Adaptive learning methods and their use in flaw classification

Sriram Chavali

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd

Part of the Structures and Materials Commons

Recommended Citation

http://lib.dr.iastate.edu/rtd/111
Adaptive learning methods and their use in flaw classification

by

Sriram Chavali

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Department: Aerospace Engineering and Engineering Mechanics
Major: Aerospace Engineering
Major Professor: Dr. Lester W. Schmerr

Iowa State University
Ames, Iowa
1996
Copyright © Sriram Chavali, 1996. All rights reserved.
Graduate College
Iowa State University

This is to certify that the Master's thesis of

Sriram Chavali

has met the thesis requirements of Iowa State University
# TABLE OF CONTENTS

## ACKNOWLEDGMENTS

viii

## ABSTRACT

ix

## 1 INTRODUCTION

1

### 1.1 NDE Flaw Classification

1

### 1.2 Background

3

### 1.3 Scope of the Thesis

4

## 2 CLUSTERING

5

### 2.1 Introduction

5

### 2.2 Similarity Measures

6

### 2.3 Hierarchical Clustering

8

#### 2.3.1 Agglomerative hierarchical clustering

9

#### 2.3.2 Implementation of clustering in CLASS

12

### 2.4 Results

13

## 3 BACKPROPAGATION NEURAL NETWORKS

17

### 3.1 Introduction

17

### 3.2 Overview of Neural Networks

17

### 3.3 Historical Perspective

18

### 3.4 Single Layer Perceptron

19

### 3.5 Multi-Layer Perceptron

23

### 3.6 Determining a Neuron's Output

25

### 3.7 Learning

29

#### 3.7.1 Adjusting the Weights of the output layer

29

#### 3.7.2 Adjusting the Weights of the hidden layers

31

### 3.8 Momentum

33
3.9 Batching ................................................................. 33
3.10 Advantages and Disadvantages of Backpropagation Learning .......... 35
3.11 Implementation of the backpropagation neural network algorithm in CLASS .................................................. 35
3.12 Results ................................................................. 37
3.12.1 The parity problem ............................................. 38

4 FEATURE DOMAINS .................................................. 43
4.1 Sampling Discrete Time Signals .................................. 43
4.2 Frequency Domain .................................................... 44
4.2.1 Fourier series ................................................. 45
4.2.2 The discrete Fourier transform .............................. 46
4.2.3 Computation of the discrete Fourier transform ............ 48
4.2.4 The Cooley-Tukey FFT algorithm ......................... 49
4.3 Cepstrum Analysis .................................................... 51
4.4 Implementation of Frequency Domain and Cepstral Domain in CLASS .................................................. 54

5 CONCLUSIONS AND FUTURE WORK ............................ 55
APPENDIX  User Manual for CLASS .................................. 57
BIBLIOGRAPHY .......................................................... 62
LIST OF TABLES

Table 2.1 Clustering algorithm results indicating the number of correctly classified samples 15

Table 3.1 XOR decision table ........................................... 23
Table 3.2 Backpropagation neural network training on XOR with $\alpha = 0.3$ and $\eta = 0.7$ . 37
Table 3.3 Backpropagation neural network training on XOR with $\alpha = 0.7$ and $\eta = 0.3$ . 38
Table 3.4 Backpropagation neural network training on XOR with $\alpha = 0.5$ and $\eta = 0.5$ . 39
Table 3.5 Parity problem of size 3 ............................................. 39
Table 3.6 Parity problem of size 4 ............................................. 40
Table 3.7 Backpropagation neural network training on parity problem of size 3 with $\alpha = 0.3$ and $\eta = 0.7$ . 41
Table 3.8 Backpropagation neural network training on parity problem of size 4 with $\alpha = 0.3$ and $\eta = 0.7$ . 41
LIST OF FIGURES

Figure 1.1 Flow chart for classification .......................................................... 2
Figure 2.1 Clustering of points in Feature space ............................................. 6
Figure 2.2 The effect of distance threshold on clustering ............................... 7
Figure 2.3 A dendogram for hierarchical clustering ....................................... 10
Figure 2.4 A flowchart of agglomerative hierarchical clustering ................... 11
Figure 2.5 Clustering algorithm folder ......................................................... 14
Figure 2.6 Clustering algorithm report generated by CLASS ......................... 16

Figure 3.1 A graphical representation of an artificial neural network ............. 18
Figure 3.2 A taxonomy of six neural nets ..................................................... 19
Figure 3.3 A simple neuron ........................................................................... 20
Figure 3.4 Linear inseparability of the XOR problem .................................... 22

Figure 3.5 A three layer perceptron .............................................................. 24
Figure 3.6 A Sigmoid function ...................................................................... 26
Figure 3.7 Types of decision regions that can be formed by single- and multi-layer perceptrons with one and two layers of hidden units and two inputs. Shading denotes decision regions for class A. All the nodes use nonlinear activation functions. . 27
Figure 3.8 An artificial neuron with an activation function .......................... 28
Figure 3.9 Training a Weight in the output layer. The subscripts p and q refer to a specific neuron. The subscripts j and k refer to a layer. ................................. 30
Figure 3.10 Training a Weight in a hidden layer. The subscripts m and p refer to a specific neuron. The subscripts i, j and k refer to a layer. ......................... 32
Figure 3.11 Flow chart illustrating the backpropagation training process ....... 34
Figure 3.12 Backpropagation neural network folder ..................................... 36
Figure 3.13 BPNN algorithm report generated by CLASS ............................. 42
Figure 4.1  Aliasing in a discrete time signal ........................................ 44
Figure 4.2  Flow graph of the decimation in time decomposition of an N point DFT into
two $\frac{N}{2}$ point DFT for $N = 8$. ........................................ 49
Figure 4.3  Flow graph of basic butterfly computation ........................ 50
Figure 4.4  Flow graph of Equations 4.28 and 4.29 .............................. 51
Figure 4.5  Tree diagram depicting bit reversed sorting ......................... 52
Figure 4.6  Flow graph of 8 point discrete Fourier transform .............. 53
Figure A.1  Class folders .................................................................... 58
Figure A.2  Clustering algorithm folder ............................................. 59
Figure A.3  Backpropagation neural network folder ............................ 61
ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Lester Schmerr for his guidance, direction and support of my research. He introduced me to the fields of Nondestructive evaluation and Artificial intelligence and was always there to help me with my numerous questions and doubts. I would also like to thank Dr. Chien-Ping Chiou for being patient with all my questions during the last year. I would also like to thank all the professors who taught me in their classes during the last two years. Finally, I would like to thank my LaTex guru, Dr. John Samsundar. But for John’s suggestions and help, I would have had a tough time getting around the idiosyncrasies of LaTex.

I would like to thank my officemates, Dr. Terry Lerch, Raju Vuppala and Dr. Alex Sedov. They were always there to help me in doing the small things which are so important. I would like to thank my brother Srikanth, who helped me stay focussed during the last year. I would also like to thank all my friends for their support and help.

Finally, I would like to thank my parents for the encouragement, support and ideals that they imparted to me.
ABSTRACT

An important goal of nondestructive evaluation is the detection and classification of flaws in materials. This process of 'flaw classification' involves the transformation of the 'raw' data into other domains, the extraction of features in those domains, and the use of those features in a classification algorithm that determines the class to which the flaw belongs.

In this work, we describe a flaw classification software system, CLASS and the updates made to it. Both a hierarchical clustering algorithm and a backpropagation neural network algorithm were implemented and integrated with CLASS. A fast Fourier transform routine was also added to CLASS in order to enable the use of frequency domain and cepstral domain features.

This extended version of CLASS is a very user friendly software, which requires the user to have little knowledge of the actual learning algorithms. CLASS can be easily extended further, if needed, in the future.
1 INTRODUCTION

1.1 NDE Flaw Classification

Non-destructive evaluation (NDE) methods place various forms of energy into materials and try to examine the materials without harming them or affecting their performance. Examples of NDE methods and the types of energy they use are: Ultrasound-acoustic energy, Eddy currents (electrical energy), and X-rays (penetrating radiation). One important application of these NDE methods is to find, classify and characterize flaws based on their response to the types of energy present. Here we are interested only in the process of classifying flaws which involves, for example, such tasks as distinguishing cracks from non-crack-like flaws.

The steps involved in the NDE flaw classification process are shown in Figure 1.1. First, the raw signals need to be captured and stored. In ultrasonics, this would be a voltage versus time signal. Features must be derived from those signals. In some cases features can be obtainable from the raw signals themselves. However, in many cases it is desirable to first transform the raw signals into other domains so that features more characteristic of the flaw type can be obtained. In ultrasonics, for example, it is often desirable to use features taken from the frequency spectrum of the signals. Once the data are available in the desired transformed domains, then the features can be extracted. Finally, through the use of of these features, the flaws are classified into types using a particular classification method. In general, more than two classes of flaws need to be identified. For example, one important application where classification of flaws of multiple types is needed is in the inspection of weldments. The detection and proper classification of weld defects such as cracks, porosity and slag inclusions is necessary since these flaws have different impacts on structural performance and reliability.

Previously, a wide variety of classification approaches have been used for ultrasonic welds inspections, including heuristic experience-based methods [1], [2], signal processing and pattern recognition schemes , [3], [4], [5] adaptive learning methods [7], and expert systems [8], [9]. More recently, neural networks [10], [11], [12], [13] have been applied to such problems. Similar classification methods are useful for many other inspection problems, particularly those associated with the aerospace and energy industries.
Figure 1.1  Flow chart for classification
1.2 Background

In previous work at the Center for Nondestructive Evaluation (CNDE) sponsored by the NSF Industrial/University program a variety of techniques for flaw classification have been considered. Some examples of these techniques are:

- Expert Systems
- Neural Networks (Probabilistic, Backpropagation, Hierarchical Nets)
- Statistical Methods (K - Nearest Neighbors)
- Decision true Methods

A flaw classification software package, CLASS, was developed in order to collect some of those techniques in a unified environment where they could be evaluated by the industrial sponsors of the Center for NDE at the Iowa State University. Initial work on CLASS was done by Loren Knutson [14], as part of his Masters' degree research work. An Intel-based personal computer was chosen as the platform for the package because it is the platform primarily used by the Center's sponsors for acquiring and analyzing their experimental data. Visual Basic was chosen as the language for implementing the Graphical User Interface. A standardized database file format was chosen to manage the substantial data used by CLASS, in order to overcome the problem of extremely slow retrieval of data from ASCII text format files, in a highly ordered sub directory tree. Using ODBC commands, the information within the database files could be accessed by the routines in CLASS.

The first version of CLASS handled only the raw time domain signals. It used six features in the time domain. Four of these six features were the first four statistical moments, namely mean, variance, skewness and kurtosis. The other two features were zero crossing and absolute mean. The definitions of these features is given below.

\[
\text{mean} = \mu = \frac{1}{N} \sum_{i} x_i \tag{1.1}
\]

\[
\text{absmean} = \frac{1}{N} \sum_{i} |x_i| \tag{1.2}
\]

\[
\text{variance} = \sigma^2 = \frac{1}{N-1} \sum_{i} (x_i - \mu)^2 \tag{1.3}
\]
skewness = \frac{1}{N} \sum \left( \frac{x_i - \mu}{\sigma} \right)^3 \quad (1.4)

kurtosis = \frac{1}{N} \sum \left( \frac{x_i - \mu}{\sigma} \right)^4 - 3 \quad (1.5)

zerocrossing = \sum P_i \quad (1.6)

P_i = 1 \text{ if } x_i x_{i+1} < 0, \quad P_i = 0 \text{ otherwise}

CLASS also contained the following Learning Algorithms:

- K-Nearest Neighbors
- Probabilistic Neural Network
- ID3 Decision Tree

For CLASS 2.0, we intended to provided it with the capability of handling data in domains other than the time domain. Also, we wanted to provide additional Learning Algorithms, of both supervised and unsupervised types for CLASS 2.0. In the following chapters, we will explain the additions that have been made to CLASS.

### 1.3 Scope of the Thesis

This chapter has briefly outlined flaw classification and described a software package, CLASS. In chapters two and three, we explain the new learning algorithms, which have been added to CLASS, namely the Clustering algorithm and a Backpropagation Neural Network algorithm. In chapter 4, we will describe the two new feature domains that have been added to CLASS. In chapter 5, we will discuss the results and conclusions from simulations on CLASS. Finally, in the appendix, we discuss the user interface developed for CLASS and some of the basic working characteristics of CLASS.
2 CLUSTERING

2.1 Introduction

The first new adaptive learning classification method that was added to CLASS was a basic clustering algorithm. This choice was made because clustering methods are useful for identifying groups of data that are similar and dissimilar. Characteristics of these groups (mean values, etc.) are potentially good features to use in the other classification algorithms. Also clustering could be used to help improve the efficiency of algorithms such as the Probabilistic neural network algorithm. Clustering is similar to the k-nearest neighbor method. Both the k-nearest neighbors algorithm and clustering are unsupervised learning algorithms. By unsupervised algorithms we mean algorithms that construct a classifier directly from the data. As a consequence unsupervised learning algorithms are usually non iterative. In contrast supervised learning algorithms build up decision surfaces based on the information learnt from training data by an iterative learning process, where the classifier architecture and parameters of the learning process are determined by the user. This generalization of the data space is then used for the classification of other testing data samples.

Clustering is one of the earliest and most intuitive methods used for classification. The samples of data often form distinctive clouds of points in n-dimensional feature space, each cloud being representative of a particular concept. The sample mean, for example constitutes the center of gravity of the cloud. It can be thought of as the single point x which is representative of all the samples. The sample covariance matrix tells us how well the sample mean describes the data in terms of the amount of scatter that exists in various directions. Figure 2.1 illustrates clustering on a 2-dimensional feature space.

By assuming that the samples come from a mixture of of c normal distributions, we can approximate a great variety of situations. In essence, this corresponds to assuming that the samples fall in hyperepilipsoidally-shaped clouds of various sizes and orientations. Clustering procedures yield a data description in terms of clusters or groups of data points that possess strong internal similarities. Clustering procedures use a criterion function, such as the sum of squared distances from the cluster centers,
and seek the grouping that extremizes the criterion function.

2.2 Similarity Measures

An obvious measure of similarity (or dissimilarity) between two samples is the distance between them. The distance between the samples in the same cluster must be significantly less than the distance between samples in different clusters, if distance were a good measure of dissimilarity.

For example, two samples can be said to belong to the same cluster if the Euclidean distance between them is less than some threshold distance $d_0$. The choice of $d_0$ is very important as can be seen from Figure 2.2. If $d_0$ is very large (Figure 2.2(a)), all the samples would be assigned to one cluster. If $d_0$ is very small (Figure 2.2(c)), each sample would form an isolated cluster. To obtain “natural” clusters (Figure 2.2(b)), $d_0$ will have to be greater than typical intra-cluster distances and less than typical inter-cluster distances.

The choice of Euclidean space implies that the feature space is isotropic. Consequently, clusters defined by Euclidean distance will be invariant to translations or rotations-rigid-body motions of the data points. However, they will not be invariant to linear transformations in general, or to other
Figure 2.2 The effect of distance threshold on clustering
transformations that distort the distance relationships. A simple scaling of the coordinate axes can result in a different grouping of the data into clusters.

One way to achieve invariance is to normalize the data prior to clustering. For example, to obtain invariance to displacement and scale changes, one might translate and scale the axes so that all of the features have zero mean and unit variance. To obtain invariance to rotation, one might rotate the axes so that they coincide with the eigenvectors of the sample covariance matrix. This transformation to \textit{principal components} can be preceded and/or followed by normalization for scale. An alternative to normalizing the data and using Euclidean distance is to use some kind of normalized distance, such as the Mahalanobis distance [15].

For the classification problem in hand, since arbitrary rescaling is unnatural, we used the Euclidean distance as the similarity measure.

\section{Hierarchical Clustering}

Basically, there are four different kinds of clustering algorithms:

\begin{itemize}
  \item Minimization of squared error.
  \item Hierarchical clustering
  \item Graph-theoretical clustering
  \item Fuzzy clustering
\end{itemize}

The clustering algorithm in CLASS is of a hierarchical nature. To discuss hierarchical clustering, let us consider a sequence of partitions of the \( n \) samples into \( c \) clusters. The first of these is a partition into \( n \) clusters, each cluster containing exactly one sample. The next is a partition into \( n - 1 \) clusters, the next a partition into \( n - 2 \), and so on until the \( n \)-th, in which all the samples form one cluster. At any level \( k \) in the sequence, there are \( c = n - k + 1 \) clusters. Given any two samples \( x \) and \( x' \), at some level they will be grouped together in the same cluster. If the sequence has the property that whenever two samples are in the same cluster at level \( k \) they remain together at all higher levels, then the sequence is said to be a \textit{hierarchical clustering}.

For every hierarchical clustering, there is a corresponding tree, called a \textit{dendogram}, that shows how the samples are grouped. Figure 2.3 shows a dendogram for a hypothetical problem involving six samples. Level 1 shows the six samples as singleton clusters. At level 2, samples \( x_3 \) and \( x_5 \) have been grouped to form a cluster, and they stay together at all subsequent levels. If it is possible to measure
the similarity between clusters, then the dendogram is drawn to scale to show the similarity between the clusters that are grouped. The similarity values are used to determine whether the groupings are natural or forced. For the example listed in Figure 2.3, the groupings at level 4 or 5 are natural, but the large reduction in similarity needed to go to level 6 makes that grouping forced.

Hierarchical clustering procedures can be divided into two distinct classes, agglomerative and divisive. Agglomerative procedures start with \( n \) singleton clusters and form the sequence by successively merging clusters. Divisive procedures start with all of the samples in one cluster and form the sequence by successively splitting clusters. The computation needed to go from one level to another is usually simpler for the agglomerative clustering procedures. Hence, a Hierarchical Agglomerative clustering algorithm was chosen for CLASS.

### 2.3.1 Agglomerative hierarchical clustering

The basic steps in Agglomerative clustering are explained in the flowchart in Figure 2.4. At the first level, all the \( n \) samples to be classified are considered to be separate clusters. Then the closest two clusters are merged to form a single cluster at the next level. Again at the next level, the two closest clusters are merged to obtain a new cluster. This procedure stops when the specified number of clusters has been obtained. There are basically four distance measures:

\[
\begin{align*}
    d_{\text{min}}(X_i, X_j) & = \min \|x - x'\| \\
    d_{\text{max}}(X_i, X_j) & = \max \|x - x'\| \\
    d_{\text{avg}}(X_i, X_j) & = \frac{1}{n_i n_j} \sum_{x \in X_i} \sum_{x' \in X_j} \|x - x'\| \\
    d_{\text{mean}}(X_i, X_j) & = \|m_i - m_j\|
\end{align*}
\]

where \( x \) belongs to cluster \( X_i \) and \( x' \) belongs to cluster \( X_j \).

All the above measures have a minimum-variance flavor, and they usually yield similar results if the clusters are compact and well separated.
Figure 2.3 A dendogram for hierarchical clustering
Figure 2.4 A flowchart of agglomerative hierarchical clustering
2.3.1.1 The nearest neighbor algorithm

If the data points were to be thought of as being nodes of a graph, with edges forming a path between nodes in the same subset $X_i$, when $d_{\text{min}}$ (equation 2.1) is used to measure the distance between subsets, the nearest neighbors determine the nearest subsets. The merging of $X_i$ and $X_j$ corresponds to adding an edge between the nearest pair of nodes in $X_i$ and $X_j$. This procedure generates a tree. If it is allowed to continue until all of the subsets are linked, the result is a spanning tree. With the use of $d_{\text{min}}$ as the distance measure, the agglomerative clustering procedure becomes an algorithm for generating a minimal spanning tree.

2.3.1.2 The furthest neighbor algorithm

When $d_{\text{max}}$ (equation 2.2) is used to measure the distance between subsets, a graph is produced in which edges connect all of the nodes in a cluster. Hence every cluster constitutes a complete subgraph. The distance between two clusters is determined by the most distant nodes in the two clusters. When the nearest clusters are merged, the graph is changed by adding edges between every pair of nodes in the two clusters.

2.3.1.3 Compromises

The minimum and maximum measures represent two extremes in measuring the distance between clusters. Like all procedures that involve minima or maxima, they tend to be overly sensitive to outliers. The use of averaging is one way of ameliorating this problem, and $d_{\text{avg}}$ (equation 2.3) and $d_{\text{mean}}$ (equation 2.4) are natural compromises between $d_{\text{min}}$ and $d_{\text{max}}$. Computationally, $d_{\text{mean}}$ is the simplest of all these measures, since the others require computing all $n_in_j$ pairs of distances $||x - x'||$.

2.3.2 Implementation of clustering in CLASS

Figure 2.5 shows the clustering algorithm folder. This folder comes up on the screen after the features have been calculated, when the user has chosen the clustering algorithm for analyzing his data. In the 'Database Information' part of the folder, the datafile in use and its path are listed. The list of chosen features are also available for viewing by the user. The user has the option of choosing between $d_{\text{min}}$, $d_{\text{max}}$ and $d_{\text{avg}}$, as the distance measures. The user can check off, either of them as the distance measure to be used in the current application. By clicking on the Train/Test command button, the user can begin the execution of the clustering algorithm. Information about processing and the up-to-date status of the execution is also conveyed to user.
One of the underlying philosophies in developing CLASS was providing a very intuitive and easy to use graphical user interface. Very little user expertise is needed in choosing from the available options in the clustering algorithm. With a minimal understanding of the learning algorithms, the NDE researcher can use CLASS for a quick analysis of the experimental data. The clustering algorithm available in CLASS is successful in attaining these goals.

2.4 Results

The clustering algorithm in CLASS was tested on an artificially created, linearly separable data set. Table 2.1 lists the performance of the clustering algorithm, with various merging schemes for two data sets. The first set of data consisted of simulated hard-alpha inclusions. The second set contained heart disease information and was downloaded from the University of California, Irvine. It is a two class data set with thirteen attributes. As can be seen in Table 2.1, the clustering algorithm was able to correctly classify 49-64 percent of the heart disease samples with the given features and 47-50 percent of the hard-alpha samples. In both these examples, clustering by itself did not produce a high percentage of correct classifications. However, this does not mean that clustering is not useful in such cases. As mentioned previously, clustering can also be used as a pre-processor of data for use by other classification methods to provide more highly leveraged features. One might use the cluster centers and radii from these centers, for example, as simple features that could help significantly in the classification process. Such applications of clustering are not pursued here, but could be in future work. A report from a sample run on the linearly separable data set by the clustering algorithm is shown in Figure 2.6. In Figure 2.6, the column ID lists the index of the sample in the database, the column Classified Concept mentions the concept evaluated by the clustering algorithm and the column Actual Concept lists the actual concept of the sample.

These results also demonstrate that, at least in the heart disease example, the choice of distance measure can have a significant effect on the classification performance. Thus, it is important to give the user different choices of distances to experiment with, as done in CLASS.
Figure 2.5  Clustering algorithm folder
Table 2.1 Clustering algorithm results indicating the number of correctly classified samples

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size of Dataset</th>
<th>Average Distance</th>
<th>Furthest Neighbors</th>
<th>Nearest Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard-Alpha</td>
<td>115</td>
<td>58</td>
<td>55</td>
<td>54</td>
</tr>
<tr>
<td>Heart Disease</td>
<td>135</td>
<td>66</td>
<td>82</td>
<td>86</td>
</tr>
</tbody>
</table>
### Clustering Algorithm Results

<table>
<thead>
<tr>
<th>ID</th>
<th>Classified Concept</th>
<th>Actual Concept</th>
<th>Correct Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 2.6 Clustering algorithm report generated by CLASS
3 BACKPROPAGATION NEURAL NETWORKS

3.1 Introduction

The second type of classification algorithm that we incorporated into CLASS uses a backpropagation neural network. This addition was made because although CLASS already contained a neural network classification algorithm (Probabilistic neural network), backpropagation networks are undoubtedly the most popular type of neural networks used today for a variety of applications.

3.2 Overview of Neural Networks

Artificial neural networks are intended to model the structure and operation of biological nervous systems. They are composed of nonlinear computational elements operating in parallel. Computational elements or nodes are connected via weights that are typically adapted during use to improve performance. Despite having only a superficial resemblance to biological neurons, artificial neural networks exhibit a surprising number of the brain's characteristics. They learn from experience, generalize from previous examples to new ones, and abstract essential characteristics from inputs containing noisy and irrelevant data.

Succinctly, a neural network can be described as a directed graph, with the nodes of the graph represented by the artificial neurons, and the edges of the graph represented by the connections between the neurons. Neural nets are specified by choosing the net topology, neuron characteristics and training rules. These rules specify an initial set of weights and indicate how weights should be adapted during use to improve performance. Each neuron receives stimuli from several sources—other neurons, the outside world, or both—and operates on its stimuli with a transfer function to produce a single output. This output becomes either the input to other neurons or the final output of the network. Figure 3.1 shows a graphical representation of a neural network with a rather arbitrary network architecture.

A taxonomy of six important neural networks that can be used for classification of static patterns is presented in Figure 3.2. This taxonomy is first divided between nets with binary and continuous valued
inputs. In NDE applications, data is almost always of a continuous nature so nets restricted to binary inputs are seldom used. In Figure 3.2 the nets are further divided between those trained with and without supervision. In the preceding chapter, the differences between supervised and unsupervised training was been explained. In unsupervised learning algorithms, the class labels for the training data set samples need not be known beforehand. The classifier is directly constructed from the data. In supervised learning algorithms, the class labels for the training data samples is known and the information learnt from training is used to build the decision surface.

3.3 Historical Perspective

The first significant contribution to the idea of emulating processing functions seen in biological organisms via the implementation of interconnected computing blocks was put forth in 1943 by Warren McCulloh and Walter Pitts [26]. This work presented their ideas of combining finite-state machines and linear threshold elements for describing some forms of behavior and memory.

A model of human learning which was most effective was proposed by D. O. Hebb [25], who in 1949 proposed a learning law that became the starting point for artificial neural networks training algorithms.
It showed how a network of neurons can exhibit learning behavior.

In the 1950s and 1960s, a group of researchers combined these biological and psychological insights to produce the first artificial neural networks. Initially implemented as electronic circuits, they were later converted to the more flexible medium of computer simulation. Early successes produced a burst of activity and optimism. Minsky, Rosenblatt, Widrow and others developed networks consisting of a single layer of neurons [25]. Often called perceptrons, they were applied to such diverse problems as weather prediction, electrocardiogram analysis and artificial vision.

### 3.4 Single Layer Perceptron

The single layered perceptron generated much interest when initially developed because of its ability to learn to recognize simple patterns. Figure 3.3 shows a perceptron that decides whether an input belongs to one of two classes (denoted A or B). The single node computes a weighted sum of the input elements, subtracts a threshold, $q$ and passes through a hard limiting nonlinearity such that the output is either $+1$ or $-1$. The decision rule is to respond class A if the response is $+1$ and class B if the output is $-1$. The perceptron forms two decision regions separated by a hyper-plane. Inputs above above the boundary line lead to class A responses and below the line lead to class B responses.

Connection Weights and the threshold in a perceptron can be fixed or adjusted using a number of dif-
Figure 3.3 A simple neuron

\[ y = f\left(\sum_{i=0}^{N} w_i x_i - q\right) \]
ferent algorithms. The original perceptron convergence procedure for adjusting Weights was developed by Rosenblatt [26]. It is applied as follows:

Step 1. Initialize Weights and threshold to random values.
Step 2. Present new input \( x_0(t), x_1(t), \ldots, x_{N-1}(t) \) and desired output \( d(t) \).
Step 3. Calculate the actual output.
Step 4. Adapt Weights, using a positive gain factor, \( \eta \) with the error.
Step 5. Repeat by going to Step 2.

Connection Weights are adapted using the following rule:

\[
    w_i(t+1) = w_i(t) + \eta[d(t) - y(t)]x_i(t)
\]

for \( 0 < \eta < 1.00 \)

According to Rosenblatt’s Perceptron Convergence theorem, if the inputs presented from the two classes are separable, then the perceptron convergence procedure converges and positions the decision hyperplane between those two classes. Such a hyperplane is illustrated in Figure 3.3.

In 1969, Minsky and Papert [22] published the book, *Perceptrons* in which they proved that the single-layer networks were theoretically incapable of solving many simple problems, including the function performed by a simple exclusive-or (XOR) gate. Any two-input function whose outputs cannot be separated into distinct classes by a single straight line, or in the case of higher dimensional input vectors, by a single hyper-plane, is not linearly separable. Minsky and Papert proved that a perceptron cannot learn to perform the correct mapping for any function which is not linearly separable. This work, for all practical purposes halted intensive artificial neural network research for many years.

The graph in Figure 3.4, shows a plot of the XOR function as shown in Table 3.1. The \( x_1 \) and \( x_2 \) axes represent the two inputs to the problem. The dark circles represents an output (\( z \)) value of 1 and the blank circles represent an output value of 0. As seen from it, a straight line cannot be drawn which separates the output responses, denoted as a filled circle for a 1 and an open circle for a 0, into two distinct classes. Hence they are linearly inseparable. A single layer perceptron cannot solve this problem.

At this time it was well known that a multi-layered perceptron was capable of performing the XOR function. However, one problem still persisted. A learning rule didn’t exist for adjusting the Weights of the 'hidden layer neurons' shown in Figure 3.5. Rosenblatt’s Perceptron Convergence theorem was valid only for single-layered perceptrons. It wasn’t known, what proportion of the error at the output layer could be attributed to the Weights for multiple layers between the inputs and outputs.

In 1986, Rumelhart, Hinton and Williams [33] presented the generalized delta rule for “back-
Figure 3.4  Linear inseparability of the XOR problem
Table 3.1 XOR decision table

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

propagating" the errors at the output layer to a hidden layer.

3.5 Multi-Layer Perceptron

Multi-layer perceptrons are feed-forward nets with one or more layers of nodes between the input and the output nodes. These additional layers contain hidden nodes. In this work, we consider those "feed-forward" perceptrons, where there are direct connections between nodes in consecutive layers only. A three-layer perceptron with two layers of hidden units is shown in Figure 3.5. As just mentioned, multi-layer perceptrons overcome many of the limitations of single-layer perceptrons, but were generally not used in the past because effective training algorithms were not available.

Figure 3.1 depicted a graphical representation of a feed forward neural network with an arbitrary network architecture. Those neurons, which receive stimuli from the outside world are termed input neurons. Neurons which direct their responses back to the outside world are called output neurons. Neurons which receive their stimuli only from other neurons in the network and send their responses to other neurons in the network are called hidden neurons.

Typically neurons are grouped into layers, with neurons in one layer connected to neurons in other layers, i.e no intra-layer connections and no recurrent connections. In this work, we talk about fully connected feed-forward neural networks, i.e a network where a neuron in one layer is connected to all neurons in the layer above. At the output layer, the product matrix of the input values to this layer with the corresponding Weight matrix is termed as NET.

The capabilities of multi-layer perceptrons stem from the nonlinear activation functions used within nodes. If nodes were linear elements, then a single-layer net with appropriately chosen Weights could
Figure 3.5 A three layer perceptron
exactly duplicate those calculations performed by any multi-layer net.

A number of functions could be used as the activation function. The only condition that the activation function must satisfy is that it must be differentiable everywhere. The activation function provides an automatic gain control for a neuron's output. For small values of $\text{NET}$, it provides a high gain. As the values of $\text{NET}$ increases, the gain decreases. The activation function thus accommodates large signals without saturation and while allowing small signals to pass through without excessive attenuation.

Figure 3.6 shows the activation function used in this work.

$$\text{OUT} = \frac{1}{1 + e^{-\text{NET}}}$$

(3.1)

This function is called a sigmoid or a logistic function. This sigmoidal function produces the required nonlinearity in the network, besides compressing the range of $\text{NET}$ between zero and one. The derivative of the sigmoid which is very important in the implementation of the backpropagation algorithm is given by

$$\frac{\partial(\text{OUT})}{\partial(\text{NET})} = \text{OUT}(1 - \text{OUT})$$

(3.2)

The capabilities of perceptrons with one, two and three layers, that use nonlinear activation functions are illustrated in Figure 3.7. The second column gives examples of decision regions which could be formed for the exclusive OR problem. The right most column gives examples of the most general decision regions that can be formed. A single-layer perceptron forms half-plane decision regions. A two-layer perceptron forms open or closed convex regions. Convex regions are formed from intersections of the half-plane regions formed by each node in the first layer of the multi-layer perceptron. These convex regions have at the most as many sides as there are nodes in the first layer. Thus, the number of nodes to be used in a two-layer perceptron must be large enough to form a decision region that is as complex as is required by the problem. A three-layer perceptron can form arbitrarily complex decision regions [31]. Using a three-layer perceptron, we can generate the disconnected and non-convex regions shown at the bottom of Figure 3.7. Thus no more than three layers are required in perceptron-like feed-forward nets because a three-layer net can in principle generate arbitrarily complex decision regions.

### 3.6 Determining a Neuron's Output

Each neuron produces a response to given input stimuli. Typically a neuron's response, or output is computed in two steps. The first step is to compute a weighted sum of inputs. This weighted sum is
Figure 3.6 A Sigmoid function
Figure 3.7  Types of decision regions that can be formed by single- and multi-layer perceptrons with one and two layers of hidden units and two inputs. Shading denotes decision regions for class A. All the nodes use nonlinear activation functions.

also known as the neuron’s state of activation, i.e. the degree to which the neuron is activated. This is done by multiplying each of the incoming signals by the Weight of the connection on which the signal is received. Figure 3.8 shows a neuron with an activation function. The state of activation termed NET is calculated as

$$NET_{p,k} = w_{p0} + \sum_{q=1}^{N} w_{pq} OUT_{q,j}$$  \hspace{1cm} (3.3)

where $w_{pq}$ is the Weight on the connection from the $q^{th}$ neuron in the preceding layer to the $p^{th}$ neuron (the neuron whose transfer is being computed) in the current layer, $OUT_{q,j}$ is the output of the $q^{th}$ neuron in the $j^{th}$ layer, $NET_{p,k}$ is the state of activation at the $p^{th}$ neuron in the $k^{th}$ layer, and $N$ is the number of input stimuli. $w_{p0}$ is a bias term, or threshold. This term offsets the origin of the sigmoid function, thereby speeding up convergence. The second step is to apply activation function, $f$, to NET in order to produce the neuron’s response to the stimuli. Denoting this neuron’s response as $OUT$:

$$OUT = f(NET)$$  \hspace{1cm} (3.4)
NET = \sum_{i} O_{i} w_{i}

OUT = f(NET)

Figure 3.8 An artificial neuron with an activation function
3.7 Learning

The computation of the outputs in a feed-forward network is essentially a mapping of the inputs to the outputs. The forward pass evaluates an equation that best expresses the outputs as a function of the inputs. The mapping function which a neural network approximates is very flexible. This function can take on many different shapes. It can be configured to the shape of any target function.

In the backpropagation algorithm, the learning process is supervised, i.e, the correct set of responses is known for a given set of inputs. The Weights of the network are adaptively adjusted by the use of the learning algorithm during the training process. In the training process, the inputs are presented to the input layer of the network. The network evaluates the response at the output layer. The outputs of the network are then compared against the desired outputs for the given inputs. Using the error at the output, the Weights are adjusted. Backpropagation uses a gradient descent, a multi-dimension optimization procedure. The Weights are changed in a direction which minimizes the error. After adjusting the Weights, the next training sample of input is presented. This procedure is repeated until the error at the output layer is within an acceptable limit over the entire training set.

Before training the Weights of the network are initialized to random values. During the forward pass, the network output of a training sample is calculated. The backpropagation algorithm minimizes the error measure between the network output and the target vectors. The error measure used in this work is the least-squared error, E.

\[ E = \frac{1}{M} \sum_{i=1}^{M} (\text{TARGET}_i - \text{OUT}_i)^2 \]  

(3.5)

where \( M \) is the number of output nodes.

3.7.1 Adjusting the Weights of the output layer

Since the target vectors at the neurons of the output layer are available, the adjustment of the associated Weights is quite easily accomplished. Figure 3.9 shows the training of a weight in the output layer. The gradient of the error, \( E \) in the \( kth \) layer (the output layer) with respect to the Weight at the \( qth \) neuron in the \( kth \) layer from the \( pth \) neuron in the \( jth \) layer (the previous layer) is evaluated as

\[ \frac{\partial E}{\partial w_{pq}} = (\frac{\partial E}{\partial \text{OUT}_{q,k}})(\frac{\partial \text{OUT}_{q,k}}{\partial \text{NET}_{q,k}})(\frac{\partial \text{NET}_{q,k}}{\partial w_{pq}}) \]  

(3.6)

We now define \( \delta \) as

\[ \delta_{q,k} = (\frac{\partial E}{\partial \text{OUT}_{q,k}})(\frac{\partial \text{OUT}_{q,k}}{\partial \text{NET}_{q,k}}) \]  

(3.7)
Figure 3.9 Training a Weight in the output layer. The subscripts $p$ and $q$ refer to a specific neuron. The subscripts $j$ and $k$ refer to a layer.
The derivative $\frac{\partial \text{OUT}_{q,k}}{\partial \text{NET}_{q,k}}$ is obtained from the equation 3.2. Using equations 3.5 and 3.6, $\delta$ is obtained as

$$\delta_{q,k} = -\text{OUT}_{q,k}(1 - \text{OUT}_{q,k})(\text{TARGET}_{q,k} - \text{OUT}_{q,k})$$  \hfill (3.8)

Using a proportionality constant, $\eta$, called the training rate coefficient, the change in $w_{pq,k}$ can be obtained such that the error $E$ is reduced.

$$\Delta w_{pq,k} = \eta \delta_{q,k} \text{OUT}_{p,j}$$ \hfill (3.9)

$\eta$ has values typically lying between 0.01 and 1.0. The above relation is known as the “Generalized Delta rule”. The value of the Weight $w_{pq,k}$ at the $(t + 1)^{th}$ iteration is now obtained as

$$w_{pq,k}(t + 1) = w_{pq,k}(t) + \Delta w_{pq,k}$$ \hfill (3.10)

### 3.7.2 Adjusting the Weights of the hidden layers

Since hidden layers do not have a target vector, the training process described in the preceding section cannot be applied to them. The lack of a training target for the hidden layers prevented multilayered neural networks from being trained, until backpropagation provided a workable algorithm. The backpropagation algorithm propagates the output error back through the network layers. Using the Generalized Delta rule, the Weights are then adjusted for the hidden layers. Figure 3.10 shows the training of a neuron in a hidden layer.

The gradient of the network error $E$ with respect to a hidden neuron Weight $w_{mp}$ is evaluated as

$$\frac{\partial E}{\partial w_{mp}} = \left( \frac{\partial E}{\partial \text{OUT}_{p,j}} \right) \left( \frac{\partial \text{OUT}_{p,j}}{\partial \text{NET}_{p,j}} \right) \left( \frac{\partial \text{NET}_{p,j}}{\partial w_{mp}} \right)$$ \hfill (3.11)

Again $\delta$ is defined as

$$\delta_{p,j} = \left( \frac{\partial E}{\partial \text{OUT}_{p,j}} \right) \left( \frac{\partial \text{OUT}_{p,j}}{\partial \text{NET}_{p,j}} \right)$$ \hfill (3.12)

$\delta$ is obtained as

$$\delta_{p,j} = \text{OUT}_{p,j}(1 - \text{OUT}_{p,j}) \left( \sum_{q=1}^{M} \delta_{q,k} w_{pq,k} \right)$$ \hfill (3.13)

Now using the Generalized Delta rule of equation 3.9, the change in the value of the Weight $w_{mp}$ is obtained. The values of $\delta$ obtained in this layer are in turn propagated to the preceding layers.

For each neuron in a given hidden layer, $\delta$’s must be calculated, and all the Weights associated with that layer must be adjusted. This process is repeated, moving towards the input layer, until all the Weights are adjusted.
Figure 3.10 Training a Weight in a hidden layer. The subscripts $m$ and $p$ refer to a specific neuron. The subscripts $i$, $j$ and $k$ refer to a layer.
The Weights of the network are updated after the values of the \( \delta \)s are obtained for all the neurons. The Weight adjustment procedure is repeated for all the samples in the training data set. The total error at the end of an epoch or a presentation of the entire training set to the training algorithm is computed. If this error falls below a sufficiently low level, training is stopped. A flow chart of the backpropagation algorithm is shown in Figure 3.11.

3.8 Momentum

Backpropagation uses gradient descent in Weight space to reduce the error for all the training samples. As with any gradient descent approach, backpropagation is susceptible to getting trapped in local minima. For certain problems, backpropagation could cause the Weights to oscillate rather than converge smoothly to a solution. A solution to this problem is called the momentum. The basic idea of momentum is that the Weight changes for the current epoch move in the direction in which they had been moving recently. In this method, first the Weight changes corresponding to the direction of steepest descent are computed for the current Weight. But the actual Weight adjustment is an average between this indicated change and the last actual change of Weights. As learning proceeds, the influence of prior gradients persists in the network. Using a momentum factor \( \alpha \) in equation 3.9, the change in the value of a Weight for the \((n + 1)^{th}\) iteration now becomes

\[
\Delta w_{pq,k}(n + 1) = \eta \delta_{q,k}OUT_{p,j} + \alpha [\Delta w_{pq,k}(n)]
\]  

(3.14)

The typical values of \( \alpha \) are between 0 and 1.00. The use of momentum is especially desirable if the network error surface has "valleys". Using the momentum method, the network tends to follow the bottom of the valley rather than criss-crossing from one side of the valley to the other. This allows the network to ignore the short-term fluctuations in the gradient, saving computational effort. In CLASS, the user can interactively choose a value for momentum.

3.9 Batching

Another way of mitigating the problem of oscillating Weight changes is the use of batching. In batching, the Weight changes are updated after the gradients over a number of training samples are accumulated. This way, the back and forth movement in the network from consecutively presenting very low and very high target outputs is avoided. This saves substantial computational effort. The backpropagation algorithm provided in CLASS allows users to use batch sizes of their own choice. The
Figure 3.11  Flow chart illustrating the backpropagation training process
value of a batch size is an integer. The batch size values can range from 1 to the number of samples in the training data set.

3.10 Advantages and Disadvantages of Backpropagation Learning

The backpropagation learning algorithm has been quite successfully applied to a variety of research problems. It has been applied in optical character recognition systems, highly improving accuracy. It has been used in NetTalk, a system for converting printed text into intelligible speech. Backpropagation has been used successfully for recognizing handwritten letters.

The major advantage of the backpropagation learning algorithm is that its basic idea is quite simple and is very easy to implement in software. The backpropagation algorithm performs quite efficiently on modestly sized problems. It uses the gradient descent approach which has been well tested in the past.

However the backpropagation algorithm is not a panacea to all problems. It has quite a few disadvantages. The training process is long and uncertain. For complex problems, the training process might take days or weeks to converge [26]. This breakdown of the training process could be due to the network becoming trapped in local minima. The network could become "paralyzed", if the Weights adjust to a large value and forces the OUT values of neurons to be high in regions with low activation function gradients. Since the error propagated in the network is proportional to this gradient, the learning process could then effectively stop.

3.11 Implementation of the Backpropagation Neural Network Algorithm in CLASS

The backpropagation neural network algorithm implemented in CLASS is fairly comprehensive. As explained in section 3.5, a three-layered perceptron (three layers of neurons) is capable of solving a general cross section of problems. Hence, for the implementation in CLASS, we chose a three-layered perceptron.

Figure 3.12 shows the backpropagation neural network algorithm folder. When the user has chosen the backpropagation neural network algorithm for testing his data, this folder comes up on the screen after the chosen features have been calculated. The name of the datafile and its path are listed in the 'Database Information' section of the folder. The list of chosen features are available for viewing by the user. The user can choose the number of neurons in the hidden layer. The user chooses values for the training rate and the momentum coefficient. The user can also choose a batch size for his
This is the list of calculated features.

Number of Neurons in the Hidden layer
1

Training Rate Co-efficient (0.01 < x < 1.0)
0.7

Momentum Co-efficient
0.30

Batch Size
1

Current Processing Information
Not Processing anything at this time.

Figure 3.12 Backpropagation neural network folder
application. After all the options have been chosen by the user, he can begin execution by clicking on the 'Train/Test' button. The status of execution is conveyed to the user in the 'Current Processing Information' part of the folder. The user can terminate the backpropagation neural network algorithm by clicking on the command button containing the stop sign.

3.12 Results

The backpropagation neural network has been successfully tested on the XOR problem, a problem which single layered perceptrons were incapable of learning. Tables 3.2, 3.3 and 3.4 illustrate some of the results that were obtained with various values of the number of neurons in the hidden layer, training coefficient and the momentum for a batch size of one. From these tables, it is to be observed that the optimal value of the number of neurons in the hidden layer is around twice the size of the number of input neurons. A similar observation can also be made from the results for the parity problem.

### Table 3.2 Backpropagation neural network training on XOR with $\alpha = 0.3$ and $\eta = 0.7$

<table>
<thead>
<tr>
<th>Number of Neurons in the hidden layer</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
</tr>
</tbody>
</table>

3.12.1 The parity problem

Another problem, which the single layered perceptron was incapable of learning was the parity problem. In this problem, the required output is 1, if the input set contains an odd number of 1's and 0 otherwise. The XOR problem is a parity problem with an input size of two. Listings for the parity
Table 3.3  Backpropagation neural network training on XOR with $\alpha = 0.7$ and $\eta = 0.3$

<table>
<thead>
<tr>
<th>Number of Neurons in the hidden layer</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
</tbody>
</table>

problems of size 3 and 4 are shown in Tables, 3.5 and 3.6 respectively. In Table 3.5, X, Y and Z are the inputs the network and R is the output. In Table 3.6, W, X, Y and Z are the inputs the network and R is the output.

The backpropagation neural network algorithm written for CLASS was successfully tested on these problems. The results obtained are tabulated in Tables 3.7 and 3.8.

Figure 3.13 lists the results' report generated by the backpropagation neural network algorithm in CLASS for the XOR problem. In Figure 3.13, the column ID lists the index of the sample in the database, the column Classified Concept mentions the concept evaluated by the clustering algorithm and the column Actual Concept lists the actual concept of the sample.
Table 3.4 Backpropagation neural network training on XOR with $\alpha = 0.5$ and $\eta = 0.5$

<table>
<thead>
<tr>
<th>Number of Neurons in the hidden layer</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>6</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 3.5 Parity problem of size 3

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 3.6  Parity problem of size 4

<table>
<thead>
<tr>
<th>W</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 3.7  Backpropagation neural network training on parity problem of size 3 with $\alpha = 0.3$ and $\eta = 0.7$  

<table>
<thead>
<tr>
<th>Number of Neurons in the hidden layer</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 3.8  Backpropagation neural network training on parity problem of size 4 with $\alpha = 0.3$ and $\eta = 0.7$  

<table>
<thead>
<tr>
<th>Number of Neurons in the hidden layer</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>58</td>
</tr>
<tr>
<td>12</td>
<td>64</td>
</tr>
<tr>
<td>11</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
</tr>
</tbody>
</table>
## Backpropagation Neural Network Results

<table>
<thead>
<tr>
<th>ID</th>
<th>Classified Concept</th>
<th>Actual Concept</th>
<th>Correct Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 3.13  BPNN algorithm report generated by CLASS
4 FEATURE DOMAINS

Many measurements from Non-destructive testing are one-dimensional signals. In ultrasonic Non-destructive testing for example, the results are discrete voltage versus time signals over a certain time period. This raw data is in the “time domain”. It is often more revealing to deal with some other representation of this signal, rather than the raw time domain signal. Some properties or characteristics of the signal might not be easily seen from the raw time domain signals. A different coordinate system or a transformation of the raw data might better show the desired properties of the signal. CLASS 2.0 is capable of transforming the raw data into the Frequency and Cepstral domains. In the following sections we explain the extraction of the Frequency and Cepstral Domain data.

4.1 Sampling Discrete Time Signals

A discrete time signal is obtained from a continuous time signal, by sampling the continuous time signal at equal intervals of time. Given a set of discrete time signal, a number of continuous time signals could be associated with it. There is an ambiguity as to which of these continuous time signals the discrete time signal refers to. This ambiguity is known as aliasing. Considering the discrete time signal obtained at a rate of five samples per second,

\[ s_n = 1, -0.81, 0.31, -0.81, 1, -0.81, 0.31 \]  

(4.1)

for fitting a cosine curve, we could associate a number of continuous cosine curves with different frequencies with equation 4.1. Referring to Figure 4.1, we could have a curve of frequency 2 Hz as indicated by the solid line. We could also have obtained the same discrete time signal from a cosine wave of frequency 3 Hz, as shown by the dashed line in Figure 4.1.

To relate a unique signal with the discrete time sample, it is assumed that the signal is a low-pass band-limited signal. If the rate of sampling the discrete time signal is greater than twice the highest frequency present in the signal, there is a unique continuous time signal associated with the discrete time signal. This theorem is known as the 'Sampling theorem'. In the example illustrated in Figure
4.1. the 2-Hz cosine is related to the discrete time sample of equation 4.1. In Nondestructive testing, the sampling frequency is typically taken to be five to six times the highest frequency associated with the ultrasound used for the test [20].

4.2 Frequency Domain

Sinusoidal and exponential sequences are useful in the representation of time domain signals. This property is due to the fact that complex exponential sequences are eigenfunctions of linear time-invariant systems. Fourier representation of signals make use of weighted combinations of the “raw” signal components with different frequencies to expose the frequency content of the signal. The Fourier series and the Fourier transform are two important representations which will be discussed in this section.
4.2.1 Fourier series

The Fourier series is a representation of a signal \( x(n) \) as a superposition of harmonically related complex exponentials. The Fourier series coefficients for a signal \( x(n) \), using this notation, are given by

\[
    c_k = \frac{1}{N} \int_{-\frac{N}{2}}^{\frac{N}{2}} x(n) e^{-j \frac{2\pi k n}{N}} dn \quad (4.2)
\]

The Fourier series of \( x(n) \) is thus an expansion over the time interval \((-N/2, N/2)\). The set of complex coefficients \( c_k \) is called the spectrum of the signal \( x(n) \).

The Fourier series can also be thought of as the an expansion of a periodic signal with a period of \( N \):

\[
    x(n) = \sum_{m=-\infty}^{\infty} c_k e^{j \frac{2\pi kn}{N}} \quad (4.3)
\]

As more frequency components are added to equation 4.2, the error for the approximation in the representation of the actual signal reduces.

The Fourier series expansion of a function has some important properties which are listed below.

- When the signal is real, the spectrum has conjugate symmetry. The spectrum has even magnitude and odd phase.

\[
    |c_{-k}| = |c_k| \quad (4.4)
\]

and

\[
    \text{phase}(c_{-k}) = -\text{phase}(c_k) \quad (4.5)
\]

- The following truncated Fourier series is the best approximation of \( x(t) \), such that the integrated squared error, \( E \) is minimized.

\[
    \hat{x}(n) = \sum_{m=-M}^{M} c_k e^{j \frac{2\pi kn}{N}} \quad (4.6)
\]

\[
    E = \int_{-\frac{N}{2}}^{\frac{N}{2}} |x(n) - \hat{x}(n)|^2 dn \quad (4.7)
\]

- Parseval's theorem for a signal \( x(t) \) states that the energy of the signal computed in either the time domain or the frequency domain is the same.

\[
    \frac{1}{N} \int_{-\frac{N}{2}}^{\frac{N}{2}} x^2(n) dn = \sum_{k=-\infty}^{\infty} |c_k|^2 \quad (4.8)
\]
4.2.2 The discrete Fourier transform

The discrete time counterpart of the Fourier series is the discrete Fourier transform. The discrete Fourier transform is a Fourier representation of discrete time signals. The discrete Fourier transform of a signal describes the periodicity of the signal. If we have a discrete time signal $x_n$ that has a periodicity of $N$, then $x_n = x_{n+rN}$ for any integer value of $r$.

The discrete time signal $x_n$ can be presented by a Fourier series corresponding to a sum of harmonically related complex exponentials. The frequencies of the complex exponentials are integer multiples of the fundamental frequency $\frac{2\pi}{N}$. The periodic exponentials have the following form for integer values of $k$:

$$e_k[n] = e^{j\left(\frac{2\pi}{N}kn\right)} = e_k[n + rN] \quad (4.9)$$

The Fourier series is then represented as

$$x[n] = \frac{1}{N} \sum_k X_k e^{j\left(\frac{2\pi}{N}kn\right)} \quad (4.10)$$

Each component of the discrete Fourier transform is the amount of the signal that varies at some specific frequency. A difference between the Fourier series representation of a continuous time signal and a periodic discrete time signal is that the continuous time signal requires infinitely many harmonically related complex exponentials while a periodic discrete time signal with period $N$ needs only $N$ harmonically related complex exponentials. This follows from equation 4.9, where the harmonically related exponentials $e_k[n]$ are identical for values of $k$ separated by $N$, i.e

$$e_k[n] = e_{k+rN}[n] \quad (4.11)$$

for integer values of $r$. The set of $N$ exponentials $e_0[n], e_1[n], \ldots, e_{N-1}[n]$ is the set of harmonically related complex exponentials with frequencies that are integer multiples of the fundamental frequency $\frac{2\pi}{N}$. To obtain the Fourier series of the discrete time signal $x_n$, we need only $N$ of the above complex exponentials. The Fourier series of a periodic discrete time signal with a periodicity of $N$ is obtained as

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\left(\frac{2\pi}{N}kn\right)} \quad (4.12)$$

Using the orthogonality of the set of complex exponential sequences, the Fourier series coefficients $X_k$ may obtained as

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\left(\frac{2\pi}{N}kn\right)} \quad (4.13)$$
Defining the complex exponent \( W_N \) as
\[
W_N = e^{-j \frac{2\pi}{N}}
\] (4.14)
we have equation 4.12 as
\[
x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_{N}^{-kn}
\] (4.15)
and equation 4.13 as
\[
X[k] = \sum_{n=0}^{N-1} x[n] W_{N}^{kn}
\] (4.16)

### 4.2.2.1 Properties of discrete Fourier series

The Discrete Fourier Series has some important properties, which are useful in signal processing problems. Many of the basic properties of the discrete Fourier series are analogous to properties of the Fourier transform. However the periodicity of the \( x_n \) and \( X_k \) results in some distinctions. One such important distinctive property of the discrete Fourier series is the duality between the time and the frequency domains in the discrete Fourier series.

- **Linearity:**

  Consider two discrete time signals \( x_1[n] \) and \( x_2[n] \) and their corresponding discrete Fourier series \( X_1[n] \) and \( X_2[n] \) respectively.

  \[
x_1[n] \leftrightarrow X_1[k]
\] (4.17)

  Then the discrete Fourier series of the the discrete time signal given by

  \[
a x_1[n] + b x_2[n] \leftrightarrow a X_1[n] + b X_2[n].
\] (4.18)

  The above relation follows from equations 4.14 and 4.15.

- **Shift of a sequence:**

  If the discrete time signal \( x[n] \) has discrete Fourier series \( X[k] \), then \( x[n-m] \) has a shifted version of \( X[k] \) as its discrete Fourier series.

  \[
x_1[n - m] \leftrightarrow X_1[k] W_{N}^{km}
\] (4.19)

  The above relation follows from the definition of Fourier coefficients. A similar shift exist in the frequency domain too

  \[
x_1[n] W_{N}^{-nl} \leftrightarrow X_1[k - l]
\] (4.20)
• Duality:

Observing equations 4.15 and 4.16, it is seen that they differ in a factor of $\frac{1}{N}$ and in the sign of the complex exponent $W_N$. Both the equations are periodic. Multiplying equation 4.15 by $N$ and reversing the sign of the complex exponent, we have

$$Nx[-n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]W_n^k$$

(4.21)

Interchanging the roles of $n$ and $k$, we have

$$N[-k] = \frac{1}{N} \sum_{k=0}^{N-1} X[n]W_n^k$$

(4.22)

We see that equation 4.22 is similar to equation 4.15. Thus there is a duality between the discrete Fourier series of a discrete time signal and the discrete time signal.

• Periodic Convolution:

If we have two discrete time signals $x_1[n]$ and $x_2[n]$ with the corresponding discrete Fourier series being $X_1[k]$ and $X_2[k]$, then the discrete Fourier series obtained by their product

$$X_3[k] = X_1[k]X_2[k]$$

(4.23)

is the Fourier representation of the product

$$x_3[n] = \sum_{m=0}^{N-1} x_1[m]x_2[n-m]$$

(4.24)

A convolution of the above type is known as the periodic convolution.

4.2.3 Computation of the discrete Fourier transform

One way of computing the Discrete Fourier transform is the direct way. In equation 4.16, we represent all the complex values in the Cartesian form and equate the real and imaginary components on the left hand side and the right hand side. This direct computation takes order $N^2$ computations for evaluating a discrete Fourier transform for a signal with periodicity $N$. This is not a very efficient way of computing the Discrete Fourier transform. The class of algorithms for efficiently evaluating the Discrete Fourier transform are known as Fast Fourier transform algorithms. An important category of Fast Fourier algorithms is the decimation in time fast Fourier transform algorithms. In these algorithms, the discrete Fourier transform is computed by successively decomposing the signal $x[n]$ into smaller sequences. They follow a divide and conquer strategy which turns out to effectively reduce the computation time. The following subsection discusses the FFT algorithm implemented in CLASS – the Cooley-Tukey FFT algorithm [17]. Figure 4.2 shows the flowgraph for the computation of an eight point DFT.
4.2.4 The Cooley-Tukey FFT algorithm

The strategy in the Cooley-Tukey decimation in time FFT is to separate the N point discrete time signal \( x[n] \) into two \( \frac{N}{2} \) point signals consisting of the even and odd numbered points in \( x[n] \). The discrete Fourier transform for this signal may now be written as

\[
X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_N^{(2r+1)k} \tag{4.25}
\]

But since \( W_N^2 = W_{\frac{N}{2}} \)

\[
X[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_{\frac{N}{2}}^{rk} = G[k] + W_N^k H[k] \tag{4.26}
\]

Each of the sums in 4.26 is an \( \frac{N}{2} \) discrete Fourier Transform. Although the index \( k \) ranges over 0 to \( N - 1 \), \( G[k] \) and \( H[k] \) must be computed over 0 to \( \frac{N}{2} - 1 \), since \( G[k] \) and \( H[k] \) are both periodic with a period of \( \frac{N}{2} \). The diagram in Figure 4.2 shows the decomposition of an 8 point DFT into two 4 point DFTs.

The process of decomposing the N point DFT into two \( \frac{N}{2} \) point DFTs can be further extended to decompose the \( \frac{N}{2} \) point DFTs to \( \frac{N}{4} \) point DFTs and so on, until we are left with only two point DFTs. If \( N = 2^v \), we can have \( v \) decompositions of the N point DFT while performing computations of the order \( N \) at each stage. Thus, this scheme of evaluating the discrete Fourier transform takes \( N \log_2 N \) computations totally.
The computation of the DFT is further be reduced by exploiting the periodicity of the coefficients $W_N^m$. From the figure of the basic 'butterfly computation' in Figure 4.3,

$$W_N^m = e^{-j\frac{(m-1)\pi}{2}} = e^{-j\pi} = -1$$

(4.27)

For the non-sequential order of the discrete time signal of Figure 4.2, the computations are all in-place and require just one array for storing the computed results at each stage. This follows from the following equations and the Figure 4.4, which illustrate the computations involved in calculating the outputs at the $m^{th}$ stage. Thus, $X_m[p]$ and $X_m[q]$ are stored in the same array as $X_{m-1}[p]$ and $X_{m-1}[q]$.

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q]$$

(4.28)

$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q]$$

(4.29)

The non-sequential order of the discrete time signal of Figure 4.2 is known as the 'bit-reversed' order. It is accomplished by writing the indices in the binary form and reversing the order of the digits. The indices of the discrete time signal are then ordered in this bit reversed form. If $(n_2, n_1, n_0)$ is the binary representation of the index of the discrete time signal $x[n]$, then $x[n_2, n_1, n_0]$ is stored in the array position $X_0[n_2, n_1, n_0]$.

To obtain bit-reversing, first the least significant digits of the indices are compared. If the least significant digit is 0, then the discrete value will appear in the top half and if the least least significant digit is 1, then the discrete value will appear in the bottom half. Then the second least significant digit
is compared in a similar way to separate the odd and even indexed discrete values of the two different sets obtained by comparing the least significant digit. This sorting is explained in the Figure 4.5.

Once bit reversing has been achieved the discrete Fourier transform is calculated. The final computations are shown in the flowgraph in Figure 4.6.

4.3 Cepstrum Analysis

The frequency spectrum of a time-domain signal can be subjected to a spectral analysis itself. This second spectrum reveals the periodicity of the frequency spectrum. It is very useful in analyzing signals whose spectra exhibit an oscillatory behavior. This extension of the spectral concept might be formulated as follows:

\[ f(t) \leftrightarrow F(\omega) \leftrightarrow \tilde{f}(l) \]  

(4.30)

Cepstrum analysis was first used by Bogert, Healy and Tukey in 1963 [18]. The term cepstrum is obtained by paraphrasing the word spectrum. The frequency counterpart of the cepstrum is called the quefrency. Bogert, et al. used the following definition to obtain the cepstrum of a signal:

\[ \tilde{f}(l) = \int_{-\infty}^{\infty} \log|F(\omega)|^2 e^{-i2\pi l \omega} d\omega \]  

(4.31)

The advantage of using the logarithm is that if \( F(\omega) \) is a product of two or more factors, the cepstrum will be a sum, which exhibits the various contributions with a better separation. Cepstrum analysis has been used in seismological applications.
Figure 4.5 Tree diagram depicting bit reversed sorting
Figure 4.6 Flow graph of 8 point discrete Fourier transform
4.4 Implementation of Frequency Domain and Cepstral Domain in CLASS

The Cooley-Tukey FFT algorithm described in section 4.2.4 was implemented in CLASS. For this implementation, the database tables containing the time domain data were modified. Additional fields for storing the magnitude and the phase components of the frequency domain data were created. If the user had chosen the “Frequency Domain” transform from the list of transforms to be computed, the frequency components of the raw data would be computed. The frequency components of the time domain data would also be computed, if the user had chosen the “Cepstral Domain” transform from the list of transforms to be computed, since the frequency domain data is needed for the computation of the cepstral domain data. Data in the Cepstral domain is obtained by using a discrete version of the continuous relation shown in Equation 4.31. The Cepstrum analysis described in the preceding chapter is performed on the frequency domain data when the user has chosen the “Cepstral Domain” transform from the list of transforms to be calculated. This “quefrency” component is then stored in the “Cepstral Domain Data” field in the database table containing the raw data.
5 CONCLUSIONS AND FUTURE WORK

Both a clustering algorithm and a backpropagation neural network have been added to CLASS and tested on a variety of problems to validate that they are working properly. The ability to generate frequency domain data (and corresponding cepstral domain data) was also added to CLASS. The FFT routine implemented was again tested on examples where the frequency domain values were known. User interfaces were generated for applying these new learning algorithms and generating and storing the frequency domain domain data, following the same "look-and-feel" of the original CLASS interfaces.

The original version of CLASS, CLASS 1.0, was designed by Loren Knutson to have "user-friendly" interfaces and to be easily extended. This was accomplished through a highly modularized design of the software. The experience obtained in writing CLASS 2.0 has confirmed that the original design goals for CLASS were substantially met. The only major problem found in CLASS 1.0 was the use of very cryptic and non-informative variable names in some of the modules. Thus, the revised version of CLASS, CLASS 2.0, has attempted to retain the basic design of CLASS 1.0. However, in developing CLASS 2.0, it was found useful to integrate the "Transformations" and "Frequency" modules. This integration has increased the computational efficiency of CLASS as the same data used by both modules need not be reread from and rewritten into the database file.

Although, a number of important additions have been made to CLASS, there is still much to be done to improve and extend the software. First, new frequency domain features need to be defined and added to CLASS to take advantage of the new ability to compute frequency domain values. Second, it would be useful to add even more learning algorithms to CLASS. It has been our experience (as well as the experience of others) that the performance of learning algorithms is highly dependent on the nature of the problem involved and the features available for classification. Thus, it is important to give a user an environment where different choices of features and classification methods can be made easily and compared.

Finally, we should point out that currently CLASS can handle real one-dimensional ultrasonic data. For complete flexibility, CLASS could be extended to handle complex data, as found in eddy current
tests, for example, and have some facilities for manipulating and displaying these different data types. It also would be useful to use some of the algorithms of CLASS, such as decision trees, to try to generate rule-based expert systems that could easily be used in "production" applications.
Using CLASS

CLASS provides a very intuitive graphical user interface. On starting CLASS, the folder shown in Figure A.1 comes up on the screen. The user should then choose the location of a data file from the appropriate path and file lists by double clicking on the file name or by dragging the filename into the box titled, 'Current Raw Data File Name with the mouse. Once the file has been chosen, the user can view or edit the 'HeaderInfo' table of the database file, using the 'View/Edit File' button option. In the 'View/Edit File' folder, there are two modes available for viewing and editing the header information contained in the datafile. The first mode for editing or viewing the header information is the spreadsheet mode. The second mode is the graph-sheet mode. These modes are obtained by clicking the appropriate icons present on the top right corner of the folder. The user then gets a list of the available transformations by clicking on the grey button which says 'Transformations'. The user can check off the transformations he wishes to use for the current application. The next step is to choose the features for the current application. The available list of features can be obtained on the button titled 'Features/Filters'. The user can then check off the names of the features needed for this application, by clicking against the appropriate feature name(s). The user can obtain a list of the available learning algorithms by clicking the button titled 'Learning Algorithms'. This brings up the Learning Algorithms folder which lists the available learning algorithms under two categories – Supervised learning algorithms and Unsupervised learning algorithms. The user can click off a learning algorithm of his choice. In this work, we discuss the folders for the clustering algorithm, the backpropagation Neural Network algorithm and the transforms folder. For a more detailed description of the basic CLASS folders and the folders for the other learning algorithms available in CLASS, the reader might refer to reference [14].
Clustering Algorithm

If the user had chosen the clustering algorithm for testing a datafile in the execution of CLASS, the clustering algorithm folder comes up on the screen after the calculation of the chosen features and transforms. A screen saver of this folder is shown in Figure A.2.

Information about the datafile is displayed in the 'Database Information' section of the folder. The user can also obtain a list of the calculated features. The user, then has to make the choice of a distance measure. The distance measure available in CLASS are:

- Nearest Neighbors
- Furthest Neighbors
- Average Distance

After making the choice of an appropriate distance measure, the user can begin the execution of the clustering algorithm by clicking on the 'Train/Test' button. Information about the about the status of the processing is displayed in the 'Current Processing Information' part of the folder. After the training and testing have been completed, the user can exit the clustering algorithm.
Figure A.2 Clustering algorithm folder
Backpropagation Neural Network Algorithm

While running CLASS, if the user had chosen the backpropagation neural network algorithm for testing the datafile of his choice, on completion of the computation of the chosen features and transforms, the backpropagation neural network folder comes up on the screen. This folder for the backpropagation neural network algorithm is shown in Figure A.3.

In the backpropagation neural network folder, the user has 4 choices to make. The first choice is the number of neurons in the hidden layer. The backpropagation neural network algorithm in CLASS uses three layers of neurons. The input layer contains as many neurons as the number of features chosen. The output layer contains as many neurons as the number of concepts for the current application. The default value for the number of neurons in the hidden layer is one more than twice the number of neurons in the input layer. The next choice that the user has to make is the training rate coefficient. Its value lies between 0.01 and 1.0. The usual value for the training rate is around 0.7. Choosing a low value of the training rate results in slower training. Then the user chooses a value for the momentum coefficient. The final option that the user has to fill in is the batch size.

After making all of the above choices, the user can begin execution of the backpropagation neural network algorithm by clicking on the command button titled 'Train/Test'. Information about the status of execution is displayed in the 'Current Processing Information' section of the folder. After the training and testing of the data has been performed, the user can view the results in a 'Crystal Reports' form. This forms has four columns. The first column indicates the index of the data sample. The second column indicates the concept that has been found from the classifier. The third column contains the actual concept and the fourth column in the report indicates whether a correction classification has been made by the classifier for the data sample in question.
Figure A.3  Backpropagation neural network folder
BIBLIOGRAPHY


