adjusted by adding or subtracting the corresponding value in the input pattern. The hope is that each adjustment will move the weights closer to the true weights.

*2.4.3 The Learning Rate.* While the perceptron convergence theorem, proved by Rosenblatt, states that for linearly separable data the perceptron will eventually converge, the speed with which the training will be completed is not known [15]. One may have to wait a very long time for an answer. The perceptron will not necessarily get closer and closer to the answer after each epoch. However, there are a number of modifications to the data and the training procedures that may speed up learning and convergence. Two of the most important are:

1. Normalize the data: Performance is often improved by normalizing the data between 0 and 1, or by using Gaussian normalization.
2. Make the learning rate adjustable by introducing a *learning rate* parameter, $\eta$ in the perceptron weight updating procedure. Instead of using $\Delta w_i$ as the correction factor, we use $\eta \Delta w_i$, where $\eta$ is usually chosen between 0 and 1.

Weiss states that, while these measures can improve the rate of convergence, there is no way of knowing in advance the value of the learning rate that will speed up convergence the most for a specific data set. He adds that, if the learning rate is too large, training may not converge and just oscillate between wrong answers. Or, it may converge to a local minimum [20:86-90]. Equation 7, when modified to incorporate a learning rate term, $\eta$, becomes Equation 8. As we shall see later in this chapter, Equation 8 will be used in the training procedures for more complicated neural nets.

$$\begin{aligned} \Delta w_i(t) &= \eta(d - f[\cdot])x_i \\ \Delta\theta(t) &= \eta(d - f[\cdot]) \end{aligned} \tag{8}$$

The perceptron cannot train correctly when the classes are not linearly separable. In addition, very few real-world applications are truly linearly separable. Hence, relying on the predictive potential of linear solutions has troubled many researchers because it is easy to come up with counter-examples of simple data sets that cannot be separated by lines.

The most commonly cited example comes from applying the "exclusive or" XOR, logical operator on two binary features. A very similar learning system that fits a line, but is less dependent on linear separability for good results, is discussed in the next section.

*2.4.4 Least Mean Square Learning System.* The least mean square learning system, known simply as LMS, is another system that finds linear solutions. The only functional difference is in the way the output, $f[\cdot]$, is computed. The LMS system uses the actual net output without any further mapping into 0 or 1. For a given input, the output of the LMS device is simply the product of the inputs and weights summed with the threshold.

$$f[\cdot] = \sum_i w_i x_i + \theta \tag{9}$$

For the LMS learning system, the correct answers are still expressed as 0 or 1, but the output is now a real number.

The goal of the LMS training procedure is to minimize the average squared distance from the true answer to the net output. This is equivalent to finding a set of weights and a threshold that minimize:

$$\sum_j (d_j - f_j[\cdot])^2 \tag{10}$$

Where $j$ ranges over the number of samples in the training set.

It should be noted that the LMS training procedure does not directly reduce the classification error rate. Rather, it reduces the distance between the output and the true answer. Weiss states that reducing this distance is *usually* strongly related to reducing classification error rates. However, it is quite possible that the classification error rate can be relatively high even with a relatively small error distance. The error distances for the correct answers may be quite small, while the erroneous answers may barely be on the other side of the boundary [20:89].

*2.4.5 Multilayer Perceptrons.* The single-output perceptron and LMS learning systems of the previous sections can be naturally extended to more complicated neural networks. For example, the outputs of several perceptrons could be used as input to

other perceptrons. These devices could then be chained together in many different ways. Figure 5 illustrates the general structure of a multilayer network structure.

The perceptron can be described as a *single-layer* neural network, where a layer represents a set of output devices. The network of Figure 5 is a two-layer network. Input nodes are *not* counted as a layer. The inputs are connected to the first layer of outputs. These outputs serve as inputs to the second layer of outputs. This is a *fully connected* network; since every node serves as input to nodes in the next layer above.

The multilayer neural network has multiple outputs and multiple layers of outputs. The final layer of outputs contains the decision results, called *output units.* The output units of the intermediate layers are referred to as *hidden units* because they are not units that are naturally defined by the application. Like the perceptron, each output unit has a threshold or bias associated with it.

The multilayer network of Figure 5 can be extended to unlimited numbers of additional layers. Potentially, this makes the multilayer neural network a very powerful classifier. Cybenko has shown that, a two-layer network with one layer of hidden units, can implement most decision surfaces, and can closely approximate any decision surface [3]. In addition, this two-layer structure allows for the formation of nonlinear decision regions, including disjoint regions. Therefore, this two-layer network structure is used in the FORTRAN program developed in this research.

*2.4.6 Back-Propagation Procedure.* The multilayer neural net can be trained by using the back-propagation training procedure. Before we discuss this procedure, a slight variation in the computation of the output $f[\cdot]$, must be considered.

In the previously described perceptron, once the weighted sum was computed, the activation of the output unit was determined by threshold logic. The activation $f[\cdot]$, was 0 or 1, depending on whether a threshold was exceeded. This threshold logic activation function creates a nonlinearity, a desirable characteristic. However, it does not provide the other desirable characteristic which we seek; that of a continuously differentiable function [11]. In order to obtain these two characteristics, an alternative activation function, $f[\cdot]$, is used; it is known as the *sigmoidal* or *logistic* function. For any real-valued numbers,

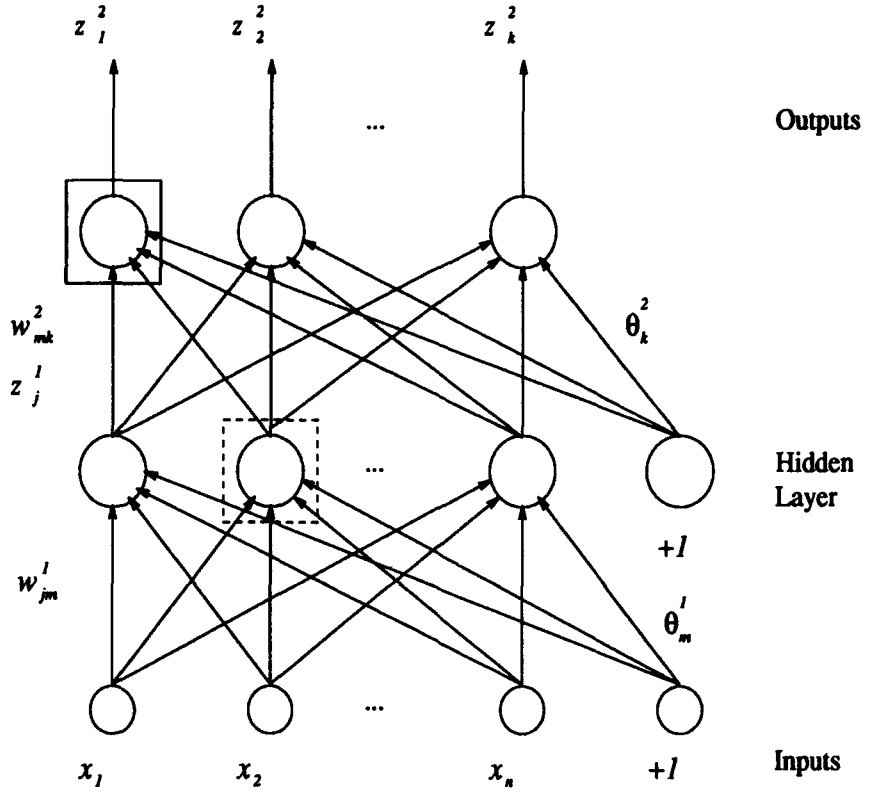



Figure 5. Multilayer Network Structure

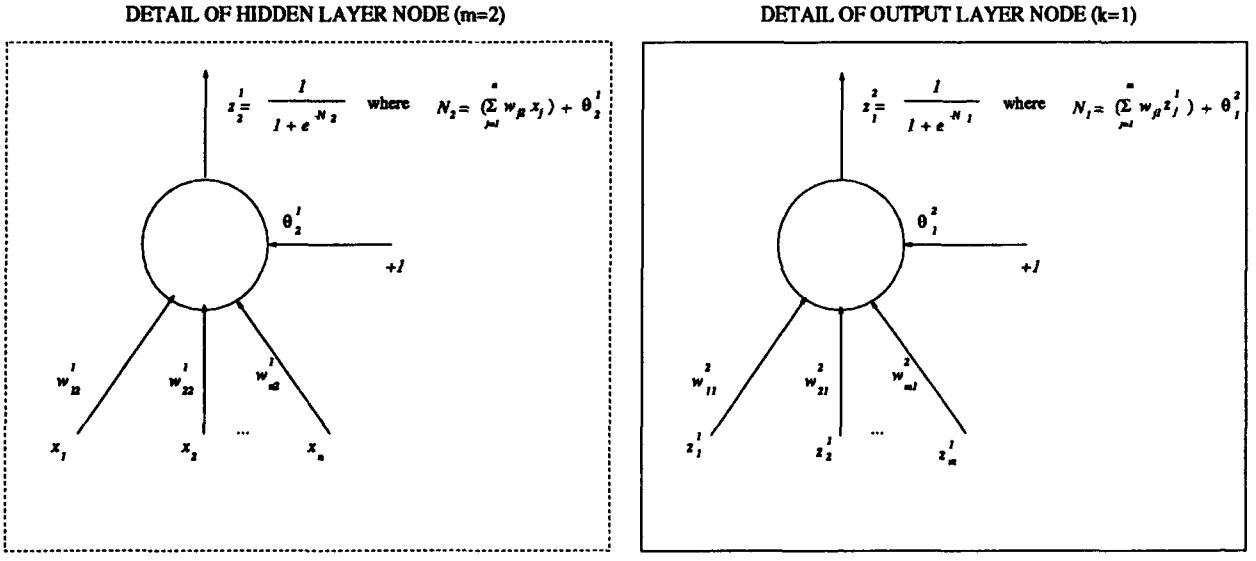



Figure 6. Detail of Hidden Layer and Output Layer Nodes

the output of the sigmoidal function is between 0 and 1. This function and its graph are shown in Equation 11 and Figure 7.

$$f(a) = \frac{1}{1 + e^{-a}} \tag{11}$$

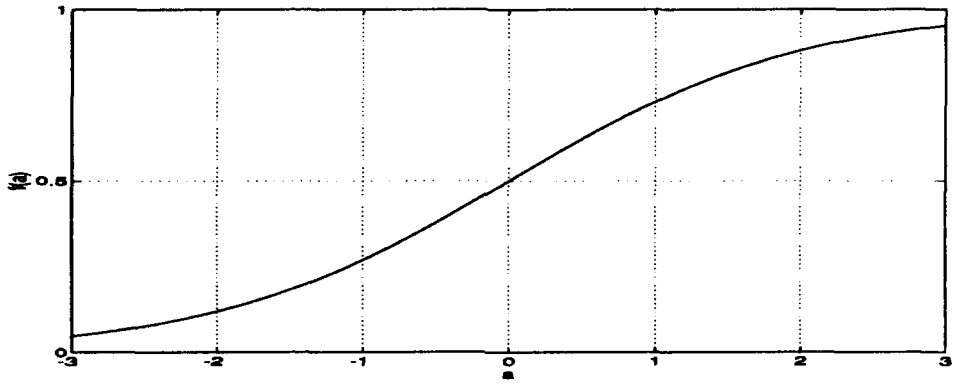



Figure 7. Nonlinear Sigmoid Function

Equations 12 and 13 show how each training vector is propagated up through the network to produce network outputs $z_m^1$ and $z_k^2$. Equations 14 and 15 show how the network outputs are then back-propagated down through the network, producing error derivatives $\delta_k^2$ and $\delta_j^1$. These error derivatives are then used to adjust the weights. After the weights have been adjusted, the next training vector is presented and the process is repeated. These equations are the heart of the back-propagation training procedure.

In Equation 12 the output or activation, $z_m^1$, of a hidden unit, $m$, is computed by applying a sigmoidal function to the net input, $N_m$, of unit $m$. The net input of hidden unit $m$ is the sum of the bias of hidden unit $m$, $\theta_m^1$, and the weighted sum of the input features, $x_j$, connected to hidden unit $m$. The weight, $w_{jm}^1$ connects input node $j$ to hidden node $m$. Refer to Figure 6 (Detail of Hidden Layer Node).

$$\begin{aligned} N_m &= \sum_{j=1}^{n} w_{jm}^1 x_j + \theta_m^1 \\ z_m^1 &= \frac{1}{1+e^{-N_m}} \end{aligned} \tag{12}$$

In Equation 13 the output or activation $z_k^2$, of an output unit, $k$, is computed by applying a sigmoidal function to the net input, $N_k$, of output unit $k$. The net input of output unit $k$ is the sum of the bias of unit $k$, $\theta_k^2$, and the weighted sum of the hidden layer outputs, $z_j^1$, connected to output unit $k$. The weight, $w_{jk}^2$ connects hidden node $j$ to output node $k$.

$$\begin{aligned} N_k &= \sum_{j=1}^{m} w_{jk}^2 z_j^1 + \theta_k^2 \\ z_k^2 &= \frac{1}{1+e^{-N_k}} \end{aligned} \tag{13}$$

Equation 14 begins the error back-propagation by calculating the error derivative, $\delta_k^2$, associated with output unit $k$:

$$\delta_k^2 = z_k^2(1 - z_k^2)(d_k - z_k^2) \tag{14}$$

where $z_k^2$ is the output of output unit $k$, and $d_k$ is the desired or true output of output unit $k$. In Equation 15 the error derivative of hidden unit $j$, $\delta_j^1$ is calculated by:

$$\delta_j^1 = z_j^1(1 - z_j^1)(\sum_k \delta_k^2 w_{jk}^2) \tag{15}$$

where $z_j^1$ is the output of hidden unit $j$, $\delta_k^2$ is the error derivative calculated in Equation 14, and $w_{jk}^2$ is the weight connecting hidden unit $j$ to output unit $k$.

We now have all the information required to calculate the new weights at time $(t+1)$ from the current weights at time $t$. Using Equations 6 and 8:

$$\begin{aligned} w_{jm}^1(t+1) &= w_{jm}^1(t) + \eta\delta_j^1 x_j \\ \theta_m^1(t+1) &= \theta_m^1(t) + \eta\delta_j^1 \end{aligned} \tag{16}$$

$$\begin{aligned} w_{jk}^2(t+1) &= w_{jk}^2(t) + \eta\delta_k^2 z_j^1 \\ \theta_k^2(t+1) &= \theta_k^2(t) + \eta\delta_k^2 \end{aligned} \tag{17}$$

where $\eta$ is the learning rate, $x_j$ is the input from input node $j$, and $z_j^1$ is the output from hidden node $j$. With the weights adjusted, we are ready to present the next training vector which starts the propagation process all over again.

*2.4.7 Momentum.* Empirical evidence supports the notion that the use of a term called *momentum* in the back-propagation revision procedure can be helpful in speeding convergence and avoiding local minima. Momentum towards convergence is maintained by making nonradical revisions to the weight change direction. Weights are revised by combining the indicated new weight revision, Equations 16 and 17 with part of the previous weight revision. Mathematically, the weights are revised as indicated in Equations 18 and 19, where $\alpha$ is a momentum term that indicates the fraction of the previous weight adjustment that is used for the current revision.

$$\begin{aligned} w^1_{jm}(t+1) &= w^1_{jm}(t) + \eta\delta^1_j x_j + \alpha[w^1_{jm}(t) - w^1_{jm}(t-1)] \\ \theta^1_m(t+1) &= \theta^1_m(t) + \eta\delta^1_j + \alpha[\theta^1_m(t) - \theta^1_m(t-1)] \end{aligned} \tag{18}$$

$$\begin{aligned} w^2_{jk}(t+1) &= w^2_{jk}(t) + \eta\delta^2_k z^1_j + \alpha[w^2_{jk}(t) - w^2_{jk}(t-1)] \\ \theta^2_k(t+1) &= \theta^2_k(t) + \eta\delta^2_k + \alpha[\theta^2_k(t) - \theta^2_k(t-1)] \end{aligned} \tag{19}$$

According to Weiss, the hope is that the momentum term will allow a larger learning rate and that this will speed convergence and avoid local minima. On the other hand, a learning rate of 1 with no momentum will be much faster when no problem with local minima or non-convergence is encountered [20:101].

## *2.5* The Saliency Metric

*2.5.1 Ruck's Saliency.* Any classifier is at the mercy of the sample data and the quality of the features. Even when no errors are made in recording the data, the predictive capabilities of some features can be quite weak. It is quite possible that revising or adding features can lead to greatly improved performance in the classifier. With this in mind, we turn to Ruck's saliency metric. Ruck describes a saliency metric which measures feature $i$'s effect on a neural network's output [16:32-38]. This metric attempts to capture the total of the partial derivatives of the network's outputs with respect to the entire M-dimensional feature space $R^M$. Ruck's saliency metric for feature $i$ is built from the exact partial derivatives of network outputs, $z^2_k$, with respect to feature inputs $x_i$ using the *trained* network.

The derivative of the outputs with respect to the input can be written as a function of only the weights and activations as follows:

$$\frac{\partial z_k^2}{\partial x_i} = z_k^2(1 - z_k^2) \sum_m w_{mk}^2 z_m^1 (1 - z_m^1) w_{jm}^1 \tag{20}$$

where, $z_k^2$ is the output of node $k$ in the output layer, $z_m^1$ is the output of node $m$ in the hidden layer, $w_{jm}^1$ is the weight connecting node $j$ in the input layer to node $m$ in the hidden layer, and $w_{mk}^2$ is the weight connecting node $m$ in the hidden layer to node $k$ in the output layer. The derivative equation above is applicable for perceptrons with a single hidden layer. As the number of hidden layers increases, the calculation of this saliency metric becomes more complex.

Ideally, the input space would be systematically sampled over its entire range of values. If $R$ points were used for each input, the total number of derivatives would be on the order of $R^M$, where $M$ is the number of feature inputs. For other than very small problems, the number of computations in $R^M$ is intractable. In fact, this is an NP-complete problem [16:34-37]. Ruck proposes a sampling method which is computationally tractable. For every training vector, each feature input is sampled over its range while the other feature inputs are held constant at their actual training vector values. For $P$ training vectors, the number of derivative evaluations is $PMRK$, where $M$ is the number of features, $R$ is the number of samples for each feature input of each training vector, and $K$ is the number of output classes. For the saliency computation of each feature, the set of $R$ "pseudo" data points remains the same. If we define $\vec{\mathbf{d}}_m$ as the vector of $R$ uniformly spaced pseudo points covering the range of the $m$th input feature, then the $r$th component, $d_r$, of $\vec{\mathbf{d}}_m$ can be defined as:

$$d_r = \min x_m + (r-1)\frac{\max x_m - \min x_m}{R-1} \qquad r = 1, 2, \ldots, R \tag{21}$$

where $\min x_m$ is the minimum value of $x_m$ taken over all $P$ training vectors, and $\max x_m$ is the maximum value of $x_m$ taken over all $P$ training vectors.

The Ruck saliency metric, $\Lambda_i$, for feature $i$ when a sigmoid nonlinearity is used is defined as:

$$\Lambda_i \equiv \sum_{p=1}^{P} \sum_{m=1}^{M} \sum_{r=1}^{R} \sum_{k=1}^{K} \left| \frac{\partial z_k^2}{\partial x_i}(\vec{\mathbf{x}}_p^{m(r)}, \vec{w}) \right| \tag{22}$$

where $P$ is the number of training vectors $\vec{\mathbf{x}}_p$; $M$ is the number of features; $R$ is the number of uniformly spaced points covering the range of each input feature found in the training set; $K$ is the number of output classes; the vector $\vec{\mathbf{x}}_p^{m(r)}$ is the vector $\vec{\mathbf{x}}_p$ with its $m$th component replaced by, $d_r$, the $r$th component of $\vec{\mathbf{d}}_m$; and $(\vec{\mathbf{x}}_p^{m(r)}, \vec{w})$ indicates that the derivative is evaluated with the feature vector $\vec{\mathbf{x}}_p^{m(r)}$ and the final estimates of the trained network weight parameters $\vec{\mathbf{w}}$. Also, the absolute value of the derivative is used, so positive and negative derivative changes do not cancel out.

Equation 22 has been modified from Ruck's original presentation to reflect that for each vector there are $PMRK$ derivative evaluations as Ruck intended, rather than just $PRK$ derivative evaluations as denoted in Ruck's notation [16:37].

*2.5.2 Tarr's Saliency.* A simpler method of determining the relative significance of the input features once the network has been trained has been suggested by Tarr. He states the following:

> When a weight is updated, the network moves the weight a small amount based on the error. Given that a particular feature is relevant to the problem solution, the weight would be moved in a constant direction until a solution with no error is reached. If the error term is consistent, the direction of the movement of the weight vector, which forms a hyper-plane decision boundary, will also be consistent. ...If the error term is not consistent, which can be the case on a single feature out of the input vector, the movement of the weight attached to the node will also be inconsistent. In a similar fashion, if the feature did not contribute to a solution, the weight updates would be random. In other words, useful features would cause the weights to grow, while weights attached to non-salient features would simply fluctuate around zero. [19:44]

Therefore, the following alternate saliency metric is proposed:

$$\tau_i = \sum_m (w_{im}^1)^2 \tag{23}$$

Which is simply the sum of the squared weights between input node $i$ and hidden node $m$.

## 2.6 High-Order Inputs and Correlation

Networks that have second, third or greater order terms for inputs are referred to as high-order networks. If two inputs $x_1$ and $x_2$ represent two separate pieces of information, then $x_1^2$ or $x_1 x_2$ may represent pieces of information more important to discrimination than either one separately. Giles suggests that an examination of correlation matrices which relate the high-order input terms with the outputs of a trained network would be useful. The entries in the correlation matrix that have the largest absolute value correspond to the high-order input terms that should be considered for inclusion in the network [6:4977-4978]. We will use the following equations to calculate the second-order correlation matrix of the product of two inputs with the output. These equations and notation were developed by Belue and Bauer [2:9].

First, define the sample covariance $C(i,(j,k))$, between the $i$th output node of the network and the second-order product of the $j$th and $k$th input nodes as:

$$C(i,(j,k)) = \sum_{s=1}^{N} [z_i^2(s) - \overline{z}(i)][y^s(j,k) - \overline{y}(j,k)] \tag{24}$$

where

$$\begin{aligned} y^s(j,k) &= x^s(j)x^s(k) \\ \overline{y}(j,k) &= \frac{\sum_{s=1}^{N} x^s(j)x^s(k)}{N} \\ \overline{z}(i) &= \frac{\sum_{s=1}^{N} z_i^2(s)}{N} \end{aligned} \tag{25}$$

and $x^s(j)$ is the value of the $j$th feature in exemplar $s$, $x^s(k)$ is the value of the $k$th feature in exemplar $s$, $z_i^2(s)$ is the output of output node $i$ for exemplar $s$ and $N$ is the number of exemplars in the *training* set.

Next, define an element of the second-order correlation matrix $R(i,(j,k))$, as the correlation between the $i$th output node of the network and the second-order product of the $j$th and $k$th input nodes where:

$$R(i,(j,k)) = \frac{C(i,(j,k))}{\sqrt{\sum_{s=1}^{N}[z_i^2(s)-\overline{z}(i)]^2}\sqrt{\sum_{s=1}^{N}[z_i^s-\overline{y}(j,k)]^2}} \tag{26}$$

The entries in the correlation matrix that are greatest in absolute value correspond to second-order terms that are highly correlated with output $i$.

## 2.7 The Shell-Mezgar Sort

Empirical evidence has shown that a *reshuffling* of training vectors at the beginning of each epoch will yield superior results [20:99-100]. With some networks requiring thousands of epochs before they are trained, it is wise to incorporate an efficient reshuffling algorithm into the software. By generating a random number for each training vector and then sorting on these random numbers, we can reshuffle the training vectors. Therefore, the efficient reshuffling algorithm that we seek transforms into an efficient sort algorithm.

According to Press, et.al., for the basic task of sorting $N$ elements, the best sort algorithms require on the order of several times $Nlog_2N$ operations. The algorithm inventor tries to reduce the constant in front of this estimate to as small a value as possible [13:226-229]. Knuth has shown that for "randomly" ordered data, the operations count on the Shell-Mezgar sort goes approximately as $N^{1.27}$, for $N < 60000$ [9]. Since our sort index consists of random numbers, and the number of training vectors will be $\ll 60000$, the Shell-Mezgar sort is an ideal candidate. In chapter 4 there is a table which compares $Nlog_2N$ and $N^{1.27}$ for various values of $N$. A FORTRAN version of this sorting routine may be found in Press, et.al. [13:226-229].

# *III.* Methodology

The overall objective of this thesis was to develop software to assist the researcher in building an appropriate back-propagation neural network. The software was to enable the researcher to quickly define a neural net structure, run the neural network, interrupt training at any point to analyze the status of the current network, re-start training at the interrupted point if desired, and analyze the final network. What follows is a synopsis of each of the subroutines found in the FORTRAN code as well as the MATLAB interface. By following the program flow we can define the methodology used. Figure 8 shows the overall program flow.

## *3.1* Defining Neural Network Parameters

The subroutine NETIN reads all network parameters from a user designated parameter file. These parameters name the raw data file, determine a stopping criteria, layout the neural network structure, and define the desired 2D and 3D graphs. An explanation of the required parameters follows. The FORTRAN program will prompt the user for the name of this *parameter file*. Appendix A gives an example of such a parameter file and defines the allowable values.

- **Data File Name** This parameter defines the name of the data set where all the raw data resides. Training, test, and validation sets will be pulled from this data set in a random order.
- **Stopping Criteria** Determines when training of the neural network will end. The user can specify number of epochs or average absolute error as the stopping criteria. If number of epochs is specified, the program will stop and present the network results as they existed at the end of the specified epoch. If average absolute error is specified, the program will stop when the average absolute error (for the training or test set) at the end of an epoch is less than or equal to the specified error rate. It should be noted that this parameter is over-ridden if the user terminates training of the network interactively. See sub-section 3.4.3.

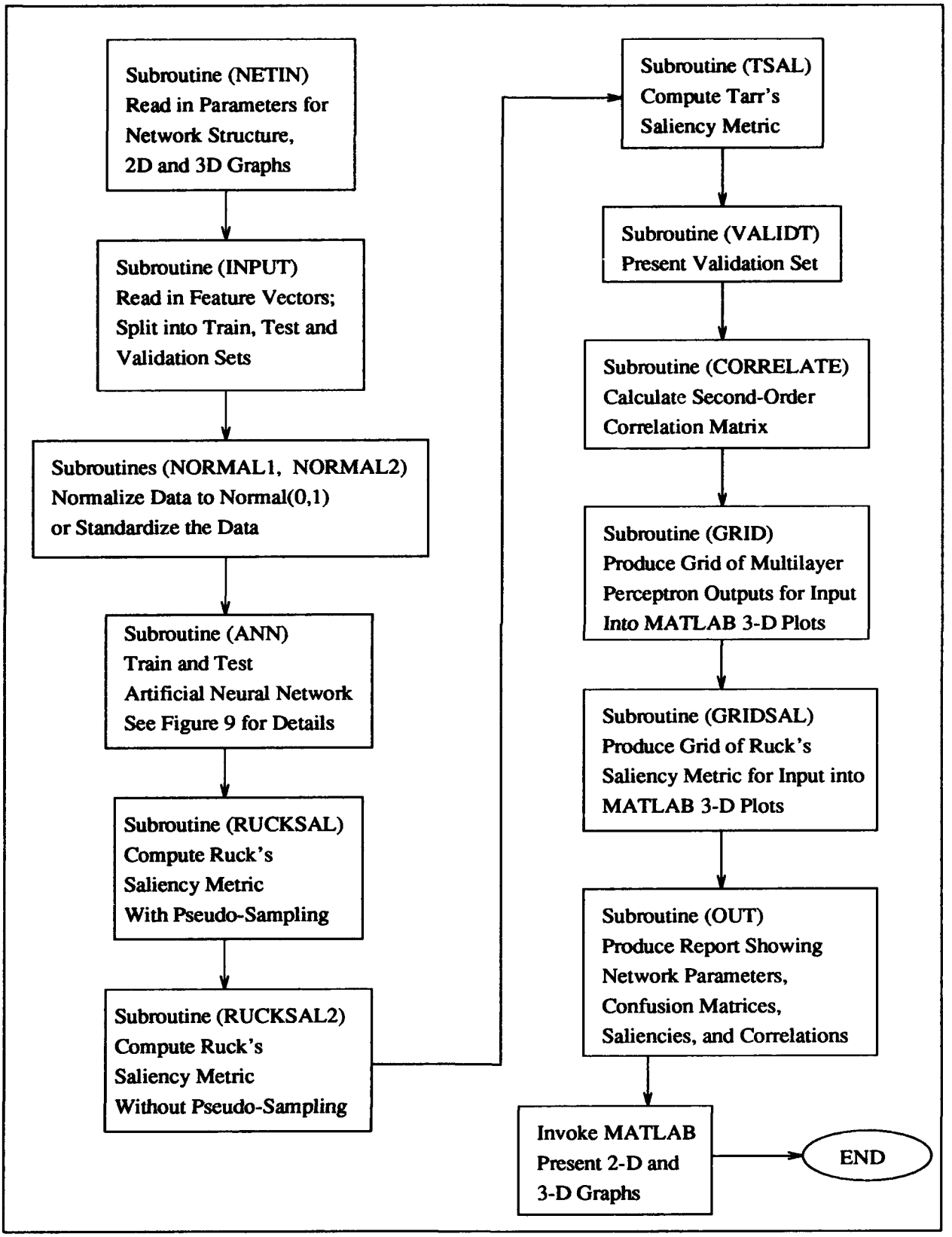



Figure 8. Overview of FORTRAN and MATLAB Program Flow

- **Number of Training Vectors** (N1) Defines the number of vectors to be put into the training set.
- **Number of Test Vectors** (N2) Defines the number of vectors to be put into the test set.
- **Number of Validation Vectors** (N3) Defines the number of vectors to be put into the validation set.
- **Number of Input Nodes** This parameter is equal to the number of features or independent variables found in each vector.
- **Number of Middle Nodes** The user must determine the number of middle nodes required in the network structure. This may have to be determined through experimentation.
- **Number of Output Nodes** This parameter is equal to the number of output classes found in each vector.
- **Type of Learning Rate** (LTYPE) Defines the type of learning rate that will be used in the back-propagation algorithm. There are six types of learning rates: constant, linear, log, log-linear, loq-square root, and log-square root linear. See Appendix A for a detailed description of the learning rates supported by this software.
- **Learning Rate** Gives the constant learning rate *value*, if (LTYPE=1) in the parameter above. If (LTYPE $\neq$ 1), this parameter is ignored.
- **Momentum Rate** Defines the constant momentum rate that will be used in the back-propagation algorithm. If no momentum rate is desired, set momentum rate equal to zero.
- **Range of Weight Initialization** Before beginning the back-propagation algorithm, the program must initialize all weights between the nodes to random numbers. The random number generator used in the FORTRAN program generates $Uniform(0, 1)$ random deviates $U$. These random deviates are then run through a transformation function of the form:

$$a + (b - a)U \tag{27}$$

where $a$ is the desired lower limit of the range initialization, and $b$ is the desired upper limit. In effect, we are generating random deviates with a $Uniform(a, b)$ distribution.

- **Random Number Seed** Defines the seed for the random number generator used in initializing weights and determining the sort order of the training set.
- **Type of Normalization of Data** Determines if data is to be normalized. User may normalize data between $(0, 1)$, standardize the data, or use the data in its original form.
- **Number of Divisions for Pseudo-sampling** This parameter is used when calculating Ruck's saliency. This is the value of $R$ as described in Equation 22.
- **Graphics Parameters** Define 2D and 3D graphs which will be created by MATLAB. See Appendix A for details.

### *3.2* Defining Train, Test and Validation Sets

The subroutine INPUT reads the raw data from a single file. Each input line is an exemplar or case. This case vector consists of feature values and the corresponding correct classification. For example, a problem with two input features $x_1$ and $x_2$, and four output classes $c_1, c_2, c_3$ and $c_4$, would have the following input vector: $(x_1, x_2, c_1, c_2, c_3, c_4)$. If a particular case had a correct classification of class 3 the vector would be: $(x_1, x_2, 0, 0, 1, 0)$

A random number is generated and attached to each vector. We then perform a Shell-Mezgar sort on the vectors using these random numbers as the sort index. We then place the first N1 vectors in the training set, the next N2 vectors in the test set, and finally the last N3 vectors in the validation set, where N1, N2 and N3 are defined in the parameter file described above. This procedure randomly creates the training, test and validation sets. To create different sets, simply change the random number seed in the parameter file. These three sets of vectors will remain fixed throughout the entire computer run. However, the vectors within the training set are reshuffled after each epoch.

### 3.3 Normalization of Data

The subroutines NORMAL1 and NORMAL2 normalize the data now residing in the train, test and validation sets. The user has the option, as specified in the parameter file, of not normalizing the data at all, normalizing between 0 and 1, or performing Gaussian normalization. Subroutine NORMAL1 normalizes each of the feature vectors to values between 0 and 1, based on the range of the *training set*. The test set and validation set are also normalized based on the range of the *training set*. By basing normalization on the training set only, we keep everything on the same scale.

Subroutine NORMAL2 statistically normalizes each of the feature vectors to values based on the mean and standard error of the *training set*. Let $x_i$ be the vector value of the $i$th feature, $\bar{x}_i$ be the mean of feature $i$ over the *training set* and $s_i$ be the standard deviation. Then for every vector in the training, test and validation set, the normalized $i$th feature is:

$$x_i' = \frac{x_i - \bar{x}_i}{s_i} \tag{28}$$

This procedure is known as *standardization*. Ruck refers to it as Gaussian Normalization [16:15-16].

### 3.4 Artificial Neural Network (ANN)

Subroutine ANN is the heart of this FORTRAN based neural network system. It is where the back-propagation algorithm is performed. Figure 9 gives a detailed flow of the subroutine ANN. This subroutine assumes the following network structure and characteristics:

- Fully connected feed-forward perceptrons
- Back-propagation training as defined in Equations 18 and 19.
- Single hidden layer
- Sigmoid non-linear transformation

The artificial neural net subroutine begins by initializing the weights connecting both the input layer and hidden layer and the hidden layer and output layer. Normally, these

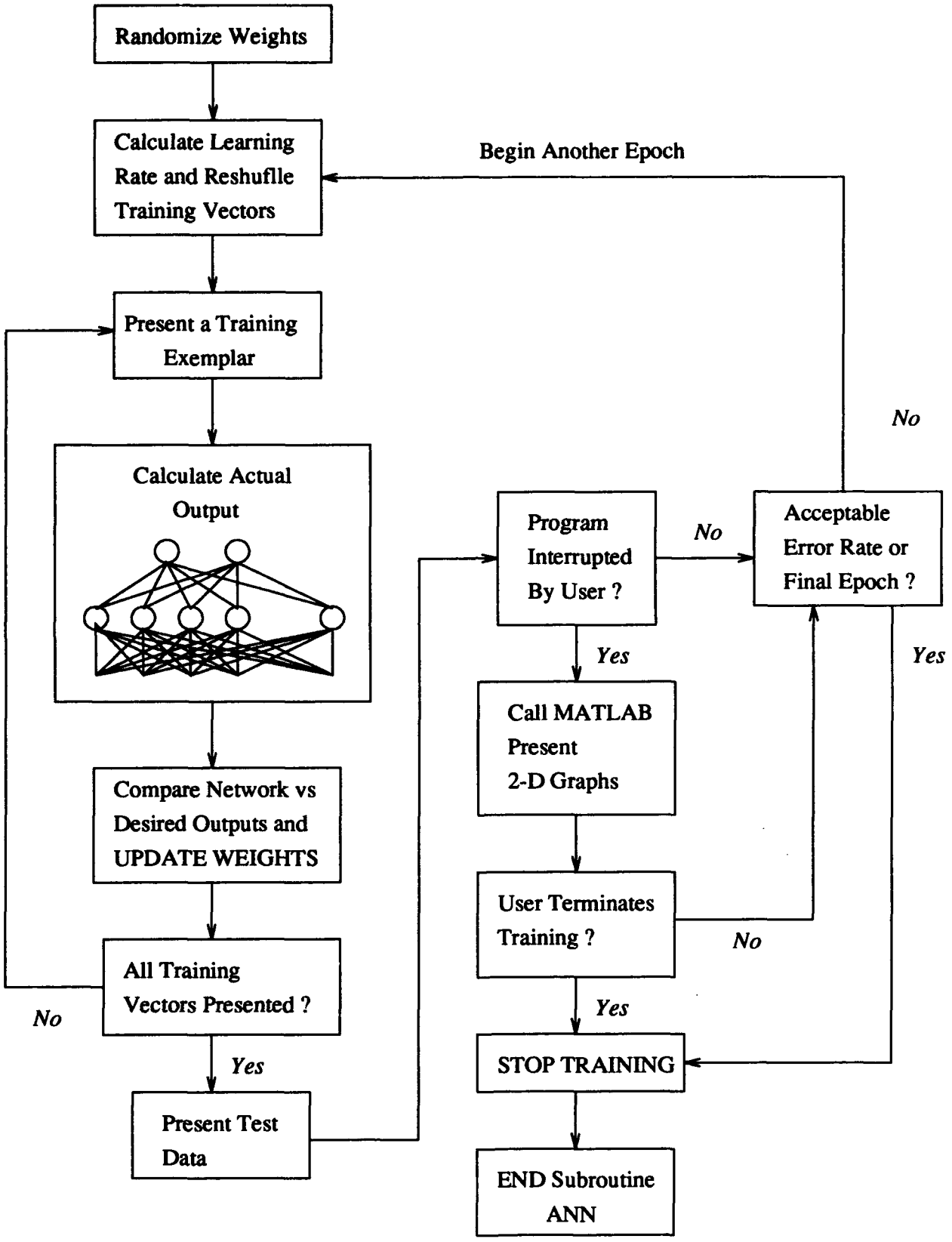



Figure 9. Subroutine ANN Program Flow

weights are random numbers between -.5 and +.5. This *range of weight initialization* is controlled by the user from the input parameter file. After the random initialization is complete, the program begins epoch number one. Each presentation of the set of training vectors and test vectors is defined as an "epoch".

*3.4.1 Calculations By Epoch.* At the beginning of each epoch a learning rate is calculated. Except for the constant learning rate, all learning rates are a function of the specific epoch number. As the epoch number increases, all learning rate functions are designed to decrease. Recall that the selection of a learning rate is of critical importance in finding the true global minimum of the absolute error. Back-propagation training with too small a learning rate is agonizingly slow, but too large a learning rate may produce oscillations between relatively poor solutions. See Appendix A, for specific learning rate functions.

The next step in the ANN subroutine is the reordering of the feature vectors in the training set into a random list. Random ordering prevents the network from learning the order of the data and may speed the training time. This random reordering is accomplished by attaching a random number to each training vector and then using the random number as a sort key. A Shell-Mezgar sort is then applied to the set of training vectors. As mentioned in Section 2.7, this sort needs to be very efficient since it is performed at the beginning of each epoch, which may number in the thousands.

With the training set randomly sorted, we are ready to begin training of the weights in the network. For each exemplar in the training set the algorithm performs the following three steps. First an activation of the hidden layer and output layers is calculated using the sigmoid non-linear transformation. Second, the activations of the output layer are compared to the desired (known) output and placed into the training set confusion matrix. Finally, the back-propagation training procedure is performed updating the network weights. After all the training vectors have been presented to the network, the weights are held constant and the test vectors are presented to the perceptron. Once again, the activations of the output layer are compared to the desired (known) output, but this time placed into the test set confusion matrix. After all training and test vectors have been

presented to the network, an average absolute error and classification error are calculated for the training and test sets. These two errors appear on the 2D graphs produced by MATLAB. They are used as an indicator of the performance of the network. By collecting the absolute error and classification error for the two sets at the end of each epoch, an error curve can be constructed and a minimum error observed somewhere along this curve. In addition, the weights specified by the user to be monitored are saved at the end of each epoch. By graphing these weights, we can see how they have changed over the entire run. Some will be trained and remain relatively constant, while others will still be increasing or decreasing.

*3.4.2 User Directed Interrupt.* The amount of time it takes to train a neural network can range from trivial to infinite. Therefore, it is desirable to design a system which allows the user to "monitor" the training of the network. In order to monitor training, the following screen output was designed for the user.

```
                                    MONITOR SELECTED WEIGHTS
                          ------------------------------------------------
                          INPUT NODE TO HIDDEN NODE  ---> IN-HN-1
                          HIDDEN NODE TO OUTPUT NODE ---> HN-ON-2
                          ------------------------------------------------
EPOCHS      TRAINING   TEST SET
COMPLETED   SET ERROR  ERROR      1- 2-1    2- 4-1    3- 2-2
---------   ---------  --------  ----------------------------------------
       38     0.0889     0.0826   -6.8170    3.3679   -2.8529
```

At the end of each epoch, these figures are updated. Epochs Completed represents the number of epochs which have been completed since training began. The Training and Test Set Errors represent the average absolute error of all exemplars found in the respective sets. The average absolute error is calculated with the following formula

$$\frac{\sum_{j=1}^{P}\sum_{k=1}^{K} \mid z_k^2 - d_{jk} \mid}{PK} \tag{29}$$

where $P$ is the number of exemplars in the respective set, $K$ is the number of output nodes, $z_k^2$ is the network output of output unit $k$, and $d_{jk}$ is the desired or true output of output unit $k$ for the $j$th exemplar. In this example the user has decided to monitor three

different weights. For example, the second monitored weight is designated "2-4-1". The code "2-4-1" indicates the user is monitoring the weight connecting input node 2 to hidden node 4 of layer 1. The third monitored weight is designated "3-2-2". The code "3-2-2" indicates the user is monitoring the weight connecting hidden node 3 to output node 2 of layer 2. To help the researcher keep these indices straight, the heading provides a simple key:

```
INPUT NODE TO HIDDEN NODE  ---> IN-HN-1
HIDDEN NODE TO OUTPUT NODE ---> HN-ON-2
```

where IN represents the input node, HN the hidden node, and ON the output node. The last digit represents the layer.

As the user watches the epochs "tick" by (or comes back to the terminal after lunch or the following day for a "large" neural net), they can interrupt the training procedure at any time. See Appendix A for details. At the end of each epoch subroutine ANN checks to see if the user has interrupted training of the network. If it detects an interrupt the FORTRAN program pauses, creates an input file for MATLAB called "plotdat1.m", and calls MATLAB. The MATLAB program automatically produces four separate graphs of error curves. See Figures 10, 11, 12, and 13. Figure 10 shows the *average absolute error* of the training and test sets for each epoch, while Figure 11 displays the same information for the last 100 epochs only. Figure 12 shows the *classification errors* of the training and test sets for each epoch, while Figure 13 displays the same information for the last 100 epochs only. In addition to the four graphs mentioned above, the user has designated specific weights to monitor in the parameter file. A graph is produced for each weight specified. See Figures 14, and 15.

The goal is to cease training at the point corresponding to a minimum error on the *test* set. The choice of this point may be difficult since it is necessary to consider both average absolute error and classification error. After analyzing the graphs the researcher must determine if additional training of the network is required, or if the network is sufficiently trained. After quitting the MATLAB program, the FORTRAN program prompts the user for their decision. If its determined that additional network training is required, the program will begin at the point where it was interrupted. *No information from prior training is lost*. The user may interrupt the training process as often as necessary. If its determined that the network is sufficiently trained, the program saves all information as

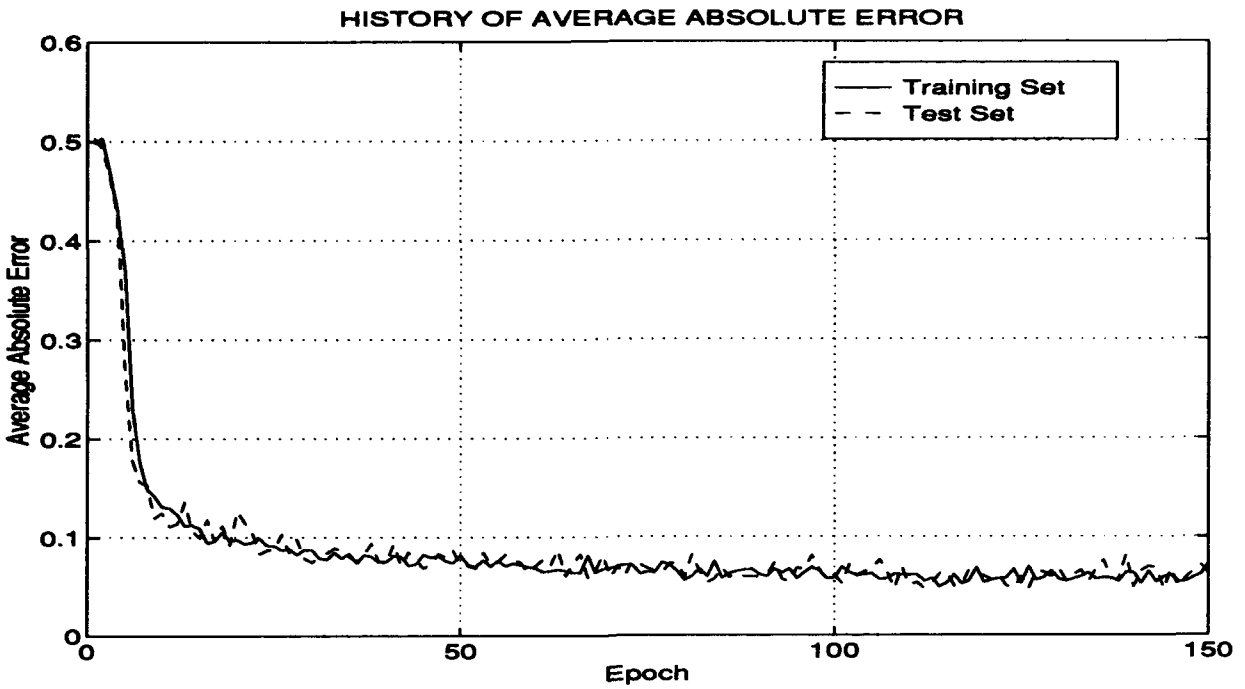



Figure 10. Sample Average Error Distance Curves for Training and Test Sets

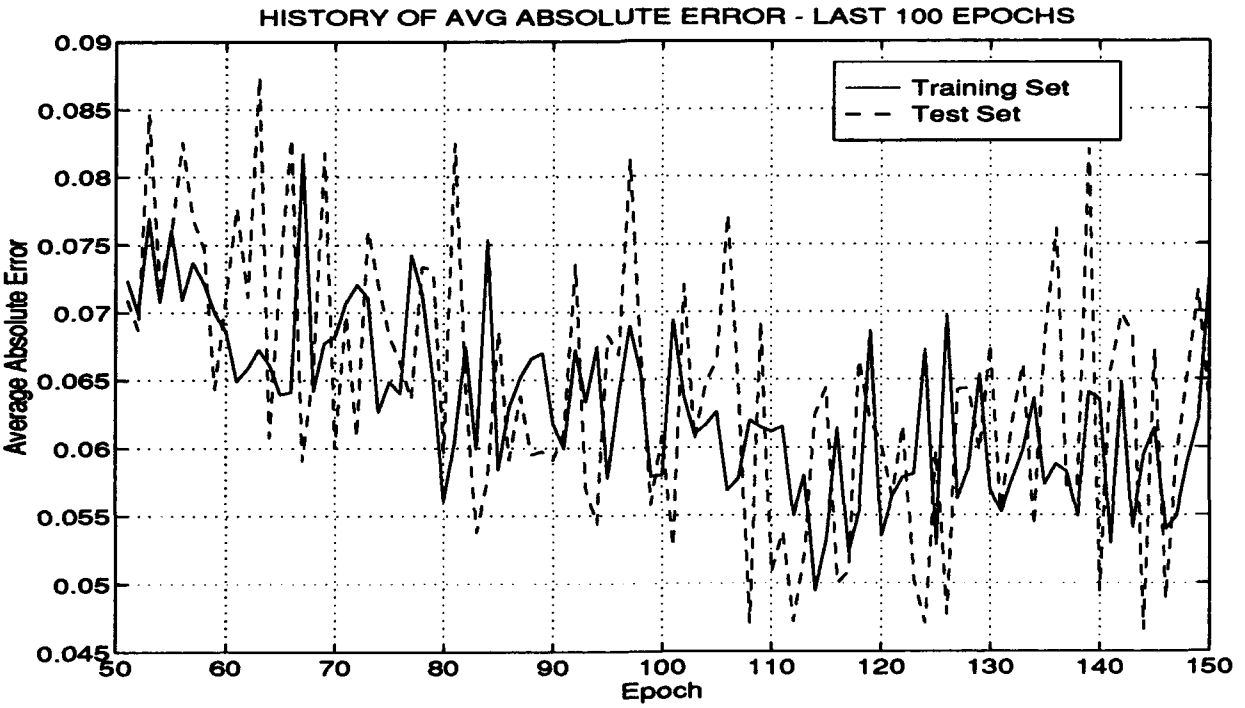



Figure 11. Sample Average Error Distance Curves - Last 100 Epochs Only

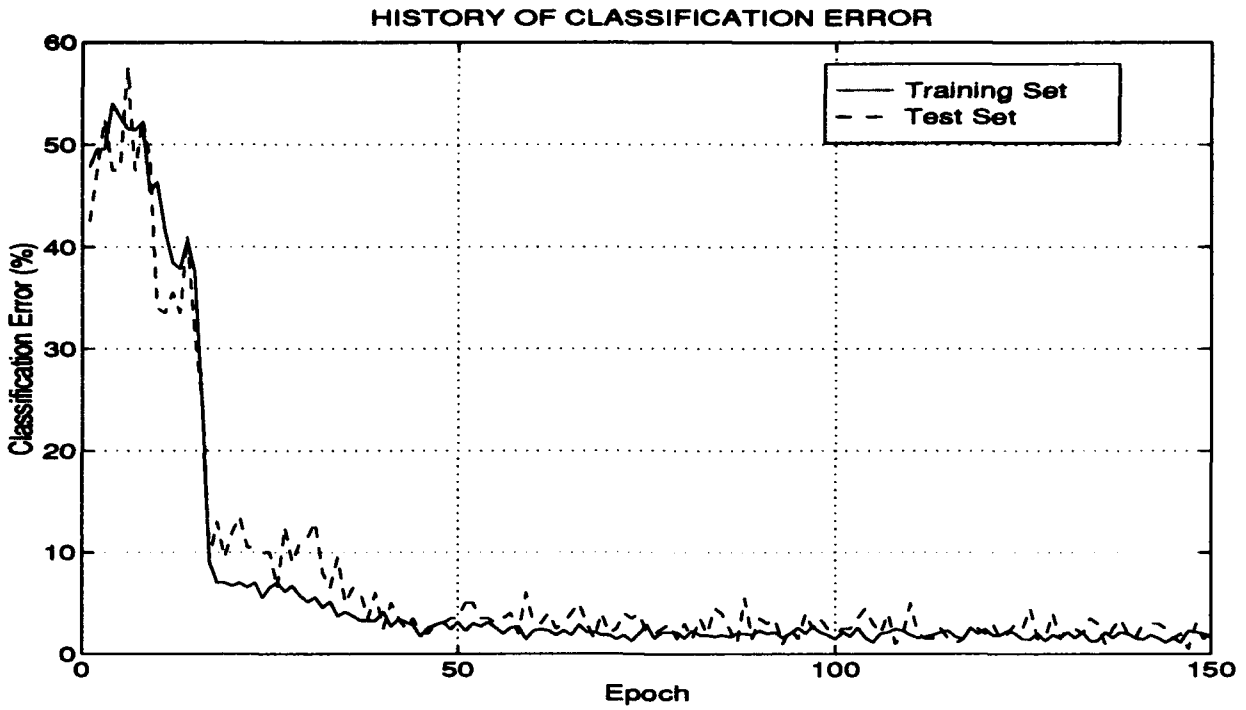



Figure 12. Sample Classification Error Curves for Training and Test Sets

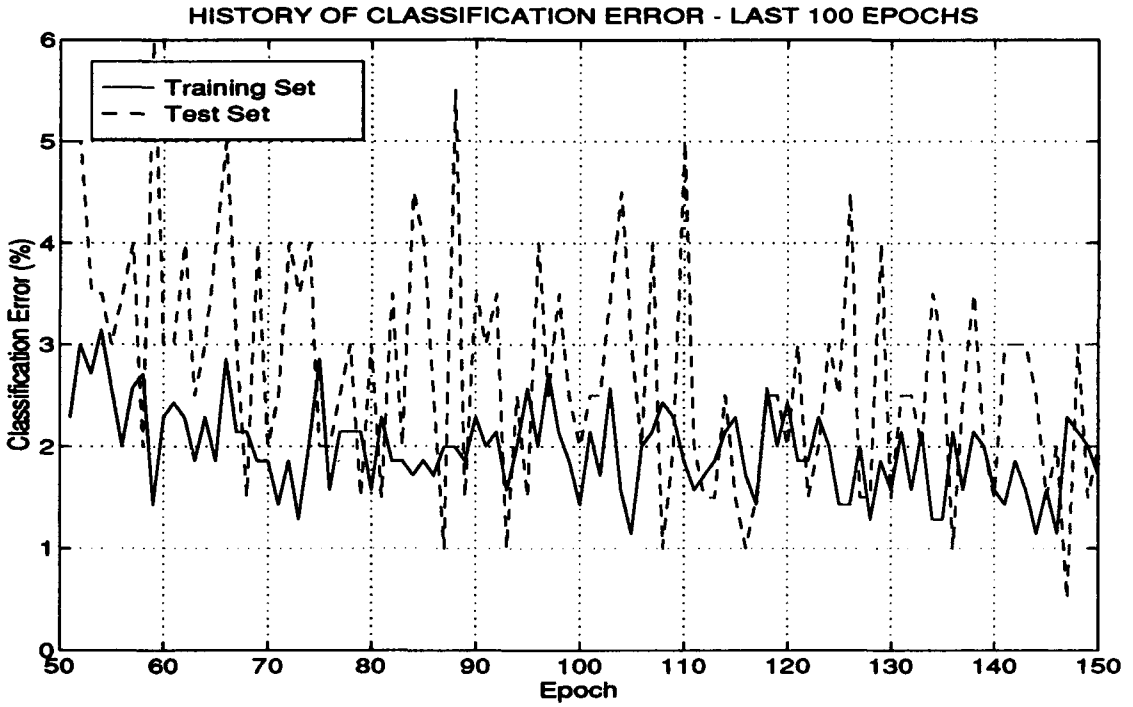



Figure 13. Sample Classification Error Curves - Last 100 Epochs Only

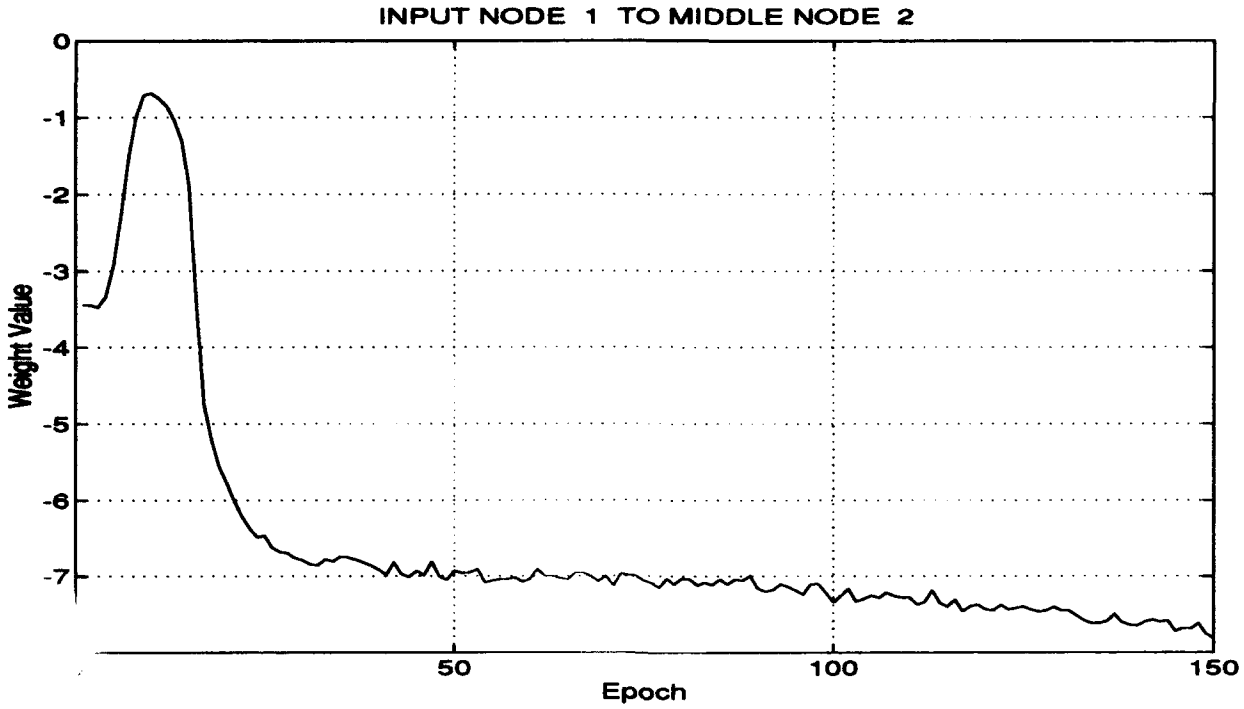



Figure 14. Sample Weight Monitoring Curve - Input Layer to Hidden Layer

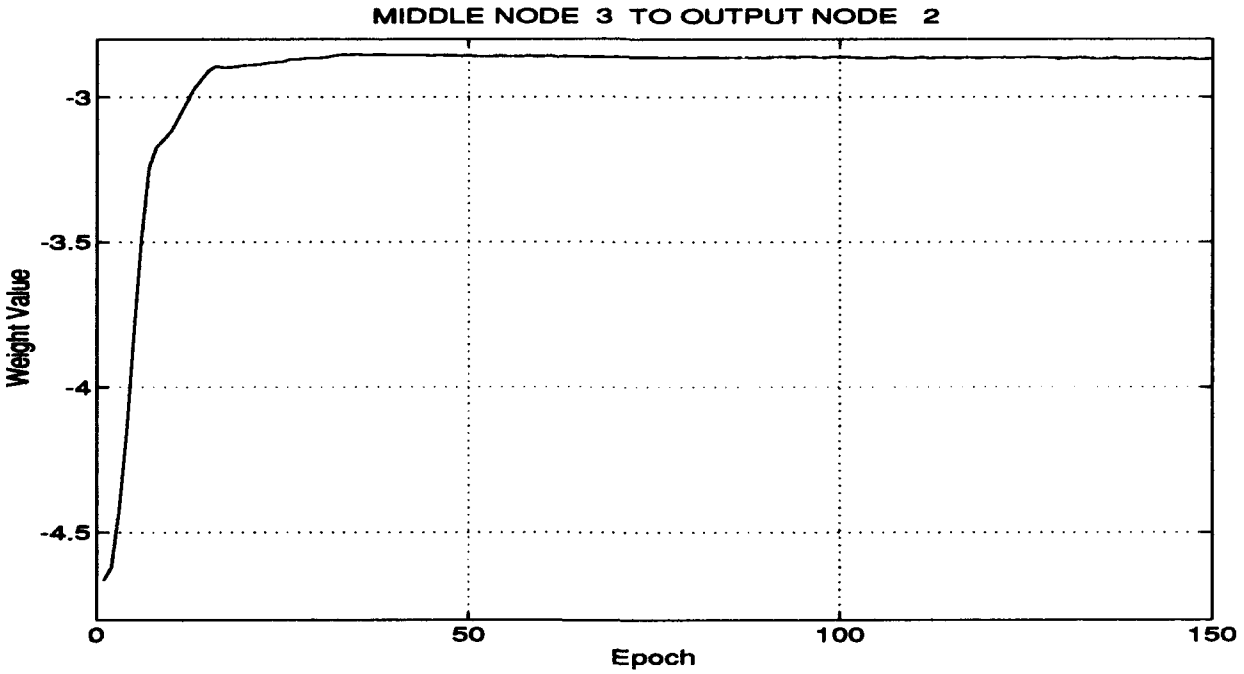



Figure 15. Sample Weight Monitoring Curve - Hidden Layer to Output Layer

of the *last epoch completed*. This information will be used to produce the final output products.

*3.4.3 Termination of Network Training.* The primary function of subroutine ANN is to train the neural network. Since training can go on indefinitely, the subroutine must know when to terminate training. There are four ways to terminate network training.

1. User Directed Termination. After interrupting the program as described above, the researcher analyzes the graphs and decides the average absolute error or the classification error is low enough. The user directs training to stop.
2. The number of epochs as specified in the parameter file is reached. The user may wish to stop training after, say, 3000 epochs.
3. The average absolute error of the training set or test set is less than or equal to the tolerance specified in the parameter file. The user may wish to stop training after the average absolute error goes below, say, 0.05.
4. The maximum number of allowable epochs is reached. This maximum number is set when the FORTRAN program is compiled. It is currently set at 10,000 epochs. If more epochs are required, the researcher must change the parameter in the FORTRAN program and recompile. See Appendix A.

Once network training has been terminated, the weights from the last completed epoch are written to the file, "weights.dat". These are considered the final network trained weights. In addition, the file "plotdat1.m' is recreated for input into MATLAB. All subsequent subroutines will use these *final* trained weights.

## *3.5* Saliency Calculations

Now that we have a final set of trained weights, we can begin the saliency calculations. Subroutines RUCKSAL, RUCKSAL2, and TSAL all calculate a saliency metric using different algorithms.

Subroutine RUCKSAL computes Ruck's saliency for each of the features based on the weights of the final trained network. Recall that saliency is a measure of the significance

that a feature has on the output of the multilayer perceptron. This subroutine uses pseudo-sampling and calculates the saliency metric using Equations 20 and 22.

Subroutine RUCKSAL2 also computes Ruck's saliency for each of the features based on the weights of the final trained network. However, no pseudo-sampling is involved. The computations in RUCKSAL2 reflect a slight modification to Equation 22. By eliminating the pseudo-sampling Equation 22 becomes:

$$\Lambda_i \equiv \sum_{p=1}^{P} \sum_{m=1}^{M} \sum_{k=1}^{K} \left| \frac{\partial z_k^2}{\partial x_i}(\vec{x}_p^m, \vec{w}) \right| \tag{30}$$

Subroutine TSAL calculates Tarr's saliency of each of the features based on the weights of the final trained network. Tarr's saliency for each feature is a function of the weights from a particular input node to all middle nodes. This subroutine calculates four types of Tarr saliencies. On all computer outputs they are referred to as TARR1, TARR2, TARR3, and TARR4 where:

1. TARR1 represents the sum of the squared weights from a particular input node to all middle nodes. This is Tarr's original saliency as defined in Equation 23.
2. TARR2 will signify the square root of the sum of the squared weights from a particular input node to all middle nodes. This can be considered the Euclidian Norm of a feature's weights.
3. TARR3 will signify the sum of the absolute value of the weights from a particular input node to all middle nodes. This is sometimes referred to as the taxi-cab norm of a feature's weights.
4. TARR4 will represent the largest weight in absolute value from a particular input node to all middle nodes. This can be considered the Infinity Norm of a feature's weights.

### *3.6* Validation Subroutine

The subroutine VALIDT presents the validation set defined in the parameter file to the *final* trained network. Recall that the validation set is presented only after the

multilayer perceptron is considered optimally trained. It verifies the performance of the trained network since its exemplars are never seen by the network during its development. The subroutine runs each exemplar through the network and creates a confusion matrix for the validation set.

### *3.7* Correlation Subroutine

The subroutine CORRELATE calculates second-order correlation matrices of the product of two inputs with the output. A matrix is produced for each output. The resulting matrices are helpful in determining which second-order terms should be included in the input vectors. The correlation matrices are calculated using Equations 24 and 26. These matrices are printed in the summary report.

### *3.8* Activation and Saliency Grids

Subroutine GRID calculates the data to be used in the MATLAB 3-D activation grids. In the parameter file, the user defines which variables are to be plotted on the $x$, $y$, and $z$ axes. The variables assigned to the $x$ and $y$ axes are features, while the variable assigned to the $z$ axes is the activation of a particular output class.

The program creates a 35x35 grid, where the $x$ axis covers the range of the feature selected for the $x$ axis using 35 equal increments, and the $y$ axis covers the range of the feature selected for the $y$ axis using 35 equal increments. If a finer or coarser mesh than 35x35 is desired, the user will have to change the parameter GRIDDIM in the FORTRAN program and recompile. These 1225 grid points are then run through the final trained network and the activation of the designated output class is recorded for graphing on the $z$ axis. We now have 1225 *3-tuples* to pass to MATLAB for 3-D processing.

When we create the 1225 grid points, we are, in effect, creating a new set of 1225 exemplars with values for the two features being graphed, only. The question arises: what values should be used in these new exemplars for the features that are not being graphed? Since we are running these 1225 exemplars through the network, each exemplar must have values for the features not being graphed. If the network has only two features, there

is no problem. However, if there is more than two features, then choices must be made by the user. The program allows the user to choose a constant value for all features not graphed, or to choose the mean value of that feature, as calculated over the training set. See Appendix A.

Subroutine GRIDSAL calculates the data to be used in the MATLAB 3-D saliency grids. In the parameter file, the user defines which variables are to be plotted on the $x$, $y$, and $z$ axes. The variables assigned to the $x$ and $y$ axes are features, while the variable assigned to the $z$ axes is the saliency, $\Lambda_i$, of the feature on the $x$ or $y$ axis.

Once again, the program creates a 35x35 grid, where the $x$ axis covers the range of the feature selected for the $x$ axis using 35 equal increments, and the $y$ axis covers the range of the feature selected for the $y$ axis using 35 equal increments. These 1225 grid points are then run through the final trained network and the saliency of the designated feature is recorded for graphing on the $z$ axis. Equations 20 and 30 are used to calculate the saliency.

As before, when we create the 1225 grid points, we are creating a new set of 1225 exemplars with values for the two features being graphed, only. Once again, the user chooses a constant value for all features not graphed, or the mean value of that feature, as calculated over the training set. See Appendix A.

### *3.9* Summary Reports

Subroutine OUT produces a summary report of the final trained network. This report summarizes the following:

- Network parameters used to build and train the network.
- Confusion matrices for training, test, and validation sets
- Summary of Ruck's saliency metric. Side by side comparison of "with" and "without" pseudo-sampling for each feature.
- Summary of Tarr's saliency metric. All four variants of Tarr's saliency metric are shown for each feature.

- a second-order correlation matrix for each output.

An example of this report can be found in Appendix B.

# *IV.* Verification and Validation

In her thesis, Belue created a flowchart which depicted a procedure the researcher can use to iterate through the myriad of multilayer perceptron parameter combinations and arrive at an optimal network structure [1:48]. Figure 16 is a reproduction of this flowchart. The primary parameters are number of epochs, number of middle nodes, learning rate, and momentum rate. Although this is not an optimal testing strategy (the interactions of the parameters are confounded), we will use this strategy to verify and validate our computer model. The alternative is to train the multilayer perceptron for all possible combinations of number of middle nodes, learning rates, and momentum rates. Time constraints do not make this approach practical. Two classical classification problems will help verify and validate the model. They are the XOR problem and the four class MESH problem. It must be emphasized that all graphs and tables found in this chapter are automatically produced by this program. That's what makes this system a very powerful analysis tool.

Before we examine these two problems, we need to define the network structure being used at each point in the flowchart. The network structure is defined by the following parameter vector:

$$(n_1, n_2, n_3, \eta, \alpha) \tag{31}$$

where $n_1$ is the number of input nodes, $n_2$ the number of middle nodes, $n_3$ the number of output nodes, $\eta$ the learning rate, and $\alpha$ the momentum rate.

## *4.1 XOR Problem*

The XOR problem is often used to test classifiers to determine their ability to classify non-linearly separable decision regions. Figure 17 illustrates the problem. We see that the regions $\theta_1$ and $\theta_2$ cannot be separated by a single line. In addition to the two significant variables $x_1$ and $x_2$, we will add two noise variables, $x_3$ and $x_4$. These noise variables will be used to see if the saliency metrics can detect them, and to see what a 3-D graph of noise saliency looks like.

The rest of this section will show the researcher how to use this system to solve the XOR problem. Real world problems can be analyzed in an analogous manner.

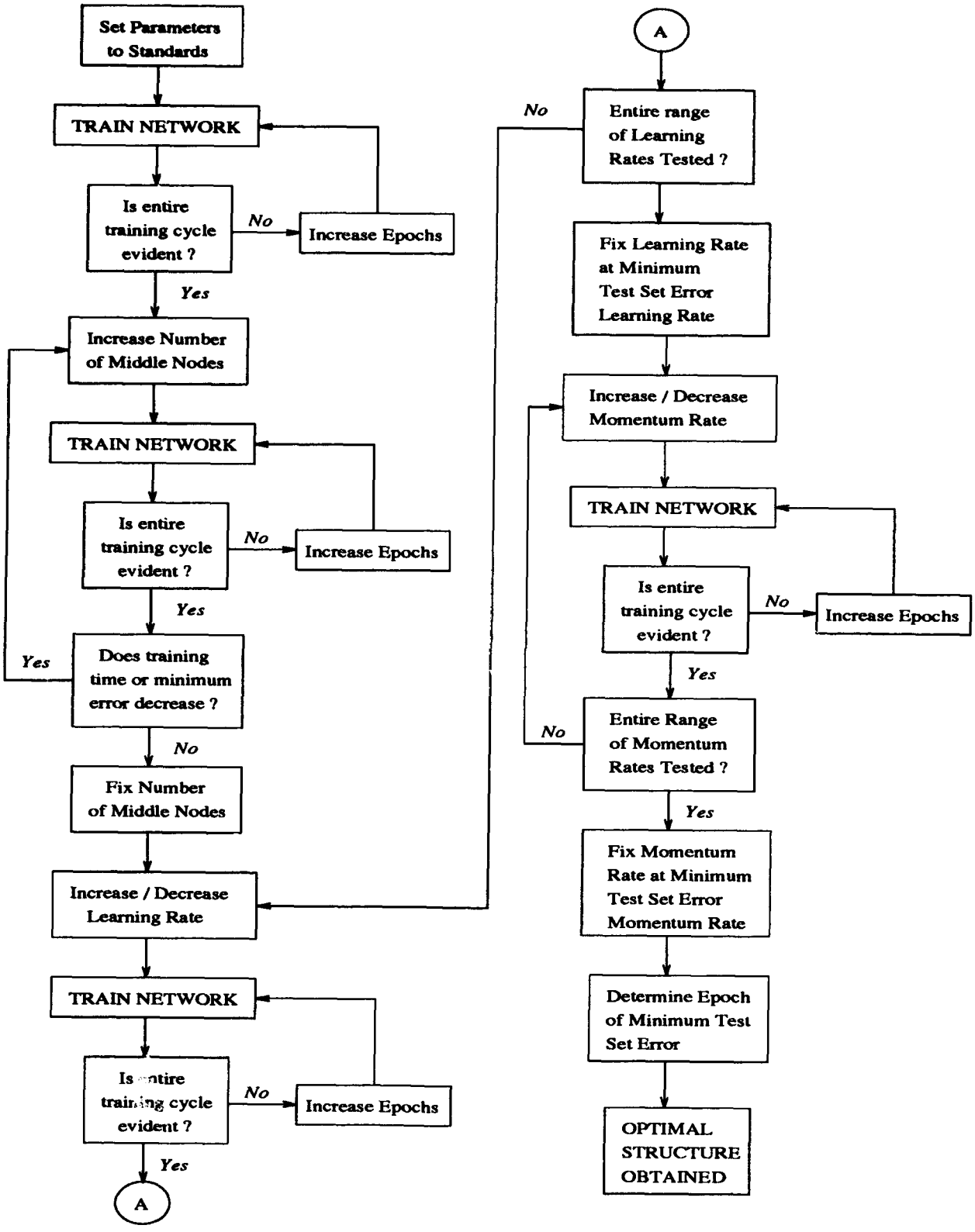



Figure 16. Building An Optimal Network

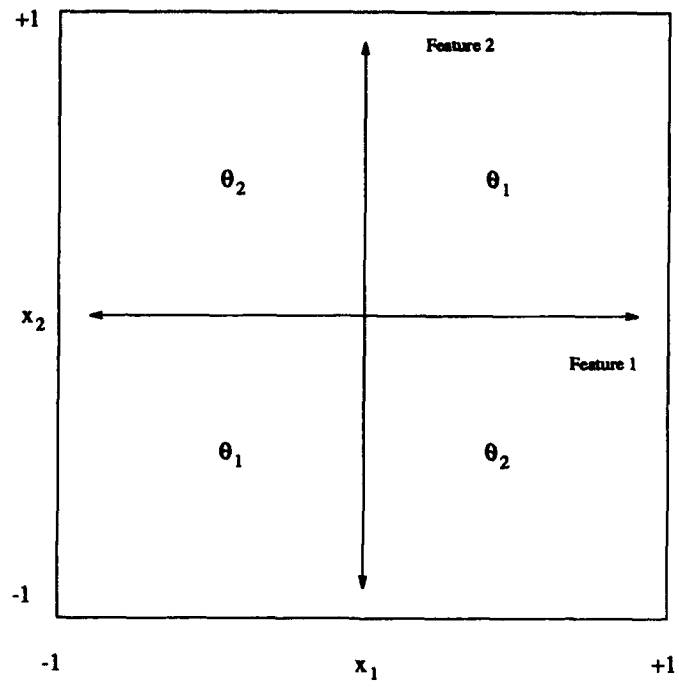



Figure 17. The XOR Problem

*4.1.1 XOR Network Structure (4,2,2,1,0).* Our first task is to determine a standard set of parameters. For purposes of illustration, we are going to assume that this is a brand new problem and that no standard parameters exist. Therefore, our initial parameter vector will consist of *best guesses*. The initial parameter vector is: $(4, 2, 2, 1, 0)$. Figures 18, 20, and 19 show selected output produced by the FORTRAN and MATLAB programs. Keep in mind that all these outputs were produced *automatically*. All the user had to do was create the parameter file which defined the network structure, the stopping criteria, and the 3-D graphs desired. See Appendix A.

Figure 18 shows a "History of Average Absolute Error" and a "History of Classification Error". The researcher has made a decision that the *entire training cycle* is evident after 300 epochs and stops training the network at this point. The average absolute error has leveled off at around 0.32 for the training set, and 0.30 for the test set. Classification error has leveled off at 30% for both sets. If the researcher wanted to continue training beyond the 300 epochs, they would simply instruct the program to continue training. The program would continue training starting at epoch 301. All information learned from the

first 300 epochs is retained. The researcher does not have to go back to square (epoch) one and start over. They can interrupt the training process as often as they like to determine if the training cycle is complete.

Figure 19 shows the average absolute error, classification error, and confusion matrices for the training and test sets at epoch 300. The training set had an average absolute error of 0.32 and a classification error of 30.8%, while the test set had an average absolute error of 0.30 and a classification error of 26.7%. We see that this network structure classifies the $\theta_1$ class (Class 1) correctly 240 out of 336 times in the training set, while the $\theta_2$ class (Class 2) is correctly classified 210 out of 314 times. The test set confusion matrix can be read in a similar manner. The researcher's objective is to drive up the numbers on the diagonal elements and reduce the off diagonal elements for both sets.

Figure 20 shows a 3-dimensional view of the network at epoch 300. A "perfectly" trained network would show the activations of Class 1 to be equal to one in the first and third quadrants, while the activations of Class 1 in the second and fourth quadrants would be equal to zero. In a similar fashion, the "perfectly" trained network would show the activations of Class 2 to be equal to one in the second and fourth quadrants, while the activations of Class 2 in the first and third quadrants would be equal to zero. Figure 20 shows that this network structure is beginning to learn how to classify exemplars in the first and fourth quadrants, but is having difficulty in classifying exemplars found in the second and third quadrants. Following the flowchart of Figure 16, our next step is to increase the number of middle nodes.

*4.1.2 XOR Network Structure (4,4,2,1,0).* In order to change the network structure from 2 middle nodes to 4 middle nodes, the researcher simply changes the middle node parameter in his parameter file. Figures 21, 23, and 22 show the output produced by the (4,4,2,1,0) network structure.

Figures 21 and 22 show that after 300 epochs the average absolute error is 0.10 for the training set and 0.13 for the test set. In addition, the classification error has been reduced to 6.8% for the training set and 10.2% for the test set. This is a substantial improvement over the (4,2,2,1,0) network structure.

Figure 23 shows the 3-dimensional view of the network at epoch 300 for the (4,4,2,1,0) network structure. We see that this network structure is beginning to learn how to classify exemplars in all four quadrants. Recall that the (4,2,2,1,0) network structure was having difficulty classifying exemplars in the second and third quadrants. This new network structure classifies the $\theta_1$ class correctly 316 out of 336 times in the training set, for a 94.05% accuracy, while the $\theta_2$ class is correctly classified 290 out of 314 times, a 92.36% accuracy rating. The test set shows 92.52% and 87.29% accuracy for the two respective classes. By simply changing the number of middle nodes from 2 to 4 in the network structure, we have greatly improved its performance. Proceeding on through the flowchart of Figure 16, we analyze our way to the following "optimal" structure.

*4.1.3 XOR Network Structure (4,10,2, 0.2, 0).* There are many subjective decisions which the researcher must make to arrive at the "optimal" network structure. After trying numerous other combinations of middle nodes, learning rates, and momentum rates, the (4,10,2, 0.2, 0) network structure was found to be "optimal". The researcher must decide when the error rates are low enough. How many additional epochs is the user willing to compute in order to squeeze out an additional percentage point. Recall that, if training goes on for too many epochs, over-fitting of the training data may occur. The error curve for the test set may begin to rise.

The top halves of Figures 24 and 25 show that the average absolute error and classification error are quickly driven toward zero. A desirable outcome. In addition, Figure 30 shows that after 400 epochs the average absolute error is 0.04 for the training set and 0.04 for the test set. The classification error has been reduced to 0.6% for the training set and 0.4% for the test set. These error rates are judged to be low enough for this illustration. This will become the optimal network structure. This network is now ready to *predict* the classification of new exemplars.

The bottom halves of Figures 24 and 25 show the last 100 epochs of their respective top halves. In effect, it allows the researcher to zoom in on the activities of the last 100 epochs. By zooming in, the bottom half of Figure 24 reveals that the average absolute

error of both sets is still decreasing. In addition, the bottom half of Figure 24 shows that the classification error for the test set was actually 0 at epoch 359.

Figure 27 shows the 3-dimensional view of the trained network at epoch 400 for the (4,10,2, 0.2 ,0) network structure. We see that fine tuning the network parameters begins to "square off" the cubical shapes indicating more precision at the class borders. In addition, the saddle point begins to rise toward an activation of one.

Figure 26 shows an example of two different weights being monitored while training is on-going. The researcher defines which weights to monitor in the parameter file. See Appendix A. The top graph monitors the weight from "Input Node 1 to Middle Node 5". Input node 1 represents the variable $x_1$, a significant variable (i.e., not a noise variable). The bottom graph monitors the weight from "Input Node 3 to Middle Node 2". Input node 3 represents the variable $x_3$, a noise variable. Note the jaggedness of the weights eminating from the noise variable compared to the smoother curve of weights eminating from the significant variable. As Tarr predicted, weights fluctuating around zero eminate from noise variables.

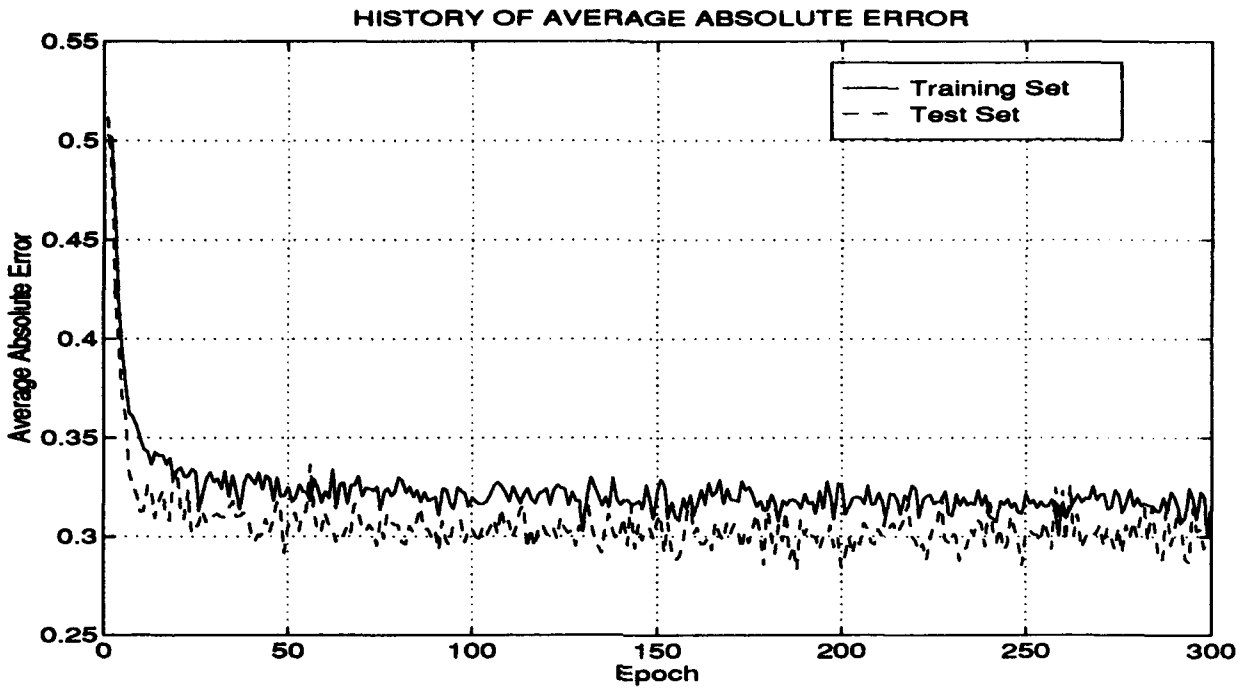
The 3-dimensional graphs of Figure 28 show the saliency of $x_1$ and $x_2$ respectively. Recall that the saliency of $x_1$ is just the change in network output (class) with respect to a change in $x_1$ network input. As $x_1$ goes from negative to positive values we see that the saliency gets very large in value. This indicates that there is a classification change along these high saliency values. In effect, we are drawing the borders of the two classes $\theta_1$ and $\theta_2$. In a similar fashion, the bottom half of Figure 28 shows the saliency of $x_2$. As $x_2$ goes from negative to positive values, we see that once again the saliency gets very large in value. This represents another border between the two classes.

The 3-dimensional graph of Figure 29 shows the saliency of $x_4$ which is named "Noise 2" on the graph. Note that the scale of the $z$-axis is very small when compared to the two previous saliency plots. If Figure 29 were redrawn using the $z$-axis scales of Figure 28 it would appear to be a flat plane. This gives the researcher another visual way of identifying noise variables.

Figure 31 shows Ruck's saliency metric with and without pseudo-sampling, as well as all four variants of Tarr's saliency. Recall that, the higher the value of a saliency metric for a particular variable, the higher its ranking as a significant variable. Ruck's saliency for features 3 and 4 are at least one order of magnitude less than the saliency's for features 1 and 2. Once again, features 3 and 4 look like noise variables. Tarr's saliency metrics also indicate that features 1 and 2 are significant, while features 3 and 4 are noise.

Figure 32 shows the confusion matrix of the final trained net when applied to the validation set. It shows that only one exemplar was misclassified out of 125 total exemplars. In addition, the second-order correlation matrix for each output class is listed. The only significant element in these matrices occurs at the $(1,2)$ position. These values of .77129 and .77133 indicate that an $x_1x_2$ term may have significant explanatory power for classifying the exemplars.

At this point, the researcher may want to restructure the exemplar data sets. Since features 3 and 4 have been shown to be noise, we can drop them from the dataset. In addition, we may wish to add an $x_1x_2$ term to the exemplar data set. This program could then be used to build an improved network structure that uses only three input features (i.e., $x_1$, $x_2$, and $x_1x_2$), or perhaps a network structure that uses only the $x_1x_2$ variable. This computer tool can help build these structures efficiently.



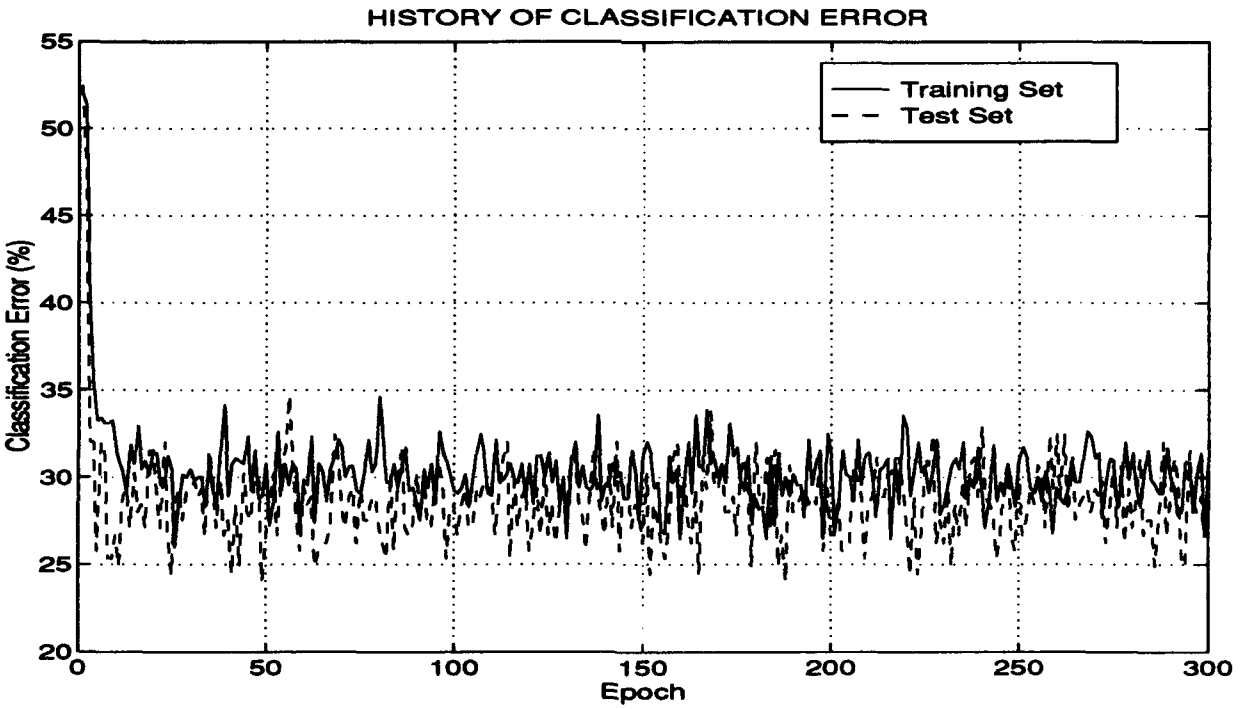




Figure 18. Network Structure (4,2,2,1,0) Absolute/Classification Error

```
---------------------------------
|          TRAINING SET          |
---------------------------------
CLASSIFICATION ERROR (%):  30.7692
AVERAGE ABSOLUTE ERROR  :   0.3172
---------------------------------
                ----------------------------
                | NETWORK  CLASSIFICATION  |
                ----------------------------
                |CLASS  1|CLASS  2| TOTAL |
---------------+--------+--------+---------
|      |CLASS  1|    240 |     96 |     336|
|  T   |        |  71.43%|  28.57%|        |
|  R   |--------+--------+--------+---------
|  U   |CLASS  2|    104 |    210 |     314|
|  E   |        |  33.12%|  66.88%|        |
|      |--------+--------+--------+---------
|      |  TOTAL |     344|     306|     650|
-------------------------------------------
*******************************************
---------------------------------
|            TEST SET            |
---------------------------------
CLASSIFICATION ERROR (%):  26.6667
AVERAGE ABSOLUTE ERROR  :   0.3002
---------------------------------
                ----------------------------
                | NETWORK  CLASSIFICATION  |
                ----------------------------
                |CLASS  1|CLASS  2| TOTAL |
---------------+--------+--------+---------
|      |CLASS  1|     49 |     58 |     107|
|  T   |        |  45.79%|  54.21%|        |
|  R   |--------+--------+--------+---------
|  U   |CLASS  2|      2 |    116 |     118|
|  E   |        |   1.69%|  98.31%|        |
|      |--------+--------+--------+---------
|      |  TOTAL |      51|     174|     225|
-------------------------------------------
```

Figure 19. Network Structure (4,2,2,1,0) Train/Test Confusion Matrices

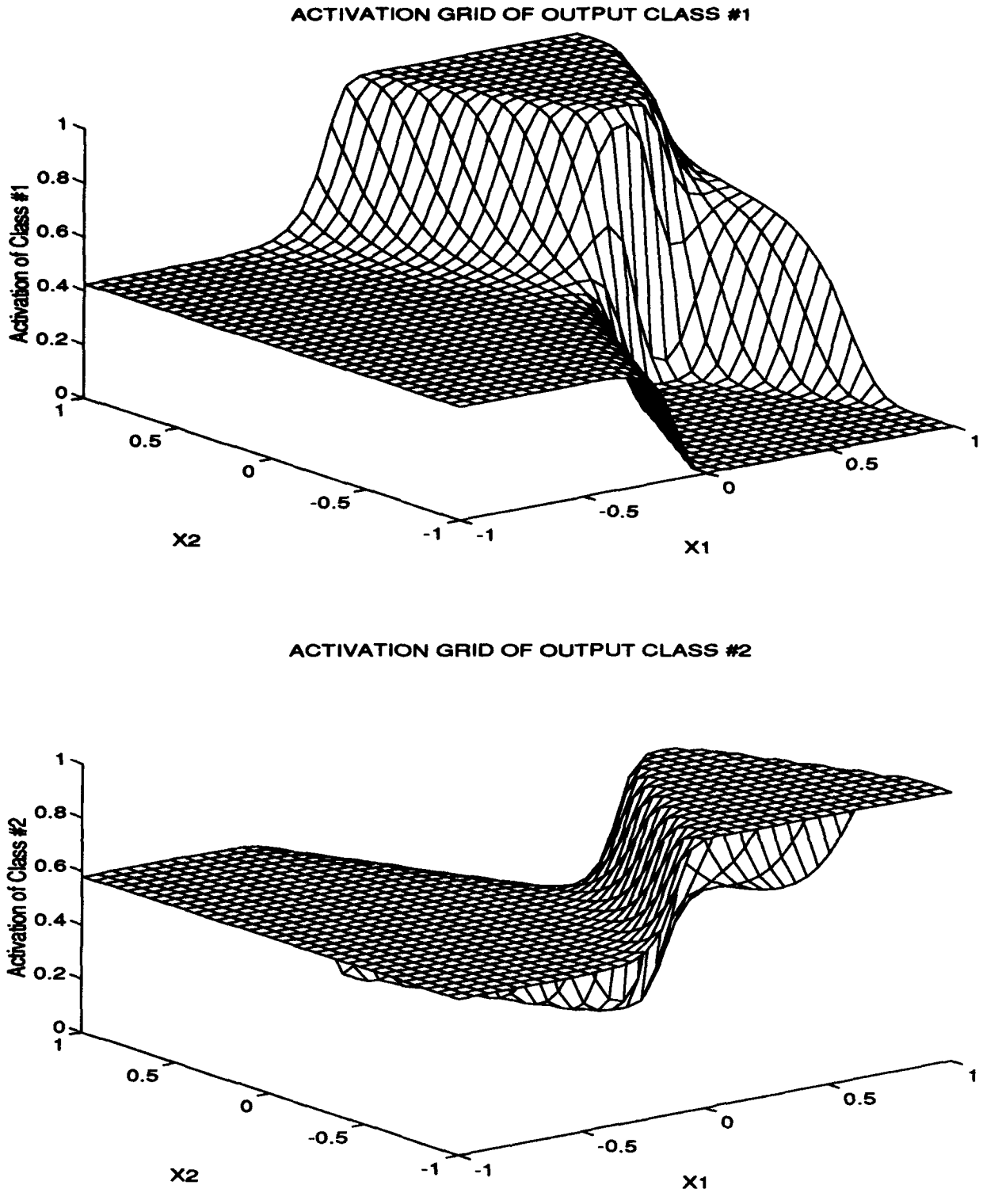
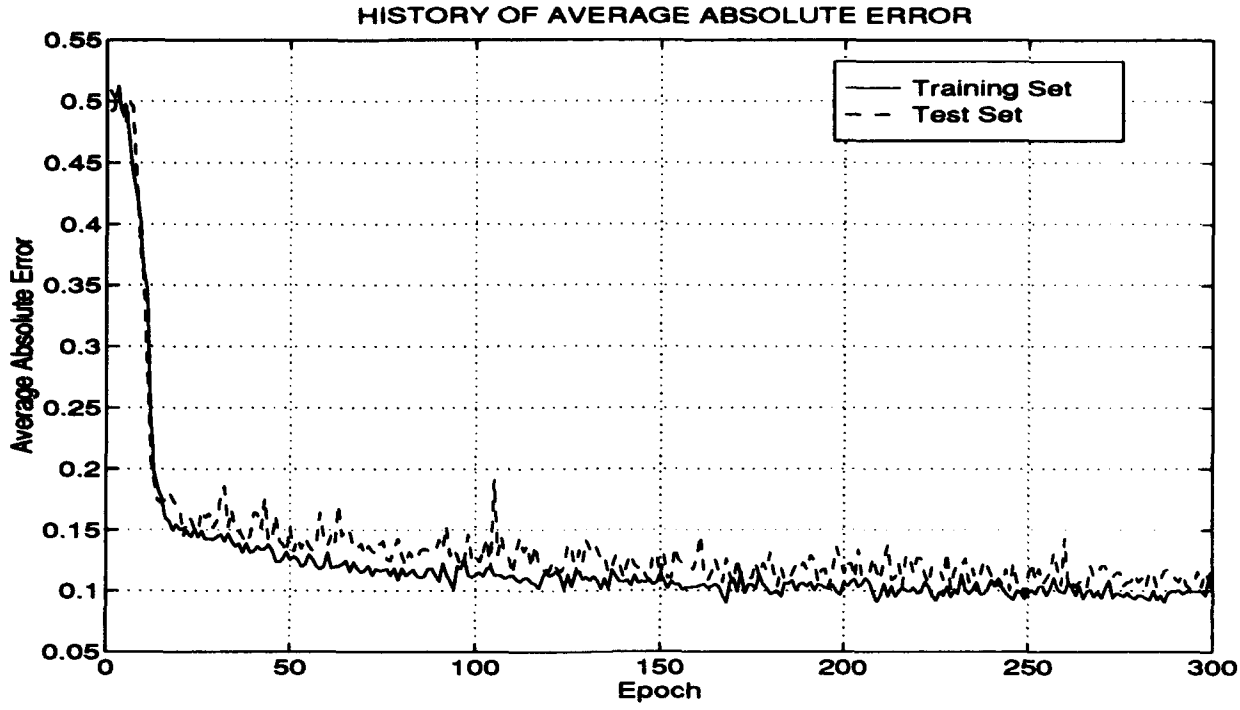



Figure 20. Network Structure (4,2,2,1,0) Activation Grids



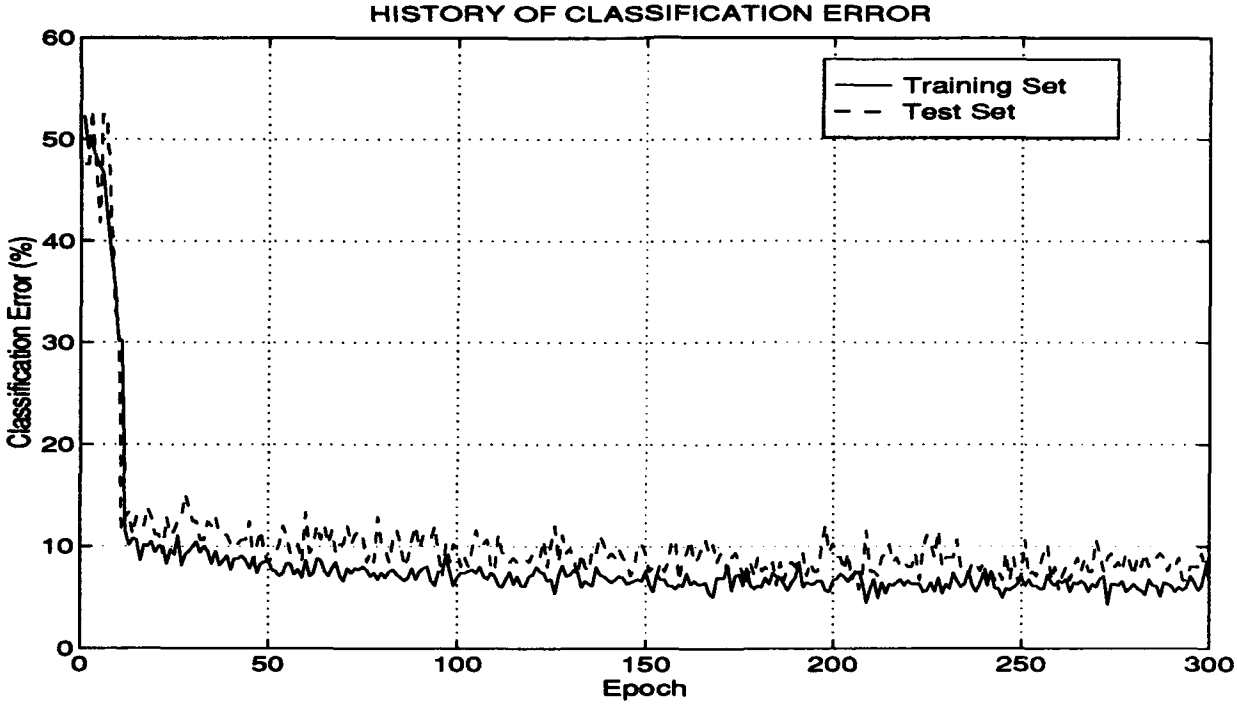




Figure 21. Network Structure (4,4,2,1,0) Absolute/Classification Error

```
---------------------------------
|          TRAINING SET         |
---------------------------------
CLASSIFICATION ERROR (%):   6.7692
AVERAGE ABSOLUTE ERROR  :   0.1008
---------------------------------

                ----------------------------
                | NETWORK  CLASSIFICATION  |
                ----------------------------
                |CLASS  1|CLASS  2| TOTAL |
---------------+--------+--------+--------
|      |CLASS  1|    316 |     20 |     336|
|  T   |        | 94.05% |  5.95% |        |
|  R   |--------+--------+--------+--------
|  U   |CLASS  2|     24 |    290 |     314|
|  E   |        |  7.64% | 92.36% |        |
|      |--------+--------+--------+--------
|      | TOTAL  |    340 |    310 |     650|
-------------------------------------------
*******************************************
---------------------------------
|           TEST SET            |
---------------------------------
CLASSIFICATION ERROR (%):  10.2222
AVERAGE ABSOLUTE ERROR  :   0.1258
---------------------------------

                ----------------------------
                | NETWORK  CLASSIFICATION  |
                ----------------------------
                |CLASS  1|CLASS  2| TOTAL |
---------------+--------+--------+--------
|      |CLASS  1|     99 |      8 |     107|
|  T   |        | 92.52% |  7.48% |        |
|  R   |--------+--------+--------+--------
|  U   |CLASS  2|     15 |    103 |     118|
|  E   |        | 12.71% | 87.29% |        |
|      |--------+--------+--------+--------
|      | TOTAL  |    114 |    111 |     225|
-------------------------------------------
```

Figure 22. Network Structure (4,4,2,1,0) Train/Test Confusion Matrices

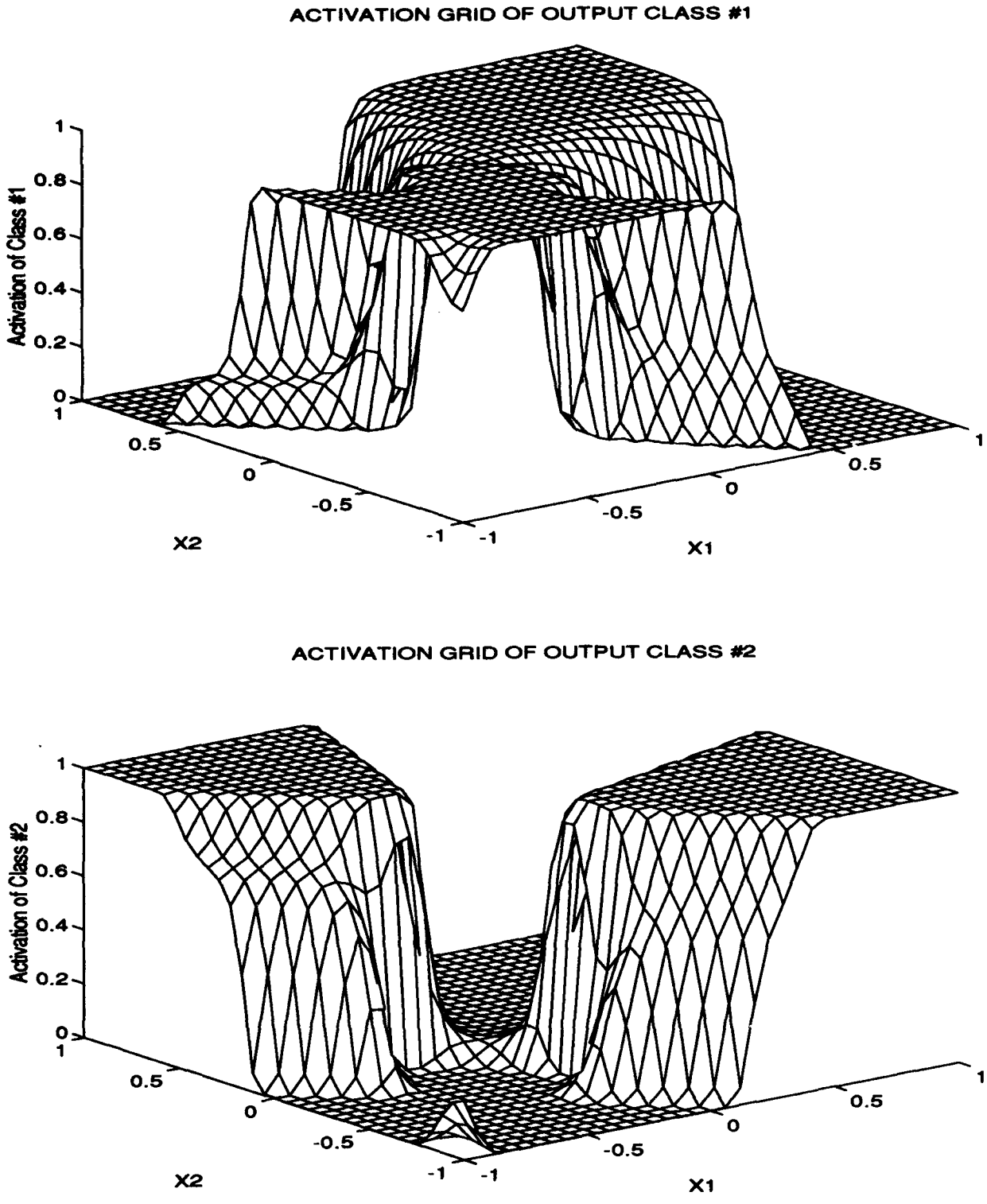
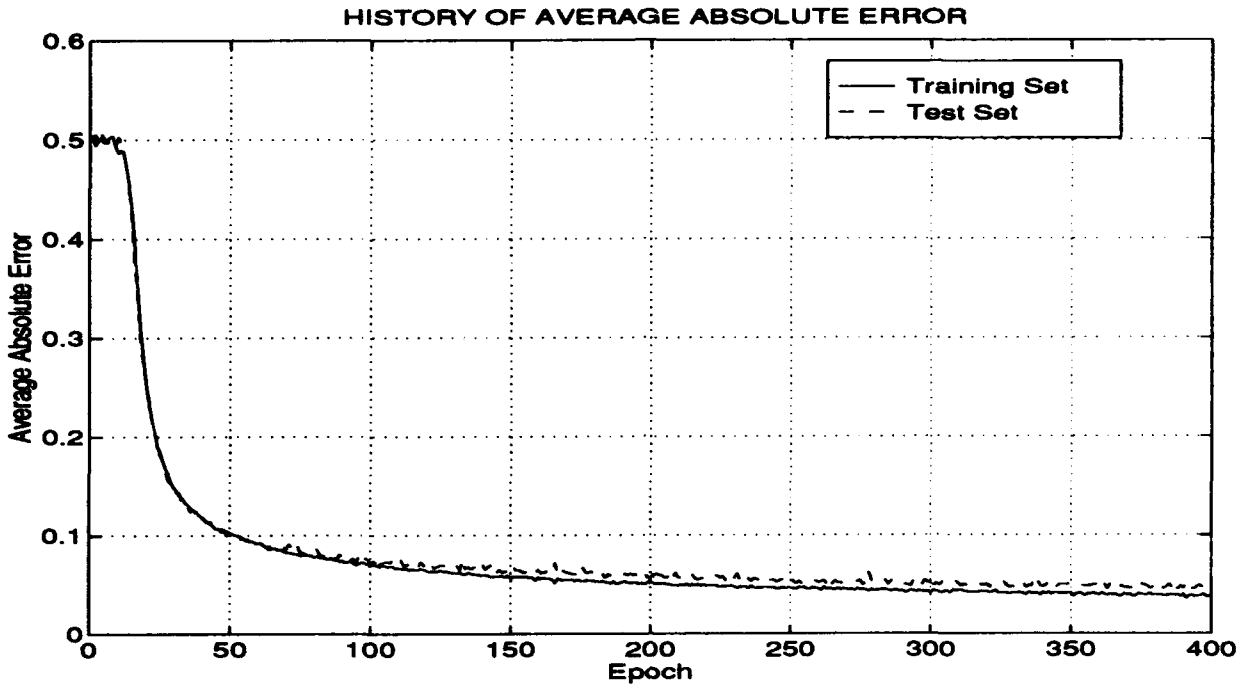



Figure 23. Network Structure (4,4,2,1,0) Activation Grids



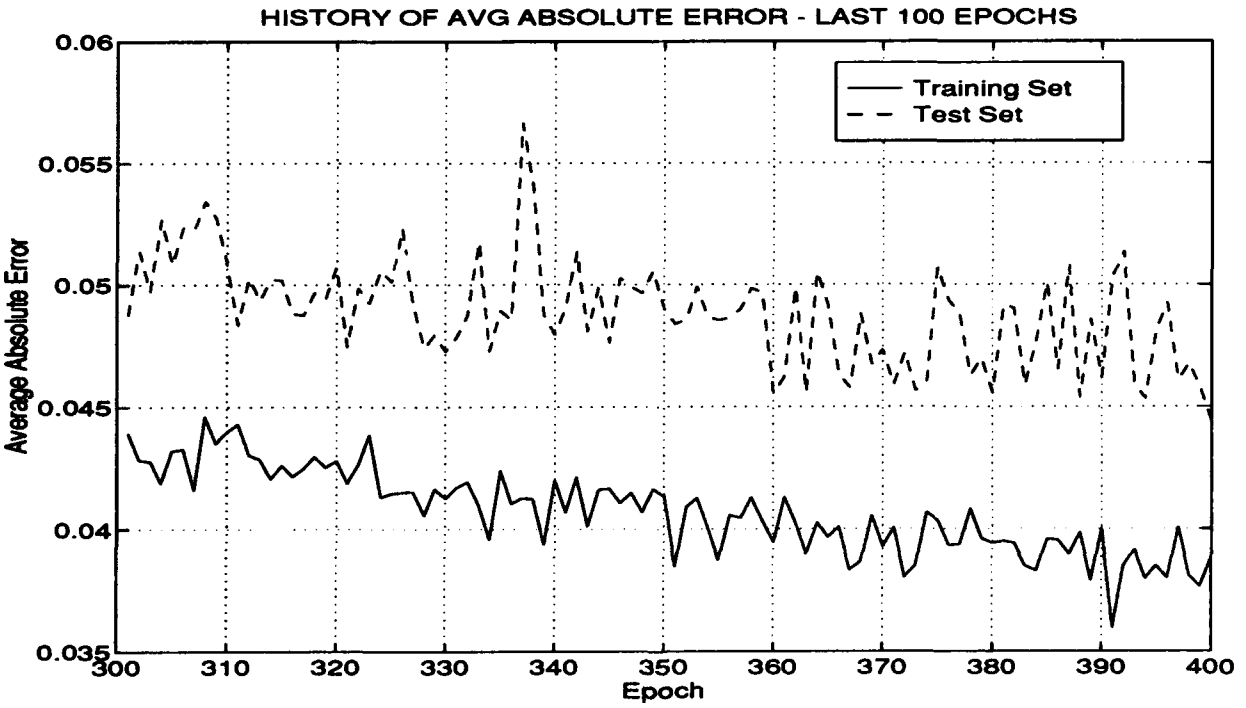
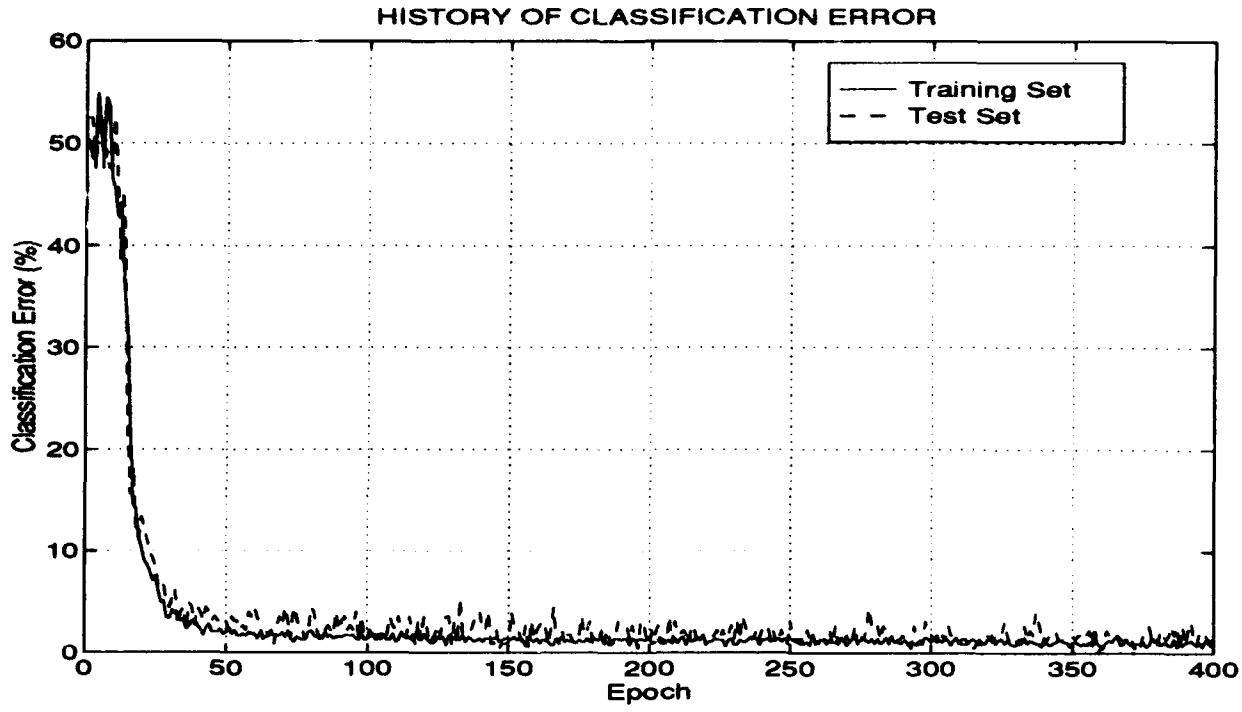




Figure 24. Network Structure (4,10,2,0.2,0) Absolute Error-Last 100 Epochs



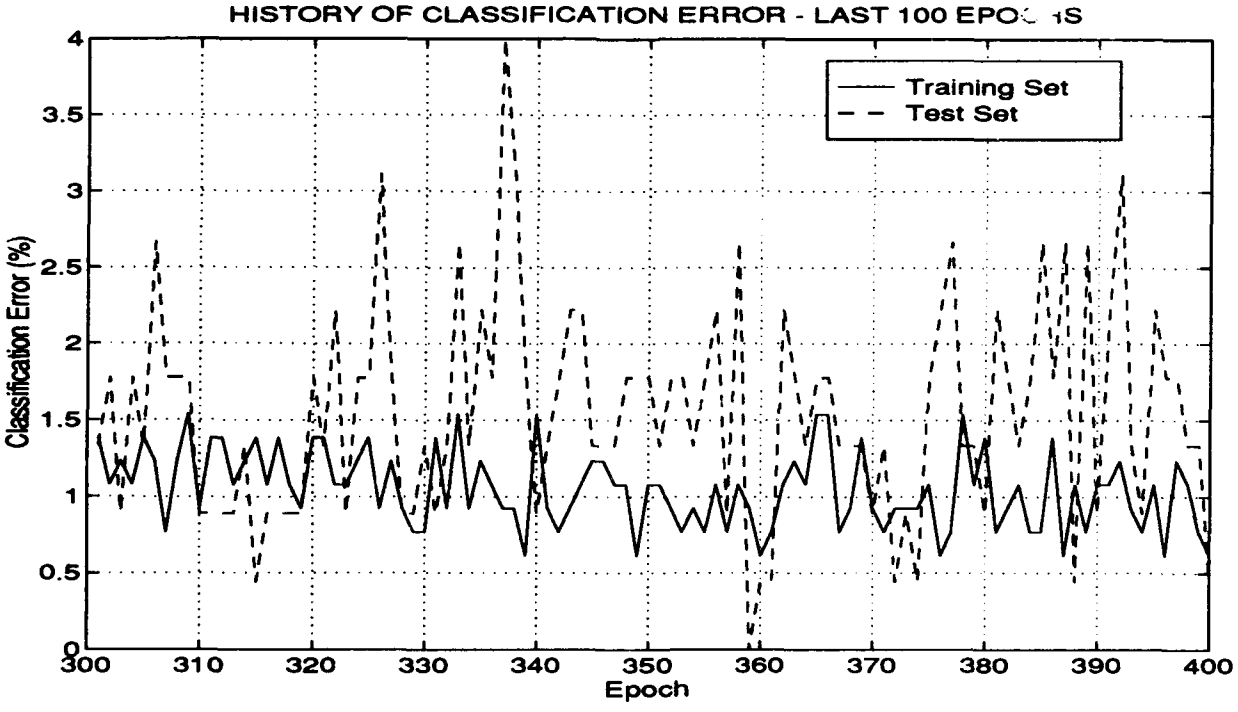
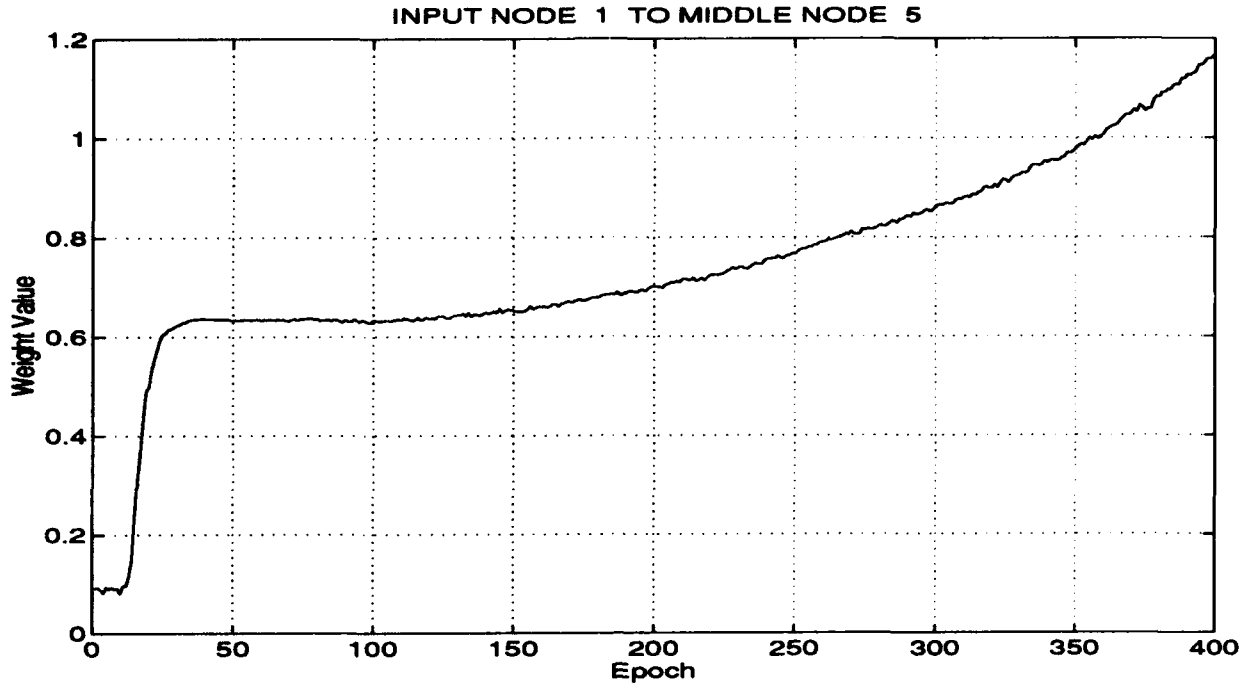




Figure 25. Network Structure (4,10,2,0.2,0) Classification Error-Last 100 Epochs



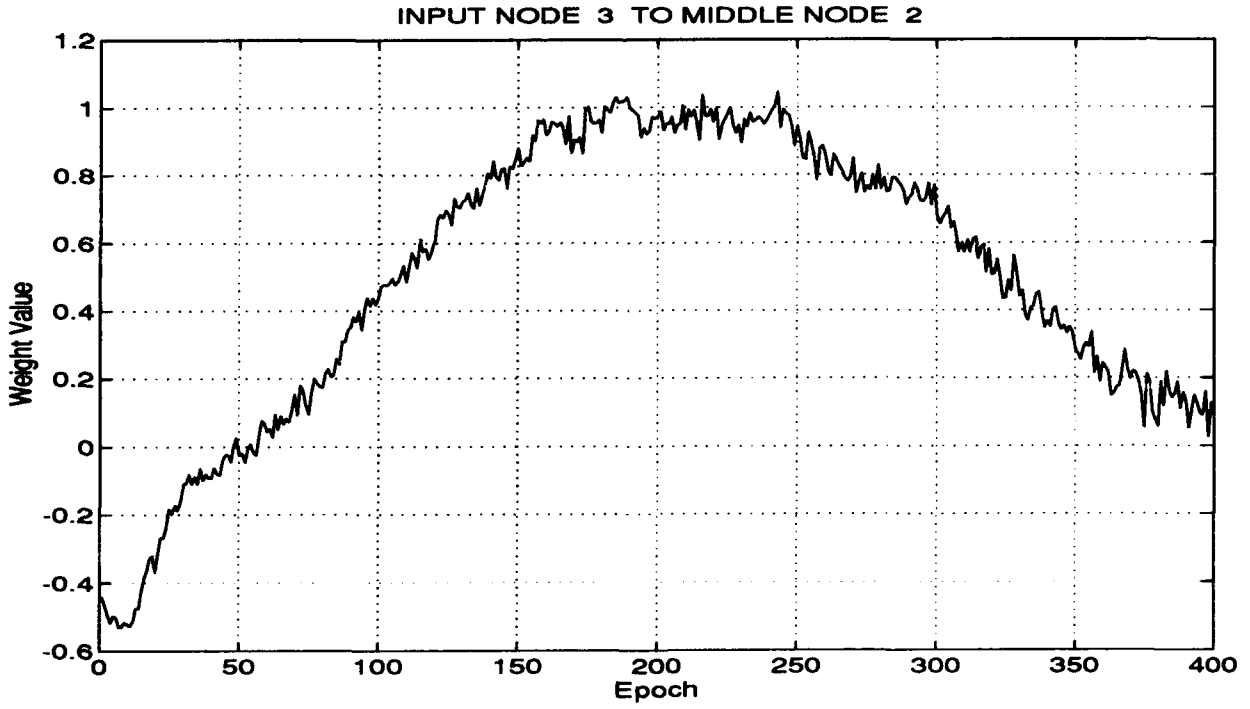




Figure 26. Network Structure (4,10,2,0.2,0) Weight Monitoring Graphs

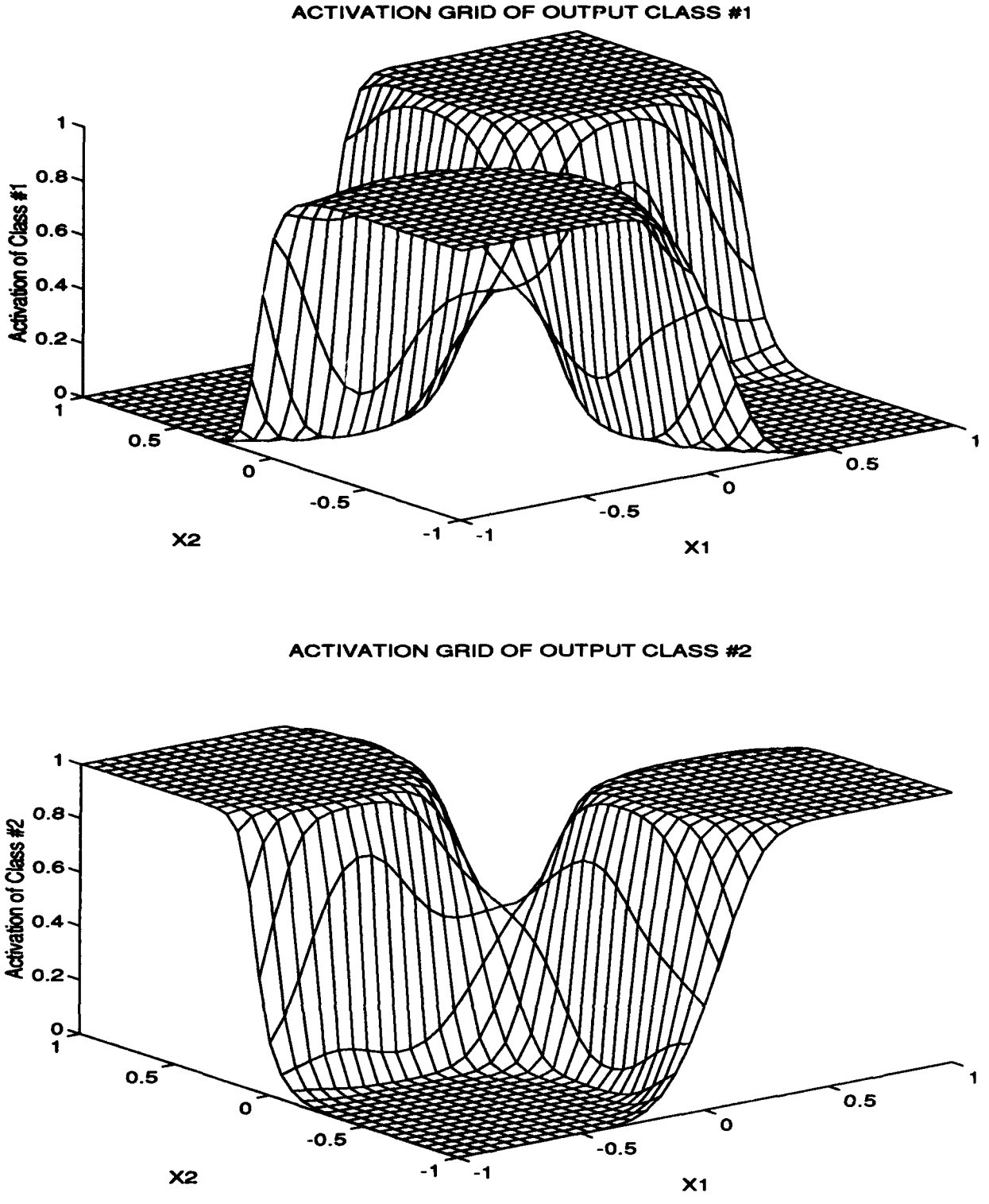



Figure 27. Network Structure (4,10,2,0.2,0) Activation Grids

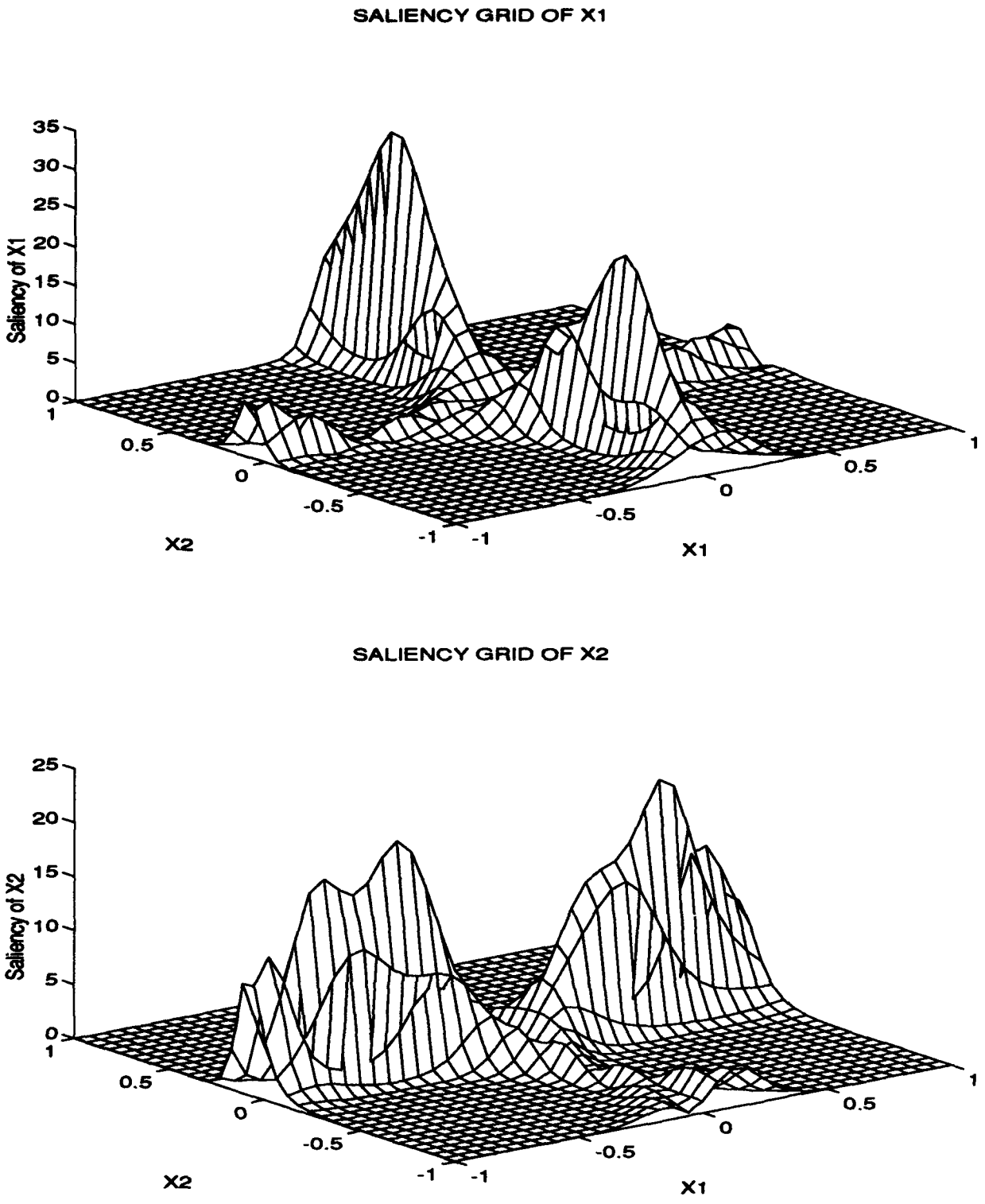



Figure 28. Network Structure (4,10,2,0.2,0) Saliency Grids

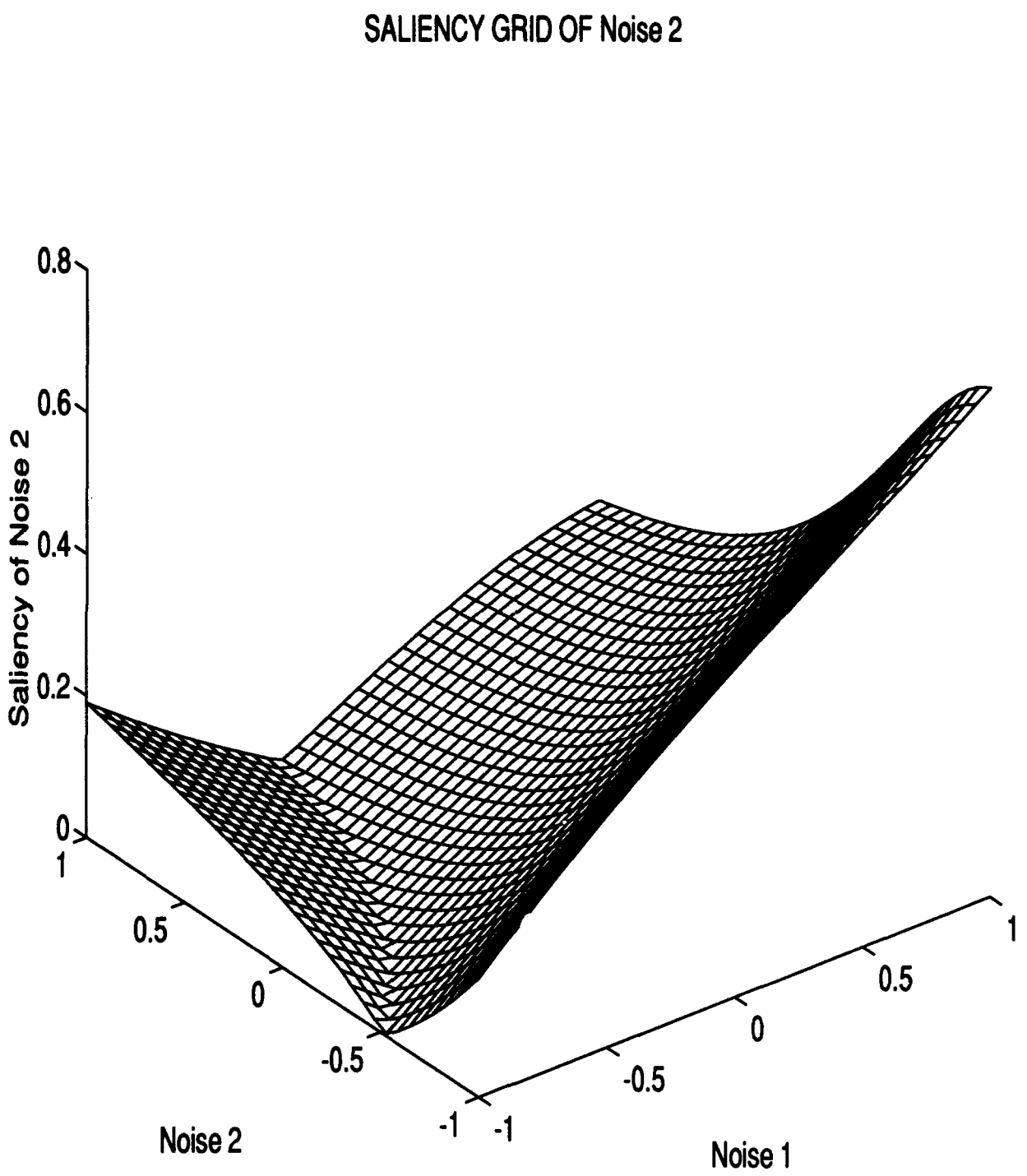



Figure 29. Network Structure (4,10,2,0.2,0) Noise Saliency Grid

```
----------------------------------
|          TRAINING SET          |
----------------------------------
CLASSIFICATION ERROR (%):   0.6154
AVERAGE ABSOLUTE ERROR  :   0.0388
----------------------------------
                ----------------------------
                | NETWORK  CLASSIFICATION  |
                ----------------------------
                |CLASS  1|CLASS  2|  TOTAL |
---------------+--------+--------+---------
|      |CLASS  1|    334 |      2 |     336|
|  T   |        |  99.40%|   0.60%|        |
|  R   |--------+--------+--------+--------|
|  U   |CLASS  2|      2 |    312 |     314|
|  E   |        |   0.64%|  99.36%|        |
|      |--------+--------+--------+--------|
|      |  TOTAL |     336|     314|     650|
-------------------------------------------
*******************************************
----------------------------------
|           TEST SET             |
----------------------------------
CLASSIFICATION ERROR (%):   0.4444
AVERAGE ABSOLUTE ERROR  :   0.0444
----------------------------------
                ----------------------------
                | NETWORK  CLASSIFICATION  |
                ----------------------------
                |CLASS  1|CLASS  2|  TOTAL |
---------------+--------+--------+---------
|      |CLASS  1|    106 |      1 |     107|
|  T   |        |  99.07%|   0.93%|        |
|  R   |--------+--------+--------+--------|
|  U   |CLASS  2|      0 |    118 |     118|
|  E   |        |   0.00%| 100.00%|        |
|      |--------+--------+--------+--------|
|      |  TOTAL |     106|     119|     225|
-------------------------------------------
```

Figure 30. Network Structure (4,10,2,0.2,0) Train/Test Confusion Matrices

**********
******* RUCK'S SALIENCY *******

| *** WITHOUT PSEUDO-SAMPLING **** | | | ***** WITH PSEUDO-SAMPLING ***** | | |
|---|---|---|---|---|---|
| FEATURE | SALIENCY | STD DEV | FEATURE | SALIENCY | STD DEV |
| 1 | 0.5284 | 2.1787 | 1 | 0.4668 | 2.0512 |
| 2 | 0.4291 | 1.6221 | 2 | 0.3879 | 1.6033 |
| 3 | 0.0194 | 0.0787 | 3 | 0.0178 | 0.0796 |
| 4 | 0.0169 | 0.0625 | 4 | 0.0168 | 0.0690 |
| Constant | 0.0926 | 0.3071 | Constant | 0.0962 | 0.3495 |

**********
*** TARR'S WEIGHT SALIENCY ****
**********

| FEATURE | TARR1 SAL | TARR2 SAL | TARR3 SAL | TARR4 SAL |
|---|---|---|---|---|
| 1 | 327.48 | 18.10 | 51.57 | 8.06 |
| 2 | 309.68 | 17.60 | 49.05 | 8.67 |
| 3 | 7.46 | 2.73 | 6.93 | 1.60 |
| 4 | 4.58 | 2.14 | 4.33 | 1.92 |
| Constant | 112.85 | 10.62 | 32.30 | 4.45 |

Figure 31. Network Structure (4,10,2,0.2,0) Ruck/Tarr Saliencies

```
---------------------------------
|         VALIDATION SET         |
---------------------------------
CLASSIFICATION ERROR (%):   0.8000
AVERAGE ABSOLUTE ERROR  :   0.0425
---------------------------------

                ---------------------------
                | NETWORK  CLASSIFICATION  |
                ---------------------------
                |CLASS  1|CLASS  2|  TOTAL |
---------------+--------+--------+---------
|      |CLASS  1|     67 |      0 |      67|
|  T   |        | 100.00%|   0.00%|        |
|  R   |--------+--------+--------+---------
|  U   |CLASS  2|      1 |     57 |      58|
|  E   |        |   1.72%|  98.28%|        |
|      |--------+--------+--------+---------
|      |  TOTAL |      68|      57|     125|
-------------------------------------------

************************************************
**** CORRELATION OF OUTPUT 1 WITH J*K INPUT *****
************************************************
-----------------------------------------------
FEATURE      1        2        3        4
-----------------------------------------------
   1     -0.08155  0.77129  0.03223 -0.07564
   2      0.77129 -0.03472 -0.00103 -0.01085
   3      0.03223 -0.00103 -0.02123  0.00846
   4     -0.07564 -0.01085  0.00846 -0.00823

**************************************      ******
**** CORRELATION OF OUTPUT 2 WITH J*K       *****
**************************************      ******
-----------------------------------------------
FEATURE      1        2        3        4
-----------------------------------------------
   1      0.08155 -0.77133 -0.03211  0.07599
   2     -0.77133  0.03526  0.00136  0.01149
   3     -0.03211  0.00136  0.02114 -0.00863
   4      0.07599  0.01149 -0.00863  0.00836
```

Figure 32. Network Structure (4,10,2,0.2,0) Correlation Matrices

### *4.2 Mesh Problem*

The second classification problem used to test the system is the four class mesh problem. Figure 33 illustrates this problem. Once again, we see that the regions $\theta_1$, $\theta_2$, $\theta_3$, and $\theta_4$ are not linearly separable. Instead of developing this problem iteration by iteration, as we did in the XOR problem, we will simply show the results from the final "optimal" trained network.

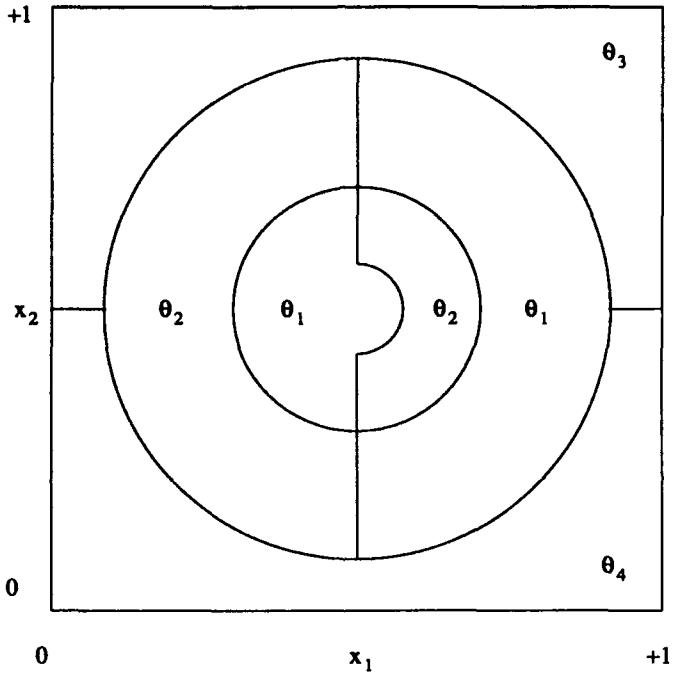



Figure 33. The Four Class MESH Problem

*4.2.1 Mesh Network Structure (2,25,4, 0.3, 0.2).* After numerous combinations of network parameters were analyzed, the $(2, 25, 4, 0.3, 0.2)$ network structure was deemed to be adequate for purposes of this validation. All outputs were produced automatically by creating a single parameter file.

Figure 34 shows a "History of Average Absolute Error" and a "History of Classification Error". The researcher has made a decision that the *entire training cycle* is evident after 1300 epochs and stops training the network at this point. The average absolute error has leveled off at around 0.04 for the training set, and 0.05 for the test set. Classification error has leveled off at 5% for the training set and 9% for the test set. Note that the

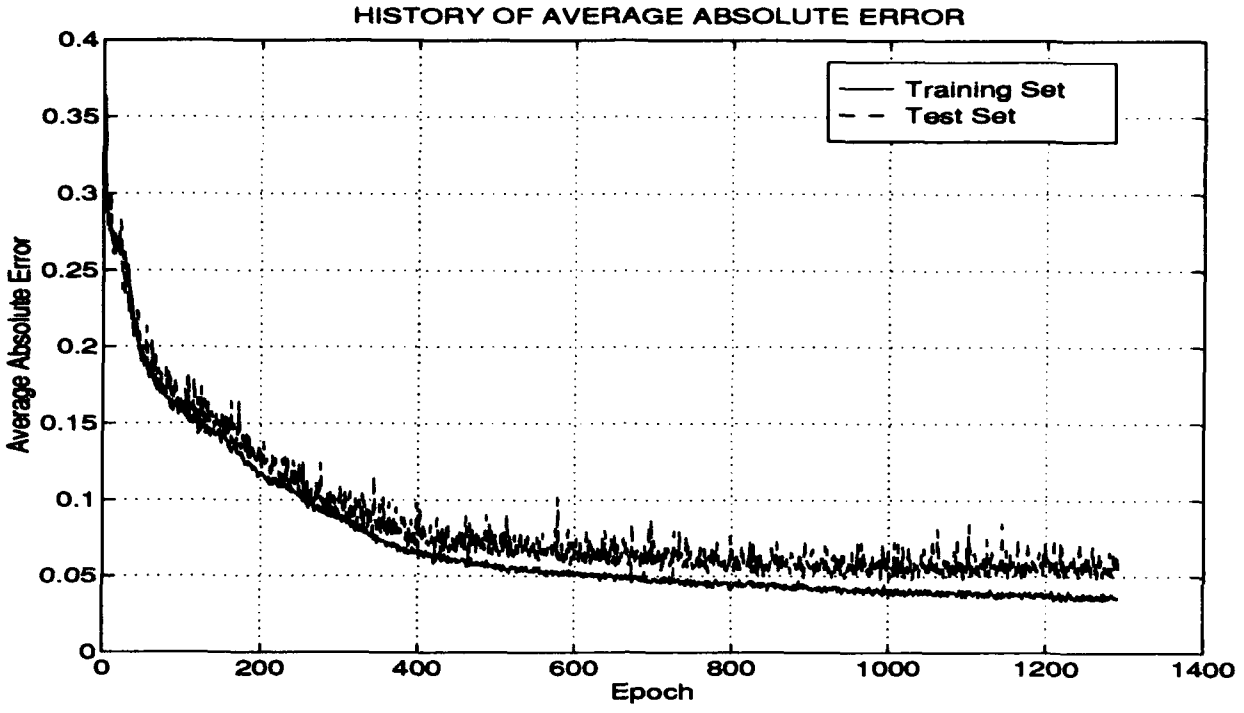
training set curve and test set curve are beginning to diverge from each other at around epoch 800. This may be an indication that over-training is beginning to occur. To test this divergence hypothesis, the researcher could simply instruct the program to continue training.

Figures 35 and 36 show the average absolute error, classification error, and confusion matrices for the training and test sets at epoch 1300. The training set had an average absolute error of 0.04 and a classification error of 4.4%, while the test set had an average absolute error of 0.05 and a classification error of 6.0%. Looking at the off-diagonal elements of these two confusion matrices shows that the network is having the most trouble distinguishing between class $\theta_1$ and $\theta_2$. Figure 38 shows Ruck's saliency metric with and without pseudo-sampling, as well as all four variants of Tarr's saliency. Recall that, the higher the value of a saliency metric for a particular variable, the higher its ranking as a significant variable. Ruck's saliency for features 1 and 2 indicates that each feature has approximately the same significance. Tarr's saliency metrics also indicate that features 1 and 2 are of equal significance.

Figures 40 and 41 show the 3-dimensional view of the trained network at epoch 1300 for the (2,25,4, 0.3 ,0.2) network structure. These four 3-dimensional plots compare quite favorably to Figure 33, except for the notch in the middle of class $\theta_1$. This may be due to the fact that there are very few exemplars (data points) in the notched region.

The 3-dimensional graphs of Figure 42 show the saliency of $x_1$ and $x_2$ respectively. Recall that the saliency of $x_1$ is just the change in network output with respect to a change in $x_1$ network input. This implies there is a classification change along high saliency values. In effect, we are drawing the borders of the four classes $\theta_1$ through $\theta_4$. These saliency plots also compare favorably to Figure 33.

In Figure 39, the second-order correlation matrix for each output class is listed. A possible significant element occurs at the $(2,2)$ position for outputs 3 and 4. These values of .69579 and $-.55727$ indicate that an $x_2^2$ term may have significant explanatory power for classifying the exemplars into class 3 or class 4.



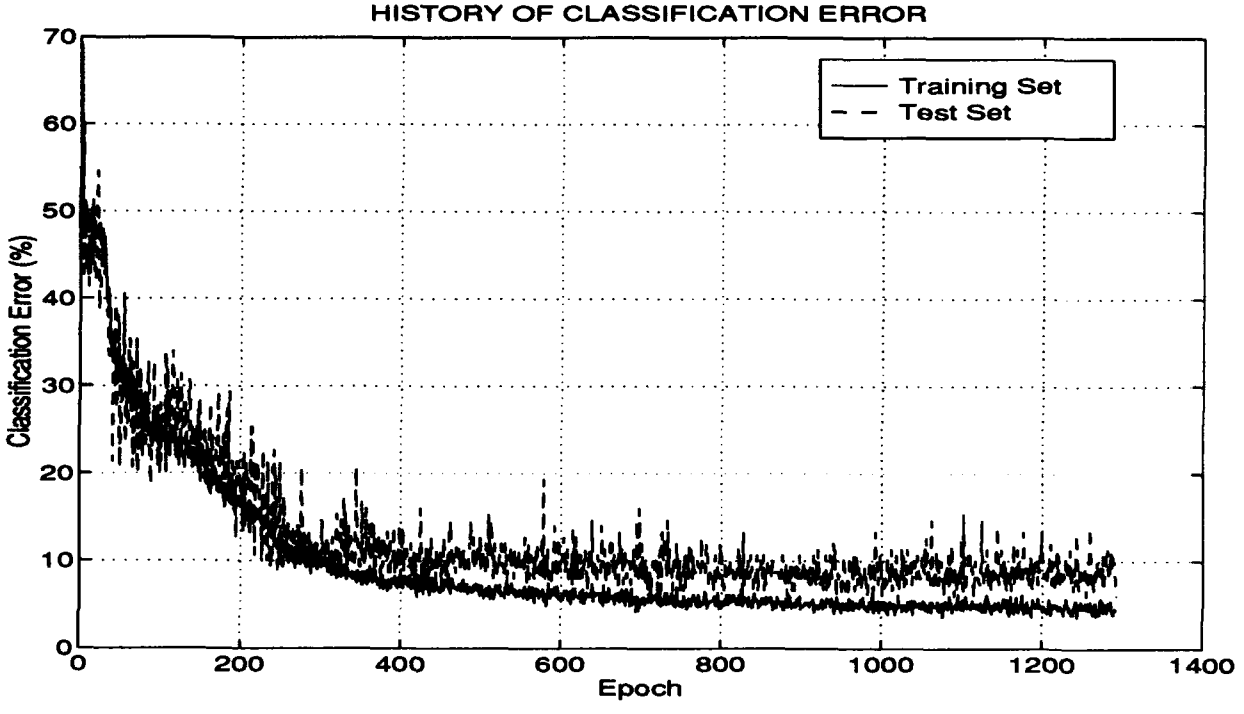




Figure 34. Network Structure (2,25,4,0.3,0.2) Absolute/Classification Error

| TRAINING SET |
|---|

CLASSIFICATION ERROR (%): 4.3750
AVERAGE ABSOLUTE ERROR : 0.0355

| | | NETWORK CLASSIFICATION | | | | |
|---|---|---|---|---|---|---|
| | | CLASS 1 | CLASS 2 | CLASS 3 | CLASS 4 | TOTAL |
| T R U E | CLASS 1 | 193<br>93.24% | 9<br>4.35% | 3<br>1.45% | 2<br>0.97% | 207 |
| | CLASS 2 | 10<br>4.69% | 201<br>94.37% | 1<br>0.47% | 1<br>0.47% | 213 |
| | CLASS 3 | 6<br>3.31% | 1<br>0.55% | 174<br>96.13% | 0<br>0.00% | 181 |
| | CLASS 4 | 1<br>0.50% | 1<br>0.50% | 0<br>0.00% | 197<br>98.99% | 199 |
| | TOTAL | 210 | 212 | 178 | 200 | 800 |

Figure 35. Network Structure (2,25,4,0.3,0.2) Confusion Matrix for Training Set

| TEST SET |
|---|
| CLASSIFICATION ERROR (%): 6.0000 |
| AVERAGE ABSOLUTE ERROR : 0.0502 |

| | | NETWORK CLASSIFICATION | | | | |
|---|---|---|---|---|---|---|
| | | CLASS 1 | CLASS 2 | CLASS 3 | CLASS 4 | TOTAL |
| TRUE | CLASS 1 | 32<br>80.00% | 7<br>17.50% | 0<br>0.00% | 1<br>2.50% | 40 |
| | CLASS 2 | 0<br>0.00% | 50<br>100.00% | 0<br>0.00% | 0<br>0.00% | 50 |
| | CLASS 3 | 0<br>0.00% | 0<br>0.00% | 29<br>96.67% | 1<br>3.33% | 30 |
| | CLASS 4 | 0<br>0.00% | 0<br>0.00% | 0<br>0.00% | 30<br>100.00% | 30 |
| | TOTAL | 32 | 57 | 29 | 32 | 150 |

Figure 36. Network Structure (2,25,4,0.3,0.2) Confusion Matrix for Test Set

| VALIDATION SET |
|---|

CLASSIFICATION ERROR (%): 8.0000
AVERAGE ABSOLUTE ERROR : 0.0512

| | | NETWORK CLASSIFICATION | | | | |
|---|---|---|---|---|---|---|
| | | CLASS 1 | CLASS 2 | CLASS 3 | CLASS 4 | TOTAL |
| TRUE | CLASS 1 | 8<br>88.89% | 1<br>11.11% | 0<br>0.00% | 0<br>0.00% | 9 |
| | CLASS 2 | 0<br>0.00% | 18<br>100.00% | 0<br>0.00% | 0<br>0.00% | 18 |
| | CLASS 3 | 1<br>6.67% | 2<br>13.33% | 12<br>80.00% | 0<br>0.00% | 15 |
| | CLASS 4 | 0<br>0.00% | 0<br>0.00% | 0<br>0.00% | 8<br>100.00% | 8 |
| | TOTAL | 9 | 21 | 12 | 8 | 50 |

Figure 37. Network Structure (2,25,4,0.3,0.2) Confusion Matrix for Validation Set

**********
******* RUCK'S SALIENCY *******

| *** WITHOUT PSEUDO-SAMPLING **** | | | ***** WITH PSEUDO-SAMPLING ***** | | |
|---|---|---|---|---|---|
| FEATURE | SALIENCY | STD DEV | FEATURE | SALIENCY | STD DEV |
| 1 | 1.3509 | 6.4056 | 1 | 1.0862 | 5.9796 |
| 2 | 1.2705 | 5.9786 | 2 | 1.0130 | 5.4914 |
| Constant | 1.1548 | 5.9985 | Constant | 0.9029 | 5.2212 |

**********
*** TARR'S WEIGHT SALIENCY ****
**********

| FEATURE | TARR1 SAL | TARR2 SAL | TARR3 SAL | TARR4 SAL |
|---|---|---|---|---|
| 1 | 8880.52 | 94.24 | 332.00 | 49.43 |
| 2 | 6106.20 | 78.14 | 272.71 | 29.52 |
| Constant | 5109.10 | 71.48 | 253.21 | 36.92 |

Figure 38. Network Structure (2,25,4,0.3,0.2) Ruck/Tarr Saliencies

```
************************************************
**** CORRELATION OF OUTPUT 1 WITH J*K INPUT *****
************************************************
--------------------------
FEATURE      1        2
--------------------------
   1      0.12846  0.17137
   2      0.17137 -0.03667

************************************************
**** CORRELATION OF OUTPUT 2 WITH J*K INPUT *****
************************************************
--------------------------
FEATURE      1        2
--------------------------
   1     -0.32094 -0.12129
   2     -0.12129 -0.05753

************************************************
**** CORRELATION OF OUTPUT 3 WITH J*K INPUT *****
************************************************
--------------------------
FEATURE      1        2
--------------------------
   1      0.10671  0.43324
   2      0.43324  0.69579

************************************************
**** CORRELATION OF OUTPUT 4 WITH J*K INPUT *****
************************************************
--------------------------
FEATURE      1        2
--------------------------
   1      0.07635 -0.45934
   2     -0.45934 -0.55727
```

Figure 39. Network Structure (2,25,4,0.3,0.2) Correlation Matrices

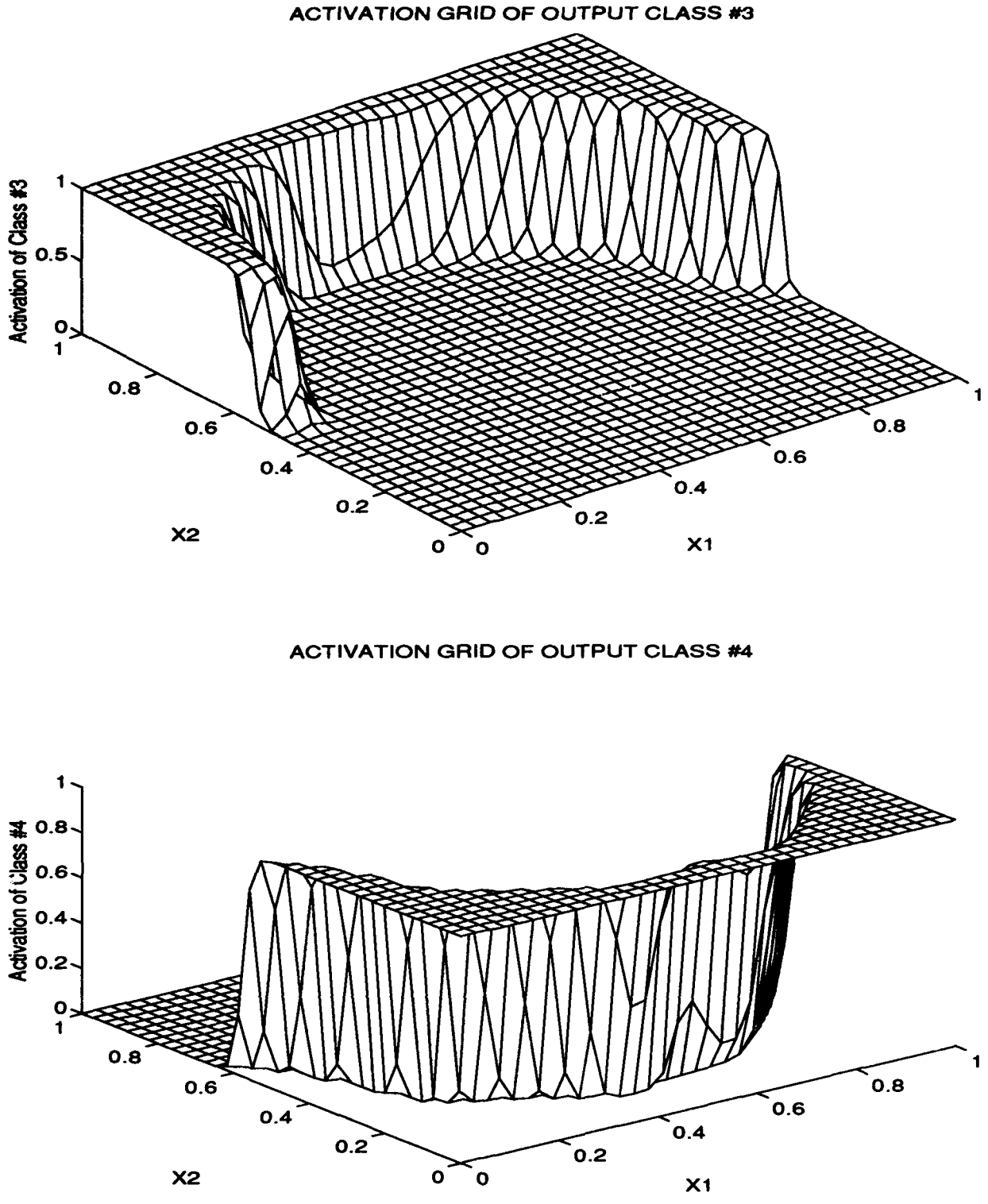



Figure 40. Network Structure (2,25,4,0.3,0.2) Activation Grids-Class 3 and 4

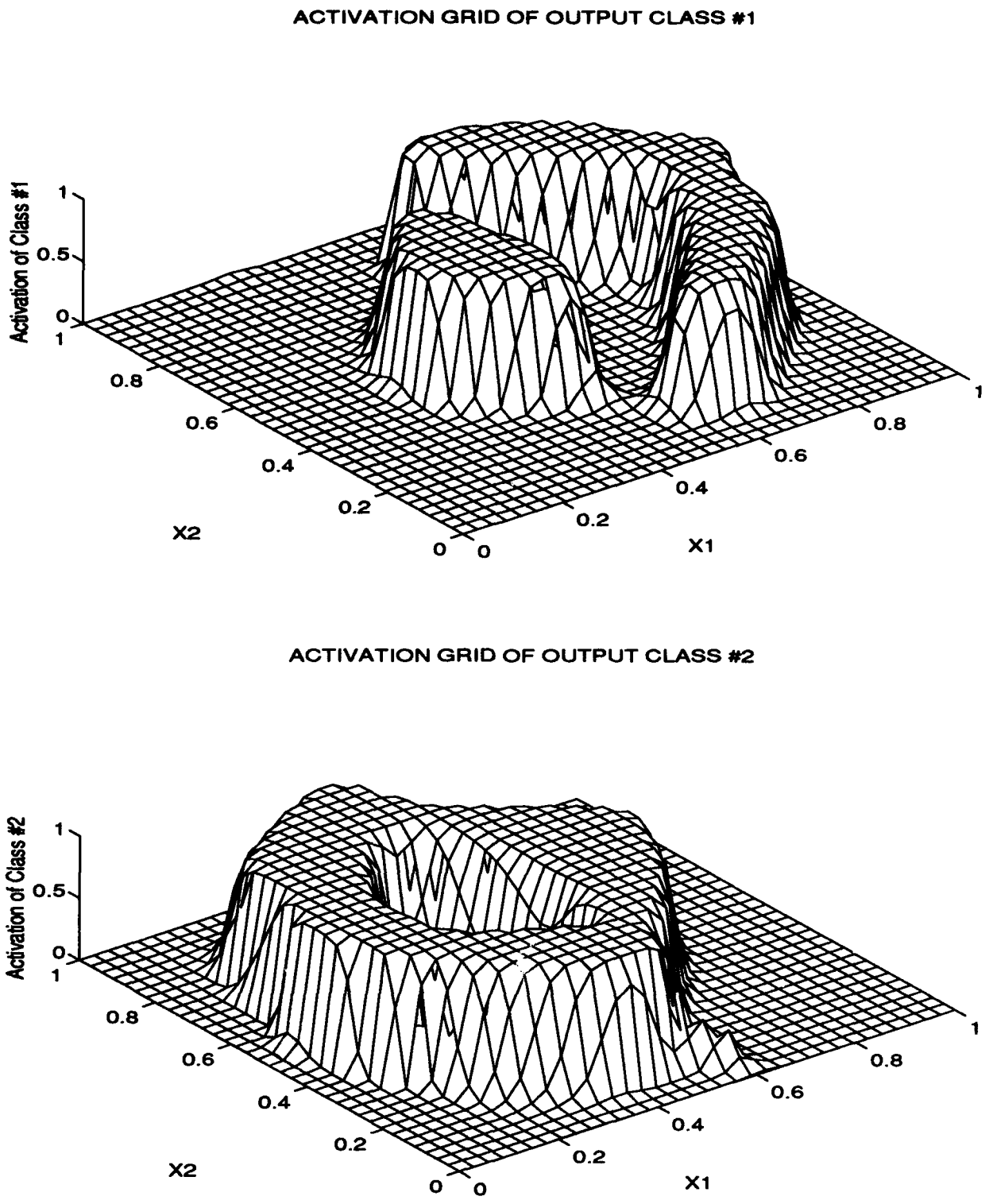
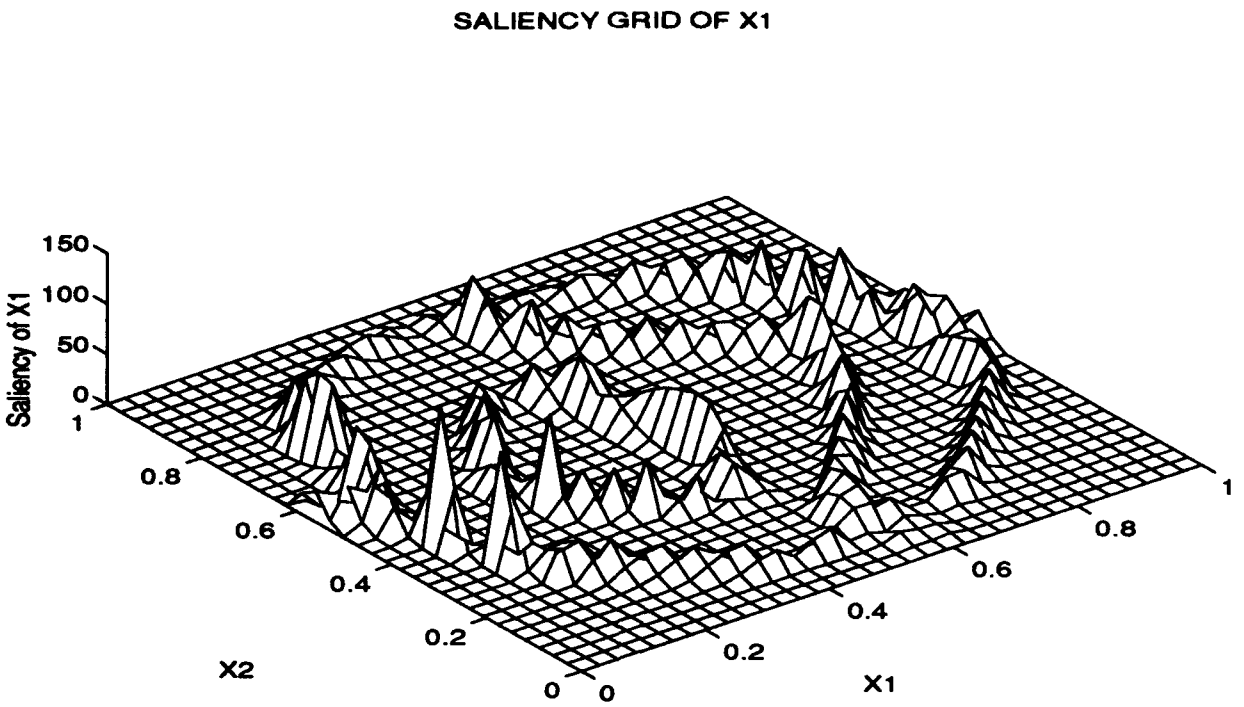



Figure 41. Network Structure (2,25,4,0.3,0.2) Activation Grids-Class 1 and 2



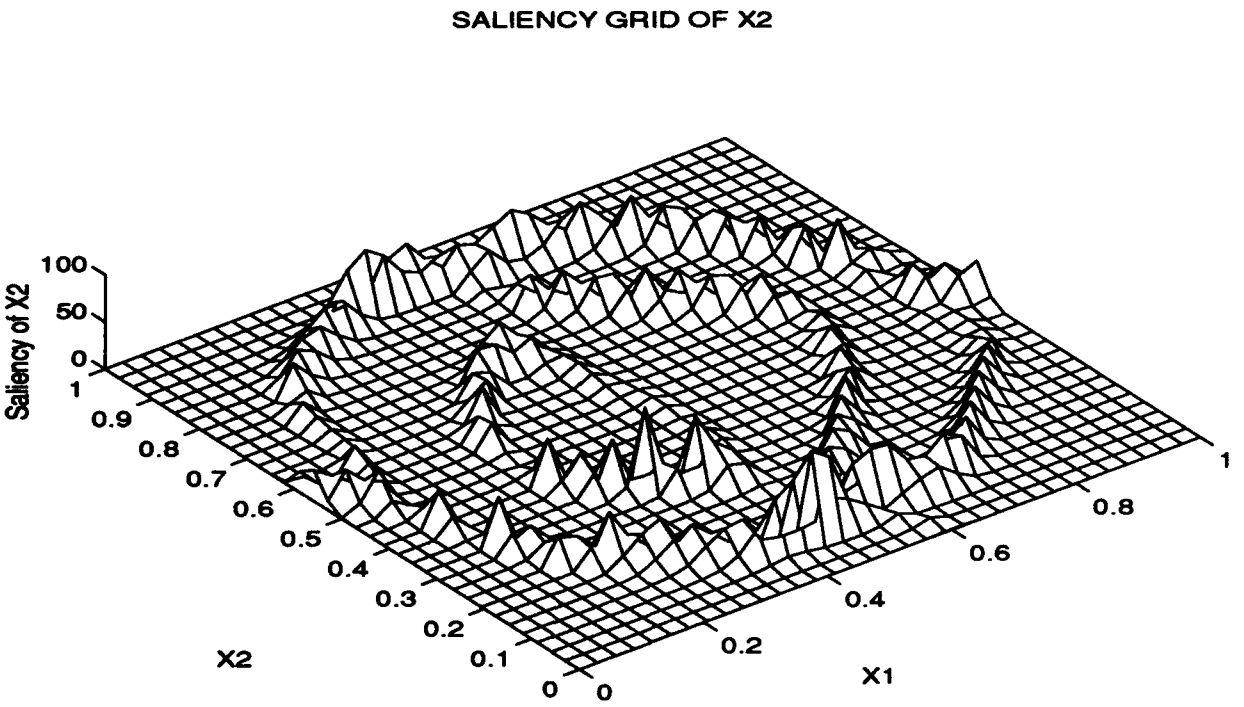




Figure 42. Network Structure (2,25,4,0.3,0.2) Saliency Grids

# V. Final Results and Recommendations

The following results and recommendations are drawn from the work involved in this thesis.

## 5.1 Final Results

- The interactive computer system developed in this thesis provides an excellent platform for determining the optimal structure of a multilayer perceptron. By following a predefined procedure, the researcher can start with a set of raw data and build a neural network structure that minimizes average absolute error and classification error. This final network structure produces a set of weights which can be used to predict the classification of future exemplars.
- The interactive capabilities of this system allow the user to quickly develop and fine-tune an "optimal" neural network structure. By allowing the user to interrupt network training at any time to see the "status" of the system, and then continue training if desired, saves countless hours of computer and research time. This "hot start" prevents valuable training information from being lost and allows the optimal network structure to be developed in a minimum amount of time.
- Error graphs produced by the system aid the user in determining when the training cycle has stopped. This helps prevent over-training and under-training of the network. This is the primary tool used to get the researcher through the flowchart of Figure 16.
- Changing network structure, data, and graph parameters is as easy as editing a single parameter file.
- Six types of variable learning rates are incorporated into the program. All six functions decrease the learning rate as the number of epochs increases. This allows the training process to take large steps in its gradient search at the beginning of the process and small steps toward the end of the process.
- The 3-dimensional graphs of class activation and saliency produce pictures never seen before. These images allow the user to visualize where the class boundaries lie

in a particular problem and the shape of each class. By interfacing the FORTRAN code with the MATLAB Graphics Package, this system allows the user to rotate the 3-dimensional object to any angle or azimuth they desire.

- Producing 3-dimensional saliency graphs of known noise variables showed that these graphs will resemble a plane. Remember to be wary of the $z$-axis scale.
- While programming the FORTRAN code, it was discoverd that Ruck's saliency formula was incorrect [17:37]. This led to the development of Equation 22 and its subsequent notation.
- The Shell-Mezgar sort was incorporated into the FORTRAN code to re-shuffle the training vectors at the beginning of each epoch. With epochs numbering in the hundreds or even thousands, this efficient sort helped speed up processing time immensely.
- The Shell-Mezgar sort also allowed us to reduce the number of random deviates generated by the random number generator by a factor ranging from 3 to 11.
- In the original FORTRAN code, three arrays had to be eliminated due to their excessive size. Although it costs more computing time, the elimination of these three arrays allows the system to tackle larger problems. These arrays were involved in computing the covariance matrix and the saliency metric.

## *5.2* Recommendations

In Chapter 4 we demonstrated the ability of this system to build an optimal network structure for the XOR and Four Class Mesh classification problems. In each case the system came up with an optimal network structure. The 3-dimensional pictures of the activations and saliencies tracked very closely to what was expected. Therefore, it is highly recommended that this system be used on a real-world classification problem.

In her thesis, Belue suggests that further study of Armor Piercing Incendiaries (API) is necessary to ensure that the optimal network structure was discovered and that the error rates cited are corr ct [1:120]. In her study, the problem was treated as a two-class problem;

complete burn or other than complete burn. In reality, there are six classifications of API projectile firings. This is an ideal classification problem for this system to analyze.

Another prime candidate would be a target recognition problem. In his dissertation, Ruck deals with the problem of identifying targets from non-targets in forward looking infrared (FLIR) images [17:41]. The target classes consist of tanks, trucks and armored personnel carriers. A set of nine input features is used.

## *Appendix A. User's Manual for Running the Program*

This appendix will provide guidance for running the FORTRAN and MATLAB programs. All MATLAB and FORTRAN interfaces are automatic. The only files the user must provide is the raw exemplar data file, and the parameter file which defines the network structure, stopping criteria, and the graphs desired.

### *A.1 Raw Exemplar Data File Format*

The raw exemplar data file "must" contain one exemplar per *logical* line. For example, a problem which has 3 features and 4 classes would have input lines that look like this:

0.5673 0.3218 2.987076 0 1 0 0 < *Return* >

or

1.2978 32 58.7 0.0 1.000 0.00 0. < *Return* >

The numbers do not have to follow any specific format, or be put in specific columns.

### *A.2 Parameter File*

All discussion that follows, refers to the example parameter file found on the last page of this appendix. The program reads the first number or word of each line only. All comments that follow the numbers are ignored. Parameters must appear in the order shown. A description of each parameter follows.

- **Name of Raw Exemplar Data File.** Exemplars from this file will be used to create the training, test and validation sets. Including the ".", a maximum of 12 characters is allowed. In the example parameter file the name of the data set is "shotsin.dat".
- **Stopping Criteria.** If this number is $\geq 1$ then it represents the number of epochs the system will train before it stops. If this number is $< 1$ then it represents a tolerance. If the average absolute error of the training set or test set falls below this tolerance the system will stop training. The user may override this criteria while

the system is running by interrupting the system and telling it to stop training. No matter how the system is stopped, all reports and requested graphs will be produced as of the last complete epoch calculated. Currently, there is an upper limit of 10,000 epochs. If the user requires more than 10,000 epochs, the parameter NNE in the FORTRAN program will have to be changed and the program recompiled. In the example parameter file we have chosen a stopping criteria of 1328 epochs.

- **Number of Training Vectors.** This is the number of exemplars which will be randomly selected from "shotsin.dat" and put into the training set. In the example we have selected 200 training vectors. Currently, there is an upper limit of 3000 training vectors. If the user requires more than 3000 training vectors, the parameter NTRAIN in the FORTRAN program will have to be changed and the program recompiled.

- **Number of Test Vectors.** This is the number of exemplars which will be randomly selected from "shotsin.dat" and put into the test set. In the example we have selected 41 test vectors. Currently, there is an upper limit of 2000 test vectors. If the user requires more than 2000 test vectors, the parameter NTEST in the FORTRAN program will have to be changed and the program recompiled.

- **Number of Validation Vectors.** This is the number of exemplars which will be randomly selected from "shotsin.dat" and put into the validation set. In the example we have selected 40 validation vectors. Currently, there is an upper limit of 1000 validation vectors. If the user requires more than 1000 validation vectors, the parameter NVALID in the FORTRAN program will have to be changed and the program recompiled.

- **Number of Input Nodes.** This is the number of input nodes in the input layer. Also equal to the number of input features in the discrimination problem. In the example we have 4 input features. Currently, there is an upper limit of 100 input features. If the user requires more than 100 input features, the parameter NNVAR in the FORTRAN program will have to be changed and the program recompiled.

- **Number of Middle Nodes.** This is the number of middle nodes in the hidden layer. In the example we have 20 middle nodes. Currently, there is an upper limit of 200

middle nodes. If the user requires more than 200 middle nodes, the parameter NNM in the FORTRAN program will have to be changed and the program recompiled.

- **Number of Output Nodes.** This is the number of output nodes in the output layer. Also equal to the number of output classes in the discrimination problem. In the example we have 2 output classes. Currently, there is an upper limit of 10 output classes. If the user requires more than 10 output classes, the parameter NNO in the FORTRAN program will have to be changed and the program recompiled.

- **Type of Learning Rate.** Can take on an integer value from 1 thru 6. This value determines the type of learning rate calculated according to the following table:

$$\begin{array}{lcll} 1 & \Rightarrow & \eta & Constant\ \ Update \\ 2 & \Rightarrow & \eta\,[1-\frac{LL}{NE+1}] & Linear\ \ Update \\ 3 & \Rightarrow & \frac{1}{\ln(LL+1)} & Log\ \ Update \\ 4 & \Rightarrow & \frac{1-\frac{LL}{NE+1}}{\ln(LL+1)} & Log-Linear\ \ Update \\ 5 & \Rightarrow & \frac{1}{\ln(\sqrt{LL}+1)} & Log-Sqrt\ \ Update \\ 6 & \Rightarrow & \frac{1-\frac{LL}{NE+1}}{\ln(\sqrt{LL}+1)} & Log-Sqrt-Linear\ \ Update \end{array} \qquad (32)$$

where $\eta$ is the constant given in the next parameter, $LL$ is the current epoch, and $NE$ is the total number of epochs expected to be run. The variable $NE$ is set equal to the number of epochs given in the stopping criteria above. If the stopping criteria is $< 1$ (a tolerance), $NE$ is set equal to $NNE$. In the example, the type of learning rate is 1 which implies the constant update is used.

- **Learning Rate.** This is the value of $\eta$ in the formulas above. If the type of learning rate above is $3, 4, 5,$ or 6, this value is ignored. In our example, the learning rate is set to 0.2 and is used since type of learning rate is 1.

- **Momentum Rate.** In our example, the momentum rate is set to 0.0. No momentum rate is desired.

- **Range of Weight Initialization.** All weights initially used by the network will be initialized between the two given numbers. In our example, we have chosen to initialize weights between $-0.5$ and 0.5.

- **Random Number Seed.** This number seeds the random number generator. In the example, the random number seed 1234567 was chosen.
- **Type of Normalization of Data.** Can take on the integer values 0, 1, or 2. A value of 0 implies no normalization, the exemplars are not transformed. A value of 1 implies all exemplars will be normalized to values between 0 and 1 based on the range of the training set. A value of 2 implies all exemplars will be standardized according to Equation 28, based on the range of the training set. The example shows that normalization between 0 and 1 was chosen.
- **Number of Divisions for Pseudo-Sampling.** This parameter is used when calculating Ruck's saliency. This is the value of $R$ as described in Equation 22. In the example, 5 divisions for pseudo-sampling were chosen. Currently, there is an upper limit of 10 divisions for pseudo-sampling. If the user requires more than 10, the parameter NNDIV in the FORTRAN program will have to be changed and the program recompiled.
- **Constant for Activation and Saliency Grids.** When we create the 35x35 grid, we are, in effect, creating a new set of 1225 exemplars with values for the two features being graphed, only. The question arises: what values should be used in these new exemplars for the features that are not being graphed? Since we are running these 1225 exemplars through the network, each exemplar must have values for the features not being graphed. If the network has only two features, there is no problem. However, if there is more than two features, then choices must be made by the user. The program allows the user to choose a constant value for all features not graphed, or to choose the mean value of that feature, as calculated over the training set. A value of 999 tells the program to use the mean of the feature. Any other value will be used as the constant for all features not being graphed. In our example, we used 999, the average values.
- **Number of Weights to Monitor.** Indicates the number of weights to monitor and the number of weight monitoring graphs the user desires. In this example, we chose to monitor 2 different weights. Note: this parameter may be 0, but remember to delete all (FROM Node/TO Node/LAYER) 3-tuples defined below. Currently,

there is an upper limit of 5 weight monitoring graphs. If the user requires more than 5, the parameter NNUMWT in the FORTRAN program will have to be changed and the program recompiled.

- **FROM Node/TO Node/LAYER.** In the parameter above we indicated how many weight monitoring graphs we wanted. For *each* graph, we must define a 3-tuple which indicates the weight we want to monitor. In the example we used 4 1 1 and 9 2 2. The first 3-tuple is designated 4 1 1. The code 4 1 1 indicates the user is monitoring the weight connecting input node 4 to hidden node 1 of layer 1. The second 3-tuple is designated 9 2 2. The code 9 2 2 indicates the user is monitoring the weight connecting hidden node 9 to output node 2 of layer 2. You *must* have the exact number of 3-tuples as indicated by the parameter "Number of Weights to Monitor".

- **Number of Activation Grids to Plot.** Indicates the number of 3-dimensional activation grids the user desires. In this example, we chose to create 2 activation plots. Note: this parameter may be 0, but remember to delete all (FEATURE for X-Axis/FEATURE for Y-Axis/Activation Class) 3-tuples defined below. Currently, there is an upper limit of 5 activation grids. If the user requires more than 5, the parameter NNGRID in the FORTRAN program will have to be changed and the program recompiled.

- **FEATURE for X-Axis/FEATURE for Y-Axis/Activation Class.** In the parameter above we indicated how many activation grids we wanted. For *each* grid, we must define a 3-tuple which indicates the feature we want on the $x$-axis, the feature we want on the $y$-axis and the activation class we want on the $z$-axis. In the example we used 3 4 1. This 3-tuple tells the system to create a 3-dimensional grid with feature 3 on the $x$-axis, feature 4 on the $y$-axis, and output class 1 on the $z$-axis. In addition, the words "Mass" and "Secant" indicate the labels which will be put on the $x$-axis and $y$-axis respectively. These labels may be up to 12 characters long. Once again, you *must* have the exact number of 3-tuples and label sets as indicated by the parameter "Number of Activation Grids to Plot". The second 3-tuple (1 4 1), and label set (Ply, Secant) define the second activation grid to be plotted.

- **Number of Saliency Grids to Plot.** Indicates the number of 3-dimensional saliency grids the user desires. In this example, we chose to create 2 saliency plots. Note: this parameter may be 0, but remember to delete all (FEATURE for X-Axis/FEATURE for Y-Axis/Feature for Saliency) 3-tuples defined below. Currently, there is an upper limit of 5 saliency grids. If the user requires more than 5, the parameter NNGSAL in the FORTRAN program will have to be changed and the program recompiled.

- **FEATURE for X-Axis/FEATURE for Y-Axis/Feature for Saliency.** In the parameter above we indicated how many saliency grids we wanted. For *each* grid, we must define a 3-tuple which indicates the feature we want on the $x$-axis, the feature we want on the $y$-axis and the saliency feature we want on the $z$-axis. In the example we used 3 4 3. This 3-tuple tells the system to create a 3-dimensional grid with feature 3 on the $x$-axis, feature 4 on the $y$-axis, and the saliency of feature 3 on the $z$-axis. In addition, the words "Mass", "Secant", "Mass" indicate the labels which will be put on the $x$-axis, $y$-axis, and $z$-axis respectively. These labels may be up to 12 characters long. Once again, you *must* have the exact number of 3-tuples and label sets as indicated by the parameter "Number of Saliency Grids to Plot". The second 3-tuple (3 4 4), and label set (Mass, Secant, Secant) define the second saliency grid to be plotted.

- **END.** The last parameter in the parameter file must be the word "END". If it is omitted, or there are an improper number of parameters, the program will give you the following error message: "ERROR IN PARAMETERS". The most probable cause is that the user indicated a certain number of graphs to be created, but forgot to define the 3-tuples or labels to put on the axes. It is best to compare this problem parameter file with one that you know is working.

### *A.3 Program Execution*

To start the FORTRAN program simply enter: "neural7.exe". The program will prompt you for the name of your *parameter file*. After entering your parameter file, a computer window will appear. This window shows the number of epochs that have been

completed as well as the absolute and classification errors. Note: Be sure to keep all executable files, data files, and parameter files in the same directory.

To interrupt the program and see the current status of the training process, simply enter: "status.exe". Note: This command must be entered from its own separate shelltool window or cmdtool window (i.e., do not enter "status.exe" in the same window as you entered "neural7.exe"). After you interrupt the program, error graphs and weight monitoring graphs will appear (3-D graphs do not appear until training is complete). At this time the analyst can use any MATLAB command they choose. After analyzing the graphs, the user types in "quit". This ends the MATLAB session and puts you back into FORTRAN. The user is then asked if they wish to continue training. If the user answers yes, training picks up from where it was interrupted. If the user answers no, the program jumps back into MATLAB and recreates the error graphs, the weight monitoring graphs, and creates all requested 3-D graphs. At this time all MATLAB commands are active again. See next section for some valuable MATLAB commands. When the analyst is finished viewing the various graphs and issuing MATLAB commands, they type in "quit". This ends both the MATLAB and FORTRAN programs.

### *A.4 MATLAB Commands*

When in the MATLAB portion of the program all MATLAB commands may be used. When MATLAB is first entered, all graphs are stacked up on the right hand side of the screen. Each graph is identified by a (Figure #) found at the top of each graph. The user may drag and drop these figures any where on the screen. Here are a few useful MATLAB commands:

- **figure(n).** This command brings figure number $n$ to the forefront of your screen. In addition, all subsequent MATLAB commands will apply to this figure (e.g. view).
- **view(AZ,EL).** 3-D graph viewpoint specification. The command view(AZ,EL) sets the angle of the view from which an observer sees the current 3-D plot. AZ is the azimuth or horizontal rotation and EL is the vertical elevation (both in degrees). Azimuth revolves about the z-axis, with positive values indicating counter-clockwise

rotation of the *viewpoint*. Positive values of elevation correspond to moving above the object; negative values move below.

Here are some examples:

AZ = -37.5, EL = 30 is the default 3-D view. AZ = 0, EL = 90 is directly overhead AZ = EL = 0 looks directly up the first column of the matrix. AZ = 180 is behind the matrix.

The best thing to do is play around with this command until you get your 3-D thought process calibrated.

- **print.** Prints designated figures directly to a printer or to a file. To print figure(5) directly to printer rm2202lps20 you would enter:

  print -f5 -Prm2202lps20

  To print figure(3) directly to an encapsulated postscript file named fg5.eps to be used later in LATEX you would enter:

  print -f3 -deps fg5.eps

- **help.** To find out more about the above commands and others, simply type in help and the name of the command while in MATLAB.

  For example: help view

```
shotsin.dat                    Name of EXEMPLAR dataset
1328     STOP CRITERIA: ( < 1 --> ERROR RATE) ( >= 1 --> NUMBER OF EPOCHS)
200      Number of TRAINING Vectors to use.
41       Number of TEST Vectors to use.
40       Number of VALIDATION Vectors to use.
4        Number of INPUT Nodes or FEATURES
20       Number of MIDDLE Nodes
2        Number of OUTPUT Nodes or CLASSES
1        TYPE OF LEARNING RATE (e.g. 1=Constant, 4=Log-Linear, etc...)
0.2      LEARNING RATE (Only used when TYPE OF LEARNING RATE = 1 or 2)
0.0      MOMENTUM RATE
-0.5 0.5    RANGE of Weight Initialization
1234567     SEED for Random Number Generator
1        Type of NORMALIZATION of data.
5        Number of Divisions for Pseudo-Sampling for RUCK'S SALIENCY
999      Constant for GRID and GRID Saliency Plots.  999 --> Use Feature Average
2        Number of WEIGHTS TO MONITOR During Training
4 1 1    FROM Node/TO Node/LAYER
9 2 2
2        Number of ACTIVATION GRIDS to Plot
3 4 1    # of FEATURE for X-AXIS / # of FEATURE for Y-AXIS / Activation Class
Mass
Secant
1 4 1    # of FEATURE for X-AXIS / # of FEATURE for Y-AXIS
Ply
Secant
2        Number of SALIENCY GRIDS to Plot
3 4 3    # of FEAT for X-AXIS/# of FEAT for Y-AXIS/# OF FEAT for SALIENCY
Mass
Secant
Mass
3 4 4    # of FEAT for X-AXIS/# of FEAT for Y-AXIS/# OF FEAT for SALIENCY
Mass
Secant
Secant
END
```

*Bibliography*


1. Belue, Capt Lisa M. *An Investigation of Multilayer Perceptrons for Classification.* MS thesis, AFIT/GOR/ENS/92M-06. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1992.
2. Belue, Lisa M. and Kenneth W. Bauer *Methods of Determining Input Features for Multilayer Perceptrons.* Working Paper. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1993.
3. Cybenko G. "Approximations by Superpositions of Sigmoidal Functions," *Mathematics of Controls, Signals, and Systems (1989).* Accepted for publication.
4. Defense Advanced Research Projects Agency (DARPA). *Neural Network Study* AFCEA International Press, Fairfax VI, November 1988.
5. Foley, Donald H. "Considerations of Sample and Feature Size," *IEEE Transactions on Information Theory, 18:* 618-626 (September 1972).
6. Giles, Lee C. and Tom Maxwell. "Learning Invariance, and Generalization in High-order Neural Networks," *Applied Optics, 26:* 4972-4978 (1 December 1987).
7. Hecht-Nielsen, Robert. *Neurocomputing.* New York: Addison-Wesley, 1990.
8. Hornik, K., M. Stinchcombe, and H. White. "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks, 2:* 359-366 (May 1989).
9. Knuth, Donald E. "Sorting and Searching," *The Art of Computer Programming, 3:* Reading, MA: Addison-Wesley, 1973.
10. Lippmann, Richard P. "An Introduction to Computing with Neural Nets," *IEEE Acoustics, Speech, and Signal Processing.* 4-22 (April 1987).
11. McClelland, J., and Rumelhart, D. *Explorations in Parallel Distributed Processing.* Cambridge, MA: The MIT Press, 1988.
12. Minsky, Marvin Lee and Seymour Papert. *Perceptrons* (Expanded Edition). Cambridge, MA: The MIT Press, 1988.
13. Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vettering. *Numerical Recipes [FORTRAN Version].* Cambridge University Press, 1989.
14. Rogers, Maj Steven K., Matthew Kabrisky, Dennis W. Ruck, and Gregory L. Tarr. *An Introduction to Biological and Artificial Neural Networks.* Unpublished Report. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, October 1990.
15. Rosenblatt, R. *Principles of Neurodynamics.* New York, Spartan Books, 1959.
16. Ruck, Capt Dennis W. *Characterization of Multilayer Perceptrons and Their Application to Multisensor Automatic Target Detection.* PhD dissertation. School of Engineering, Air Force Institute of Technology (AU), Wright- Patterson AFB OH, December 1990 (AD-A229035).
17. Ruck, Capt Dennis W. "Feature Selection Using a Multilayer Perceptron," *Journal of Neural Network Computing, 20:* 40-48 (Fall 1990).

18. Steppe, Capt Jean M. *Feature Selection in Feedforward Neural Networks* PhD dissertation prospectus. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, October 1992.

19. Tarr, Capt Gregory L. *Multi-layered Feedforward Neural Networks for Image Segmentation.* PhD dissertation. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1991.

20. Weiss, Sholom M. and Casimir A. Kulikowski. *Computer Systems That Learn.* Morgan Kaufmann Publishers, Inc., 1991.

## *Vita*

Captain Gregory L. Reinhart was born on 27 September 1955 in Buffalo, Minnesota. In 1973, he graduated from Loyola High School of Mankato Minnesota. In 1977, he graduated magna cum laude from Mankato State University with a Bachelor of Science Degree in Mathematics. A Distinguished Graduate from Undergraduate Navigator Training, his first assignment was as a C-141 navigator at Norton AFB, California, where he earned his qualification as a Special Operations Low Level instructor navigator. A subsequent assignment took him to 21st Air Force, AMC, McGuire AFB, New Jersey, where he supported the flight planning and diplomatic clearance section and became assistant chief of the Special Operations Division. Captain Reinhart entered the Air Force Institute of Technology in August of 1992.

Permanent address: 1016 Summit Avenue
New Ulm, Minnesota 56073

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | March 1994 | Master's Thesis |

4. TITLE AND SUBTITLE

A FORTRAN BASED LEARNING SYSTEM USING MULTILAYER BACK-PROPAGATION NEURAL NETWORK TECHNIQUES

5. FUNDING NUMBERS

6. AUTHOR(S)

Gregory L. Reinhart, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GOR/ENS/94M-11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

An interactive computer system which allows the researcher to build an "optimal" neural network structure quickly, is developed and validated. This system assumes a single hidden layer perceptron structure and uses the back-propagation training technique. The software enables the researcher to quickly define a neural network structure, train the neural network, interrupt training at any point to analyze the status of the current network, re-start training at the interrupted point if desired, and analyze the final network using two-dimensional graphs, three-dimensional graphs, confusion matrices and saliency metrics. A technique for training, testing, and validating various network structures and parameters, using the interactive computer system, is demonstrated. Outputs automatically produced by the system are analyzed in an iterative fashion, resulting in an "optimal" neural network structure tailored for the specific problem. To validate the system, the technique is applied to two, classic, classification problems. The first is the two-class XOR problem. The second is the four-class MESH problem. Noise variables are introduced to determine if weight monitoring graphs, saliency metrics and saliency grids can detect them. Three dimensional class activation grids and saliency grids are analyzed to determine class borders of the two problems. Results of the validation process showed that this interactive computer system is a valuable tool in determining an optimal network structure, given a specific problem.

14. SUBJECT TERMS

Neural networks, Pattern recognition, Back-propagation, Learning system, Perceptron

15. NUMBER OF PAGES

98

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |