A radial basis neural network for the analysis of transportation data

David P. Aguilar

University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
A Radial Basis Neural Network for the Analysis of Transportation Data

by

David P. Aguilar

A thesis submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Rafael Perez, Ph.D.
Srinivas Katkoori, Ph.D.
Dewey Rundus, Ph.D.

Date of Approval:
November 4, 2004

Keywords: radial basis function, learning optimization, gaussian units, back propagation, regression analysis

© Copyright 2004, David P. Aguilar
Table of Contents

List of Tables iii

List of Figures v

Abstract vii

Chapter 1. Introduction 1
  1.1. Motivation for this Thesis 2
  1.2. Contribution to the Discipline 3
  1.3. Structure of Thesis Report 4

Chapter 2. Theoretical Foundations 5
  2.1. The Basics of Neural Networks 5
  2.2. The Radial Basis Function 8
  2.3. Training Methods 13
    2.3.1. Output Layer – The Delta Rule 13
    2.3.2. Hidden Layer 1 – Movable Centers 14
    2.3.3. Hidden Layer 3 – Adjustable Sigmas 15
  2.4. Training Algorithm 16

Chapter 3. Implementation and Training 18
  3.1. Program Specifications 18
    3.1.1. Program Overview 18
    3.1.2. The File System 20
    3.1.3. The Train Function 21
    3.1.4. The Test Function 27
    3.1.5. The Run Function 28
    3.1.6. The Classification and Report Module 29
  3.2. Implementation of Mathematical Formulae 31
  3.3. Basic Functions A – Delta Rule Training (Fixed Centers) 33
  3.4. Basic Functions B – Delta Rule vs. Fixed Centers and Delta 35
  3.5. Basic Functions C 37
    3.5.1. Manually Adjusting the Sigma 38
    3.5.2. automatically Adjusting the Sigma 39
  3.6. Analysis of Preliminary Training (OR, AND, XOR) 41
Chapter 4. Application Initialization 42
  4.1. Description of Data 42
  4.2. CUTR Analysis Results 43
    4.2.1. Forward Regression 43
    4.2.2. Backward Regression 43
    4.2.3. Stepwise Regression 44
    4.2.4. Force-entered Regression 44
  4.3. Selection of Network Architecture 48
    4.3.1. Number and Selection of Inputs 48
    4.3.2. Number of Neurons and Epochs 48

Chapter 5. Execution and Analysis 53
  5.1. Training and Testing Results on Application Data 53
  5.2. Analysis of Performance 56

Chapter 6. Performance Comparisons 58
  6.1. Comparison with Regression Results 58
  6.2. Comparison with Backpropagation Networks 59

Chapter 7. Conclusions 61
  7.1. Final Analysis 61
  7.2. Future Work 62

References 64

Appendices 66
  Appendix A: Table of Incentive Variables and Dependent Variable 67
  Appendix B: Code Modules 68
  Appendix C: Sample Files 71
List of Tables

Table 1. Level 1 Training Results During Network Testing 35
Table 2. Level 2 Training Results During Network Testing 37
Table 3. SSE and Accuracy After 300 Epochs For Level 1 Training 38
Table 4. SSE and Accuracy After 300 Epochs For Level 2 Training 38
Table 5. Level 3 Training Results During Network Testing 41
Table 6. Summary of Training Results on Binary Functions 41
Table 7. Classification of Data and The Bin Ranges 42
Table 8. Force-Entered Regression Results 45
Table 9. Force-Entered Network Results (0 Hidden Units) 45
Table 10. Backward Regression Results 45
Table 11. Backward Regression Network Results (0 Hidden Units) 46
Table 12. Backward Network Results (With Hidden Units) 46
Table 13. Forward Regression Results 46
Table 14. Forward Regression Network Results (0 Hidden Units) 46
Table 15. Forward Regression Network Results (With Hidden Units) 47
Table 16. Stepwise Regression Results 47
Table 17. Stepwise Regression Network Results (0 Hidden Units) 47
Table 18. Stepwise Regression Network Results (With Hidden Units) 47
Table 19. Stepwise Regression Network Results With Costs 48
Table 20. Stepwise Results: Hidden Units/Reduced Dataset 48
Table 21. Varying Percentage of Data For Neuron Number (Bin3) 50
Table 22. Varying The Number of Epochs For Classifying Bin 3 52
Table 23. Network Training Results 53
Table 24. Network Classification Results 54
Table 25. Error Analysis 55
Table 26. Comparison of RBF Network and Regression Results 58
Table 27. Comparison of RBF Network and Regression Networks 1 59
Table 28. Comparison of RBF Network and Regression Networks 2 59
Table 29. Table of Incentive Variables and Dependent Variable 67
List of Figures

Figure 1. Linear Separability in 2D Space and 3D Space 6
Figure 2. A Two-Layer Network Trained For The XOR Function 7
Figure 3. Threshold and Sigmoid Functions vs. Gaussian Function 9
Figure 4. TLU Decision Boundaries vs. Gaussian Boundaries 10
Figure 5. A Radial Basis Function (RBF) Neural Network 10
Figure 6. The XOR Problem in 2 and 3 Dimensional Space 12
Figure 7. The Main Menu of The Network Interface 19
Figure 8. Training Level 1 – Adjusting Output-Layer Weights 22
Figure 9. Training Curve Displaying The SSE Values 23
Figure 10. Network Diagram Output 24
Figure 11. Training Level 2 – Adjusting Weights and Centers 25
Figure 12. Training Level 3 – Adjusting Weights, Centers and Sigmas 26
Figure 13. Flowchart of The Test Function 27
Figure 14. Diagram of The Run Function Algorithm 29
Figure 15. SSE Curves For OR, AND and XOR Respectively (Set 1) 34
Figure 16. SSE Curves For OR, AND and XOR Respectively (Set 2) 36
Figure 17. SSE Curves For OR, AND and XOR Respectively (Set 3) 40
Figure 18. Diagram of Training Results with Variable Neurons 51
Figure 19. Effects of Training on SSE Values 54
Figure 20. Effects of Training on Accuracy 54
Figure 21. Radial Basis Networks and Regression Analysis 58
Figure 22. Radial Basis Networks and Backpropagation Analysis 59
Figure 23. Radial Basis Networks / Hidden Backpropagation Analysis 59
A Radial Basis Neural Network for the Analysis of Transportation Data

David P. Aguilar

ABSTRACT

This thesis describes the implementation of a Radial Basis Function (RBF) network to be used in predicting the effectiveness of various strategies for reducing the Vehicle Trip Rate (VTR) of a worksite. Three methods of learning were utilized in training the Gaussian hidden units of the network, those being a) output weight adjustment using the Delta rule b) adjustable reference vectors in conjunction with weight adjustment, and c) a combination of adjustable centers and adjustable sigma values for each RBF neuron with the Delta rule. The justification for utilizing each of the more advanced levels of training is provided using a series of tests and performance comparisons.

The network architecture is then selected based upon a series of initial trials for an optimum number of hidden Radial Basis neurons. In a similar manner, the training time is determined after finding a maximum number of epochs during which there is a significant change in the SSE.

The network was compared for effectiveness against each of the following methods of data analysis: force-entered regression, backward regression, forward regression, stepwise regression, and two types of back-propagation networks based upon the attributes discovered to be most predictive by these regression techniques.
A comparison of the learning methods used on the Radial Basis network shows the third learning strategy to be the most efficient for training, yielding the lowest sum of squared errors (SSE) in the shortest number of training epochs. The result of comparing the RBF implementation against the other methods mentions shows the superiority of the Radial Basis method for predictive ability.
Chapter 1

Introduction

According to the Air Resources Board established in 1967, the emission of cars and trucks is the single most significant factor contributing to the air quality issue facing urban areas within the state of California. Although the cars and trucks produced today generate significantly less pollutants than those manufactured in the 1970s, Californians continue to lose billions of dollars a year due to air pollution and its related problems [1]. These problems are not only directly related to human health, but also involve environmental concerns affecting both wildlife and cultivated resources, therefore this issue is of consequence to a wide range of interest groups.

The technology involved in creating more fuel-efficient vehicles has come a long way, however one of the most effective methods of maintaining the standards of air quality is to simply burn less fuel. In a highly motorized area such as Los Angeles, however, most people prefer the convenience of providing their own transportation to and from their places of employment. In order to offset this tendency, many large companies have attempted to implement “trip reduction strategies,” which provide incentives for individuals to use public transportation, group transportation or non-automotive means of travel such as bicycles. Wherever possible, telecommuting programs are also introduced.

The Air Resources Board (ARB) and other agencies have suggested a wide array of such strategies, and in order to maintain the pollution level below acceptable standards, have required large employers to submit plans of action which they must undertake in
order to influence their Average Vehicle Ridership (AVR), which is the ratio of employees to motor vehicles that arrive at a given worksite. An increase in the AVR level, which corresponds to a decrease in the Vehicle Trip Rate (VTR) discussed during the course of the thesis, indicates an effective strategy. The large volume of data collected on these various methods to date facilitates the detailed analysis of these techniques’ effectiveness using various the predictive methods at our disposal.

1.1. Motivation for this Thesis

Even a cursory glance at a spreadsheet containing the obtained information on trip reduction strategies reveals the problem associated with determining the effectiveness of any particular method. A large set of factor combinations and the difficulty in determining the degree to which each factor influences the outcome of the strategies result in a significant barrier facing those companies wishing to implement effective incentive plans to increase their average vehicle ridership (AVR). The issue facing large employers in light of this situation is how to provide the best set of incentives for their employees in order to affect compliance with emissions regulations. Most of the ARB suggestions are not inexpensive, and it is therefore essential that the data accumulated thus far be examined for useful information.

A number of methods have been proposed and implemented for the analysis of this particular set of data, however the results have not been spectacular. Statistical techniques such as regression analysis and backpropagation neural networks have predicted the effectiveness of the various incentives with levels of accuracy in the order
of 20%, although a recent RBF network using the techniques that form the basis for this thesis has managed to increase that number somewhat [2].

Radial Basis networks have shown the most promise so far in their ability to find the relationship between the sets of programs and their impact on VTR. The purpose of this thesis is to continue exploration in that area, utilising a refined data set and an alternative statement of the problem than has been previously used in an attempt to obtain more accurate prediction results. The thesis also presents a comparison of the results of this procedure with previous attempts at building a model of the data, as well as more recent attempts using other approaches to the analysis of this current, reduced data set.

1.2. Contribution to the Discipline

A further exploration of the Radial Basis Function network, which has been the most effective means thus far of building a model of the data, is undertaken in this body of work. Aside from refinements made to the data set upon which the model is being built, the problem statement and results themselves are stated in a new way. Levels of effectiveness are divided into 8 stages or “bins,” and one approach is to use the RBF networks as binary classifiers, to determine which of these levels of impact the data records (which indicate combinations of incentive programs) fit. Another innovation allowed by converting the analysis into a set of binary functions is the ability of the implemented software to select its own threshold for the admission of a record into a particular bin. Thus, the maximum possible percentage accuracy is obtained by any given method of training for the initial neural network.
The results of the thesis confirm that the more sophisticated methods of RBF network training, involving the movement of reference vectors within the neurons and the adjustment of sigma values controlling the slope of the Gaussian (comparison) function, are most effective. They also reveal the effectiveness of using Radial Basis networks as classifiers in dealing with large, sophisticated patterns of attributes within a data set.

1.3. Structure of Thesis Report

The remainder of the thesis report is organized as follows: The basic theory of Artificial Neural Networks is discussed in chapter 2. Also presented there are the benefits of the Radial Basis Function, the training methods associated with networks using this technique, and the algorithms used during in the course of this thesis. Chapter 3 presents the specific implementation used to study the transportation data, providing initial results and comparisons of the training methods on simple binary.

Chapter 4 describes the data set dealing with the various incentives being studied, the methods used to reduce and refine this information, and the procedures used in preparing the data for network analysis. Chapter 5 gives the results of training and testing the data on the RBF networks, as well as a statistical analysis of the various methods. Chapter 6 compares these results with alternative examination techniques used by the Center for Urban Transportation Research (CUTR). Chapter 7 gives the conclusions of the thesis and a discussion of potential future work.
Chapter 2
Theoretical Foundations

The basic principles of Artificial Neural Networks and their structures are discussed in this chapter, as well as the factors influencing the decision to use Radial Basis Functions for analysing the transportation data. The structures and mathematical bases of the implemented network are discussed in detail, providing the formulas used and a brief description of how they were derived. The training methods provided by the software implementation are introduced, and the specific algorithm used is described.

2.1. The Basics of Neural Networks

Put in simple terms, an Artificial Neural Network (ANN) is a software application that alters certain variables in response to a set of corresponding input and output patterns. Beginning with an initial set of internal values, the network modifies these quantities in order to find a position of “best fit,” thereby generating from the input patterns their expected results.

For example, a basic network can be trained to act as an AND logic gate, giving an output of “1” when the binary input vector (1, 1) is applied.

Even the most simple of these networks can quickly learn to identify sets of patterns that exhibit the quality of “linear separability.” This indicates that the outputs can be classified according to a decision boundary defined by the equation:

$$y = ax_1 + bx_2 + cx_3 \ldots + nx_n + z$$  \hspace{1cm} (1.1)
where \((n+1)\) is the number of dimensions of the input vector. This can be determined and demonstrated graphically, as in Figure 1, by attempting to find a line, plane or hyperplane (depending on the dimensionality of the input pattern) that separates the outputs into distinct classes.

![Figure 1. Linear Separability in 2D Space and 3D Space](image)

Most classification problems, however, are far more complex than this, and the networks are required to be correspondingly more sophisticated in order to correctly identify these patterns. A simple binary function such as XOR, for example, is non-linearly separable, and therefore presents a problem for single-layer systems or “perceptrons.” The solution to this is to add another layer, which acts as an inhibitor, in order to more effectively define a specific solution space. Theoretically, a 3-layer neural network is sufficiently advanced to separate any grouping of continuous inputs, provided the network is large enough [3].

Multi-layer networks, such as that shown in Figure 2, also called feed-forward networks or multi-layer perceptrons, have experienced a great deal of success in solving problems where a large amount of data is involved. The ability of the networks to generalize relationships between inputs and outputs is key to their effectiveness. The
limitation of multi-layer networks is found in the absence of a method to efficiently train them to adapt to new data. Due to the inherent complexity of the architecture, deciding which values to adjust in order to arrive at more accurate classifications is an NP-complete problem in a network consisting of only 3 nodes or “neurons” [4].

![Figure 2. A Two-Layer Network Trained For The XOR Function](image)

Algorithms such as the Back Propagation method have been effective at providing high levels of accuracy on difficult problems of various types, and although certain inefficiencies in these algorithms have been pointed out recently, refinements are being made with considerable success [5]. For all these related algorithms, the basic method of learning consists of calculating an “error term” dependent on the difference between the expected and actual outputs, adjusting the weights between the neurons in order to lower that term, and propagating the effects of the error term through previous layers of weights.

This process is repeated until the level of difference between obtained and expected outputs is within an acceptable boundary [6]. The applications of the resulting multi-layer perceptron systems are found in diverse fields of industry and research, and the numerous refinements and modifications to the basic algorithms that exist [7] are often used to carry out highly specialized operations.
2.2. The Radial Basis Function

The basic model of multi-layer neural networks utilizes either sigmoid or threshold neurons. These threshold logic units (TLUs) generate a signal, or “fire,” when the sum of their input signals exceeds a certain level. Sigmoid units provide a more graceful degradation of output signals, and because the derivative of this function can be taken the back propagation method discussed above, which relies on this value, can be implemented for training purposes.

Radial Basis Function (RBF) neural networks operate on a somewhat different principle. Instead of having threshold units with a single value against which to compare accumulated sums of input signals, each RBF neuron has a set of values called a “reference vector” for comparison with an input set of the same cardinality. The driving formula of each neuron used in this thesis work is the multivariate Gaussian function:

\[ 
\phi(r) = e^{-\frac{1}{2\sigma_j^2}||x - t_j||^2} 
\]  

(2.1)

Where \( x \) is the input vector for the neuron, \( t_j \) is the set of reference values, \( \sigma_j \) is the standard deviation (\( \sigma^2 \) is the variance) of the function for each of the centers (\( j \)), and the value \( r (||x - t_j||) \) is the Euclidean distance between a center vector and the set of data points [6][8].

The output of each neuron is reflective of the similarity between input and reference vectors according to the “bell” curve shown in Figure 3 below. A high degree of similarity produces an output (\( \phi \)) that approaches 1. Figure 4 provides a graphical representation of the type of classification ability afforded by a two-level threshold logic system compared with a radial basis function.
It should be understood that the Gaussian function is not the only selection available for RBF networks. Other formulas considered typical are the thin-plate-spline function:

\[ \varphi(r) = r^2 \log(r) \]  
(2.2)

the multiquadric function:

\[ \varphi(r) = (r^2 + \beta^2)^{1/2} \]  
(2.3)

and the inverse multiquadric function:

\[ \varphi(r) = 1/(r^2 + \beta^2)^{1/2} \]  
(2.4)

The justification for using the standard Gaussian units for this thesis work is found in the property of the networks using this formula to effectively make local adjustments; they adapt to new data without significantly altering the results generated from the input patterns that have already been learned. This local refinement property makes it ideal for use on data with a considerable number of input fields and a large set of records [8].
The most common architecture for an RBF network, which is the one used in the software associated with this thesis, is the fully connected, single hidden layer arrangement, an example of which is provided by Figure 5.

In such an arrangement, each of the neurons of the hidden layer operate on the Gaussian formula given above, comparing a reference vector of a length equal to the number of inputs. The output layer in this case consists of a single neuron, although radial basis networks are fully capable of supporting numerous such units, and is defined by a linear function of the hidden units’ φ values. Networks such as this are able to perform complex pattern recognition tasks by transforming the data into higher-
dimensional space in which, according to Cover’s separability theorem [9], makes finding linear separability for classification more likely.

A simple example of this can be demonstrated by use of the XOR function discussed in Section 2.1. As Figure 2 illustrates, no single, linear decision boundary can be drawn that will separate the two types of data (designated by red and green dots) into distinct classes. If we add a third dimension, however, which is a function of the inputs, a boundary between the two classes can be found.

The set of input values $x$ for a binary XOR function may be listed as follows:

$x_1 = (0,0); x_2 = (0,1); x_3 = (1,0); x_4 = (1,1)$. The expected outputs of the function would therefore be 0, 1, 1, and 0 respectively. During Haykins’ discussion of Cover’s theorem, he demonstrates the Gaussian function’s ability to differentiate between the two classes of inputs with only two neurons [6]. A similar (and simplified) network producing identical results may be obtained, for purposes of explanation, by defining a function:

$$\varphi(x_i) = ||x_{i1} - x_{i2}||$$ (2.5)

resulting in the mapping:

$x_1 = (0,0) \rightarrow 0$

$x_2 = (0,1) \rightarrow 1$

$x_3 = (1,0) \rightarrow 1$

$x_4 = (1,1) \rightarrow 0$

This allows a bisection of the classes of inputs provided, and in this simple example the resulting $\varphi$ of each input set corresponds precisely with the expected outputs of the XOR function. By providing a function of the existing inputs as another dimension
of the input vector, a plane can be found in the resulting 3-dimensional space that allows linear classification to take place (see Figure 6).

Figure 6. The XOR Problem in 2 and 3 Dimensional Space

In feedforward networks, this is done by adding another layer of neurons, or by preprocessing the data, extracting derived functions before applying the inputs to the neurons [10]. In an RBF network, this principle applies in the following way: each of the hidden neurons contains a complete reference vector corresponding to the number of inputs. Each comparison between the input vector and these reference vectors represents a dimension for use by the output layer in determining classification. When the number of neurons is equivalent to the cardinality of the input vector, each neuron may be used to respond to a single pattern of inputs. While this is ideal where there are limited, finite input patterns, a network that has become familiar with the input variables to this degree is often not optimal for continuous data.

If a network perfectly classifies training data, yet is unable to accurately recognize the patterns when provided with new but similar input patterns, the system is described as being overtrained. This overtraining effect, also called “overfitting,” results from a loss of generality. The neurons become attuned to precise patterns of information, focusing on non-generic aspects of the training set, and are therefore unable to recognize those sets
of data which deviate in the slightest manner from the sequences on which it was trained [11]. The obvious remedy to this problem is to reduce the number of hidden units, and the selection of the number of neurons in a given network is therefore an important aspect of its architecture. This aspect required due consideration when implementing the systems used for obtaining the results presented in this thesis, and is discussed in more detail in Chapter 4.

The methods employed in order to train such a reduced neuron system are based upon the properties of Green’s functions, which allow a process of supervised selection of centers to regulate the variables within the reference vectors of those neurons. The Gaussian function, an example of such, takes advantage of the techniques outlined below in order to exhibit the quality known as approximation, in which each neuron is able to identify multiple patterns, which are further differentiated by the output layers. An exposition of the approximation techniques based upon Gaussian properties is provided by Haykin [6], and the formulas themselves with their derivations are discussed in [12].

2.3. Training Methods

2.3.1. Output Layer – The Delta Rule

The networks employed during the course of this thesis consist of two types of neurons, the Gaussian units of the hidden layer and the linear unit of the output. The training of the output weights can be accomplished by using the Delta Rule, a common technique for adjusting the values of single-layer perceptrons and feed-forward networks. The formula employed is:

\[ w_j(n+1) = w_j(n) + \mu_1 \Delta w \]  

(2.6)
where $w$ is the weight connecting a hidden unit with the output layer, $j$ is the hidden unit to which the weight is connected and $n$ is the current epoch, or iteration index of the training set. The learning constant employed by the network for the delta rule is designated by $\mu$, and $\Delta w$ is the change in weight generated by the error term.

This error term $e_i$ is a function:

$$e_i = T_i - Z_i$$  \hspace{1cm} (2.7)

where $T_i$ is the target value and $Z_i$ is the obtained value from the network, the linear sum of all outputs from the Gaussian layer:

$$Z_i = \sum_{i=1}^{M} w_i \phi_i(r)$$  \hspace{1cm} (2.8)

$M$ is the number of neurons in the system and $\phi(r)$ is defined by Equation 2.1.

The error term $e$ is used to determine the $\Delta w$ value using the following equation:

$$\Delta w = \sum_{j=1}^{N} e_j \phi_j(r)$$  \hspace{1cm} (2.9)

where $N$ is the total number of input patterns from the data set.

### 2.3.2. Hidden Layer 1 – Movable Centers

While using the Delta rule to modify output weights may prove sufficient for many applications, data consisting of high dimensions such as that analysed during the course of this thesis require more sophisticated techniques to obtain a reasonable degree of classification accuracy. A refinement of network training involves modifying the reference vectors contained within the Gaussian neurons in order to more accurately reflect the input patterns to which they are exposed. This provides a set of more
significant (and potentially significantly different) $\phi$ values from the hidden layer to be summed by the output layer for classification and prediction.

In a similar manner to the Delta rule, the formula for changing the reference vector $t$ within a neuron is reflected by the formula:

$$t_j(n+1) = t_j(n) + \mu_2 \Delta t$$  \hfill (2.10)

As before, the $n$ value is the epoch, and the $\Delta t$ indicates the change based upon the error term provided by Equation 2.7. The $\mu_2$ is the learning constant for this phase of the training. The $\Delta t$ value is obtained from the function:

$$\Delta t = 2\psi_j(\phi) \sum_{i=1}^{N} e_i(\phi) \phi_j'(r) [x_i - t_j(\phi)].$$  \hfill (2.11)

The $\phi'$ value is the first derivative of the Gaussian function (2.1):

$$\phi'(r) = -\frac{1}{2\sigma_j^2} e^{-\frac{1}{2\sigma_j^2} (x_j - \mathbf{t}_j)^2}$$  \hfill (2.12)

### 2.3.3. Hidden Layer 3 – Adjustable Sigmas

The sigma value of the RBF units determines the “spread” of the bell curve by which the input vectors are compared, the radius of the decision boundary for grouping patterns (see Figures 3 and 4). A large $\sigma$ results in a wide spread, and large areas of identified input values. This can generate good training results if there are few outputs and a large range of acceptable values. On the other hand, this will also generate a lot of “false positives” when the network is actually used. Conversely a small $\sigma$ considerably narrows the range of deviation for identifiable input patterns, giving fewer false positives. The drawback is that acceptable values will be missed frequently, particularly if the pattern of expected outputs has a high degree of variability.
Manually adjusting sigma values allows a considerable difference in network performance to be detected, as shown in Section 3.4 (Tables 3 and 4), and a method of allowing the system to automatically adjust its spread in response to the patterns of data on which it is being trained is provided by the following formula:

$$\sigma_j(n+1) = \sigma_j(n) + \mu_3 \Delta \sigma$$  \hspace{1cm} (2.13)

The $\Delta \sigma$ multiplied by the learning constant $\mu_3$ in order to provide the change in the sigma per epoch is defined as follows, using the same $\phi'$ value from Equation 2.12:

$$\Delta \sigma = -\nu \phi_j(y) \sum_{i=1}^{M} \phi_i(y) \phi_j(y) [x_i - t_j(y)][x_i - t_j(y)]^T$$  \hspace{1cm} (2.14)

During the actual implementation of the networks used for this thesis, three levels of training are used. The first stage involves training only the output weights, and the other two methods add moveable centers and adjustable sigmas cumulatively.

### 2.4. Training Algorithm

The basic method of training for the system of networks used during this research follows the following steps:

1) The input vector is applied to the hidden layer of Gaussian neurons.

2) The resulting $\phi$ values are multiplied by the output weights and summed at the output layer.

3) The output value is compared with the expected value for that input vector.

4) This error term is used to determine how much the system’s parameters are to be adjusted in the following steps.

5) The Delta Rule (Equation 2.6) is applied to the output weights.
6) If level 2 or 3 training is selected, Equation 2.10 is used to modify the reference vectors of each of the M neurons of the network.

7) If level 3 training is being used, the sigma value of each RBF unit is altered according the adjustment Equation 2.13.

8) If the error rate is above the acceptable level, or if the number of epochs run is less than a predetermined limit, steps 5 to 7 are repeated.

Figures 8, 11 and 12 displayed in the following chapter give graphical representations of the above algorithm, and the various stages are discussed along with their Java implementations on which the transportation data was run.
Chapter 3
Implementation and Training

3.1. Program Specifications

3.1.1. Program Overview

A Gaussian function radial basis neural network was implemented in Java for the purpose of this thesis work. The object-oriented structure of Java programs lends itself naturally to the modular architecture of the system [13]. Each hidden neuron of the networks is implemented as an object within the virtual environment, and the creation of new units is thus made a simple process, and results in a freely modifiable architecture. Each unit produces a $\phi$ value based upon the input vector, using the function \texttt{NEURON.COMPUTE} (see Appendix B) and the sum of the hidden layer outputs, adjusted based upon the output weights, are summed to provide a network output as described in Equations 2.1 and 2.8.

The highly portable nature of the Java environment also enables the network to be executed on a number of different platforms, which could allow for performance optimization analysis in future work.

A graphical user interface (GUI) is provided by the system, and allows the user to choose from among the following operations: Train, Test, Run, Save, Help and Exit (see Figure 7).
The Exit function needs no particular explanation, and the Help feature allows for runtime text screens to be viewed by the user providing information about the file system and functions that are available. The Save option allows the user to write the reference vectors, sigma values and output weights of trained networks to output files for use in data analysis, which is accomplished using the Run function. Selecting Test allows for quick assessment of a newly trained network, processing a keyboard generated input pattern and doing a quick comparison with an expected, user-defined output.

The Train procedure, in which most of the system’s complexity lies, is explained in the following sections. More details of the other relevant modules are also discussed below.
3.1.2. The File System

Aside from the six picture icons in the main menu, provided by graphic interlace files (.gif images), 5 file types are necessary to the basic functionality of this program when it is run as an application in the Java environment. These are:

1) RBNet.java - the main program
2) NNI.rvf - a default reference vector
3) NN1.swf - default sigma and weight values
4) NN1.ipf - a default set of input patterns
5) NN1.itf - a default set of target values

Four file types are used to store information about a neural network:

1) *filename*.rvf - This file stores an $n \times m$ matrix of values. The number of rows ($n$) is the number of neurons and the number of columns ($m$) is the number of inputs to the network. (*rvf* = reference vector file)

2) *filename*.swf - This file stores a $2 \times n$ matrix of values: each of the $n$ rows contains two values. The first value is an initial Sigma, and the subsequent value is an initial weight from neuron to output. The number of rows is therefore 1 more than the $n$-value from the .rvf file. The number of rows in the file sets the number of neurons in the network architecture. (*swf* = sigma & weight file)

These first two files are produced when a newly trained network is saved. Two files, independent of the networks’ variables are used to contain training and testing data:
3) *filename*.ipf - This file contains an $n \times m$ matrix of values in which the number of rows ($n$) is the number of input patterns and the number of columns ($m$) is the dimensionality of each vector. (*ipf* = input pattern file)

4) *filename*.itf - This file contains a $1 \times n$ matrix of values, each row containing a single value: a target corresponding to an input pattern in the .ipf file. The number of targets listed here sets the number of patterns processed by the network during training. (*itf* = input target file)

The file *filename*.rrf is the report generated by the Run function, and is given a detailed examination in Section 3.1.5.

Appendix C contains samples of each type of file used by the network for training and analysis.

### 3.1.3. The Train Function

The training option of the system brings into play the theoretical foundations discussed in Chapter 2, and produces networks based upon four files provided by the user. When the Train function is selected, the system responds with a prompt requesting a *filename*. This variable is used to automatically identify the file containing the input patterns (*filename*.ipf), the list of expected outputs (*filename*.itf), the initial reference vector for the network to be trained (*filename*.rvf), and a set of sigmas for the neurons along with their associated weights for the output layer (*filename*.swf).

With these four files, the network is trained according to the user’s selection from among three learning methods. The first implements the algorithm described in Section 2.4 according to the diagram shown in Figure 8. This is the simplest training method.
After the iterations contained within the Training Module are completed, two on-screen outputs are provided for the user. The training curve (Figure 9) is displayed, indicating the times at which the training began and ended, based upon the computer’s local clock. The $x$-axis is the training epoch, and the $y$-axis is the SSE calculated from the cumulative difference between the expected and obtained outputs ($e$) as shown in the formula:

$$SSE(n) = \sum_{i=1}^{N} e_i$$

(3.1)

where $n$ is the epoch and $N$ is the number of neurons.

The calibration of the $y$-axis is determined from the initial SSE of the untrained network. The cumulative error of the first epoch is used as the highest $y$-value (since the network is anticipated to reduce this quantity), and the axis displays 10% decrements of this total from the initial value down to 0 rounded off at two decimal places.
The second display provided for the user is a diagram of the network’s architecture as exemplified by Figure 10. The size of the diagram is dependent upon the number of inputs and neurons of which the network is composed, and while this is more useful as a display feature for smaller networks, it provides information on the Sigma value on the final epoch of training for one of the input patterns which is selected by the user, and the final SSE achieved by the trained system.

The final weights of the output layer are also displayed, as well as the expected and obtained value of the user’s selected input pattern. In addition to this, the number of neurons and inputs is displayed at the bottom of the diagram.
At this point, if the Save feature is selected from the main menu (see Figure 7) the user is prompted for a string variable *network*, and the trained network is saved, its distinctive values encapsulated in the files *network.rvf* and *network.swf*.

If level 2 training is selected, the learning algorithm follows the flowchart in Figure 11, the difference being found in an added complexity of the Training Module.
During training, in addition to the Delta Rule being used to modify the output layer weights (Equation 2.6), the reference vector of each neuron is adjusted according to the input patterns via Equation 2.10. Output screens such as those shown in Figures 9 and 10 are provided to the user when the iterative learning is complete, and the user may save the network at this point. As with all three of the training methods, the Save function must be invoked at this point; if the system is exited, or if Train is selected again, the current network will be lost or overwritten respectively.
Figure 12 shows the algorithm of Section 2.4 being run on a system using level 3 training. In addition to the procedures involved with the previous two levels of training, the sigma values ($\sigma$) are adjusted using this technique, altering the variance of the Gaussian function and affecting the width of the decision boundary area (see Equation 2.13 and Figure 4). This directly affects the quality of the classification of input patterns, and serves to further tune the effect of the reference vectors and input weights.
3.1.4. The Test Function

The Test function, which invokes the algorithm described by Figure 13, allows a saved network to be quickly tested. The name of the network is requested from the user, at which point the corresponding swf and rvf files are loaded into memory. The user is then prompted for a set of inputs, the number of which corresponds to the size of the input vector anticipated by the system. A target value is then entered from the keyboard, and the input values are compared against the reference vector of each neuron and a Z-value is obtained from the system.

No output files are written, but a diagram of the network is provided, showing the $\varphi$ from each neuron, the values of the output weights, and the final network output compared to the expected value obtained from the user. This tool is effective particularly in smaller networks to verify the accuracy of the system’s classifications.
3.1.5. The Run Function

The Run feature of the system is that which is used to produce report files used for data analysis. It differs from the Test function in the following ways: it obtains its input patterns and targets from external files, it processes numerous input vectors in an iterative manner, and it provides a report file in addition to the on-screen output, which contains analysis information on the data. The report file is explained in the section following this one.

In Figure 14, filename1 represents the network and filename2 the data set comprised of input patterns and their expected output values. For each input pattern, the values of the vectors are processed by the Gaussian neurons, and the linear summation of these outputs (Z) as affected by the output weights is compared with each expected value.
Each obtained $Z$ value, also called the “predicted value” is stored in an array along with each corresponding target from the itf file, the “expected result,” and this information is passed on to the Run Report File (filename2.rf).

### 3.1.6. The Classification and Report Module

The following text is an actual rrf that resulted from the analysis of XOR files for input and output on a network trained to recognize that function:

```
***************************************************************
NNet Files: XORt.swf & XORt.rvf
Data Files: XOR.ipf & XOR.ipf
Date: 06/09/03 Time: 10:06:45 PM
***************************************************************
For each pattern, T is the Target and Z is the predicted value.

Pattern 0: T: 0.0 | Z: 0.0035512790504252673
Pattern 1: T: 1.0 | Z: 1.0071977444230102
Pattern 2: T: 1.0 | Z: 0.9979889896543631
```
Pattern 3: T: 0.0 | Z: 0.08665013438300243
Threshold: 0.9979889896543631
Training SSE: 0.0 (if trained this session)
Note: If Z < Threshold, it is considered a 0.

Rounded Z scores:
Pattern 0: T: 0.0 | Z: 0.0
Pattern 1: T: 1.0 | Z: 1.0
Pattern 2: T: 1.0 | Z: 1.0
Pattern 3: T: 0.0 | Z: 0.0

Correct: 4, Misclassified: 0, Accuracy: 100.0%

The main sections of the report consist of two columns. The first column is the list of target values for a set of input patterns obtained from an itf file. The second is the list of corresponding Z-values resulting from those patterns being classified by a network. The first half of the report provides the actual values obtained from the system, and the second part shows these outputs rounded to a threshold value.

This number is obtained automatically by the Output Data module of the Run feather, which uses each actual Z-value iteratively as a threshold, keeping track of which score provides the best final percentage accuracy. Any number less than the threshold is considered a 0 for classification purposes, and the other Z-scores are rounded up to 1. For networks in which a binary classification is not being performed, this part of the report can be ignored.

At the bottom of the file is a report summary consisting of the number of rightly and wrongly identified patterns and a percentage figure representing the network’s accuracy in regards to that data set. For another example of this and other types of files, please see Appendix C.
3.2. Implementation of Mathematical Formulae

The output of each neuron ($\varphi$) is calculated within the module NEURON.COMPUTE, which is coded in this way:

$$R = 0;$$
$$\text{for } (\text{int } q = 0; q < \text{inpnum}; q++)$$
$$\{$$
$$R += ((\text{invector}[q] - \text{refvector}[n][q])*(\text{invector}[q] - \text{refvector}[n][q]));$$
$$\}$$
$$R = \text{Math.sqrt}(R);$$
$$\text{outvector}[n] = \text{Math.exp}(-((R)*(R))/(\text{Sigma}[n]*\text{Sigma}[n]));$$

Referring to Equation 2.1, $R$ is the term $||x - t_j||$, $\text{outvector}[n]$ is the $\varphi$ value associated with each $n^{th}$ neuron, and $\text{inpnum}$ is the size of the input vector.

Equation 2.12, which provides the $\varphi'$ term used in both the moving centers formula (Equation 2.10) and the adjustable Sigma technique (Equation 2.13) is calculated in NEURON.COMPUTE by means of the following code in which $\text{toutvector}[n]$ is $\varphi'$:

$$\text{for } (\text{int } m = 0; m < \text{inpnum}; m++)$$
$$\{$$
$$\text{toutvector}[n][m] = (2-(2*\text{refvector}[n][m]))*\text{outvector}[n];$$
$$\}$$

for each neuron $n$ and each input pattern $m$. This code segment immediately follows the derivation of $\text{outvector}[n]$, and uses that value in its own mathematical calculations. For the code of the entire NEURON.COMPUTE module, please see Appendix B.

The following module COMPUTE, not to be confused with the internal neuron function NEURON.COMPUTE, obtains the $Z$ values (output of the system) by summing the outvectors ($\varphi$ terms) for each neuron as they are affected by the output weights:

$$\text{public void Compute(int inptrn) throws IOException }$$
$$\{$$
$$\text{Zval} = 0;$$
$$\text{for } (\text{int } q = 0; q < \text{inpnum}; q++)$$
$$\{$$
$$\text{invector}[q] = x[inptrn][q];$$
$$\}$$
for (int n = 0; n < neunum; n++)
{
    NeuronCell[n].Compute(n);
    Zval += (outvector[n]*weight[n]);
}
}

where *inptrn* is the row of the data set, corresponding to one record of input values and

*inpnum* is the total number of inputs associated with each input vector.

The error term (*e*) , which is key to all levels of learning, is computed in the

TRAIN function using the Z-value from the COMPUTE function by simply subtracting it from the target value associated with each row of the data set:

```
Compute(row);
    e = (target[row] - Zval);
```

The Delta rule for training the output weights, defined in Equation 2.6, is also implemented within the function TRAIN, by means of the following code:

```
Wdelta = e*outvector[n];
    weight[n] = weight[n] + MU1*Wdelta; // Change the weights for each neuron n.
```

The formula used to adjust the reference vectors of the neurons (Equation 2.10) is coded as follows:

```
if (tsel > 1) // If training method is 2 or 3...
    { for (int m = 0; m < inpnum; m++) // Change the ref. vectors
        { Rdelta = 2*weight[n]*e*outvector[n][m]*(invector[m] - refvector[n][m]);
            refvector[n][m] = refvector[n][m] + MU2*Rdelta;
        }
    }
```

The spread-adjustment performed on the neurons by modifying the sigma values, shown in Equation 2.13 is represented in Java by this segment:

```
if (tsel > 2) // If training method is 3...
    { Sdelta = (-1)*weight[n]*e*Mprod; // Change Sigma values
```
The TRAIN module in its entirety, from which the $M_{prod}$ value is obtained, can be found in Appendix B.

3.3. Basic Functions A – Delta Rule Training (Fixed Centers)

In order to test the effectiveness of the neural network here described, the system was applied to the binary functions OR, AND and XOR. Using arbitrarily assigned reference vectors for 4 neurons, the training curves shown in Figure 15 were obtained after 500 epochs:
Figure 15. SSE Curves For OR, AND and XOR Respectively (Set 1)
The results of running the network on a full set of the possible binary inputs yielded the data shown in Table 1.

Table 1. Level 1 Training Results During Network Testing

<table>
<thead>
<tr>
<th>Function</th>
<th>Final SSE</th>
<th>Network Accuracy</th>
<th>Last non-0 SSE epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0.84</td>
<td>100%</td>
<td>00</td>
</tr>
<tr>
<td>AND</td>
<td>0.43</td>
<td>100%</td>
<td>00</td>
</tr>
<tr>
<td>XOR</td>
<td>0.05</td>
<td>100%</td>
<td>00</td>
</tr>
</tbody>
</table>

A 00 in the “Last non-0 SSE epoch” column indicates that the SSE did not drop to 0 within the maximum number of epochs for which the network was trained.

3.4. Basic Functions B – Delta Rule vs. Fixed Centers and Delta

Although the network performed perfectly in classifying the input patterns for the three binary test functions, it was observed that the SSE values did not fall to an acceptable level within a reasonable amount of time. For a network with four neurons and such elementary binary functions, the rate of learning was anticipated to be more dramatic. Since the purpose of the system was to analyse data with a much higher degree of complexity, Level 2 training was next used to adapt networks to these functions in an attempt to generate a sharper reduction in the cumulative errors of the training epochs. See Figure 16 for the learning curves thus obtained.
Figure 16. SSE Curves For OR, AND and XOR Respectively (Set 2)
Using the Run function on each of the three networks, using the complete set of binary inputs provided the information shown in Table 2 below:

### Table 2. Level 2 Training Results During Network Testing

<table>
<thead>
<tr>
<th>Function</th>
<th>Final SSE</th>
<th>Network Accuracy</th>
<th>Last non-0 SSE epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0.97</td>
<td>100%</td>
<td>00</td>
</tr>
<tr>
<td>AND</td>
<td>0.00</td>
<td>100%</td>
<td>233</td>
</tr>
<tr>
<td>XOR</td>
<td>0.00</td>
<td>100%</td>
<td>414</td>
</tr>
</tbody>
</table>

Although the OR function was classified less accurately than the other two binary processes, which may be related to the random nature of the initial centers, the performance of the network was appreciably improved when the center adjustment feature was implemented. The curves in the XOR test show the most dramatic improvements, the SSE dropping so sharply after certain epochs that the display curve is dotted.

### 3.5. Basic Functions C

Both the curves and the tabular results obtained from level 2 training reveal that the adjustment of the reference vectors within the Gaussian neurons provides higher rates of learning on the binary functions used to test the system. In both these methods, however, the spread of the radial basis function remained constant at 0.4, and these variables were modified and the networks re-trained on each function to demonstrate the significance of the $\sigma$ value to network performance.
3.5.1. Manually Adjusting the Sigma

The following SSE and accuracy values were obtained after 300 epochs by varying the Sigma value for each of the three functions OR, AND and XOR, and a reduced data set consisting of 50 records of the transportation data (LA50) in both level 1 and 2 training:

<table>
<thead>
<tr>
<th>Function</th>
<th>$\sigma$</th>
<th>0.01</th>
<th>0.05</th>
<th>0.10</th>
<th>0.20</th>
<th>0.40</th>
<th>0.70</th>
<th>1.00</th>
<th>2.00</th>
<th>5.00</th>
<th>7.00</th>
<th>10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>3.0</td>
<td>2.99</td>
<td>2.99</td>
<td>2.32</td>
<td>0.96</td>
<td>0.13</td>
<td>0.23</td>
<td>0.49</td>
<td>0.76</td>
<td>0.77</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.99</td>
<td>0.51</td>
<td>0.2</td>
<td>0.33</td>
<td>0.64</td>
<td>0.75</td>
<td>0.76</td>
<td>0.77</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>2.0</td>
<td>1.99</td>
<td>1.37</td>
<td>0.42</td>
<td>0.08</td>
<td>0.44</td>
<td>0.87</td>
<td>1.04</td>
<td>1.03</td>
<td>1.03</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>LA50</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>10.96</td>
<td>10.36</td>
<td>6.69</td>
<td>6.48</td>
<td>7.6</td>
<td>8.75</td>
<td></td>
</tr>
<tr>
<td>LA50run</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>86</td>
<td>88</td>
<td>86</td>
<td>88</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>LAver1</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>$\sigma$</th>
<th>0.01</th>
<th>0.05</th>
<th>0.10</th>
<th>0.20</th>
<th>0.40</th>
<th>0.70</th>
<th>1.00</th>
<th>2.00</th>
<th>5.00</th>
<th>7.00</th>
<th>10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>3.0</td>
<td>2.99</td>
<td>2.99</td>
<td>1.36</td>
<td>0.98</td>
<td>0.04</td>
<td>0.25</td>
<td>0.39</td>
<td>0.72</td>
<td>0.75</td>
<td>0.76</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.99</td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
<td>0.33</td>
<td>0.67</td>
<td>0.72</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>2.0</td>
<td>1.98</td>
<td>0.99</td>
<td>0.00</td>
<td>0.01</td>
<td>0.11</td>
<td>0.49</td>
<td>1.03</td>
<td>1.03</td>
<td>1.03</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>LA50</td>
<td>11.0</td>
<td>11.0</td>
<td>11.0</td>
<td>11.02</td>
<td>11.0</td>
<td>10.93</td>
<td>9.01</td>
<td>6.22</td>
<td>7.12</td>
<td>7.24</td>
<td>8.69</td>
<td></td>
</tr>
<tr>
<td>LA50run</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>82</td>
<td>92</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>LAver1</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.85</td>
<td>84.84</td>
<td>84.84</td>
</tr>
</tbody>
</table>

The first three rows of Tables 3 and 4 consist of the SSE values obtained after the indicated number of epochs for each action listed in the Function column on the left. The fourth row shows this value for the reduced data set. Row five is the percentage of accuracy achieved by training, and row six is the percentage of accuracy the trained network obtained when run on a validation set of 99 random records from the original spreadsheet.
As these tables indicate, there is a definite point for each type of training at which the sum of squared errors is lowest, and the training accuracy is at a peak. For the subset of the actual data, the LA50 training set, using level 1 training and a consistent sigma value of 5.0 in each Gaussian neuron provides the best results. For level 2 training, that which adjusts both the output weights and the reference vectors, the lowest SSE value and the highest degree of accuracy are both observed when the neurons use a Sigma of 2.0 for calculations. Further testing using even higher σ values than 10.0 provided a consistent degradation from these peak values.

The different values of Sigma for which each function performs best should be noted. There is not one particular σ value that provides optimum learning, but the final SSE depends also on the initial reference vector and weights, the training data provided, and the nature of the function itself.

3.5.2. Automatically Adjusting the Sigma

The data from Section 3.5.1 justifies the use of further refinements to the training algorithms utilized by the network, since manually adjusting Sigmas for each new network would be a time consuming task. This would be rendered even more impractical if the σ variable of each neuron was adjusted individually for optimum performance. Because of this, a third level of training was implemented within the system. This automatic adjustment of the sigma value for each neuron during the course of the training generated the results shown in Figure 17 after 500 epochs:
Figure 17. SSE Curves For OR, AND and XOR Respectively (Set 3)
### Table 5. Level 3 Training Results During Network Testing

<table>
<thead>
<tr>
<th>Function</th>
<th>Final SSE</th>
<th>Network Accuracy</th>
<th>Last non-0 SSE epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0.39</td>
<td>100%</td>
<td>00</td>
</tr>
<tr>
<td>AND</td>
<td>0.00</td>
<td>100%</td>
<td>181</td>
</tr>
<tr>
<td>XOR</td>
<td>0.00</td>
<td>100%</td>
<td>250</td>
</tr>
</tbody>
</table>

#### 3.6. Analysis of Preliminary Training (OR, AND, XOR)

### Table 6. Summary of Training Results on Binary Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Lv1 F-SSE</th>
<th>Lv1 LNZE</th>
<th>Lv2 F-SSE</th>
<th>Lv2 LNZE</th>
<th>Lv3 F-SSE</th>
<th>Lv3 LNZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
<td>0.84</td>
<td>00</td>
<td>0.97</td>
<td>00</td>
<td>0.39</td>
<td>00</td>
</tr>
<tr>
<td>AND</td>
<td>0.43</td>
<td>00</td>
<td>0.00</td>
<td>233</td>
<td>0.00</td>
<td>181</td>
</tr>
<tr>
<td>XOR</td>
<td>0.05</td>
<td>00</td>
<td>0.00</td>
<td>414</td>
<td>0.00</td>
<td>250</td>
</tr>
</tbody>
</table>

The Lv value is the level of training as described in Sections 3.3 to 3.5, the F-SSE term is the final sum of squared errors value, and LNZE represents the last epoch of training in which the SSE was not 0 (Last Non-Zero Epoch).

This information shows a significant improvement with each new level of training used by the system, and the superior performance of level 3 training in every test case run. In 500 epochs of Level 1 training, the average SSE of the three binary functions listed is 44, and the network is unable in this time to recognize any of them perfectly. By adding the ability to shift reference vectors within neurons to the training method, the average SSE becomes 32.33, with both AND and XOR patterns learned perfectly.

Level 3 training cuts the average SSE down to a mere 13, and while the OR function does not converge in this period, its error is minimized using automatic Sigma adjust and the patterns which do converge do so in an average of 108 less training cycles.
Chapter 4

Application Initialization

4.1. Description of Data

Appendix A gives a full listing of the variables comprising the reduced data set used for this thesis work, and their descriptions. The dependent variable, which the various regression and network techniques are attempting to predict, is the DELTA_VTR, or the change in the Vehicle Trip Rate over the course of time in which the incentive plans and their associated costs, represented by the independent variables (i.e. predictors), were implemented.

In order to aid in the analysis, different degrees of impact were sorted into 8 classes or “bins,” as described by Table 7 below. A drop in VTR by more than 7 indicates that the combination of incentives was very effective, and this record is sorted into Bin 1. A record of incentive programs that produced a reduction between 4 and 7 is classified as belonging to Bin 2. Very minor changes identify an incentive combination as Bin 4 or 5, and if the VTR begins to increase, the record is sorted into Bins 5-7, depending on the degree of impact.

Table 7. Classification of Data and The Bin Ranges

<table>
<thead>
<tr>
<th>Range</th>
<th>&lt; -7</th>
<th>-7 to -4</th>
<th>-4 to -2</th>
<th>-2 to 0</th>
<th>0 to 1</th>
<th>1 to 2.5</th>
<th>2.5 to 5</th>
<th>&gt; 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin #</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
The data on which the RBF network analysis is being performed consists of 16,302 such records, already divided into bins for training. The number of records in Bin 1 is 2543. Bin 2 holds 2310, Bin 3 has 2192 and there are 2635 records that are classified as Bin 4. 1414 records belong to Bin 5, 1479 patterns result in changes within the range of Bin 6, 1714 records are sorted into Bin 7, and Bin 8 holds 2015 data patterns. This relatively even split of patterns among the bins renders padding the data by duplicating random records unnecessary, although this technique was used to balance the training methods used on previous, unrefined sets of the same data.

4.2. CUTR Analysis Results

The accuracies of previous attempts to predict the bin into which data patterns of this set are presented in the following tables. Four types of classification were attempted using regression techniques by researchers at the Center for Urban Transportation Research (CUTR). Models were built using the software *SPSS 11.1 for Windows*, and they were designed according to the following techniques:

4.2.1. Forward Regression

The system begins by computing which of the predictor variables has the largest bivariate correlation with (i.e. impact on) the DELTA_VTR value, and then other variables are selected based upon their relative contribution to the variance in the dependent variable. Predictors that do not have a significant impact are ignored.

4.2.2. Backward Regression

This method begins by using all the potential predictor variables, and then deleting those that do not correlate to the variance within a significance level of 90%.
The deletion of variables stops when all the factors calculated to be relatively useless are gone.

4.2.3. Stepwise Regression

Stepwise Regression combines both the forward and backward concepts, providing a more complex selection process. This method has proven to be the most effective at actual classification, and provided the figures against which the Radial Basis Network was most effectively compared. This procedure both adds and subtracts predictor variables based upon inter-correlations between those already selected. A simple example of this is that if the inclusion of one variable weakens the impact of an already-selected variable, the new one is not included in the final set produced.

4.2.4. Force-entered Regression

This simplest of techniques merely accepts all the variables specified, regardless of their correlation with the dependent variable.

Based on the variables selected by all these techniques, and the full set accepted by force-entered, two sets of neural networks were developed - one with no hidden units and the other with a number of such units selected by the SPSS software as optimal - and the results of both the regression and network analyses are included in this report.

In the tables below, Exact T. is the accuracy of the network in classifying the records of its training set precisely. That is, if a record from the training set is expected to produce a Bin value of 8, the network correctly classifies it if and only if it produces a value of 8 for that record. 1-Off T. is the ability of a technique to classify the pattern with an error range of ± 1.
Exact V. is the ability of a network to precisely predict the Bin into which a data record of an unseen Validation set falls. 1-Off V. considers a classification “correct” if, for example, the expected Bin number is 4, and the returned value \((Z)\) is \(4 \leq Z \leq 6\). The values given describe both the accuracy of various regression techniques and neural network analyses based on the variables obtained from the respective regression methods.

Set A: Forced Regression

Table 8. Force-Entered Regression Results

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.75</td>
<td>0.86</td>
<td>18.88</td>
<td>33.71</td>
<td>30.09</td>
<td>12.50</td>
<td>14.77</td>
<td>9.16</td>
<td>6.35</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.63</td>
<td>1.75</td>
<td>16.81</td>
<td>33.28</td>
<td>30.37</td>
<td>11.31</td>
<td>14.03</td>
<td>10.41</td>
<td>6.50</td>
</tr>
<tr>
<td>Exact T.</td>
<td>53.92</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>65.75</td>
<td>44.56</td>
<td>40.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>47.36</td>
<td>27.15</td>
<td>50.56</td>
<td>76.10</td>
<td>71.73</td>
<td>54.61</td>
<td>34.17</td>
<td>27.98</td>
<td>23.88</td>
</tr>
</tbody>
</table>

Table 9. Force-Entered Network Results (0 Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>15.92</td>
<td>0.29</td>
<td>16.08</td>
<td>31.74</td>
<td>33.81</td>
<td>12.66</td>
<td>15.53</td>
<td>8.40</td>
<td>2.21</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>15.81</td>
<td>0.91</td>
<td>14.21</td>
<td>32.69</td>
<td>33.00</td>
<td>12.16</td>
<td>15.00</td>
<td>9.46</td>
<td>1.69</td>
</tr>
<tr>
<td>Exact T.</td>
<td>46.51</td>
<td>23.05</td>
<td>50.35</td>
<td>74.72</td>
<td>76.36</td>
<td>62.45</td>
<td>32.95</td>
<td>26.34</td>
<td>17.40</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>46.68</td>
<td>24.10</td>
<td>47.12</td>
<td>75.84</td>
<td>74.97</td>
<td>61.53</td>
<td>35.88</td>
<td>27.34</td>
<td>16.68</td>
</tr>
</tbody>
</table>

Set B: Backward Regression

Table 10. Backward Regression Results

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>15.96</td>
<td>8.93</td>
<td>33.92</td>
<td>28.09</td>
<td>20.35</td>
<td>9.62</td>
<td>10.98</td>
<td>7.63</td>
<td>4.42</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.81</td>
<td>1.49</td>
<td>17.01</td>
<td>34.57</td>
<td>30.28</td>
<td>11.47</td>
<td>13.99</td>
<td>10.52</td>
<td>6.61</td>
</tr>
<tr>
<td>Exact T.</td>
<td>45.80</td>
<td>42.65</td>
<td>68.53</td>
<td>75.28</td>
<td>62.39</td>
<td>39.90</td>
<td>24.24</td>
<td>20.99</td>
<td>18.23</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>47.22</td>
<td>26.72</td>
<td>50.39</td>
<td>76.38</td>
<td>71.75</td>
<td>53.98</td>
<td>33.69</td>
<td>28.05</td>
<td>23.81</td>
</tr>
</tbody>
</table>
Table 11. Backward Regression Network Results (0 Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.16</td>
<td>0.29</td>
<td>15.03</td>
<td>33.43</td>
<td>34.52</td>
<td>10.97</td>
<td>15.53</td>
<td>8.78</td>
<td>3.04</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.08</td>
<td>0.65</td>
<td>14.42</td>
<td>35.13</td>
<td>32.72</td>
<td>11.93</td>
<td>13.36</td>
<td>10.07</td>
<td>2.65</td>
</tr>
<tr>
<td>Exact T.</td>
<td>46.75</td>
<td>23.34</td>
<td>51.05</td>
<td>75.28</td>
<td>76.12</td>
<td>59.49</td>
<td>31.82</td>
<td>25.57</td>
<td>21.27</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>46.95</td>
<td>23.58</td>
<td>48.50</td>
<td>76.35</td>
<td>75.03</td>
<td>58.42</td>
<td>35.27</td>
<td>26.99</td>
<td>20.39</td>
</tr>
</tbody>
</table>

Table 12. Backward Network Results (With Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>17.26</td>
<td>5.48</td>
<td>19.93</td>
<td>25.28</td>
<td>36.17</td>
<td>18.14</td>
<td>12.88</td>
<td>8.78</td>
<td>5.26</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>17.66</td>
<td>5.99</td>
<td>17.28</td>
<td>26.03</td>
<td>37.09</td>
<td>16.38</td>
<td>14.56</td>
<td>9.76</td>
<td>5.78</td>
</tr>
<tr>
<td>Exact T.</td>
<td>48.09</td>
<td>28.82</td>
<td>47.90</td>
<td>75.00</td>
<td>72.58</td>
<td>66.24</td>
<td>36.36</td>
<td>27.86</td>
<td>22.93</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>48.15</td>
<td>31.00</td>
<td>45.64</td>
<td>72.41</td>
<td>73.09</td>
<td>65.71</td>
<td>38.14</td>
<td>27.95</td>
<td>23.11</td>
</tr>
</tbody>
</table>

Set C: Forward Regression

Table 13. Forward Regression Results

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.40</td>
<td>0.58</td>
<td>16.08</td>
<td>34.55</td>
<td>30.53</td>
<td>10.58</td>
<td>14.02</td>
<td>9.54</td>
<td>6.35</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.73</td>
<td>1.10</td>
<td>16.84</td>
<td>34.60</td>
<td>30.80</td>
<td>10.58</td>
<td>13.77</td>
<td>10.29</td>
<td>6.78</td>
</tr>
<tr>
<td>Exact T.</td>
<td>46.43</td>
<td>25.65</td>
<td>53.50</td>
<td>74.72</td>
<td>71.24</td>
<td>54.33</td>
<td>31.06</td>
<td>27.10</td>
<td>22.65</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>47.42</td>
<td>26.34</td>
<td>50.76</td>
<td>76.65</td>
<td>72.46</td>
<td>54.35</td>
<td>34.13</td>
<td>27.55</td>
<td>23.98</td>
</tr>
</tbody>
</table>

Table 14. Forward Regression Network Results (0 Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.12</td>
<td>8.07</td>
<td>18.18</td>
<td>22.75</td>
<td>33.10</td>
<td>10.55</td>
<td>15.15</td>
<td>7.63</td>
<td>6.35</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.12</td>
<td>0.52</td>
<td>14.15</td>
<td>34.66</td>
<td>33.50</td>
<td>11.93</td>
<td>14.34</td>
<td>10.11</td>
<td>2.03</td>
</tr>
<tr>
<td>Exact T.</td>
<td>46.12</td>
<td>35.73</td>
<td>50.70</td>
<td>62.36</td>
<td>72.81</td>
<td>60.34</td>
<td>30.30</td>
<td>23.28</td>
<td>24.03</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>46.85</td>
<td>23.10</td>
<td>48.20</td>
<td>76.21</td>
<td>74.97</td>
<td>58.79</td>
<td>35.40</td>
<td>27.22</td>
<td>20.12</td>
</tr>
</tbody>
</table>
### Table 15. Forward Regression Network Results (With Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>17.07</td>
<td>11.82</td>
<td>16.08</td>
<td>17.70</td>
<td>39.95</td>
<td>18.14</td>
<td>12.12</td>
<td>5.73</td>
<td>6.63</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>18.30</td>
<td>7.42</td>
<td>17.62</td>
<td>25.38</td>
<td>39.84</td>
<td>18.61</td>
<td>14.12</td>
<td>8.58</td>
<td>5.57</td>
</tr>
<tr>
<td>Exact T.</td>
<td>47.22</td>
<td>38.33</td>
<td>50.35</td>
<td>61.24</td>
<td>74.47</td>
<td>67.51</td>
<td>31.82</td>
<td>22.90</td>
<td>23.20</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>48.56</td>
<td>31.65</td>
<td>45.64</td>
<td>73.70</td>
<td>75.23</td>
<td>68.03</td>
<td>37.96</td>
<td>25.00</td>
<td>22.46</td>
</tr>
</tbody>
</table>

### Set D: Stepwise Regression

### Table 16. Stepwise Regression Results

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.20</td>
<td>0.58</td>
<td>15.73</td>
<td>33.99</td>
<td>30.31</td>
<td>10.58</td>
<td>13.64</td>
<td>9.54</td>
<td>6.35</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.81</td>
<td>1.49</td>
<td>17.01</td>
<td>34.57</td>
<td>30.28</td>
<td>11.47</td>
<td>13.99</td>
<td>10.52</td>
<td>6.61</td>
</tr>
<tr>
<td>Exact T.</td>
<td>46.24</td>
<td>25.65</td>
<td>53.50</td>
<td>74.16</td>
<td>70.58</td>
<td>54.33</td>
<td>30.68</td>
<td>27.86</td>
<td>22.38</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>47.22</td>
<td>26.72</td>
<td>50.39</td>
<td>76.38</td>
<td>71.75</td>
<td>53.98</td>
<td>33.29</td>
<td>28.05</td>
<td>23.81</td>
</tr>
</tbody>
</table>

### Table 17. Stepwise Regression Network Results (0 Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>17.23</td>
<td>17.87</td>
<td>22.73</td>
<td>20.22</td>
<td>25.06</td>
<td>8.86</td>
<td>18.56</td>
<td>11.07</td>
<td>9.12</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>16.21</td>
<td>0.42</td>
<td>13.74</td>
<td>34.49</td>
<td>34.45</td>
<td>11.79</td>
<td>14.42</td>
<td>10.30</td>
<td>1.93</td>
</tr>
<tr>
<td>Exact T.</td>
<td>45.09</td>
<td>48.13</td>
<td>53.15</td>
<td>58.15</td>
<td>55.56</td>
<td>53.16</td>
<td>34.09</td>
<td>25.57</td>
<td>27.62</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>46.65</td>
<td>22.38</td>
<td>47.79</td>
<td>76.55</td>
<td>75.03</td>
<td>58.61</td>
<td>35.40</td>
<td>26.72</td>
<td>19.88</td>
</tr>
</tbody>
</table>

### Table 18. Stepwise Regression Network Results (With Hidden Units)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>17.07</td>
<td>10.37</td>
<td>19.58</td>
<td>19.94</td>
<td>40.43</td>
<td>12.66</td>
<td>12.88</td>
<td>6.11</td>
<td>5.25</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>17.59</td>
<td>5.54</td>
<td>18.22</td>
<td>25.45</td>
<td>39.77</td>
<td>15.03</td>
<td>15.58</td>
<td>9.49</td>
<td>2.10</td>
</tr>
<tr>
<td>Exact T.</td>
<td>47.22</td>
<td>38.90</td>
<td>51.40</td>
<td>58.43</td>
<td>70.45</td>
<td>64.98</td>
<td>32.95</td>
<td>25.19</td>
<td>28.45</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>47.89</td>
<td>29.87</td>
<td>44.43</td>
<td>75.03</td>
<td>72.57</td>
<td>68.12</td>
<td>37.61</td>
<td>27.41</td>
<td>20.67</td>
</tr>
</tbody>
</table>
Table 19. Stepwise Regression Network Results With Costs (8G1)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.00</td>
<td>3.17</td>
<td>16.08</td>
<td>23.88</td>
<td>43.50</td>
<td>14.77</td>
<td>8.71</td>
<td>6.11</td>
<td>1.66</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>17.52</td>
<td>4.83</td>
<td>16.84</td>
<td>25.59</td>
<td>41.83</td>
<td>18.19</td>
<td>13.36</td>
<td>7.31</td>
<td>2.10</td>
</tr>
<tr>
<td>Exact T.</td>
<td>46.31</td>
<td>25.65</td>
<td>44.41</td>
<td>73.88</td>
<td>75.65</td>
<td>71.73</td>
<td>34.09</td>
<td>23.66</td>
<td>14.92</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>47.87</td>
<td>27.79</td>
<td>43.89</td>
<td>73.56</td>
<td>77.76</td>
<td>72.30</td>
<td>36.99</td>
<td>24.20</td>
<td>17.85</td>
</tr>
</tbody>
</table>

Table 20. Stepwise Results: Hidden Units/Reduced Dataset (8IOB1)

<table>
<thead>
<tr>
<th>Classtype</th>
<th>Total</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>16.92</td>
<td>7.19</td>
<td>19.55</td>
<td>27.59</td>
<td>35.52</td>
<td>13.29</td>
<td>12.10</td>
<td>6.99</td>
<td>6.56</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>18.30</td>
<td>7.75</td>
<td>22.68</td>
<td>30.34</td>
<td>33.02</td>
<td>14.29</td>
<td>12.64</td>
<td>9.33</td>
<td>8.83</td>
</tr>
<tr>
<td>Exact T.</td>
<td>47.82</td>
<td>31.85</td>
<td>54.09</td>
<td>77.78</td>
<td>67.24</td>
<td>56.65</td>
<td>36.94</td>
<td>27.42</td>
<td>23.94</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>49.53</td>
<td>37.48</td>
<td>54.37</td>
<td>76.05</td>
<td>68.92</td>
<td>58.77</td>
<td>35.36</td>
<td>27.19</td>
<td>27.89</td>
</tr>
</tbody>
</table>

4.3. Selection of Network Architecture

4.3.1. Number and Selection of Inputs

Appendix A describes the list of variables considered to be relevant by a Stepwise Regression analysis technique, which corresponds to Table 20 of Set D. As the “Total” column of the tables indicates, this technique provides the most accurate prediction results, particularly when the variables it selects are used to train a neural network that contains hidden units. The Radial Basis network implemented for this thesis uses this list of variables for training in an attempt to utilize this already established information.

4.3.2. Number of Neurons and Epochs

As mentioned in Section 2.2, over-fitting of a network is a very real concern when doing analysis of data. Networks may show very good results in classifying their training sets, however when applied to a validation set results may be obtained which are less desirable than was expected. Using too large a network or too long a training period can
lead to redundancy, and such a network will amplify the effects of “noise data” within the input patterns resulting in this poor performance [14]. In order to avoid this, a network containing as few neurons as possible, while maintaining good training results, is used to obtain the highest degree of generalization, and thus the best validation results.

Testing to discover the optimal network size was performed by varying the number of network neurons in an attempt to classify the records of Bin 3, the most accurately classified class of patterns in the CUTR experiments. Table 21 reveals that significant changes relating to training accuracy are obtained in reasonable time with about 2000 neurons, and this number was used for obtaining the main training and validation results of this thesis.

It should be noted that a rare case with certain values causes a NaN (not a number) error during calculations. This is most likely due to a rounding problem when dealing with very small doubles, and thus divisions by 0 are an occasional concern. Testing has shown that the easiest way to alleviate the difficulty is to slightly increase or decrease the number of neurons in the system. This does not affect the training or validation accuracies to any detectable degree, and allows the network to avoid the potential rounding difficulty.
Table 21. Varying The Number of Neurons For Classifying Bin 3

<table>
<thead>
<tr>
<th>Neurons</th>
<th>Time (seconds)</th>
<th>Initial SSE</th>
<th>Final SSE</th>
<th>Delta SSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>17</td>
<td>2055</td>
<td>1923</td>
<td>132</td>
</tr>
<tr>
<td>200</td>
<td>123</td>
<td>2000</td>
<td>1781</td>
<td>219</td>
</tr>
<tr>
<td>300</td>
<td>229</td>
<td>2031</td>
<td>1690</td>
<td>341</td>
</tr>
<tr>
<td>400</td>
<td>611</td>
<td>1999</td>
<td>1607</td>
<td>392</td>
</tr>
<tr>
<td>500</td>
<td>563</td>
<td>1980</td>
<td>1517</td>
<td>463</td>
</tr>
<tr>
<td>600</td>
<td>613</td>
<td>1949</td>
<td>1427</td>
<td>522</td>
</tr>
<tr>
<td>700</td>
<td>1024</td>
<td>1909</td>
<td>1329</td>
<td>580</td>
</tr>
<tr>
<td>800</td>
<td>1440</td>
<td>1895</td>
<td>1250</td>
<td>645</td>
</tr>
<tr>
<td>900</td>
<td>783</td>
<td>1845</td>
<td>1185</td>
<td>660</td>
</tr>
<tr>
<td>1000</td>
<td>1201</td>
<td>1809</td>
<td>1097</td>
<td>712</td>
</tr>
<tr>
<td>1100</td>
<td>957</td>
<td>1780</td>
<td>1026</td>
<td>754</td>
</tr>
<tr>
<td>1200</td>
<td>997</td>
<td>1751</td>
<td>951</td>
<td>800</td>
</tr>
<tr>
<td>1300</td>
<td>1056</td>
<td>1721</td>
<td>880</td>
<td>841</td>
</tr>
<tr>
<td>1400</td>
<td>1094</td>
<td>1690</td>
<td>805</td>
<td>885</td>
</tr>
<tr>
<td>1500</td>
<td>118</td>
<td>1669</td>
<td>735</td>
<td>934</td>
</tr>
<tr>
<td>1600</td>
<td>124</td>
<td>1636</td>
<td>663</td>
<td>973</td>
</tr>
<tr>
<td>1700</td>
<td>1567</td>
<td>1604</td>
<td>597</td>
<td>1007</td>
</tr>
<tr>
<td>1800</td>
<td>1646</td>
<td>1577</td>
<td>527</td>
<td>1050</td>
</tr>
<tr>
<td>1900</td>
<td>1565</td>
<td>1552</td>
<td>454</td>
<td>1098</td>
</tr>
<tr>
<td>2000</td>
<td>3034</td>
<td>1520</td>
<td>391</td>
<td>1129</td>
</tr>
</tbody>
</table>
As Table 23 indicates, the SSE values are not always directly proportional to the classification accuracy of the system, therefore when testing the network for a reasonable number of epochs without resulting in over training, the SSE values were not considered, only the degree of accuracy resulting from the report module. These results are shown in Table 22. Based on the data in Tables 21 and 22, a system of 2000 neurons, trained for 1000 epochs, was selected for use by the Radial Basis Network.
Table 22. Varying The Number of Epochs For Classifying Bin 3

<table>
<thead>
<tr>
<th>No. of Epochs</th>
<th>Training Accuracy</th>
<th>Validation Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>99.049</td>
<td>85.75</td>
</tr>
<tr>
<td>600</td>
<td>99.055</td>
<td>85.74</td>
</tr>
<tr>
<td>700</td>
<td>99.055</td>
<td>85.75</td>
</tr>
<tr>
<td>800</td>
<td>99.067</td>
<td>85.75</td>
</tr>
<tr>
<td>900</td>
<td>99.049</td>
<td>85.75</td>
</tr>
<tr>
<td>1000</td>
<td>99.086</td>
<td>85.75</td>
</tr>
<tr>
<td>1100</td>
<td>99.092</td>
<td>85.75</td>
</tr>
<tr>
<td>1200</td>
<td>99.092</td>
<td>85.75</td>
</tr>
</tbody>
</table>
Chapter 5

Execution and Analysis

5.1. Training and Testing Results on Application Data

Two distinct types of analysis are attempted by the Radial Basis Network. The first is an 8-level binary classification. In essence, the problem of classifying data patterns into eight bins is split into eight separate problems. Eight different networks are used, each of which is designed to determine whether or not a particular data pattern belongs to a specific bin. For example, the Bin1 network produces a binary result: if a data pattern belongs to Bin 1, the Bin1 network will output a 1, or it will produce a 0 if the pattern is of another Bin. The same holds for Bins 2 – 8.

Table 23. Network Training Results

<table>
<thead>
<tr>
<th>Bin Trained</th>
<th>Initial SSE</th>
<th>Final SSE</th>
<th>Delta SSE</th>
<th>Initial Accuracy</th>
<th>Final Accuracy</th>
<th>Delta Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin1</td>
<td>1790</td>
<td>368.99</td>
<td>1421.01</td>
<td>87.72</td>
<td>97.49</td>
<td>9.77</td>
</tr>
<tr>
<td>Bin2</td>
<td>1570</td>
<td>86.09</td>
<td>1483.91</td>
<td>87.52</td>
<td>99.48</td>
<td>11.96</td>
</tr>
<tr>
<td>Bin3</td>
<td>1540</td>
<td>147.41</td>
<td>1392.59</td>
<td>88.55</td>
<td>99.09</td>
<td>10.54</td>
</tr>
<tr>
<td>Bin4</td>
<td>2106</td>
<td>564.45</td>
<td>1541.55</td>
<td>84.27</td>
<td>96.09</td>
<td>11.82</td>
</tr>
<tr>
<td>Bin5</td>
<td>948</td>
<td>2.79</td>
<td>945.21</td>
<td>93.74</td>
<td>99.98</td>
<td>6.24</td>
</tr>
<tr>
<td>Bin6</td>
<td>919</td>
<td>2.44</td>
<td>916.56</td>
<td>92.22</td>
<td>99.98</td>
<td>7.76</td>
</tr>
<tr>
<td>Bin7</td>
<td>1083</td>
<td>3.13</td>
<td>1079.87</td>
<td>91.84</td>
<td>99.98</td>
<td>8.14</td>
</tr>
<tr>
<td>Bin8</td>
<td>1286</td>
<td>17.61</td>
<td>1268.39</td>
<td>90.31</td>
<td>99.88</td>
<td>9.57</td>
</tr>
<tr>
<td>Average</td>
<td>1405.25</td>
<td>149.114</td>
<td>1256.136</td>
<td>89.521</td>
<td>98.996</td>
<td>9.475</td>
</tr>
<tr>
<td>Macro1</td>
<td>2080</td>
<td>634.46</td>
<td>1445.54</td>
<td>15.32</td>
<td>23.01</td>
<td>7.69</td>
</tr>
<tr>
<td>Macro2</td>
<td>2080</td>
<td>629.95</td>
<td>1450.05</td>
<td>15.32</td>
<td>23.39</td>
<td>8.07</td>
</tr>
</tbody>
</table>
The second type of analysis, corresponding to the rows labelled Macro1 and Macro 2 in Table 23, is an attempt to directly duplicate the Regression and Back-propagation Network results by training one network to recognize all eight types of Bins. The results of these two approaches are discussed in section 5.2.

![Figure 19. Effects of Training on SSE Values](image1)

![Figure 20. Effects of Training on Accuracy](image2)

Table 24. Network Classification Results

<table>
<thead>
<tr>
<th>Type</th>
<th>Bin1</th>
<th>Bin2</th>
<th>Bin3</th>
<th>Bin4</th>
<th>Bin5</th>
<th>Bin6</th>
<th>Bin7</th>
<th>Bin8</th>
<th>Avg.</th>
<th>OVR</th>
<th>Macro1</th>
<th>Macro2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExactV.</td>
<td>84.22</td>
<td>85.90</td>
<td>85.75</td>
<td>83.01</td>
<td>91.67</td>
<td>91.46</td>
<td>89.83</td>
<td>85.91</td>
<td>91.46</td>
<td>0.435</td>
<td>16.72</td>
<td>16.72</td>
</tr>
<tr>
<td>1-OffV.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>0.653</td>
<td>46.38</td>
<td>45.89</td>
<td></td>
</tr>
<tr>
<td>ExactT.</td>
<td>97.49</td>
<td>99.48</td>
<td>99.09</td>
<td>96.09</td>
<td>99.98</td>
<td>99.88</td>
<td>99.88</td>
<td>98.996</td>
<td>23.01</td>
<td>23.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-OffT.</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>92.203</td>
<td>57.04</td>
<td>57.87</td>
<td></td>
</tr>
</tbody>
</table>

The numbers provided in Table 24, and shown in Figures 19 and 20, indicate the percentage of accuracy with which the various bins are able to classify the data records.
The Avg. field contains the simple average over the eight bins for each of the binary classification types, and OVR stands for the overall accuracy if multiple classification and null classification errors are taken into account. The results are labelled in the comparison tables below as RBF Net A. The Macro1 and Macro2 fields contain the accuracies derived from training the network to recognize all 8 bins at once. Macro1 was trained with 730 epochs and Macro2 with 1030 to show the benefits of training and to provide a measure of potential over-fitting. The results for Macro2, which are used for comparison, translate into the following tables under the heading RBF Net B.

Table 25. Error Analysis

<table>
<thead>
<tr>
<th>E-Type</th>
<th>OVR</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null V.</td>
<td>98.749</td>
<td>N/A</td>
</tr>
<tr>
<td>Mult V.</td>
<td>0.000</td>
<td>N/A</td>
</tr>
<tr>
<td>MultZ V.</td>
<td>0.000</td>
<td>N/A</td>
</tr>
<tr>
<td>Null T.</td>
<td>7.545</td>
<td>N/A</td>
</tr>
<tr>
<td>Mult T.</td>
<td>0.307</td>
<td>N/A</td>
</tr>
<tr>
<td>MultZ T.</td>
<td>0.184</td>
<td>N/A</td>
</tr>
<tr>
<td>Missed</td>
<td>N/A</td>
<td>42.13</td>
</tr>
</tbody>
</table>

In the error analysis displayed in Table 25, a Null error indicates that no bin was assigned to the record in either V (validation set) or T (training set) testing. A Mult error indicates multiple classification. A MultZ error indicates multiple classifications outside of the 1-off (acceptable error) range. These apply to the binary approach. The Missed error type is a simple misclassification outside of the 1-off (acceptable error) range, and this type of error applies to the Macro2 network.
5.2. Analysis of Performance

The data obtained from the two approaches to classification using the Radial Basis network yields some notable results. At first glance it appears as if the binary approach, splitting the problem into eight “yes-or-no” type segments, produces a high level of accuracy. Error analysis, however, indicates that this appearance is given by the fact that for each of the 8 bins, only an eighth of the data patterns on average will produce 1s, while all the others produce 0s. During training, therefore, the initial accuracy if all outputs were set to 0 is already 7 out of 8, or 87.5%. By modifying relatively few outputs, the Radial Basis network reported an accuracy of nearly 100% for the training set. The problem with this approach was revealed by subtracting out the Null errors (reporting expected 1s as 0s for all bins), and the resulting accuracy, when applied to the validation sets, falls off to practically 0%.

The Macro approaches, on the other hand, report much lower training accuracy. However, the prediction results of training all 8 bins at once does not diminish significantly when errors are taken into account. The ability of the network to actually classify data is much higher, and much more consistent. While the overall result of the binary approach on the validation set is an extremely low 0.65%, the Macro network reports 45.89%. The significance of these numbers when compared to the previous approaches to classification is discussed in the following section.

In comparing the two Macro networks, that which was trained for 730 epochs vs. that which was trained for 1030, we find that there is actually a slight decrease in the reported accuracy with more training time. While the difference between the two was less than a percent in each case, and thus Macro2 was a valid choice for comparison
against the prior approaches to the classification problem, it does indicate that the actual optimum performance of the RBF Network may not have been achieved.
Chapter 6

Performance Comparisons

6.1. Comparison with Regression Results

Table 26. Comparison of RBF Network and Regression Results

<table>
<thead>
<tr>
<th>Classtype</th>
<th>RBF NetA</th>
<th>RBF NetB</th>
<th>Forced</th>
<th>Backward</th>
<th>Forward</th>
<th>Stepwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>0.435</td>
<td>16.75</td>
<td>16.75</td>
<td>15.96</td>
<td>16.40</td>
<td>16.20</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>0.653</td>
<td>45.89</td>
<td>16.63</td>
<td>16.81</td>
<td>16.73</td>
<td>16.81</td>
</tr>
<tr>
<td>Exact T.</td>
<td>92.056</td>
<td>23.39</td>
<td>53.92</td>
<td>45.80</td>
<td>46.43</td>
<td>46.24</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>92.203</td>
<td>57.87</td>
<td>47.36</td>
<td>47.22</td>
<td>47.42</td>
<td>47.22</td>
</tr>
</tbody>
</table>

Figure 21. Radial Basis Networks and Regression Analysis

Table 26 and Figure 21 above show comparisons between the system implemented for the thesis work (RBF Net) and the methods used by CUTR for predicting the VTR change based upon the list of incentives. Forced is the heading of the column containing the force-entered regression results, using all the attributes regardless of their apparent effectiveness as predictors. Backward shows the results of backward regression; Forward and Stepwise in a similar manner show the results in the “total” column of the tables for forward and stepwise regression. The Tables for this data are 8, 10, 13 and 16.
6.2. Comparison with Backpropagation Networks

Table 27. Comparison of RBF Network and Regression Networks 1

<table>
<thead>
<tr>
<th>Classtype</th>
<th>RBF Net A</th>
<th>RBF Net B</th>
<th>Forced</th>
<th>Backward</th>
<th>Forward</th>
<th>Stepwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>0.435</td>
<td>16.75</td>
<td>15.92</td>
<td>16.16</td>
<td>16.12</td>
<td>17.23</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>0.653</td>
<td>45.89</td>
<td>15.81</td>
<td>16.08</td>
<td>16.12</td>
<td>16.21</td>
</tr>
<tr>
<td>Exact T.</td>
<td>92.056</td>
<td>23.39</td>
<td>46.51</td>
<td>46.75</td>
<td>46.12</td>
<td>45.09</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>92.203</td>
<td>57.87</td>
<td>46.68</td>
<td>46.95</td>
<td>46.85</td>
<td>46.65</td>
</tr>
</tbody>
</table>

Figure 22. Radial Basis Networks and Backpropagation Analysis

Table 28. Comparison of RBF Network and Regression Networks 2

<table>
<thead>
<tr>
<th>Classtype</th>
<th>RBFNet A</th>
<th>RBFNet B</th>
<th>Backward</th>
<th>Forward</th>
<th>Stepwise1</th>
<th>Stepwise2</th>
<th>Stepwise3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact V.</td>
<td>0.435</td>
<td>16.75</td>
<td>17.26</td>
<td>17.07</td>
<td>17.07</td>
<td>16.00</td>
<td>16.92</td>
</tr>
<tr>
<td>1-Off V.</td>
<td>0.653</td>
<td>45.89</td>
<td>17.66</td>
<td>18.30</td>
<td>17.59</td>
<td>17.52</td>
<td>18.30</td>
</tr>
<tr>
<td>Exact T.</td>
<td>92.056</td>
<td>23.39</td>
<td>48.09</td>
<td>47.22</td>
<td>47.22</td>
<td>46.31</td>
<td>47.82</td>
</tr>
<tr>
<td>1-Off T.</td>
<td>92.203</td>
<td>57.87</td>
<td>48.15</td>
<td>48.56</td>
<td>47.89</td>
<td>47.87</td>
<td>49.53</td>
</tr>
</tbody>
</table>

Figure 23. Radial Basis Networks / Hidden Backpropagation Analysis
A comparison of total (overall) accuracies is considered once more in Tables 27 and 28. Table 27 shows the difference between the results of the thesis’ system and the backpropagation networks built using the attributes of the type of regression with which each column is labelled. This table uses the networks for which the results are fully described in Tables 9, 11, 14 and 17, those with no hidden units in their architecture.

Table 28 shows the comparison results between the Radial Basis network and networks built with regression-determined attributes and a number of hidden units, as described in Section 4.2. Figures 22 and 23 correspond to the data in Tables 27 and 28, providing a graphical representation of the numbers.

The force-entered regression results were not considered significant enough to merit a hidden-unit network, however the stepwise regression technique produced three backpropagation architectures, labelled in Table 28 as Stepwise1, Stepwise2 and Stepwise3. These consist of the Total column from Tables 18, 19 and 20 respectively. Stepwise1 is a network built upon only those attributes selected as being predictive by the stepwise regression.

Stepwise2 builds a similar network, including all the costs associated with each incentive program; some of the costs were not considered to be significant by the technique. Stepwise3 builds a network on the reduced data set used in this thesis. The attributes selected from this refined set of information is found in Appendix A, and makes this comparison one of the most significant, as it is a direct comparison of network performance on identical data patterns.
Chapter 7

Conclusions

7.1. Final Analysis

As the results of Chapter 6 indicate, the two approaches employed by utilizing Radial Basis Function networks vary greatly in effectiveness. The binary approach to the problem reported extremely high accuracy when applied to the training set, however it was unable to perform with any degree of validity when it was exposed to data patterns not previously encountered. For the validation tests, the binary classification network performed the most poorly, and this work has demonstrated that splitting the categorization problem into binary components does not constitute an effective form of analysis using current techniques.

The Radial Basis Function, when applied to all bins at once, produces results that are far more useful in practice. The network implemented during this thesis reported predictions on the data that were comparable to the best of the Regression and Back Propagation techniques for Exact classification on both the training and validation sets. The usefulness of the RBF Network is more clearly revealed by examining the 1-Off classification accuracy for both testing types. In both the training and validation tests, the Radial Basis network outperforms all the other approaches by a wide margin (250% better than the next-best approach), and the 1-Off validation accuracy more than doubles that of the Stepwise Regression Back Propagation Network with hidden units, against which it can be most closely compared.
The divergence of values for the RBF Network between the Exact and 1-Off accuracies may be a natural product of the approximation characteristic of the Green’s function used by the network. (See Chapter 2.2 and [6])

While further testing is needed to show the exact limits of the Radial Basis function’s ability to classify data of this type, particularly as overtraining may have been a minor factor, the results are significant enough to conclude that the techniques presented in this work constitute the most effective method yet examined for analyzing the type of data sets provided.

7.2. Future Work

The low accuracy reported by all techniques used to classify the validation set of the traffic information indicates that further refinements to the data set may be in order. The training time required by the RBF Network was significant, the application often taking several days to produce a result. Testing on different platforms may be effective in deciding how to reduce the training time.

Further data sets from Washington, DC and Tucson, AZ will be analyzed using the same techniques. This will further test the Radial Basis Network’s ability to deal with this type of information.

In terms of the implementation itself, two improvements may be made. First, an internal rounding method may be introduced in order to eliminate the rare division-by-zero error reported in Chapter 4.3.2. Second, automatic selection of the number of hidden units as described in [14] may be implemented to increase the overall accuracy of
the system. Testing will reveal whether or not the trade-off between accuracy and training time if this addition is made will be worthwhile.
References


Appendices
# Appendix A: Table of Incentive Variables and Dependent Variable

## Table 29. Table of Incentive Variables and Dependent Variable

<table>
<thead>
<tr>
<th>Input Code</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFO</td>
<td>Facility improvements</td>
</tr>
<tr>
<td>BFR</td>
<td>Bike racks and lockers</td>
</tr>
<tr>
<td>BGC</td>
<td>Company vehicle guaranteed return trip</td>
</tr>
<tr>
<td>BGO</td>
<td>Other guaranteed return trip program</td>
</tr>
<tr>
<td>BGR</td>
<td>Rental car guaranteed return trip</td>
</tr>
<tr>
<td>BGT</td>
<td>Taxi guaranteed return trip</td>
</tr>
<tr>
<td>BGT_DOLLAR_AMT</td>
<td>Cost associated with implementing BGT</td>
</tr>
<tr>
<td>BGU</td>
<td>Unscheduled overtime guaranteed return</td>
</tr>
<tr>
<td>BIKE</td>
<td>Ratio of bike users to employees</td>
</tr>
<tr>
<td>BMM</td>
<td>Posted materials (Marketing)</td>
</tr>
<tr>
<td>BMN</td>
<td>New hire orientation (Marketing)</td>
</tr>
<tr>
<td>BMO</td>
<td>Other marketing elements</td>
</tr>
<tr>
<td>BMR</td>
<td>Company recognition (Marketing)</td>
</tr>
<tr>
<td>BMS</td>
<td>Special interest clubs (Biking, Walking)</td>
</tr>
<tr>
<td>BUS</td>
<td>Ratio of employees who use a bus system</td>
</tr>
<tr>
<td>CAR2</td>
<td>Two people per vehicle</td>
</tr>
<tr>
<td>CAR4</td>
<td>Four people per vehicle</td>
</tr>
<tr>
<td>COMPRESSED</td>
<td>Binary value: Are compressed weeks used?</td>
</tr>
<tr>
<td>CWW336</td>
<td>Compressed work week: 3 days, 36 hours</td>
</tr>
<tr>
<td>CWW440</td>
<td>Compressed work week: 4 days, 40 hours</td>
</tr>
<tr>
<td>CWW980</td>
<td>Compressed work week: 9 days, 80 hours</td>
</tr>
<tr>
<td>DFT_DOLLAR_AMT</td>
<td>Cost associated with implementing DFT*</td>
</tr>
<tr>
<td>DNT</td>
<td>Additional time off with pay</td>
</tr>
<tr>
<td>DPC</td>
<td>Increased parking costs for drive-alones</td>
</tr>
<tr>
<td>DPO</td>
<td>Other park-management strategies</td>
</tr>
<tr>
<td>DW4</td>
<td>Compressed work week: 4 days, 40 hours</td>
</tr>
<tr>
<td>DW4_DOLLAR_AMT</td>
<td>Cost associated with implementing DW4</td>
</tr>
<tr>
<td>DWO</td>
<td>Other compressed work week schedules</td>
</tr>
<tr>
<td>ISS</td>
<td>Cafeterias, ATMs, Postal, Fitness Centers</td>
</tr>
<tr>
<td>IST_DOLLAR_AMT</td>
<td>Cost associated with implementing IST**</td>
</tr>
<tr>
<td>RS_MATCH</td>
<td>A program to match drivers and riders</td>
</tr>
<tr>
<td>TARGET_AVR</td>
<td>Average Vehicle Ridership goal</td>
</tr>
<tr>
<td>TELECOMMUTE</td>
<td>Binary value: Is telecommuting available?</td>
</tr>
<tr>
<td>TRANSIT</td>
<td>Binary value: Are shuttle systems available?</td>
</tr>
<tr>
<td>DELTA_VTR</td>
<td>Vehicle Trip Rate; Dependent variable</td>
</tr>
</tbody>
</table>

*DFT, not used as a predictor, stands for On-going transit subsidies

**IST, not used as a predictor, stands for Transit information or pass sales
Appendix B: Code Modules

```java
public void Train() throws IOException {
    String outstring = "Times:\n";
    int n, Epoch;

    for (n = 0; n < MAXE; n++) {
        epochSSE[n] = 0;
    }
    maxSSE = 0;
    date = new Date();
    outstring += "Start: " + sdf2.format(date);
    for (Epoch = 0; Epoch < MAXE; Epoch++) {
        for (int row = 0; row < patterns; row++) {
            Compute(row);
            e = (target[row]-Zval);
            for (n = 0; n < neunum; n++) //For each neuron
            {
                Wdelta = e*outvector[n];
                weight[n] = weight[n] + MU1*Wdelta; // Change the weights
            }
            if (tsel > 1) // If training method is 2 or 3...
            {
                for (int m = 0; m < inpnum; m++) // Change the ref. vectors
                {
                    Rdelta = 2*weight[n]*e*toutvector[n][m]*(invector[m] - refvector[n][m]);
                    refvector[n][m] = refvector[n][m] + MU2*Rdelta;
                }
            }
            if (tsel > 2) // If training method is 3...
            {
                Mprod = 0;
                for (int m = 0; m < (inpnum - 0); m++) // Calculate Mprod value
                {
                    Qval = (invector[m] - refvector[n][m])*(invector[m] - refvector[n][m]);
                    Mprod += toutvector[n][m]*Qval;
                }
                Sdelta = (-1)*weight[n]*e*Mprod;
                Sigma[n] = Sigma[n] - Sdelta; // Change the Sigma values
            }
        }
    }
}
```
Appendix B (continued)

epochSSE[Epoch] += e*e;
if (epochSSE[Epoch] > maxSSE)
    maxSSE = epochSSE[Epoch];

// truncating the output at 4 decimal places.
Zval = (int)(Zval*10000);
Zval = Zval/10000;
}
if (epochSSE[Epoch] < MINSSE)
    Epoch = MAXE-1; // Break out of training
epochSSE[Epoch] = (int)(epochSSE[Epoch]*100);
epochSSE[Epoch] = epochSSE[Epoch]/100;
//outstring += " " + epochSSE[Epoch] + " | ";
//if (Epoch % 20 == 0) outstring += "\n";
}
date = new Date();
outstring += "\nEnd  : " + sdf2.format(date);
//Last computation, for diagram only!
Compute(display);

// JOptionPane.showMessageDialog(null, refoutput, "Zephon: Reference
Vector", JOptionPane.INFORMATION_MESSAGE);
JOptionPane.showMessageDialog(null, refoutput, "Zephon: Reference
Vector", JOptionPane.INFORMATION_MESSAGE);
JOptionPane.showMessageDialog(null, outstring, "Zephon: Timing
report", JOptionPane.INFORMATION_MESSAGE);
Errorgraph.setVisible(false);
Errorgraph.Drawgraph();
}

public class Neuron extends Object
{
public Neuron(double rep[][], int rownum)
{
    refoutput += rownum + ": ";
    for (int n = 0; n < inpnum; n++)
    {
        refvector[rownum][n] = rep[rownum][n];
        refoutput += refvector[rownum][n] + " ";
    }
    refoutput += "\n";
}

//Internal compute for a single neuron
public void Compute(int n) throws IOException
{

    //Calculation formulas for output of neuron
    R = 0;
    for (int q = 0; q < inpnum; q++)
Appendix B (continued)

```java
{  
    R += ((invector[q] - refvector[n][q])*(invector[q] - refvector[n][q]));
}
R = Math.sqrt(R);
outvector[n] = Math.exp(-((R)*(R))/((Sigma[n]*Sigma[n])));
for (int m = 0; m < inpnum; m++)
{
    toutvector[n][m] = (2-(2*refvector[n][m]))*outvector[n];
}
}

public void Compute(int inptrn) throws IOException
{
    Zval = 0;
    for (int q = 0; q < inpnum; q++)
    {
        invector[q] = x[inptrn][q];
    }
    for (int n = 0; n < neunum; n++)
    {
        NeuronCell[n].Compute(n);
        Zval += (outvector[n]*weight[n]);
    }
}
```
Appendix C: Sample Files

Trained Level 3 AND files:

```
AND.ipf
0 0 \ 1 1 \ 1 0 \ 0 1 
AND.itf
0 \ 1 \ 0 \ 0 \ 0 \ 0 
ANDt.rvf
0.765080709630533 1.0 \ 1.0 0.20697885367991298 \ 0.1043296078870729 1.0 \ 0.355591024503458 1.0989041737714167 \ 
ANDt.swf
0.4 1.3880162594688192 \ 0.4 0.00501572825766795 \ 0.4 -0.12242247314478774 \ 0.4 0.1854413995478036 \ 
```

Report file from running the above files:

```
**Report file from Network ANDt**
Pattern 0: T: 0.0 | Z: 3.974412885673406E-4
Pattern 1: T: 1.0 | Z: 0.29875977082415794
Pattern 2: T: 0.0 | Z: 0.001905613025937431
Pattern 3: T: 0.0 | Z: 0.2753843358457124

Threshold: 0.29875977082415794; Training SSE: 0.0
(if Z >= Threshold, it is considered a 1)

Rounded Z scores:
Pattern 0: T: 0.0 | Z: 0.0
Pattern 1: T: 1.0 | Z: 1.0
Pattern 2: T: 0.0 | Z: 0.0
Pattern 3: T: 0.0 | Z: 0.0

Correct: 4, Misclassified: 0, %Accuracy: 100.0
```

Segments of trained traffic files (those used for obtaining the Macro1 thesis results):

```
LADAT.ipf
0 0 0 1 1 0 0 1 -0.007940522 0 1 1 0 0 -0.019212455 -0.027867407 0 0 0 1 1 0 0 -0.010011575 -0.028660744 0.366610289 -0.267901421 \ 0 0 1 1 1 1 1 0.060433615 1 0 0 0 0 -0.019212455 -0.01555501 0.077500544 1 -0.020790547 -0.034706559 -0.027867407 -0.1308460153 0 0 0 1 0 0 -0.010011575 -0.028660744 -0.083310895 -0.002928499 \ 0 1 0 0 0 0 0 0.230751121 1 1 0 0 0 -0.019212455 0.069081128 -0.067711264 0 -0.020790547 -0.039273381 -0.027867407 0.028433111 0 1 0 0 0 1 -0.010011575 -0.028660744 0.698257327 -0.654390574 \ 0 1 0 1 0 1 0 0.518388152 1 1 0 0 0 -0.019212455 -0.167085022 -0.03824828 0 -0.020790547 -0.039273381 -0.027867407 -0.1308460153 0 0 0 0 0 0 0 1 -0.010011575 -0.028660744 0.060874447 0.070689775 \ 0 0 0 0 0 0 0 0.114467531 0 0 0 0 -0.019212455 -0.485587955 -0.067711264 0 -
```
Appendix C (continued)

0.020790547 -0.039273381 -0.027867407 0.346979648 0 0 0 0 0 1 -
0.010011575 -0.028660744 -0.124674529 0.462849259 \\n0 1 0 0 0 0 0 -0.089525841 0 0 0 1 0.30067265 -0.047616865
0.085912019 1 -0.020790547 0.051448803 0.065387353 -0.130840153 0 0 0 1
0 0 0 -0.010011575 0.75112633 0.299193382 -0.20265168 \\n0 0 1 1 0 1 1 -0.127898186 0 1 1 0 0 -0.019212455 -0.129917115 -
0.031334825 0 -0.020790547 -0.039273381 -0.027867407 -0.130840153 0 0 0 0
0 0 1 0 -0.010011575 -0.028660744 -0.067623451 0.172842011 \\n...  

LADAT.itf
0.2 \
0.7 \
0.8 \
0.3 \
0.1 \
0.8 \
0.4 \
0.2 \
0.2 \
0.1 \
0.4 \
0.6 \
0.2 \
0.3 \
... 

LAALL1.rvf
-4.965484990048543E-10 -6.66887053094437E-5 7.684886216322444E-7 1.0
1.0 1.0 1.0 1.0 -0.12793036253118942 -9.60461774948704E-7 1.0 1.0 -
1.7697754013871617E-5 2.065417866663245E-10 -0.019212295477742385 -
0.22137033413551901 0.019255600499939858 -1.1162027586523322E-8 -
0.02078531687905132 -0.02501093401337006 -0.027867407967155335
0.20389487061978853 1.0 -9.30839843204276E-10 2.878930268491265E-10 -
4.273381292915034E-14 4.866283406818378E-16 -3.908177593333534E-5 -
6.321628658834787E-7 -0.0100945812077394 -0.028754769876708875
0.05259669153504068 0.07591030159550938 \
-1.805538472298853E-12 -1.1803537335965208E-6 -5.8398005087401E-9 1.0
-3.2923712169722443E-9 1.0 1.0 1.0 -0.1279892962165396
2.0993503531088973E-8 1.0 -1.0538081181617227E-6 3.7924354272978295E-9 -
7.43669809414538E-12 -0.01921245614915573 -0.09377485819900505
0.007297816209911526 1.0 -0.020790703986505395 0.039273554801801024 -
0.027867399571237832 -0.13084015063389592 -1.030377385752942E-12
4.0268508492189316E-13 1.0 1.0 -1.94669855419051544E-14 -
2.941965632774466E-7 3.4734038397182486E-10 -0.010011574991574087 -
0.0286608444613393 -0.0593187625276591 0.2118086770043505 \
1.0 1.0 5.641330986370547E-11 9.019210385937662E-11 1.0 1.0 1.0 -
8.076393353804181E-16 0.00796894903450453 7.00300890892641E-15 1.0
1.0 7.500379134316337E-11 1.1095614447219561E-10 -0.01921245500002862 -
0.1631868479230969 -0.0436284429853249 1.0 -0.015635638002750137 -
0.025207566010307575 -0.01605276899824657 -0.13084015306436786
1.4362575952867387E-13 1.0 -3.378580223265652E-17
1.04.070262650749548E-18 1.0 6.160495228315241E-14 -
Appendix C (continued)

0.010011574999997318 0.02153789803587048 -0.07101665398171253
0.2604999239734895 \ ...

LAALL.swf
0.3871851681555741 -0.16955602936620337 \ 
0.39985109929256546 -0.17626976563578114 \ 
0.4000000136209009 -0.02961121820668597 \ 
0.462035915091099 0.4012768757586607 \ 
0.38213811339493137 0.31090134134182545 \ 
0.3894535368944984 0.12202782854167503 \ 
0.6673373699036712 -0.1842181363543176 \ 
0.399999901020371 -0.0431161453152646 \ 
0.38020375075833585 0.256181682973236 \ 
0.34595870487782326 0.7347200387778526 \ 
0.45225560161449635 0.12650345955148362 \ 
0.5113718604607151 0.28348429488390287 \ ...

LAALL.rvf
***************************************************************
NNet Files: LAALL1.swf & LAALL1.rvf
Data Files: BaseVal.ipf & BaseVal.itf
Date: 28/08/04 Time: 11:04:17 PM
***************************************************************

For each pattern, T is the Target and Z is the predicted value.

Pattern 0: T: 0.4 | Z: 0.5185153349260078
Pattern 1: T: 0.4 | Z: 0.26966354013311056
Pattern 2: T: 0.2 | Z: 0.17900644562544815
Pattern 3: T: 0.2 | Z: 0.13956177448770138
Pattern 4: T: 0.1 | Z: 0.327052150599257
Pattern 5: T: 0.7 | Z: 0.16430814991579615
Pattern 6: T: 0.8 | Z: 0.5039561275422964
Pattern 7: T: 0.5 | Z: 0.3036430006801017
Pattern 8: T: 0.7 | Z: 0.3197963135612663
Pattern 9: T: 0.3 | Z: 0.5052671476679972
Pattern 10: T: 0.5 | Z: 0.36140455728779936
Pattern 11: T: 0.4 | Z: 0.3339178336851348
Pattern 12: T: 0.7 | Z: 0.25293456145556376
...

73