3-27-1997

Mode choice modeling with neural network : Boston area case study

Soon Chung
Florida International University

DOI: 10.25148/etd.FI14060829

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

MODE CHOICE MODELING WITH NEURAL NETWORK: BOSTON AREA CASE STUDY

A thesis submitted in partial satisfaction of the

requirements for the degree of

MASTER OF SCIENCE

IN

CIVIL ENGINEERING

by

Soon Chung

1997

To:Gordon Hopkins
Engineering and Design

This thesis, written by Soon Chung, and entitled "Mode Choice Modeling with Neural Network: Boston Area Case Study", having been approved in respect to style and intellectual content, is referred to you for judgement.

We have read this thesis and recommend that it be approved.

L. David Shen

Nii O. Attoh-Okine

Young-Kyun Lee, Major Professor

Date of Defence: March 27, 1997

The thesis of Soon Chung is approved.

Dean Gordon Hopkins
College of Engineering and Design

Dr. Richard L. Campbell
Dean of Graduate Studies

Florida International University, 1997

I dedicate this thesis to my parents. Without their patience, understanding, support, and most of all love, the completion of this work would not have been possible.

ACKNOWLEDGMENTS

I wish to thank the members of my committee for their helpful comments and patience. I also want to thank Federico Frigerio for his help with computer programming.

A special thanks must go to my major professor, Dr. Y-K Lee, for his support and encouraging comments, and especially for having the confidence in me to give me the chance to do this project.

ABSTRACT OF THE THESIS

MODE CHOICE MODELING WITH NEURAL NETWORK: BOSTON AREA CASE STUDY

by

Soon Chung

Florida International University, 1997

Miami, Florida

Professor Y-K Lee, Major Professor

Demand forecasting is an essential element in the analysis of transportation systems. It is concerned with the behavior of consumers of transportation services and facilities. We choose geographic, demographic, and socioeconomic characteristics of consumers that may affect the travel demand of each selected. We use an artificial neural network to predict travel demand with characteristics selected from three different database sources: Census Summary Tape files, TIGER/Line files, and Federal Transit Administration's National Transit GIS database.

A neural network is an information processing system that is intensely parallel and neural networks are capable of learning how to classify and associate input/output patterns. This capability makes neural network a suitable approach for mode choice modeling for this study.

A neural network has two phases: the training and the testing. In the training phase, we find weights between inputs and outputs, and in the testing phase, neural network calculates outputs representing travel demand with weights from the training phase.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

I.1 BACKGROUND

Demand forecasting is an essential element in the analysis of transportation systems. It is concerned with the behavior of consumers of transportation services and facilities. Users include travelers and shippers of goods in urban, interurban, and international transport markets.

Early studies of major investments in regional highway networks and rapid progress in computational technology have led to the development of procedures to forecast trips by origins and destinations and traffic volumes on the links of a network. These modeling procedures have been continually extended and improved and are being applied worldwide.

A major innovation in the analysis of transportation demand was the development of disaggregate travel demand models based on discrete choice analysis methods. We are clearly concerned, however, with the forecasting of aggregate demands. The usual analytic approach is to subdivide the geographical area under study into zones. Then the origins and destinations of all travelers in a zone are represented by a single point within the zone, usually the geographical centroid. Aggregate travel demand models are directly estimated, with data on travel behavior at some level of aggregation within a geographical zone that serves as an observational unit for the statistical analysis.

It has been shown that for a linear regression model of an aggregation of the data prior to estimation will result in a loss of precision of the estimated parameters if the aggregate groups are not homogeneous with respect to the values of the explanatory or independent variables.

Despite these difficulties Berkson's method which is one of linear-in-parameters binary choice model and least squares procedure should be considered when appropriate and will generally be most useful under the following conditions (Ben-Akiva and Lerman, 1985):

- The available sample is extremely large, as is occasionally the case with air traffic data or in some freight demand problems, where large samples of waybills may be available.
- The data are only available in aggregated form, as is often the case with the U.S. Census data (aggregation in this case is to preserve the anonymity of respondents).
- The model structure uses only a small number of categorized variables so that the number of cells is reasonably small.
- Each respondent to a survey is observed making a large number of repeated decisions so that each individual's choices form a natural basis for grouping.

A neural network is an information processing system that is intensely parallel (Caudill and Butler, 1992). Unlike traditional computer systems, neural networks are capable of learning how to classify and associate input/output patterns (Lippmann, 1987). This capability alone makes neural networks a suitable approach for solving complex problems like forecasting travel demand from socioeconomic and demographic components pattern received from several different sources. In addition, neural networks are highly fault tolerant in the sense that given an input pattern with noise or disturbance, neural networks are still capable of recognizing that input and providing a correct output (Wasserman, 1989).

I.2 OBJECTIVE

Understanding those backgrounds mentioned above, we intend to examine an Artificial Neural Network (ANN) for the analysis of travel demand with following issues:

- Is ANN applicable to travel demand analysis?
- What is the effect if the number of hidden or output units is changed?
- What is the effect if the order of input sequence is randomized?

CHAPTER II

ARTIFICIAL NEURAL NETWORKS

II.1 INTRODUCTION

An Artificial Neural Network (ANN) can be intuitively considered a mathematical model that can mimic the relations between given inputs and outputs for any system of interest. By learning these relations, the ANN adjusts itself to fit the designated system. In other words, the ANN learning can be viewed as a function fitting process without the need of defining the necessary functional form. Since the relations between the inputs and outputs may be quite complex, most often it is difficult, if not impossible, to identify the functional form of such complex relations. The ANN can be used to accurately predict the corresponding outcomes of any inputs to the system. The learning process of the ANN is called training. The ANN must be trained specifically for a particular system before it may be used. Through the intensive connections between adjacent layer neurons, causal relations between any input and output may be adequately captured. Therefore, ANN models may sometimes provide more accurate predictions than conventional statistical models (Wang et al., 1992; Sharda, 1992).

There are two main factors that describe an Artificial Neural Networks: its architecture and its algorithm. There are many types of architecture such as perceptron, multi-layer perceptron, Hopfield Net, and Kohonen Feature Map. The most basic one would be the perceptron, a one layer network which is limited in the mappings it can learn, while a multilayer net (with one or more hidden layers) can learn any continuous mapping to an arbitrary accuracy. In this paper, the most common single hidden layer feedforward ANN is chosen. The training algorithms are almost all based on the backpropagation (BP) algorithm, which is essentially a gradient descent method to minimize the total squared error of the output computed by the net. Beside BP algorithm, there are the forwardpropagation algorithm and the selforganization algorithm (Kinnebrock, 1992).

II.2 ARCHITECTURE

Determining an appropriate configuration of hidden neurons for a given problem is usually troublesome. Hecht-Nielsen (1989) provides a proof that one hidden layer of neurons (operating sigmoidal activation functions) is sufficient to model any solution surface of practical interest, though his line of reasoning does not reflect the way in which existing training algorithms operate, and in this sense is of little more than academic interest. An alternative, more pragmatic proof that two hidden layers are sufficient is provided by Lapedes and Farber (1988). In this case, it is shown that an appropriate combination of the sigmoidal-shaped steps generated at neurons in the first hidden layer can be combined to produce a bump-shaped feature (of any width and with its peak located at any position) at a neuron in the second hidden layer.

Despite Hecht-Nielsen's (1989) proof, there are many solution surfaces that are extremely difficult to model using a sigmoidal network comprising one hidden layer. Certainly, two hidden layers provide the greater flexibility necessary to model complex-shaped solution surfaces, and are thus recommended as a starting point when developing a layered feedforward network of sigmoidal neurons. In both cases, it needs to be shown that a solution can be found readily using a single hidden layer, thus recommending such a configuration as a starting point (Flood and Kartam, 1994).

Generally, there is no direct and precise way of determining the most appropriate number of neurons to include in each hidden layer, and this problem becomes more complicated as the number of hidden layers in the network increases. Graphically, it appears that increasing the number of hidden neurons provides a greater potential for developing a solution surface that fits closely to that implied by the training patterns. In practice, however, a large number of hidden neurons can lead to a solution surface that, while fitting the training points closely, deviates dramatically from the trend of the surface at intermediate points or that provides too literal of an interpretation of the training points. In addition, a large number of hidden neurons slow down the operation of the network, both during training and in use. Conversely, an accurate model of some or all features in the solution surface may not be achieved if too few hidden neurons are included in the network.

In an attempt to resolve this dilemma, a range of different configurations of hidden neurons is normally considered, and that with the best performance is accepted (Flood and Kartam, 1994).

A multi-layer ANN with one layer of hidden units (Z units) is shown in **Figure 2.1**. In this study, output units are representing travel demand for each mode and input units are containing socioeconomic and demographic variables[1]. The output units (Y units) and the hidden units also may have biases. The bias on a typical output unit Y_k is denoted by w_{0k}; the bias on a typical hidden unit Z_j is denoted v_{0j}. These bias terms act like weights on connections from units whose output is always 1. The arrows denote the direction of propagation of the input signal, hence the term feedforward. During the backpropagation phase of learning, signals are sent in the reverse directon. This architecture can further be extended to two or more hidden layers, but for this application to predict travel demand one hidden layer might be enough.

II.3 BACKPROPAGATION ALGORITHM

Training an ANN by backpropagation (BP) involves three stages: the feedforward of the input training pattern, the backpropagation of the associated error, and the adjustment of the weights.

During feedforward, each input unit (X_i) receives an input signal and broadcasts this signal to each of the hidden units $Z_1,\ldots,Z_p$. Each hidden unit then computes its activation and sends its signal (z_j) to each output unit. Each output unit (Y_k) computes its activation (y_k) to form

[1] All variables used for input vectors are listed in Appendix A.

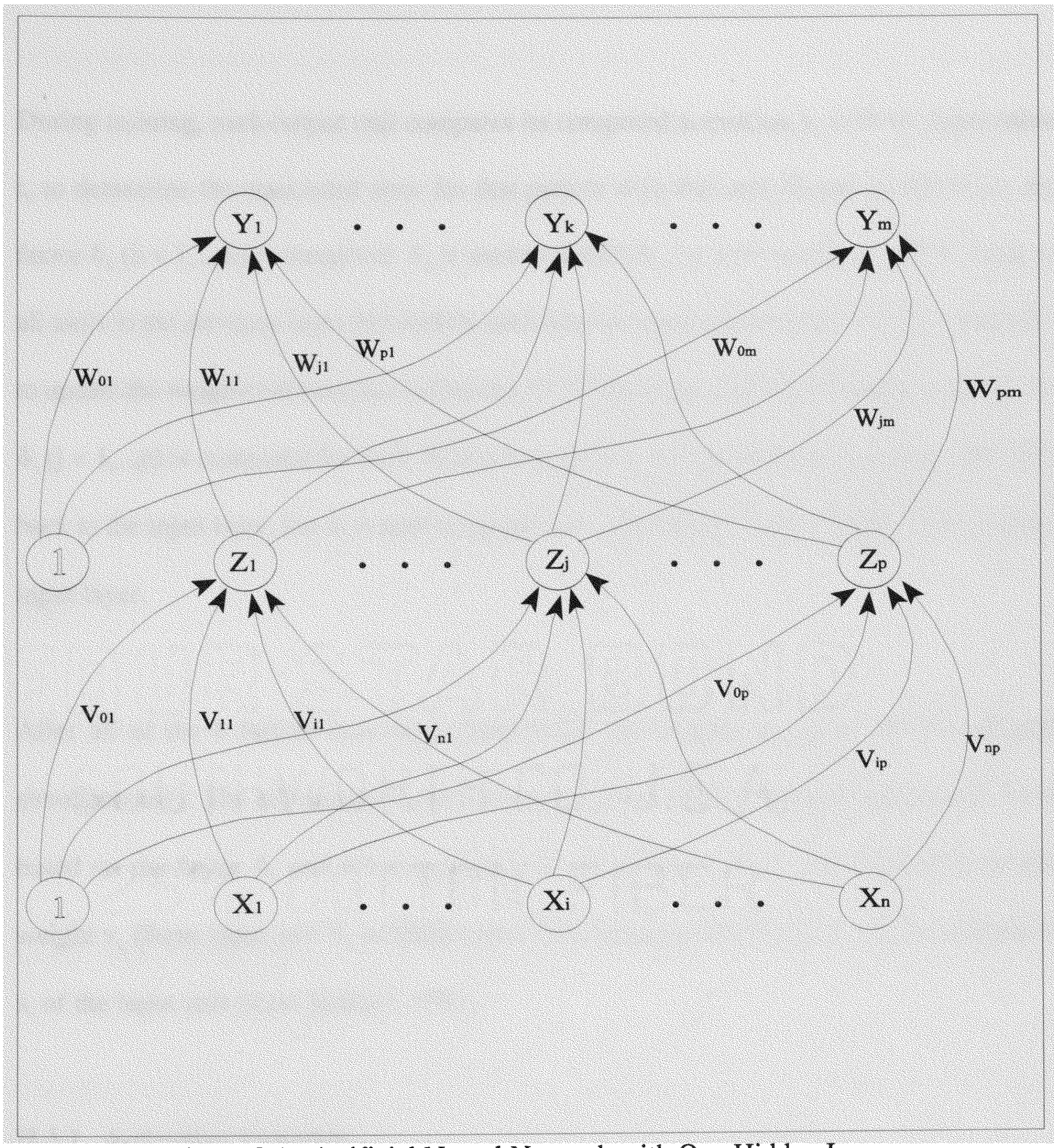


Figure 2.1 Artificial Neural Network with One Hidden Layer

the response of the net for the given input pattern.

During training, each output unit compares its computed activation y_k with its target value t_k to determine the associated error for that pattern with that unit. Based on this error, the factor δ_k ($k = 1,..,m$) is computed. δ_k is used to distribute the error at output unit Y_k back to all units in the previous layer (the hidden units that are connected to Y_k). It is also used later to update the weights between the output and the hidden layer. In a similar manner, the factor δ_j ($j = 1,..,p$) is computed for each hidden unit Z_j. It is not necessary to propagate the error back to the input layer, but δ_j is used to update the weights between the hidden layer and the input layer.

After all of the δ factors have been determined, the weights for all layers are adjusted simultaneously. The adjustment to the weight w_{jk} (from hidden unit Z_j to output unit Y_k) is based on the factor δ_k and the activation z_j of the hidden unit Z_j. The adjustment to the weight v_{ij} (from input unit X_i to hidden unit Z_j) is based on the factor δ_j and the activation x_i of the input unit (Ravi Kothari, 1993).

II.3.1 Activation Function

An activation function for a backpropagation should have several important characteristics: It should be continuous, differentiable, and monotonically nondecreasing. Furthermore, for computational efficiency, it is desirable that its derivative be easy to compute. For the most commonly used activation functions, the value of the derivative (at a particular value of the independent variable) can be expressed in terms of the value of the function (at the value of the independent variable). Usually, the function is expected to saturate, i.e., approach finite maximum and minimum values asymptotically (Ravi Kothari, 1993).

One of the most typical functions, and the one we used, is the binary sigmoid function, which has a range of (0, 1) and is defined as:

$$f_1 = \frac{1}{1 + e^{-x}}$$

with:

$$f_1' = f_1(1 - f_1)$$

This function is shown in **Figure 2.2**.

II.3.2 Initial Weights and Biases

The choice of initial weights will influence whether the ANN reaches an absolute minimum (or only a relative -local minimum) of the error and, in that case, how fast it converges. The update of the weight between two units depends on both the derivative of the upper unit's activation function, and the activation of the lower unit. For this reason, it is important to avoid choices of initial weights that would make it likely that either activations or derivatives

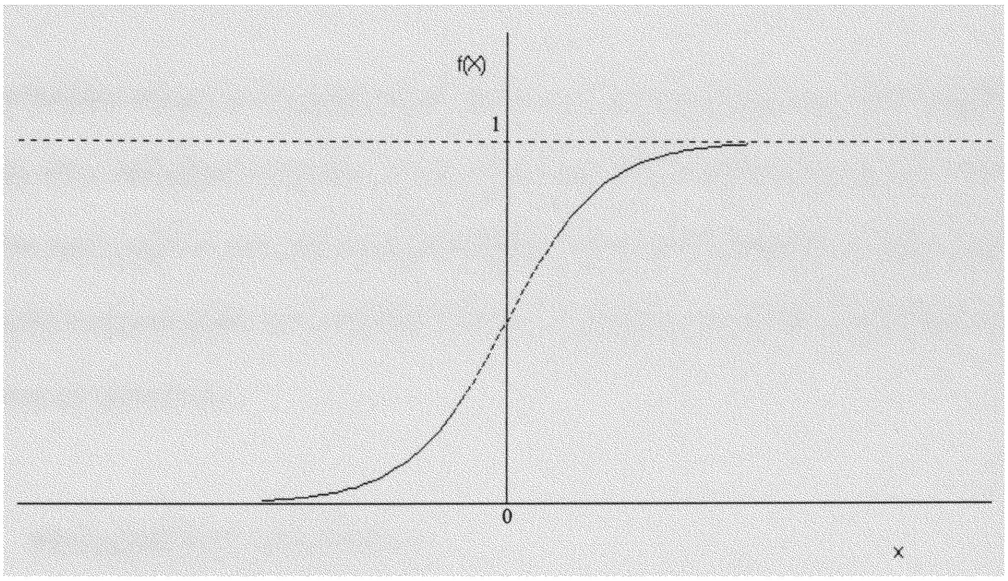


Figure 2.2 Binary Sigmoid, range (0, 1)

are zero, since the weight correction would be zero. The values for the initial weights must not be too large, or the initial input signals to each hidden or output unit will be likely to fall in the region where the derivative of the sigmoid function has very small value (saturation) and the correction would be negligible. On the other hand, if the initial weights are too small, the ANN input to a hidden or output unit will be close to zero and the correction would also be negligible.

The procedure chosen to initialize the ANN was the most common one. The weights (and biases) were initialized to random values between -0.5 and 0.5. The values need to be positive and negative because the final weights after training may be of either sign. The computer's random generator is seeded with the computer's clock and then pseudo random numbers are generated.

II.4 PROGRAM ALGORITHM

II.4.1 Training Procedure

The classic sigmoid activation function was chosen and the standard BP algorithm was used for net training. The algorithm employed (in meta-code)[2] is the following:

[2] The complete source code for the ANN is listed in Appendix B.

Step 0. Generate Neural Net in memory and initialize weights to random numbers in the (-0.5, 0.5) range.

Step 1. While stopping condition is false, do Steps 2-9.

The stopping condition is that the average squared error be less than a given maximum.

Step 2. For each training set, do Steps 3-8.

Feedforward:

Step 3. Each input unit (X_i, $i = 1, \ldots, n$) receives an input signal x_i and broadcasts this signal to all units in the layer above (hidden units).

Step 4. Each hidden unit (Z_j, $j = 1, \ldots, p$) sums its weighted input signals,

$$z\_in_j = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

applies its activation function to compute its output signal,

$$z_j = f(z\_in_j) = \frac{1}{1 + e^{-z\_in_j}}$$

and sends this signal to the output unit.

Step 5. The output unit (Y_k, $k = 1, \ldots, m$) sums its weighted input signals,

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and applies its activation function to compute its output signal,

$$y_k = f(y\_in_k)$$

Backpropagation of error:

Step 6. The output unit receives a target value corresponding to the input training

set, and computes its error information term.

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

calculates its weight correction term (used to update w_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j$$

calculates its bias correction term (used to update w_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k$$

and sends δ_k to hidden layer units.

Step 7. Each hidden unit (Z_j, $j = 1, \ldots, p$) sums its delta input (from units in the

layer above),

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

multiplies by the derivative of its activation function to calculate its error

information term,

$$\delta_j = \delta\_in_j f'(z\_in_j)$$

calculates its weight correction term (used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i$$

and calculates its bias correction term (used to update v0j later),

$$\Delta v_{0j} = \alpha \delta_j$$

Update weights and biases:

Step 8. The output unit updates its bias and weights ($j = 0, \ldots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$$

Each hidden unit (Z_j, $j = 1, \ldots, p$) updates its bias and weights ($i = 0, \ldots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$$

Step 9. Test stopping condition.

Step 10. Write net description, weights and biases to an output file.

Step 11. Test the trained net with a set of values not used for training.

where,

X_i	Input Layer Vector's *i*th neuron
x_i	*i*th input neuron's response
Z_j	Hidden Layer Vector's *j*th neuron
z_j	*j*th hidden layer neuron's response
Y_k	Output Layer Vector's *k*th neuron
y_k	*k*th output layer neuron's response
t_k	*k*th output layer neuron's target value
v	weight between input layer and hidden layer

w	weight between hidden layer and output layer
α	Momentum Term
δ	Delta correction error

An iteration is one cycle through the entire set of training vectors. Typically 6,000 to 10,000 iterations were required for training the ANN. The previous program updates the weights after each training pattern is presented, as opposed to a common variation in which the weight updates are accumulated over an entire iteration before being applied (batch updating).

The mathematical basis for the backpropagation algorithm is the optimization technique known as gradient descent. The gradient of a function (in this case, the function is the error and the variables are the weights of the net) gives the direction in which the function increases more rapidly; the negative of the gradient gives the direction in which the function decreases most rapidly. In this manner, by finding the gradient on a given point in the hyperplane of the function one can find a minima where the squared errors are minimized. This is similar to a river flowing down a mountain following the minimum resistance path. It is then obvious that the minima found is not necessary an absolute minima (the lowest point in the hole hyperplane) but might be a relative minima (a point which is lower than the points around it in a small radius). This is one of the most critical drawbacks of the BP algorithm.

II.4.2 Feedforward Procedure

After training, a backpropagation neural net is applied by using only the feedforward phase of the training algorithm. The application procedure is as follows:

```
Step 0. Initialize weights (from training algorithm).
Step 1. For each input vector, do Steps 2-4.
    Step 2. For i = 1, . . . , n: set activation of input unit
        $x_i$;
    Step 3. For j = 1, . . . , p:
        $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$
        $z_j = f(z\_in_j)$
    Step 4. For k = 1, . . . ,m:
        $y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$
        $y_k = f(y\_in_k)$
```

CHAPTER III

DATABASE

III.1 STUDY AREA

Boston is the capital of Massachusetts and New England's largest city. Named for the English port in Lincolnshire, England, the city is situated on a hilly peninsula, where the Mystic and Charles rivers flow into Massachusetts Bay. The city proper is small in terms of area and has a population of 574,283. The metropolitan area of Greater Boston is, however, 1,038 square miles, with a population of 2.6 million (Shen, 1995).

Boston is a major financial, insurance, educational, and industrial center in the New England region. The development of facilities in the manufacturing of sophisticated electronics, computers, and chemical has a great impact on the industrial structure of the metropolitan area. Tourist industry has a significant impact in the Boston economy as well.

Being a city with over three hundred years of history, Boston has a densely developed central business district, which remains strong till today. In addition to downtown Boston, Cambridge is also a major employment center. Because the land has been highly developed, large number of parking facilities are rare and street parking is limited. The lack of parking combined with the road congestions provides the strongest incentive for using public transit. In fact, Boston is ranked the fourth among the major U.S. metropolitan areas as having the highest number of workers (31.5%) using public transportation, following New York City/New Jersey, Washington, D.C., and San Francisco (APTA, 1994).

Massachusetts Bay Transportation Authority (MBTA) was created in 1964 as the forerunner of many regional transport planning and operating agencies. MBTA provides services to 78 towns and cities within its operating district and 52 non-member communities, serving 4.15 million people in 2,344 square miles of service area. The MBTA now operates 159 bus routes, three heavy rail lines (Blue Line, Orange Line, and Red Line) and four routes, four light rail (Green Line) routes, four trackless trolley routes, 11 commuter boat routes (Shen,1995).

Three counties such as Suffolk County, Middlesex County, and Norfolk County were selected for describing the metropolitan area of Boston, and they consist of total 586 census tracks. Geographical and demographic factors of those census tracks that may affect the travel demand were selected based on the study (Shen et al. 1994). Factors are described as below:

• **Distance to guideway transit station**

Distance to the nearest guideway transit station is very important factor in choosing a transit mode for commuting. In this study, the distance is defined as the distance from the station to the centroid of the census tract studied.

• Income

Total income is the algebraic sum of the amounts reported separately for wage or salary income; net non-farm self-employment income; net farm self-employment income; interest, dividend, or net rental or royalty income; Social Security or railroad retirement income; public assistance or welfare income; retirement or disability income; and all other income. Receipts from the following sources are not included as income; money received from the sale of property; income from food stamps, public housing subsidies, medical care, withdrawal of bank deposits; money borrowed; tax refunds; exchange of money between the relatives living in the same household; insurance payments.

Family income is the summation of the incomes of all members of the family that are 15 years old and older. All persons with income not reported were assigned the reported income of persons with similar characteristics.

• Place of work

Data on place of work refer to the geographic location at which workers carried out their occupational activities during the reference week. The comparability of the 1980 and 1990 place of work census category was affected, because the data from the 1980 census was based on one half of the full census sample and the data from the 1990 census was based on the full sampling.

• Travel time to work

Travel time to work refers to the total number of minutes that it took the person to get from home to work during public transportation, picking up passengers in carpools, and time spent in other activities related to getting to work.

• **Race**

The data for race represent self-classification by people according to the race with which they most closely identify. Race categories include both racial and national origin or socio-cultural groups. If a person could not provide a single race response, the race of the mother was used. If a single race response could not be provided for the person's mother, the first race reported by the person was used. The races include the following classification: White; Black; American Indian, Eskimo, or Aleut; Asian or Pacific Islander, and Other Race (includes entries such as multiracial, multiethnic, mixed, interracial, or a Spanish/Hispanic origin group).

III.2 DATA PRE-PROCESSING

The data employed in the ANN training and testing was obtained from three sources: 1990's Census Summary Tape Files, TIGER/Line files, Federal Transit Administration's National Transit Geographic Information System database. The data were processed at the geographic area levels called census tracts. Census tracts are subdivisions of counties with similar or homogeneous population characteristics, economic status and living conditions. They were delineated with the intention of being maintained over a long period of time so that statistical comparisons can be made from census to census. Census tracts are identified by a four-digit basic number and may have two-digit basic suffix (U.S. Department of Commerce and Bureau of the Census, 1992).

III.2.1 1990's Census Summary Tape Files (STF)

The Census Bureau offers the summary tape files (STF) which contain the census tabulations of specific areas. STF1 and STF3 are the most commonly used. STF2 and STF4 include detailed tabulations by race and Hispanic origin that are rarely used in transportation applications. STF1 is from the 100 percent count and includes information on household relationship, sex, race, Hispanic origin, age and marital status. STF3 is from the sample count or census "long form" (which in 1990 was a sample of 1 in 6 households nationwide, and in 1980 also a sample of 1 in every 6, except for areas with population less than 2,500). The long form included the same basic demographic and housing items as the short form, as well as additional questions on social and economic characteristics and many housing characteristics (GIS/Trans. Ltd., 1994).

For this study, the STF3 files were selected to perform the analysis. The Census data file is segmented into 35 dBaseIII files, designated STF300ss.dbf through STF334ss.dbf, where *ss* is the two-character state abbreviation and *ma* is for Massachusetts. The STF300 segment contains the full 67 field identification section. Segments STF301 through STF334, each contains eight identification fields repeated from the STF300 segment. Fields in the numeric data tables are named according to a convention which identifies the tables and the sequence of the data items within the table (Bureau of the Census, 1992).

Since each state's information is contained in multiple multimegabytes database files, a method had to be devised to extract the valuable information into more manageable files. After carefully considering the available tools at the time, Microsoft's Microsoft Query (MQ) was chosen due to its flexibility. With MQ, we were able to perform Structured Query Language (SQL) queries to the database, thus allowing us to formulate the searches of the database in a plain English format. For example:

```
SELECT stf300ma.STATEFP+stf300ma.CNTY+stf300ma.TRACTBNA 'ID',
stf300ma.POP100 'Population'
FROM d:\stf300ma.dbf stf300ma
WHERE (SUMLEV = '140')
```

This previous example generates a column with the ID of the tract in a format that matches the geographical data naming convention of TIGER/Line files; and also a column with the population from the stf300ma.dbf file which corresponds to Boston. In this case, it is also used a classification of SUMLEV, which allows to extract only the data at the census tract level. MQ also has the added benefit of being able to handle multiple file formats (dBaseIII, Plain Text, FoxPro, and Excel to name a few), thus serving a double function of extracting the data and translating it to a useful format. In this manner, once the data was extracted it is available to save it as both a TAB separated text file and a dBaseIII database file.

III.2.2 TIGER/Line Files

The geographic database used for this study is from the Census Bureau's Topologically Integrated Geographic Encoding and Referencing (TIGER) system. The geographic database is in the so called TIGER/Line files. This database system was created by the Census Bureau in conjunction with the US Geological Survey. TIGER/Line files simply contain every street and block type in the United States. TIGER/Line files' data is divided into three major types of data: 1) Line features (roads, railroads, hydrography, etc.) 2) Landmarks (schools, churches, parks, cemeteries) 3) polygons (geographic entity codes for areas). The line feature and polygon information form the majority of data in the TIGER/Line files (U.S. Department of Commerce and Bureau of the Census, 1992).

The TIGER/Line file structure consists of 14 record types that collectively contain geographic information (attributes) such as latitude/longitude coordinates of linear and point features, and area and polygon boundaries. A separate file exists for each of the 14 record types for each county or county equivalent. The files are named TGR*st*cty.f5x, where *st* is the Federal Information Processing Standard (FIPS) State Code, cty is the FIPS county code, and x is the record type (U.S. Department of Commerce and Bureau of the Census, 1992). For this study, *st* is 25 and cty can be 017 for Middlesex County, 021 for Suffolk County, and 025 for Norfolk County.

TIGER/Line has a relationship with the 1990 Census Statistical Data. TIGER's digital description of the Nation's legal and statistical entities includes FIPS codes and the Census Bureau codes so that these can be matched easily with the 1990 census data. For this study, we are interested in performing the analyzes on geographic area levels called tracts. Census tracts are identified by a four-digit basic number and may have two-digit basic suffix (U.S. Department of Commerce and Bureau of the Census, 1992). The main information employed from this database is the latitude and longitude of an internal point of the tract. There are two available points per tract, one of them is the center of the tract which might not be contained within the tract depending on the tract's shape, and a properly internal point which in most cases is the same as the center but which in any circumstance is guaranteed to be internal.

III.2.3 F.T.A.'s National Transit Geographic Information System database

The Federal Transit Administration is conducting a three-year project together with some representatives from organizations such as the American Association of State Highway and Transportation Officials, the Conference of Minority Transportation Officials, and others, in order to develop a National Transit Geographic Information System. The Transit Geographic Information System (GIS) will benefit the surface transportation planning management; since, the system will facilitate the access to and association of relevant and numerous transit data that will speed the decision making process (The Urban Transportation Monitor, 1994).

The Transit GIS is being implemented in three stages. In the first stage, the most important to this study, the Transit GIS was established in headquarters, with a core group of in-house GIS users that apply the new technology to enhance the quality and the content in FTA reporting. The year of completion was 1994 and the GIS includes urban area maps showing Rapid, Light, and Commuter Rail System Networks; HOV Facilities; People Mover System Networks; Ferry Terminals and Routes; Intermodal Terminals; Routes maps for up to one-half of public bus routes, stations, and maintenance facilities; and other features (The Urban Transportation Monitor, 1994).

This data was available in the form of GIS databases compatible with the TransCAD GIS package. The information provided consisted of two type of entities: points and lines. The line entities describe elements such as Rapid, Light, and Commuter Rail Systems and Routes. The point entities describe Junctions, Crossings and most important Stations. Each data record was labeled with the type of element it described, so by applying a filter we were able to obtain only records of STATION type. Once we had those records, we were able to export this information to a dBaseIII database file. The information we required was the latitude/longitude of the stations.

III.3 DATA PROCESSING

Once we had these three different data sources from the STF files, the TIGER/Line files, and the FTA GIS database in their raw state, it was needed to consolidate this information into one database. The information, however, required more processing in certain cases. For instance, the geographical information was meaningless as it was. It was necessary to transform the latitude/longitude information into distances. However, the distance required was the minimum distance to the closest station from the tract inner point. A program needed to be developed to calculate this distance.

III.3.1 Distance Program

The program language chosen to develop this program[3] was Visual Basic. The main factor determining the choice was the fact that the required information (Station position and tract center point) was contained in two different dBaseIII files. Visual Basic provided us with an interface in the form of the Open Data Base Connectivity Layer (ODBC) which shielded us from the intricacies of the database format, allowing us to simplify the programming task and concentrate only on the distance algorithm. The algorithm essentially calculates, for each tract, the distance to all the stations, writing to an output text file the shortest distance. The distance was calculated by using the maximum arc method which takes into account Earth curvature thus yielding a more precise value. The program also contemplates the use of the Euclidean distance that although been less precise is much faster to calculate. However, it was never required since the run time for the program in most cases was less than 15 minutes.

[3] The complete source code for the distance is listed in Appendix C.

III.3.2 Data Manipulation

After processing the data, it was required to further analyze it, and consolidate it in one single file. This was done using Microsoft Excel 5.0. Since it was expected that certain operations would be required to be performed on the data, a spreadsheet with its capability of displaying and manipulating the information as a whole was the perfect tool for the task. However, due to the extent of the information, the size of files grew roughly to 5 megabytes in the end.

Once the data files were imported into Excel, it was needed to correlate the STF data with the distance to the station data extracted from the TIGER/Line and the FTA GIS database information. This was done based in the 'ID' field. The 'ID' field is the indexing number given to each tract. It consists in a series of numbers resulting from the concatenation of the State FIPS code, plus the county code plus the tract code within that county. For instance, 250173001 means:

1. State FIPS:	25	(Massachusetts)
2. County Code:	017	(Middlesex County)
3. Tract Number:	3001	

In this manner each tract is indexed by a unique number. By sorting both data sets and placing them together in one single worksheet, the final raw data was obtained. Further checks were made to ensure that there was no mistake in the data. For instance, the number of records in one table matched the number of records in the second. Also, every so many records the information was carefully checked to make sure the numeration was synchronized. In this manner, Excel files for each of two sets of output were generated.

III.3.3 Generation of Input Files

The generation of the final files is a lengthy process in which the characteristics of the input data must be taken into account. Once the input data was analyzed a mapping function was devised to convert the data into an appropriate form suitable to be used in an ANN.

The obtained data from pre-processing, was analyzed to examine its suitability for ANN training. It was discovered that input variables are generally distributed around the quite small value of 0.05, as we can see in **Figure 3.1**. The graphs were generated with the input variables and they represent the general tendency of the input data vectors. The data that could be represented as a percentage of population, was divided by the maximum value for each variable to obtain a normalized value. This value will have a range of (0, 1), thus being exceptionally well suited for ANN training. However, due to the fact that in most cases most of the data is concentrated in the lower (0, 0.1) range, a mapping function needs to be introduced.

The main purpose of the mapping function would be to transform the input vectors into a set of vector for which the distribution is closer to a normal distribution with the mean around 0.5. The function must also have a simple inverse function to in that manner transform the

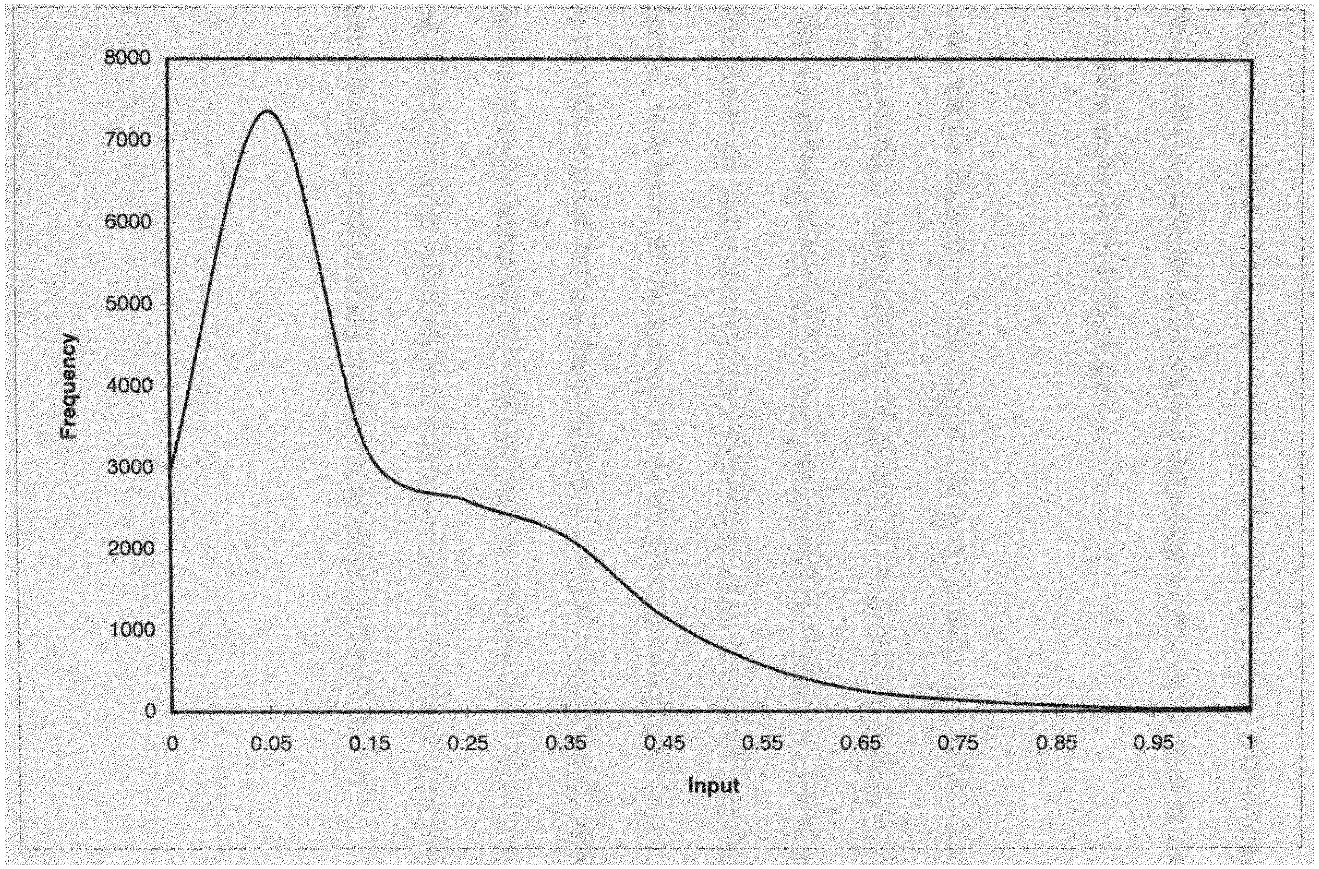


Figure 3.1 Distribution of Input Vectors

results from the ANN into a vector which can then be compared to the expected result to assess the ANN quality.

Simply, a linear transformation was used. The linear transformation provided us with the simplest function capable of changing the range of the input vectors so that most of them were located in the (0.3, 0.7) range.

Once the Excel files were generated, it was necessary to extract the data into properly formatted text files. The program for neural network was designed to process ASCII files. ASCII is a standard method of encoding alphanumeric characters, essentially a standard DOS text file. Excel provides an extremely simple export command that creates files with exactly this format. However, all the data could not be used for training. It was necessary to further divide the information into two separated files, one for training and another for testing. It was decided to use approximately 50% of the data for training and 10% of the rest of the data for testing. The files[4] were saved in their preprocessed format, ready to be fed into the ANN for the actual training and evaluation and it was done by Excel macro[5].

[4] Sample file for training and testing is in Appendix D.

[5] Excel macro to generate training and testing files is in Appendix E.

III.3.4 Scenario

To apply the ANN to forecasting travel demand with different assumptions such as alteration of the number of hidden neurons and output, randomization of the order of input sequence, and application of transformation of method: normalization and transformation, four scenarios were provided for preparing training and testing files.

In the first scenario, we decided to provide two sets of outputs which are 5 outputs and 6 outputs. 6 outputs consist of 6 different travel mode such as drive alone, carpool, bus, subway and rail, walk, and other. Percentages of travel demand for each mode are described in **Table 3.1**. For 5 outputs, bus and subway and rail are added together as public transit.

After we had database for two sets of output, second scenario was applied to transform both database with normalization and linear-transformation. Therefore, we had 4 training and testing files: normalized database for 5 output, linear-transformed database for 5 output, normalized database for 6 output, and linear-transformed database for 6 output.

In third scenario, we randomized the order of input sequence for each case. Eight training and testing files were prepared for assumptions through 3 scenarioes. However, changing the number of hidden neuron was not performed because the ANN program was coded for this case. We are able to change the number of hidden neurons when we train the ANN. By the way, fourth scenario was needed for alteration of the number of hidden neurons and we choosed 3 different numbers such as 5, 10, and 20.

Table 3.1 Demand of Travel Mode to Work in Boston

Mode / ID	Drive Alone	Carpool	Bus	Subway & Rail	Walk	Other	Sum
25017	536383[a]	73600	28585	37765	39406	31357	747096
Percent (%)	**71.80**	**9.85**	**3.83**	**5.05491**	**5.27**	**4.19718**	**100**
25021	234515	30614	5540	23606	12375	16134	322784
Percent (%)	**72.65**	**9.48**	**1.72**	**7.31325**	**3.83**	**5.00**	**100**
25025	137137	34648	41614	45709	41812	23189	324109
Percent (%)	**42.31**	**10.69**	**12.84**	**14.103**	**12.90**	**7.1547**	**100**

Note: a. person

CHAPTER IV

ARTIFICIAL NEURAL NETWORK MODELING

Once the training files were created, the actual training of the ANN was conducted. Even though there seems to be a linear path followed, there were many iterations of algorithms, programs, and input pre-processing before this final form was obtained. Essentially, the ANN program requires the input of not only the training and testing vectors, but also certain parameters of the ANN to be trained. Also, after training, the resulting ANN network needed to be serialized in a manner that it could be regenerated for further use in the future.

IV.1 ARTIFICIAL NEURAL NETWORK TRAINING

To train the ANN, the program needs a description of the ANN to be employed and we need to put parameters into a dialog box[6] provided by the program. The parameters required for training and testing are listed below:

•Number of Input Neurons (Number of variables)

•Number of Hidden Neurons

•Number of Output Neurons

•Learning Rate

•Momentum Factor (α)

[6] Graphical Interface is shown in Appendix F.

•Minimum Training Error

•Input file with a complete path

•Output file with a complete path

The number of input neurons is determined by the number of variables and since it was decided that a certain set be employed for all the cases, a fixed number of 30 input was established. However, even then it was decided not to hard code this into the program. Therefore, the number of input variables is one of the parameters that must be passed to the training program.

The learning rate determines how big the *jump* would be in the direction of the decreasing error. Mathematically it determines the delta by which each weight should be corrected given the direction of the decreasing error gradient. If the learning rate is too small, the ANN might become trapped in a local minima, if too big it might never stabilize and oscillate indefinitely. The learning rates employed varied from 0.5 to 0.001 and 0.5 was used for this study.

The momentum factor is the parameter which helps networks learn faster by basing the current weight changes proportionally on the previous weight changes. Generally, values between 0.0 and 1.0 are used and for this study, 0.01 was used. If 0.0 is selected. the momentum factor term is not used.

The minimum training error enables the network to stop training when the specified error value is reached. The training error is the sum of the squared differences between the outputs and target values. Although values in the range of 0.1 are often used, 0.01 was used for this study to get the extra accuracy.

The input file is the file obtained after the procedures explained in Chapter III. This file contains the input vectors and the target values for the given vectors. It is with these values that the ANN trains. The output file will contain the description of the ANN and the resulting weights after training. This file is of importance because it allows the experimenter to save a given network to be employed at a later time.

The error output file was saved automatically with ASCII text file format after each 100 iterations in the directory the program of ANN is located. It is important because it records each 100 iterations average square error. The average square error determines the average error squared between the ANN response to the training vectors and the target values and shows how the training procedure progresses. **Figure 4.1** shows how the average squared error converges and the difference of the progress of the ANN training between with 5 outputs and with 6 outputs. In the legend, titles consist of three digit numbers and one alphabet character. First number of title is for number of outputs and following two numbers are for the number of hidden neurons. Last alphabet character is for the method of transformation: N stands for normalization, T stands for linear-transformation, and R stands for randomization. **Figure 4.2** shows the difference of the progress of the ANN training between with the normalized training files and the transformed training files. Also, **Figure 4.3** shows the difference of the progress of the ANN training with the training files between in the normal order and in randomized order.

The main decision the training algorithm must take, is when to consider the ANN to be fully trained. It was decided to stop the training whenever the average squared error reached a critical value. The result of the training procedure is summarized in **Table 4.1**.

IV.2 ARTIFICIAL NEURAL NETWORK TESTING

After a particular network was successfully trained, it was tested with the file generated for testing. The output file from an ANN training contains a description of the network and the values for the different weights. With this file and the error output file, we were able to see how well the ANN performed. Performance is understood as how high (worse) or low (best) the average squared error for this ANN was when confronted to data it was not trained with but that is closely related to the data used to train it. Results of the ANN's testing are summarized in the following **Table 4.2** and **Table 4.3**.

IV.3 DISCUSSION

After the training and testing of the ANN was completed, we had 14 files containing average squared errors through the training procedure and 14 output files containing predicted travel demand from the testing procedure, according to four scenarioes assumed in Chapter 3. These files were opened in Excel for further analysis and output values for travel demand were compared to actual travel demands to evaluate the ANN's performance in forcasting travel demand.

From average squared errors in error files, we are able to see the progress of the ANN's learning procedure which is called the "ANN's training". **Figure 4.1** contains errors curves for 6 different cases. First error curve is for 5 outputs, 5 hidden neurons, and normalization. Second error curve is for 5 outputs, 10 hidden neurons, and normalization. Third error curve is for 5 outputs, 20 hidden neurons, and normalization. Fourth error curve is for 6 outputs, 5 hidden neurons, and normalization. Fifth error curve is for 6 outputs, 10 hidden neurons, and normalization. Sixth error curve is for 6 outputs, 20 hidden neurons, and normalization. Therefore, the effect of alteration of the number of hidden neuron and outputs is described in this figure. We are able to see that average squared errors for 5 outputs are generally lower than those for 6 outputs. Number of iterations of training for 5 outputs are less than number of iterations for 6 outputs.

Figure 4.2 contains error curves for 6 different cases. First error curve is for 6 outputs, 5 hidden neurons, and normalization. Second error curve is for 6 outputs, 10 hidden neurons, and normalization. Third error curve is for 6 outputs, 20 hidden neurons, and normalization. Fourth error curve is for 6 outputs, 5 hidden neurons, and transformation. Fifth error curve is for 6 outputs, 10 hidden neurons, and transformation. Sixth error curve is for 6 outputs, 20 hidden neurons, and transformation. Therefore, the effect of transformation is described in this figure. We are able to see that average squared errors for transformation are generally lower than those for normalization.

Figure 4.3 contains errors curves for 4 different cases. First error curve is for 5 outputs, 10 hidden neurons, and normalization. Second error curve is for 5 outputs, 10 hidden neurons, normalization, and randomization. Third error curve is for 5 outputs, 10 hidden neurons, and transformation. Fourth error curve is for 5 outputs, 10 hidden neurons, normalization, and randomization. Therefore, the effect of randomization of the input sequence is described in this figure. We are able to see that randomization of input sequence did not affect the ANN's training procedure.

Table 4.1 shows results of the ANN's training for all cases. 293 census tracts are used for training. Numbers for high water mark are numbers of census tract satified minimum error given at the beginning stage of training and if the ANN was fully trained numbers are underlined as its appear in the table. Numbers without underline represent that the ANN was not fully trained and the training process did not progress. The ANN training procedure was stopped intentionally in these cases: the case for 5 outputs, 5 hidden neurons, and normalization, the case for 5 outputs, 5 hidden neurons, and transformation, the case for 6 outputs, 5 hidden neurons, and normalization, the case for 6 outputs, 10 hidden neurons, and normalization, the case for 6 outputs, 5 hidden neurons, and transformation, and the case for 6 outputs, 10 hidden neurons, and transformation.

For the ANN's testing, 50 census tracts were selected, and input variables of these census tracts are included in the testing files. With a testing file and an weight file which is the result of training for each case, the ANN's testing was conducted. As results of testing, we have 14 output files containing normalized values for predicted travel demands.

Figure 4.4 shows actual travel demands for 'drive alone' with three predicted travel demands for three different cases: the first case for 5 outputs and 5 hidden neurons, the second case for 5 outputs and 10 hidden neurons, and the third case for 5 outputs and 20 hidden neurons. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0018 for the first case, 0.002 for the second case, and 0.0012 for the third case.

Figure 4.5 shows actual travel demands for 'carpool' with three predicted travel demands for three different cases: the first case for 5 outputs and 5 hidden neurons, the second case for 5 outputs and 10 hidden neurons, and the third case for 5 outputs and 20 hidden neurons. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0091 for the first case, 0.0094 for the second case, and 0.01 for the third case.

Figure 4.6 shows actual travel demands for 'public transit' with three predicted travel demands for three different cases: the first case for 5 outputs and 5 hidden neurons, the second case for 5 outputs and 10 hidden neurons, and the third case for 5 outputs and 20 hidden neurons. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0052 for the first case, 0.0039 for the second case, and 0.0036 for the third case.

Figure 4.7 shows actual travel demands for 'walk' with three predicted travel demands for three different cases: the first case for 5 outputs and 5 hidden neurons, the second case for 5 outputs and 10 hidden neurons, and the third case for 5 outputs and 20 hidden neurons. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0047 for the first case, 0.0022 for the second case, and 0.0012 for the third case.

Figure 4.8 shows actual travel demands for 'other' with three predicted travel demands for three different cases: the first case for 5 outputs and 5 hidden neurons, the second case for 5 outputs and 10 hidden neurons, and the third case for 5 outputs and 20 hidden neurons. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0039 for the first case, 0.0042 for the second case, and 0.0036 for the third case.

Figure 4.9 shows actual travel demands for 'drive alone' with two predicted travel demands for two different cases: the first case for 5 outputs and 20 hidden neurons and the second case for 6 outputs and 20 hidden neurons. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0012 for both cases.

Figure 4.10 shows actual travel demands for 'drive alone' with two predicted travel demands for two different cases: the first case for 5 outputs, 20 hidden neurons, and normalization and the second case for 5 outputs, 20 hidden neurons, and transformation. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.0012 for the first case and 0.0027 for the second case.

Figure 4.11 shows actual travel demands for 'drive alone' with two predicted travel demands for two different cases: the first case for 5 outputs, 10 hidden neurons, and normal sequence and the second case for 5 outputs, 10 hidden neurons, and randomized sequence. Travel demands in this figure are those of 50 census tracts selected for testing and are normalized to have a range of (0, 1). Predicted demands are closely matched to actual demands with average errors: 0.002 for the first case and 0.0023 for the second case.

As shown in **Table 4.2** and **Table 4.3**, average squared errors becomes smaller as the number of hidden neurons are increased. It seems that linear-transformation did not affect results of the ANN's testing and randomization of the order of input sequence did not bring much difference. However, we are able to have better testing results with 5 outputs than with 6 outputs. Even though the ANN was not fully trained, the testing results were acceptable with average squared errors around 0.006.

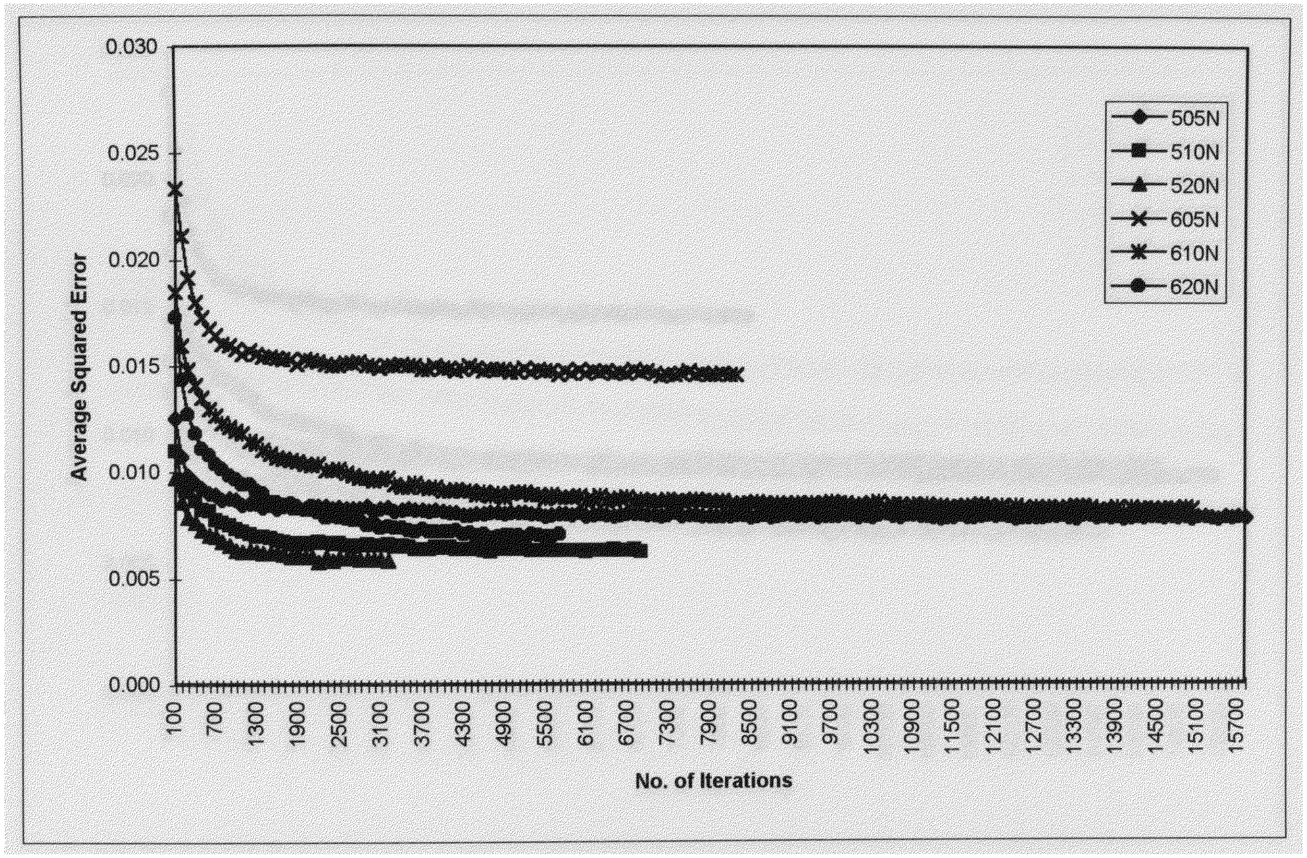


Figure 4.1 Average Squared Error of 5 Output Neurons and 6 Output Neurons

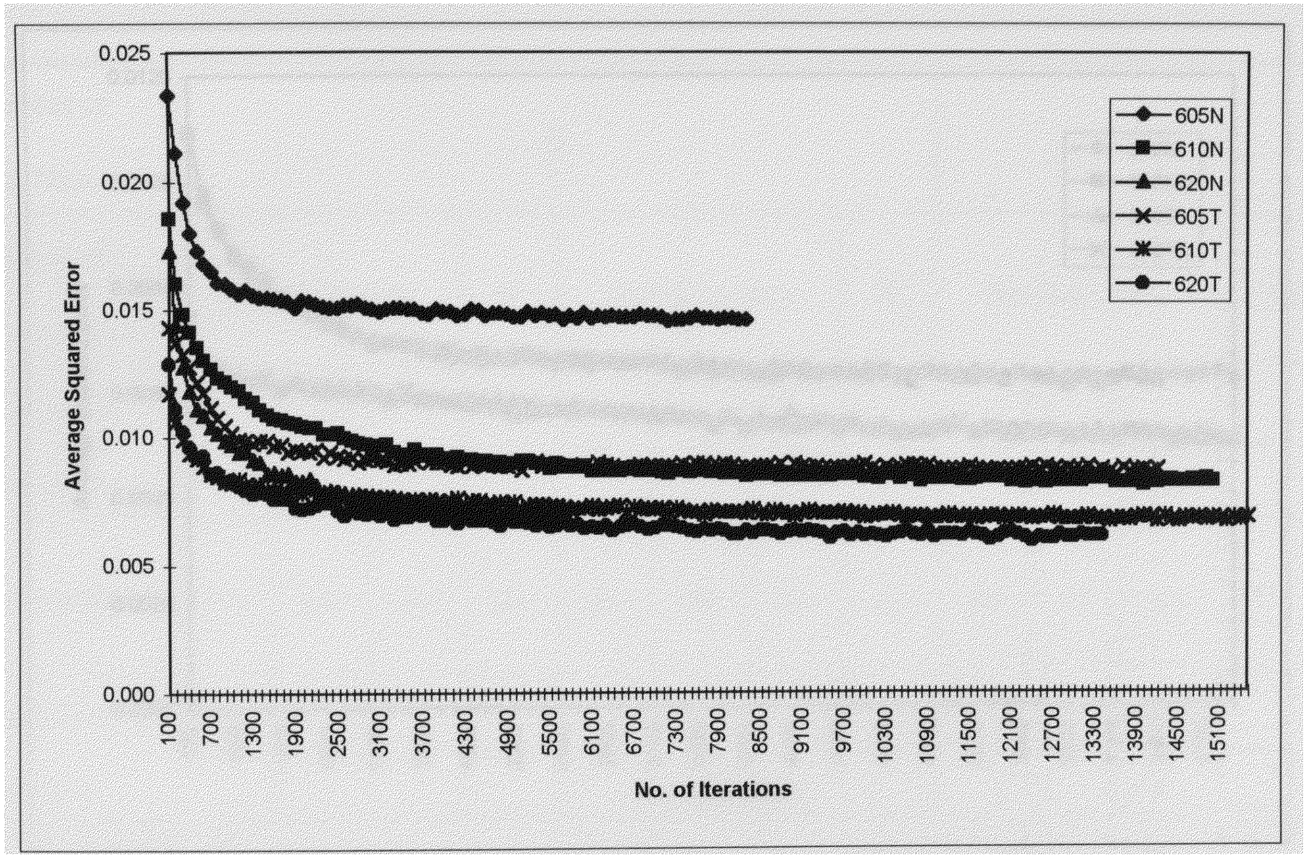


Figure 4.2 Average Squared Error of the Normalization and the Transformation

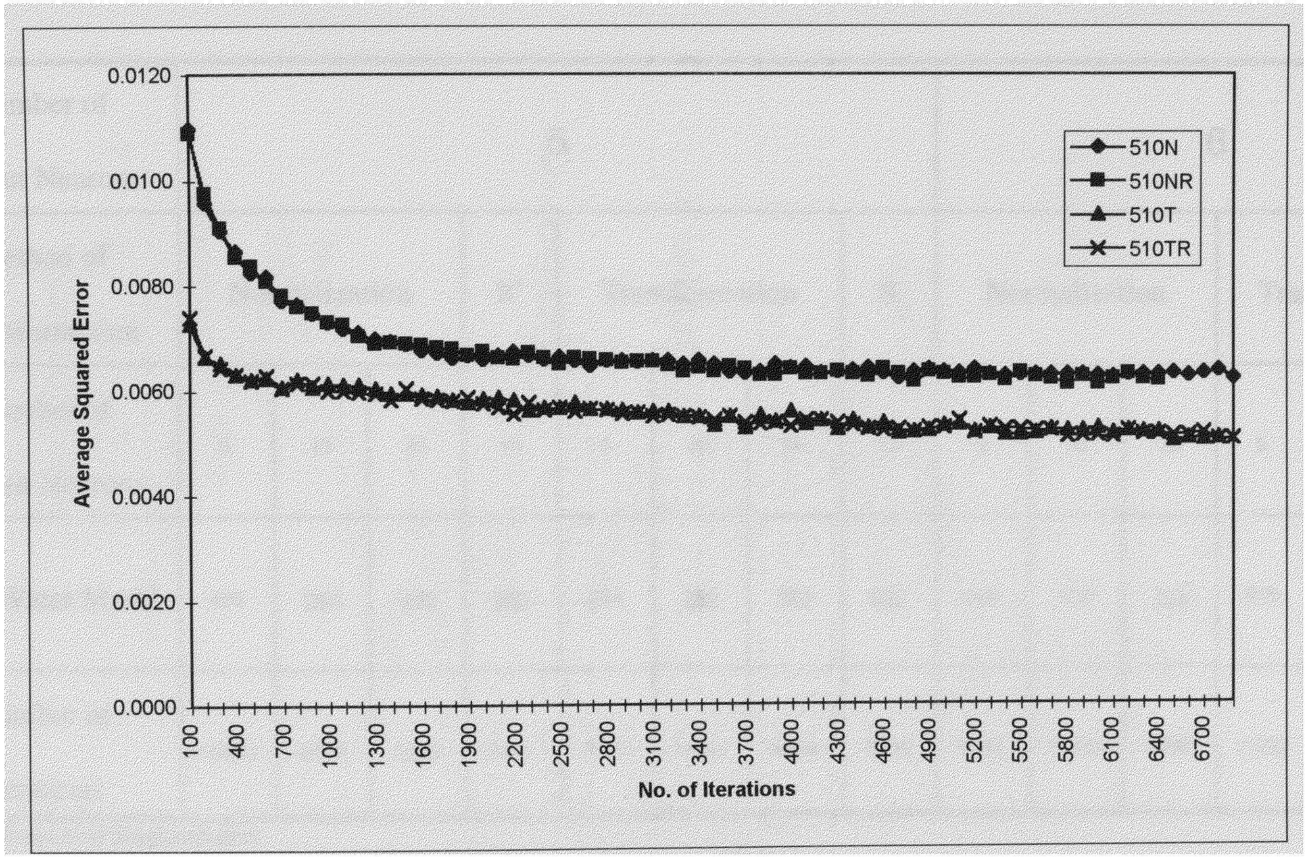


Figure 4.3 Average Squared Error of the Normal Sequence and the Randomized Sequence

Table 4.1 Result of the Training Procedure of ANN

Number of Output Neurons	5								6					
Method of Transformation	Normalization			R[a]	Transformation			R	Normalization			Transformation		
Number of Hidden Neurons	5	10	20	10	5	10	20	10	5	10	20	5	10	20
High Water Mark[b]	239	293	293	293	284	293	293	293	206	233	293	219	262	293
Number of Iterations	15886	6928	3299	6481	16921	6869	5469	6985	8456	15049	5785	14280	15548	13403

Note. a: R stands for Randomization.
b: High Water Mark means the number of census tract satisfying both error conditions.

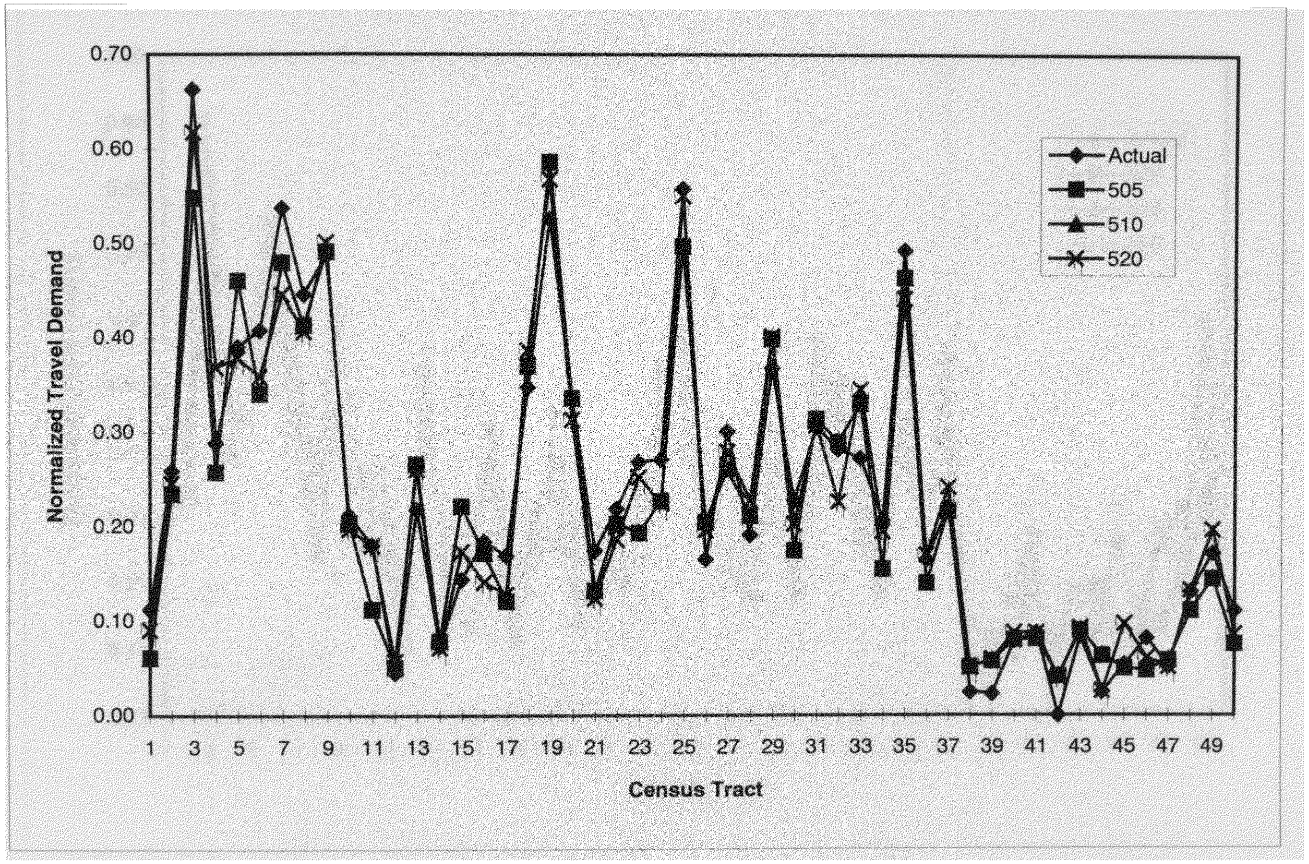


Figure 4.4 Distribution of Normalized Travel Demand of Drive Alone

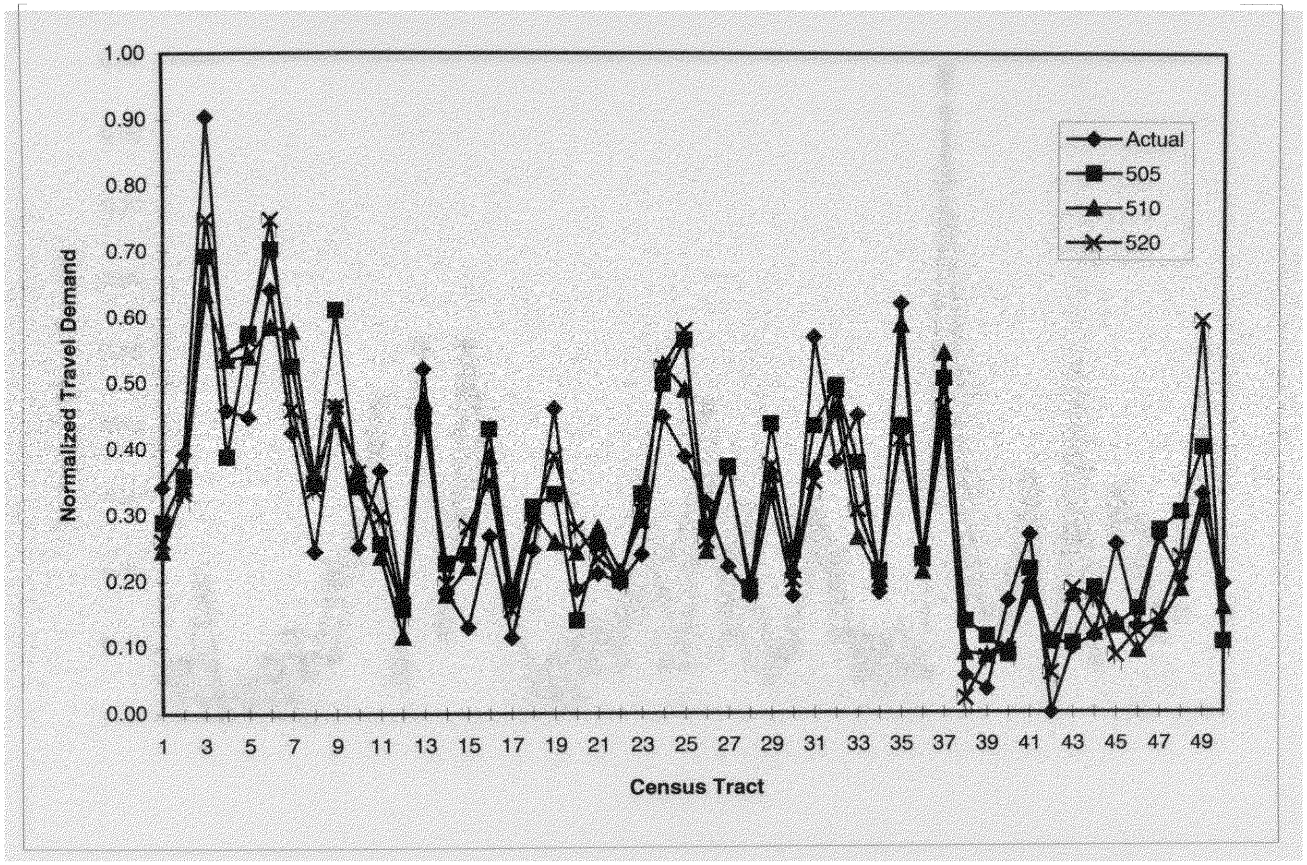


Figure 4.5 Distribution of Normalized Travel Demand of Carpool

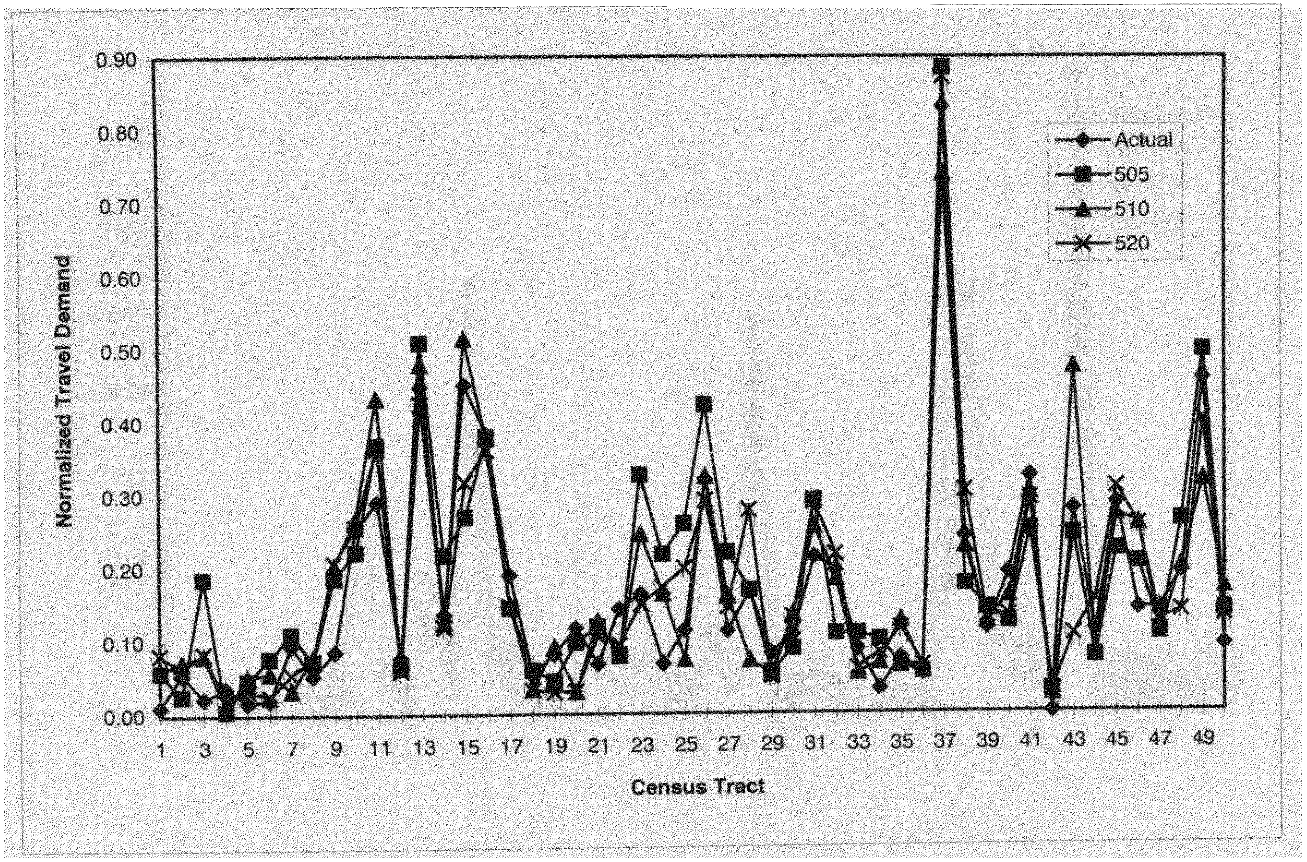


Figure 4.6 Distribution of Normalized Travel Demand of Public Transit

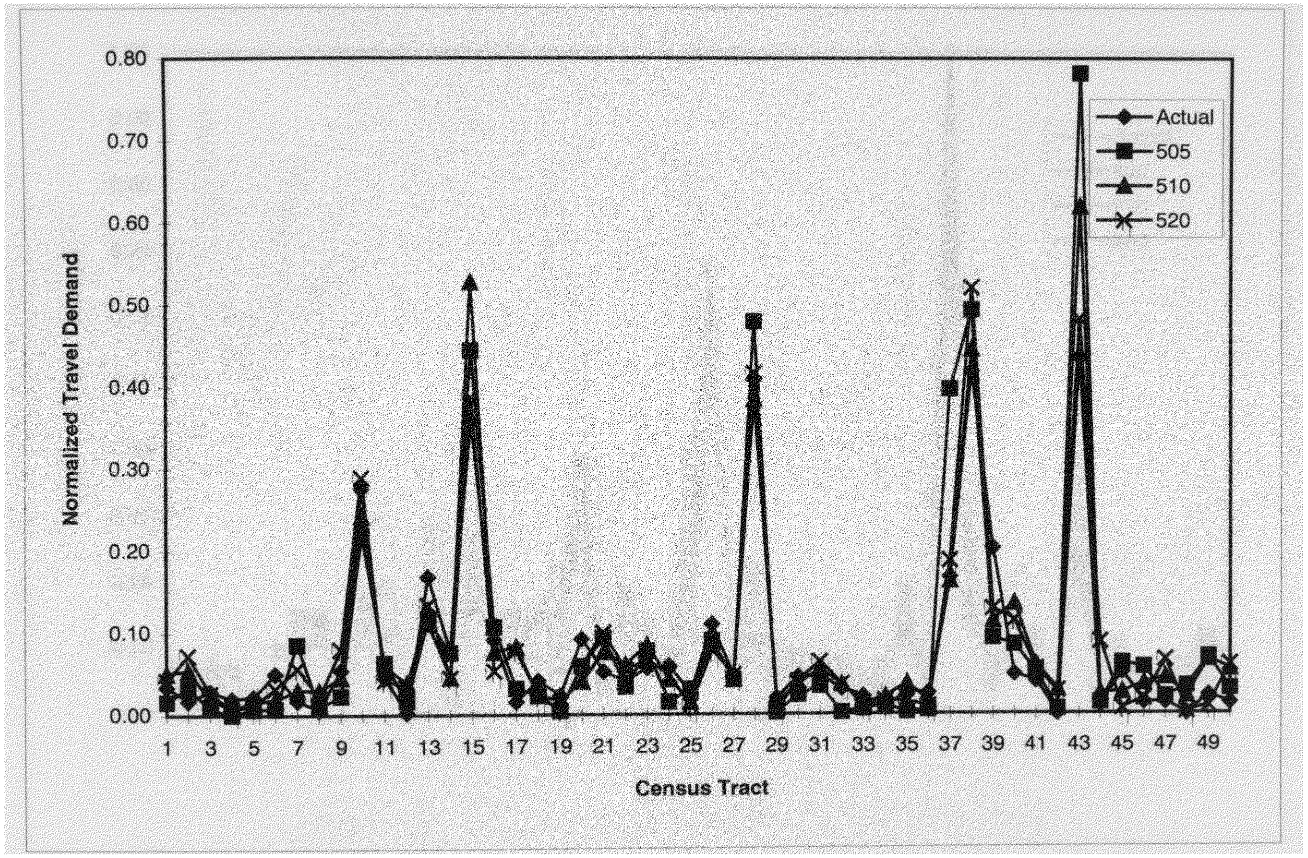


Figure 4.7 Distribution of Normalized Travel Demand of Walk

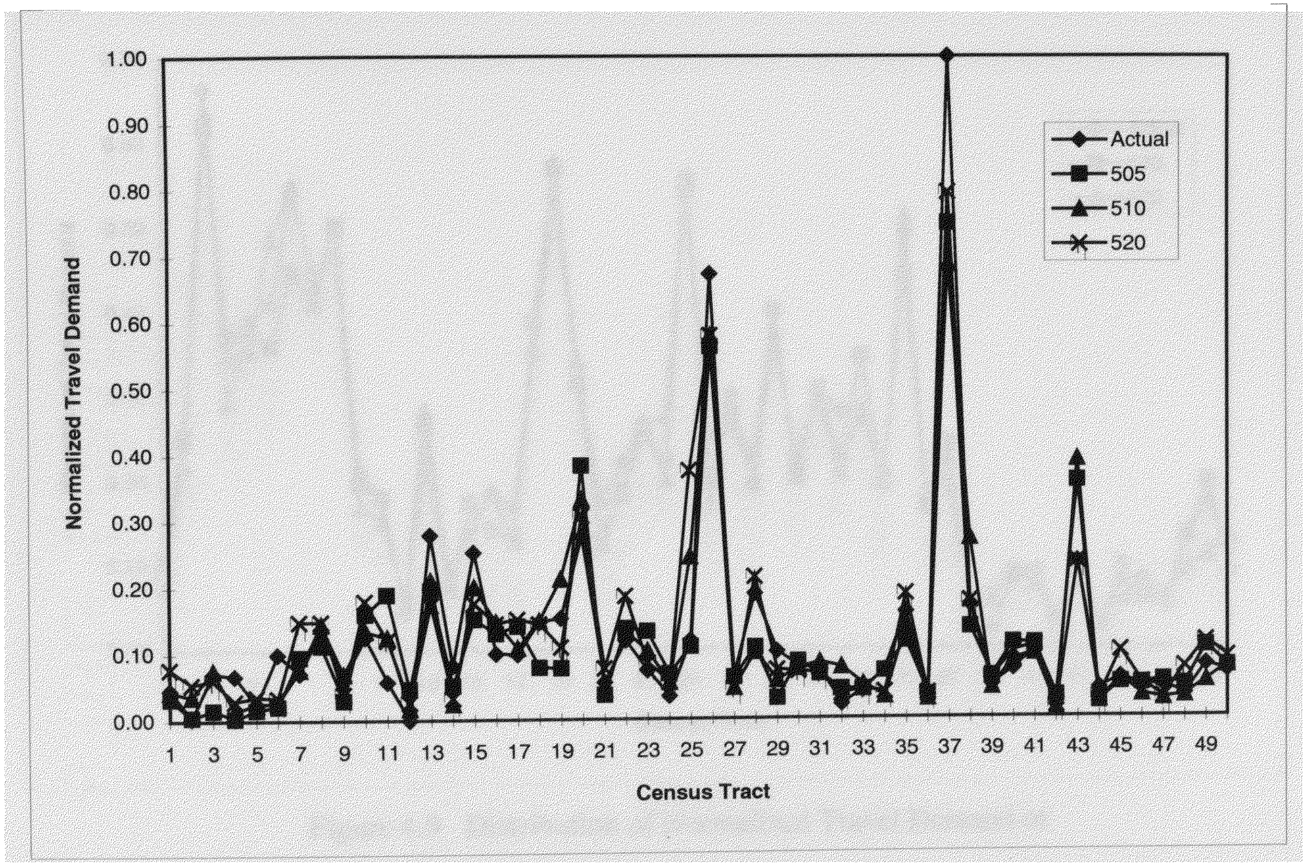


Figure 4.8 Distribution of Normalized Travel Demand of Other

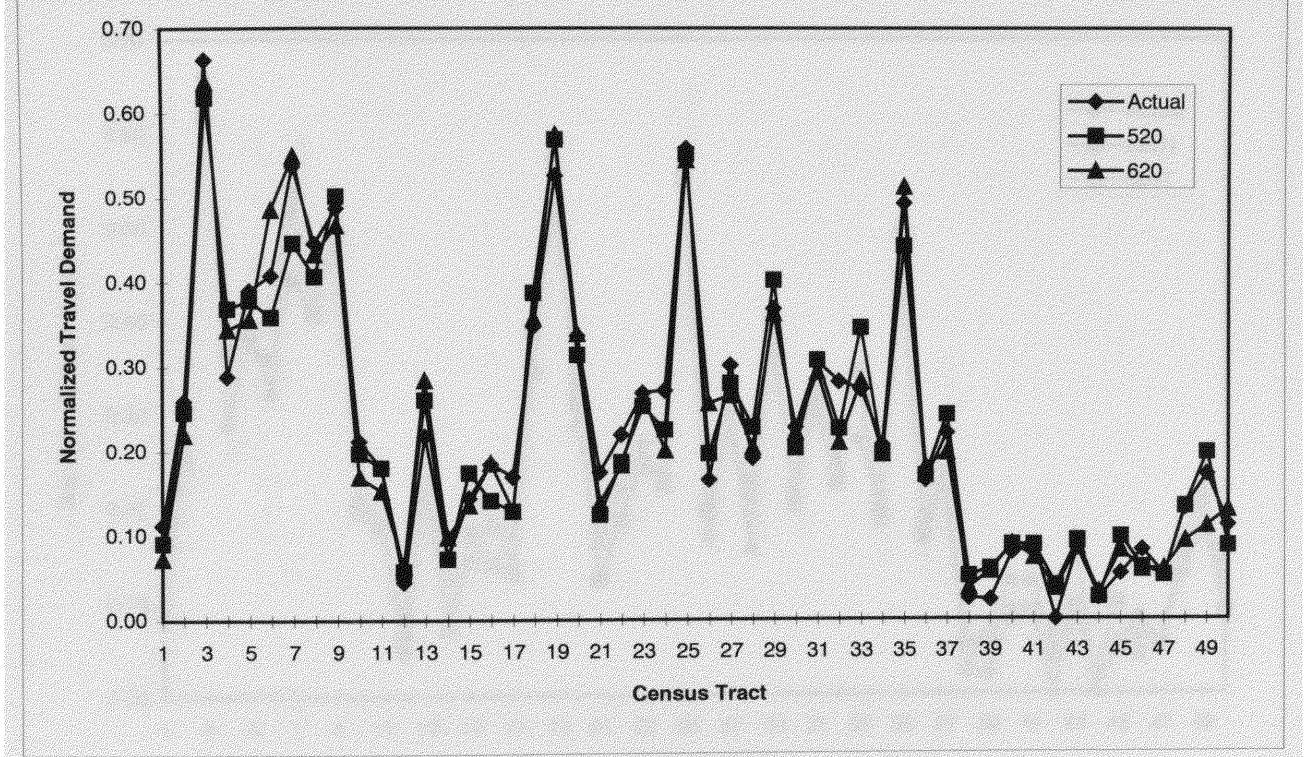


Figure 4.9 Distribution of Normalized Travel Demand of

5 Output Neurons with 20 Hidden Neurons and 6 Output Neurons with 20 Hidden Neurons

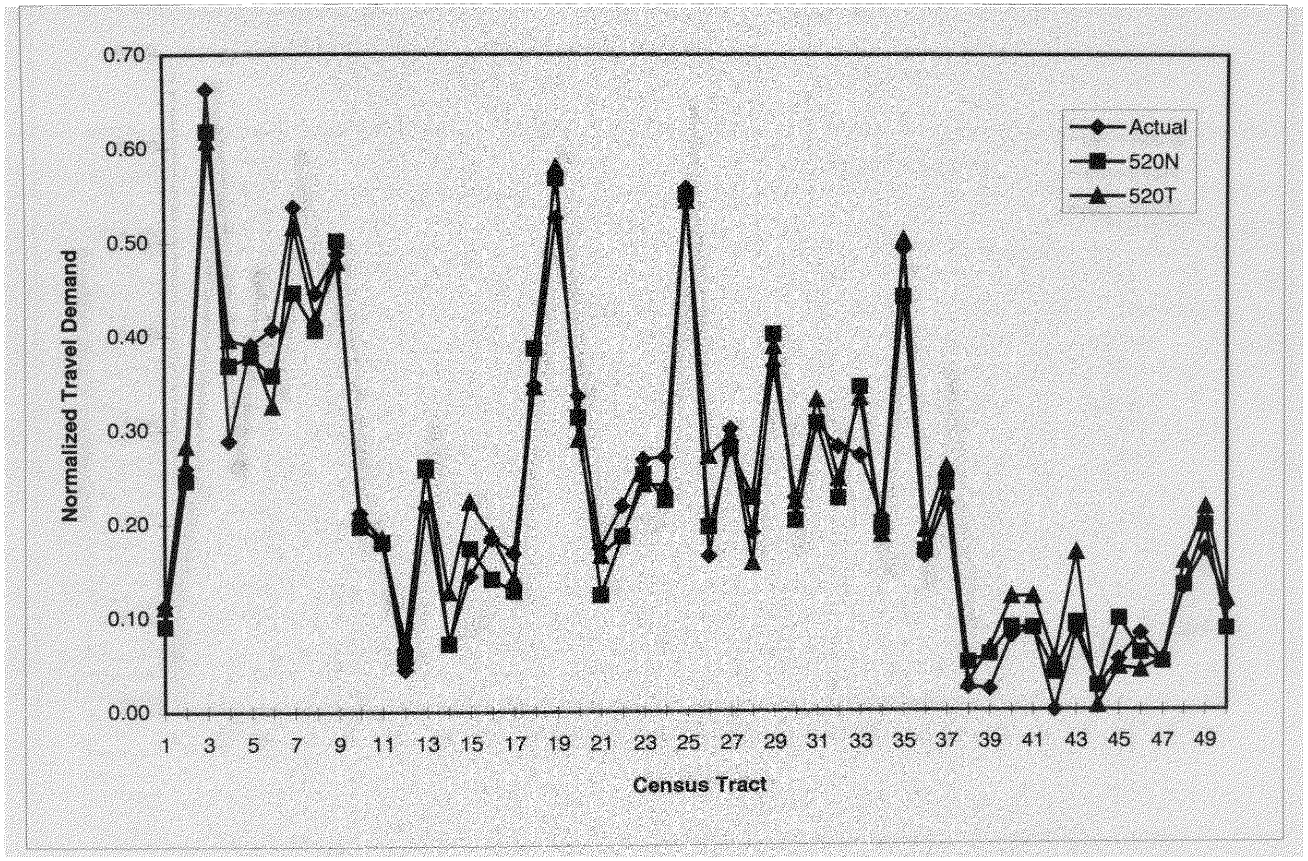


Figure 4.10 Distribution of Normalized Travel Demand of Normalization and Transformation

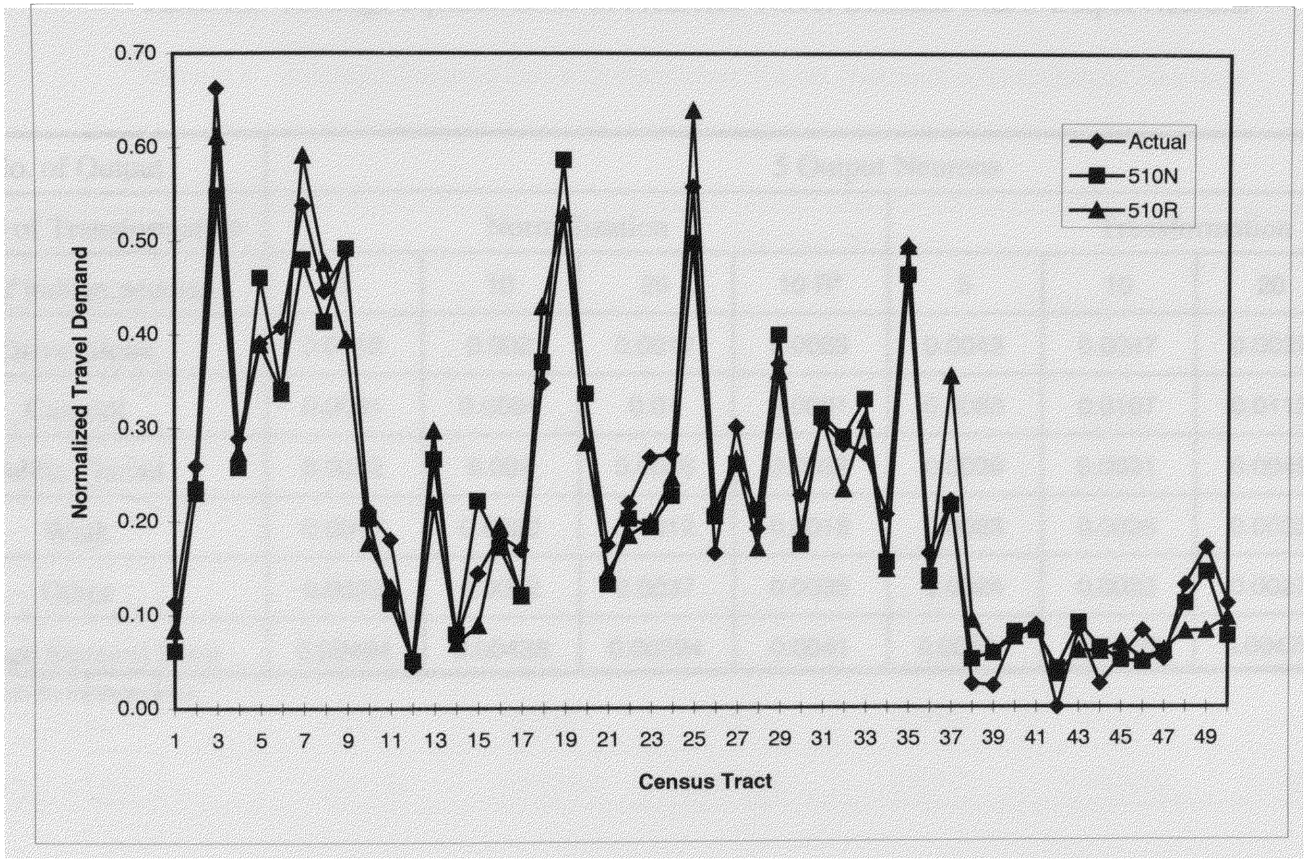


Figure 4.11 Distribution of Normalized Travel Demand of Normal Sequence and Randomized Sequence

Table 4.2 Average Squared Error of Predicted Travel Demand with 5 Output Neurons

No. of Output	5 Output Neurons							
Method of Transformation	Normalization				Transformation			
No. of hidden neurons	5	10	20	10 R[a]	5	10	20	10 R
Drive Alone	0.0018	0.002	0.0012	0.0023	0.0043	0.0047	0.0027	0.0045
Carpool	0.0091	0.0094	0.01	0.0091	0.0083	0.0107	0.0112	0.0113
Public Transit	0.0052	0.004	0.0036	0.0038	0.0039	0.0051	0.0049	0.0051
Walk	0.0047	0.0022	0.0012	0.0018	0.0023	0.0036	0.0023	0.0034
Other	0.0039	0.0042	0.0037	0.0035	0.0024	0.0023	0.0027	0.0021
Average Squared Error	0.00494	0.00436	0.00394	0.0041	0.00424	0.00528	0.00476	0.00528

Note: a. R stands for Randomization.

Table 4.3 Average Squared Error of Predicted Travel Demand with 6 Output Neurons

No. of Output	6 Output Neurons					
Method of Transformation	Normalization			Transformation		
No. of hidden neurons	5	10	20	5	10	20
Drive Alone	0.002	0.0027	0.0012	0.0052	0.0039	0.0045
Carpool	0.011	0.0103	0.0093	0.0117	0.0066	0.0107
Bus	0.0093	0.0233	0.0079	0.0086	0.0086	0.0091
Subway + Rail	0.0039	0.0125	0.0053	0.004	0.0053	0.0068
Walk	0.0034	0.0027	0.003	0.0025	0.0027	0.002
Other	0.0072	0.0023	0.0014	0.0022	0.0031	0.0021
Average Squared Error	0.006133	0.008967	0.004683	0.0057	0.005033	0.005867

CHAPTER V

CONCLUSION

For this study, a Pentium Class computer with 24 MB of RAM and Windows95 were employed. 6000 iterations of the ANN's training took about 24 hours, but testing the ANN was performed in less than 10 seconds. The ANN needs long time for training to find out weights between layers. Once we have weights, however, it doesn't take long time to apply the ANN for forecasting.

Main objective of this study is to show whether the ANN is applicable to travel demand analysis. Overall, it was demonstrated that the ANN is capable of forecasting travel demands in the cases tested with an average of average squared errors of 0.0048 when the ANN was satisfying both stopping criteria and 0.0058 when the ANN was satisfying one of stopping criteria, but still an average squared error is less than error threshold.

Secondary objectives of this paper are to find out the effects of alteration of the number of hidden neurons and output neurons and transformation of input vectors. Three transformation methods were examined: Normalization, Transformation, and Randomization. It was found that the training and testing of the ANN with smaller number of output neurons came up with better results. Two sets of input database were prepared and each of these had 5 output neurons and 6 output neurons. Comparing results of training and testing the ANN of these two input database, average squared errors for 5 output neurons were generally lower than those for 6 output neurons and the number of training iterations for 5 output neurons was less than the number of training iterations for 6 output neurons. In the case of hidden neurons, increasing number of hidden neurons made average squared error converge to a given minimum value, 0.01, faster so that the number of iterations of training was decreased.

Transformation of input vectors affect the ANN's performance by lowering the initial average squared error during the ANN's training, but it did not affect the final result of testing. It seems that randomization did not bring much differences for both training and testing the ANN.

To improve the performance of the ANN adopted in this study, there are possibly several changes can be recommended such as increasing the number of hidden layers, using other activation functions (bipolar sigmoid function), using other transformation functions (sigmoid function), and training the ANN by other algorithm.

LIST OF REFERENCES

Ben-Akiva, Moshe and Steven R. Lerman. *Discrete Choice Analysis.* The MIT Press, Cambridge, M.A. 1985.

Bureau of the Census. "Census of Population and Housing." *Summary Tape File 3 on CD-ROM Technical Documentation.* Washington: The Bureau. 1992

Caliper Corporation. TransCAD Reference Manual Version 2.0. Newton, Massachusetts. 1992.

Caudill, M., and Butler, C. *Understanding Neural Network - computer explorations.* Volumes 1 & 2. The MIT Press, Cambridge, M.A. 1992.

Flood, I. and Kartam, N. *Neural Networks in Civil Engineering*. Part I: Principles and Understanding, Journal of Computing in Civil Engineering, ASCE, Nol. 8, No. 2, pp. 131-148. 1994

Flood, I. and Kartam, N. *Neural Networks in Civil Engineering*. Part II: Systems and Application. Journal of Computing in Civil Engineering, ASCE, Nol. 8, No. 2, pp. 149-162. 1994

GIS/Trans. Ltd. *Census Mapbook for Transportation Planning.* FHWA-PL-94-035. Silver Spring, Maryland. 1994

Hecht-Nielsen, R. *Theory of the Backpropagation Neural Network*, Proc. Int. Joint Conf. on Neural Networks, IEEE, Washington, D.C., Vol. I. pp. 593-605. 1989

Hoffmann, N. *Simulation neuronaler Netze: Grundlagen, Modelle, Programme in Turbo Pascal Vieweg Verlagsgesellschaft*, Braunschweig, Wiesbaden 1992

Hua, J. and Faghri, A. *Applications of ANNs to IVHS*, Transportation Research Board, 73rd Annual Meeting, Paper No. 940622, January 9-13, Washington, D.C. 1994

JHK & Associates. *CTPP HANDBOOK: An Instructional Guide to the 1990 Census Transportation Planning Package.* FHWA-PD-95-019. Washington, D.C. 1995

Kinnebrock, W. *Neuronale Netze*, Grundlagen, Anwendungen, Beispiele Oldenbourg Verlag; München, Wien. 1992

Kothari, Ravi, Powsiri Klinkhachorn, and Roy S. Nutter. *Back Propagation Training Algorithm.* Proceedings of the IEEE. 1993

Lapedes, A., and Farber, R. *How Neural Networks Work*, Neural Information Processing Systems, American Institute of Physics, pp. 442-456. 1988

Lippman, R. *An Introduction to computing with Neural Nets*. IEEE ASSP Magazine, pp 4-21. April 1987

Sharda, R. *Neural Networks for the OR/MS Analyst: An Application Bibliography*, Working Paper 92-3, College of Business Administration, Oklahoma State University, July 1992

Shen, L.D., Y. Xiong, and Y-K. Lee. *Evaluation of the Impacts of National Demographic and Technological Trends on Future Transit Efficiency.* Miami, Florida. 1994

Shen, L.D., and Zhao, F. Guideway Transit and Intermodalism: Function and Effectiveness, Case Study, Boston. 1995

The Urban Transportation Monitor. "FTA Developing a National Transit Geographic Information System." *The Urban Transportation Monitor vol8,no5*, pp 1-2. 1994

U.S. Department of Commerce and Bureau of the Census. "TIGER/Line Files, 1992 Technical Documentation." Washington D.C. 1993

Wasserman, P. *Neural Computing - Theory and Practice*. Van Nostrand Reinhold, New York. 1989

APPENDICES

APPENDIX A

List of Input Variables

· White Male 16-29
· White Male 30-60
· White Male 65-Over
· White Female 16-29
· White Female 30-64
· White Female 65-Over
· Black Male 16-29
· Black Male 30-64
· Black Male 65-Over
· Black Female 16-29
· Black Female 30-64
· Black Female 65-Over
· Other Male 16-29
· Other Male 30-64
· Other Male 65-Over
· Other Female 16-29
· Other Female 30-64
· Other Female 65-Over
· Worked in MSA/PMSA of residence Central city
· Worked in MSA/PMSA other
· Travel Time Less Then 14
· 14 to 29 minutes
· 30 to 44 minutes
· 45 or more minutes
· Worked at home
· Distance to nearest Station
· Population Density (inh/sqmi)
· Per Capita Income 1989
· Per Capita Income White 1989
· Per Capita Income Black 1989

APPENDIX B

Program Code for ANN

```
////////////////////////////////////////////////////////
// Backpropagation training algorithm
////////////////////////////////////////////////////////

#ifndef FFBACKPR_C
#define FFBACKPR_C

#include <conio.h>
#include <malloc.h>
//1// #include <mem.h>

#include "ut.h"
#include "uttypes.h"
#include "ff.h"

void BackpropagationTraining(FFSetupRec setup, WeightsRec weights,
                                        TrainingRec TrRec, GetDataProc
GetNetworkInputOutput);

void Backpropagation(NetworkRec net, FVector DesiredOut, WeightsRec wts,
                                WeightsRec momentum, FFSetupRec setup){
        // one hidden layer
        int i,j,k;
        float sum = (float)0.0, wtChange = (float)0.0;
        double term = (double)0.0;
        DVector iDelta = NULL;
        FVector oDelta = NULL;
        iDelta = AllocateDVector(1,setup.NOfHlNodes);
        oDelta = AllocateFVector(1,setup.NOfOutputs);
        for(j=1;j<=setup.NOfOutputs;j++)
                oDelta[j]=DesiredOut[j]-net.yOut[j];
        for(j=1;j<=setup.NOfHlNodes;j++){
                sum = (float)0.0;
                for(k=1;k<=setup.NOfOutputs;k++){
                        term=oDelta[k]*wts.wo[j][k];
                        if(setup.SigmoidOut)
                                term *=net.yOut[k]*(1.0 - net.yOut[k]);
                        sum += (float)term;
                        }
                iDelta[j] = sum;
                }
        for(i=1;i<=(setup.NOfInputs + 1);i++)
                for(j=1;j<=setup.NOfHlNodes;j++){
                wtChange =
(float)iDelta[j]*(float)net.xInp[i]*(float)net.hl[j]*((float)1.0-(float)net.hl[j]);
                if(!setup.SigmoidOut)
                        wtChange *= (float)0.1;
                wts.wi[i][j] +=setup.LearningRate*((1.0 - setup.alpha)*wtChange +
                                setup.alpha * momentum.wi[i][j]);
                momentum.wi[i][j]=wtChange;
                }
        for(i=1;i<=setup.NOfHlNodes;i++){
                for(j=1;j<=setup.NOfOutputs;j++){
                        wtChange = oDelta[j]*(float)net.hl[i];
                        if(setup.SigmoidOut)
                                wtChange*=net.yOut[j]*((float)1.0-net.yOut[j]);
                        else
                                wtChange *= (float)0.1;
                        wts.wo[i][j]+=setup.LearningRate*
                                        ((1.0 - setup.alpha) * wtChange+
                                                setup.alpha*momentum.wo[i][j]);
                        momentum.wo[i][j]=wtChange;
                        }
                }
        FreeFVector(oDelta,1);
        FreeDVector(iDelta,1);
        return;
        }

void BackpropagationTraining(FFSetupRec setup, WeightsRec weights,
                                                TrainingRec TrRec, GetDataProc
GetNetworkInputOutput){
        int i,k;
        float err,sumerr;
        int *ShuffleList;
        WeightsRec momentum = {0,NULL,NULL};
        NetworkRec network = {(short)0,NULL,NULL};
        FVector dOut = NULL;
        DisplayRec display = {0,0,0,0,0,0};
        FILE            *outfile;
```

```
#define OUT_FORMAT      "%9.4E\n"

AllocateNetwork(setup,&network);
momentum = InitializeWeights(setup,0);
sumerr = (float)0;
dOut = AllocateFVector(1,setup.NOfOutputs);
ShuffleList=(int *)malloc((size_t)(TrRec.NOfTrainingItems+1)*sizeof(int));
display.iter = setup.StartIter;
display.NbrTrItems = TrRec.NOfTrainingItems;
SetupScreen (setup,display);
if(!AppOutputTextFile(&outfile, "errors.txt"))
        exit(0);
while ((display.RightCount < TrRec.NOfTrainingItems) && !kbhit()){
        display.RightCount = 0;
        display.WrongCount = 0;
        display.iter++;
        if ((display.iter % 100) == 0){
                if(!WriteWeights(weights, setup.files.OutWtsFileName, setup))
                  goto ExitProc;
                fprintf(outfile,OUT_FORMAT,sumerr/TrRec.NOfTrainingItems);
                }
        ResetScreen (display);
        gotoxy(1,25);
        sumerr = (float)0;
        if(setup.Shuffle)
                shuffl(TrRec.NOfTrainingItems,ShuffleList);
        for(i=1;i<=TrRec.NOfTrainingItems;i++){
                if(setup.Shuffle)
                        k=ShuffleList[i - 1]+1;
                else
                        k=i;
                GetNetworkInputOutput(setup,TrRec,k,network.xInp,dOut);
                Feedforward(network,weights,setup);
                err = FFError(dOut,network.yOut,setup);
                sumerr += err;
                if(err>setup.ErrorThreshold){
                        display.WrongCount++;
                        Backpropagation(network,dOut,weights,momentum,setup);
                        }
                else
                        display.RightCount++;
                DisplayResults(setup,display,dOut,network.yOut);
                }
```

```
				if(display.RightCount > display.HighWater)
					display.HighWater = display.RightCount;
				display.LastRight = display.RightCount;
				display.LastWrong = display.WrongCount;
				}
		if(!kbhit())
				cprintf("%c",BEEP);
		else
				getch();
		clrscr();
		gotoxy(1,4);
		cprintf("Total number of iterations used: %ld",display.iter);
		gotoxy(1,6);
		cprintf("High water mark: %3d out of %3d.", display.HighWater,
TrRec.NOfTrainingItems);
		gotoxy(1,8);
		if(!WriteWeights(weights,setup.files.OutWtsFileName,setup))
				goto ExitProc;
		gotoxy(1,15);
		cprintf("Weights saved to file: %s", setup.files.OutWtsFileName);
		gotoxy(1,15);
		cputs("Press key to clear: ");
		getch();
ExitProc:
		fclose(outfile);
		clrscr();
		free (ShuffleList);
		FreeFVector(dOut,1);
		FreeNetwork(&network);
		FreeWeights(&momentum);
		return;
		}

#endif
```

```
/////////////////////////////////////////////////////
// Feedforward algorithm implementation
/////////////////////////////////////////////////////

#ifndef FFEEDFW_C
#define FFEEDFW_C

#include <conio.h>
#include <math.h>
#include <malloc.h>

#include "ut.h"
#include "ff.h"

void AllocateNetwork(FFSetupRec setup, NetworkRec *network);
void FreeNetwork (NetworkRec *network);
float FFError (FVector desired, FVector NetOut, FFSetupRec setup);
void Feedforward (NetworkRec TheNetwork, WeightsRec TheWeights,
                                FFSetupRec TheSetupRec);

void AllocateNetwork(FFSetupRec setup, NetworkRec *network){
        network->xInp = AllocateFVector(1,setup.NOfInputs + 1);
        network->yOut = AllocateFVector(1,setup.NOfOutputs + 1);
        network->hl= AllocateDVector(1,setup.NOfHlNodes);
        network->allocated = TRUE;
        return;
        }

void FreeNetwork(NetworkRec *network){
        if(network->allocated){
                FreeFVector(network->xInp,1);
                FreeFVector(network->yOut,1);
                FreeDVector(network->hl,1);
                }
        network->allocated = FALSE;
        return;
        }

double sigmoid(double x){
        if(x>50.0) return 1.0;
        if(x<-50.0) return 0.0;
        return 1.0/(1.0 + exp(-x));
        }
```

```
float FFError(FVector desired, FVector NetOut, FFSetupRec setup){
        int i;
        float sum = (float)0.0;
        for (i=1;i<=setup.NOfOutputs;i++)
                sum += (desired[i] - NetOut[i])*(desired[i] - NetOut[i]);
        return (float)0.5*sum;
        }

void Feedforward(NetworkRec net, WeightsRec weights,
                                FFSetupRec setup){
        int i,j;
        double wx = 0.0, whl=0.0;
        net.xInp[setup.NOfInputs + 1] = (float)1.0;
        for(i=1;i<=setup.NOfHlNodes;i++){
                wx=0.0;
                for(j=1;j<=(setup.NOfInputs+1);j++)
                        wx += weights.wi[j][i]*net.xInp[j];
                net.hl[i]=sigmoid(wx);
                }
        for(i=1;i<=setup.NOfOutputs;i++){
                whl = 0.0;
                for(j=1;j<=setup.NOfHlNodes;j++)
                        whl+=weights.wo[j][i]*net.hl[j];
                if(setup.SigmoidOut)
                        net.yOut[i] = (float)sigmoid(whl);
                else
                        net.yOut[i] = (float)whl;
                }
        return;
        }

#endif
```

APPENDIX C

Program Code for Distance

```
Sub Extract ()
Dim SQLQuery As String
Dim OutLine As String
Dim ssSTF As Snapshot
Dim ssFTA As Snapshot
Dim dFTALon, dFTALat As Double
Dim dSTFLon, dSTFLat As Double
Dim ID As Integer


   SQLQuery = "SELECT STATEFP, CNTY, TRACTBNA, BLCKGR, INTPTLAT,
INTPTLNG "
   SQLQuery = SQLQuery & "FROM " & tblSTFInput
   SQLQuery = SQLQuery & " WHERE (SUMLEV = '140')"
   MsgBox (SQLQuery)
   Set ssSTF = dbSTFInput.CreateSnapshot(SQLQuery)
   ssSTF.MoveFirst

   SQLQuery = "SELECT LATITUDE, LONGITUDE, TYPE, ID "
   SQLQuery = SQLQuery & "FROM " & tblFTAInput
   SQLQuery = SQLQuery & " WHERE (TYPE='STA')"
   MsgBox (SQLQuery)
   Set ssFTA = dbFTAInput.CreateSnapshot(SQLQuery)
   ssFTA.MoveFirst

   Open szOutFile For Output As 1 Len = 1024

Do Until (ssSTF.EOF)
   dSTFLat = ssSTF("INTPTLAT") * .000001 * 3.14159 / 180
   dSTFLon = ssSTF("INTPTLNG") * .000001 * 3.14159 / 180


   ssFTA.MoveFirst
   dFinal = 10000000
   Do Until (ssFTA.EOF)
      dFTALat = ssFTA("LATITUDE") * .000001 * 3.14159 / 180
      dFTALon = ssFTA("LONGITUDE") * .000001 * 3.14159 / 180
```

```
            If frmMain.optEuclidean.Value = True Then
              distance = EuclDist(dSTFLat, dSTFLon, dFTALat, dFTALon)
            Else
              distance = SpherDist(dSTFLat, dSTFLon, dFTALat, dFTALon)
            End If
            If distance < dFinal Then
              dFinal = distance
              ID = ssFTA("ID")
            End If
            DoEvents
            ssFTA.MoveNext
        Loop

        OutLine = ssSTF("STATEFP") & ssSTF("CNTY") & ssSTF("TRACTBNA")
        OutLine = OutLine & "," & dFinal & "," & ID
        Print #1, OutLine
        ssSTF.MoveNext
    Loop

        MsgBox ("Process Finished")

    End Sub

    Function Arccos (angle As Double) As Double
     Arccos = Atn(-angle / Sqr(-angle * angle + 1)) + 1.5708
    End Function



    Sub dbOpen (ByVal szRawFile As String)

    Dim szPath As String
    Dim szfile As String
    Dim i, iPos As Integer

        i = Len(szRawFile) - 3
        While Not (InStr(i, szRawFile, "\") <> 0)
          i = i - 1
        Wend
        szPath = Left(szRawFile, i)
        szfile = Mid(szRawFile, i + 1, Len(szRawFile) - i - 4)
        Set dbSTFInput = OpenDatabase(szPath, False, True, "dBASE III")
        Set tblSTFInput = dbSTFInput.OpenTable(szfile)
```

```
End Sub



Function EuclDist (ByVal pOneLat As Double, ByVal pOneLon As Double, ByVal
pTwoLat As Double, ByVal pTwoLon As Double) As Double

   EuclDist = 3440.4 * Sqr((pOneLon - pTwoLon) ^ 2 + (pOneLat - pTwoLat) ^ 2)

End Function



Sub FTAOpen (ByVal szRawFile As String)
Dim szPath As String
Dim szfile As String
Dim i, iPos As Integer

   i = Len(szRawFile) - 3
   While Not (InStr(i, szRawFile, "\") <> 0)
      i = i - 1
   Wend
   szPath = Left(szRawFile, i)
   szfile = Mid(szRawFile, i + 1, Len(szRawFile) - i - 4)
   Set dbFTAInput = OpenDatabase(szPath, False, True, "dBASE III")
   Set tblFTAInput = dbFTAInput.OpenTable(szfile)

End Sub



Function SpherDist (ByVal pOneLat As Double, ByVal pOneLon As Double, ByVal
pTwoLat As Double, ByVal pTwoLon As Double) As Double

   SpherDist = 3440.4 * Arccos(Sin(pOneLat) * Sin(pTwoLat) + Cos(pOneLat) *
Cos(pTwoLat) * Cos(pOneLon - pTwoLon))

End Function
```

APPENDIX D

Sample File for Training and Testing

```
4     2              No. of input   No. of outputs
minimum
0.000000             #Input minimum for inpout 1
0.000000             #Input minimum for inpout 2
0.000000             #Input minimum for inpout 3
0.000000             #Input minimum for inpout 4
-
0.000000             #Input minimum for output 1
0.000000             #Input minimum for output 2
maximum
1000.000             #Input maximum for input 1
1.000000             #Input maximum for input 2
0.300000             #Input maximum for input 3
0.400000             #Input maximum for input 4
-
100.0000             #Input maximum for output 1
56.00000             #Input maximum for output 2
training data
232.0000             #Input Data 1 for Training Set 1
2.000000             #Input Data 2 for Training Set 1
32323.00             #Input Data 3 for Training Set 1
2323.000             #Input Data 4 for Training Set 1
-
2.000000             #Output Data 1 for Training Set 1
3.000000             #Output Data 2 for Training Set 1
23.00000             #Input Data 1 for Training Set 2
2.000000             #Input Data 2 for Training Set 2
```

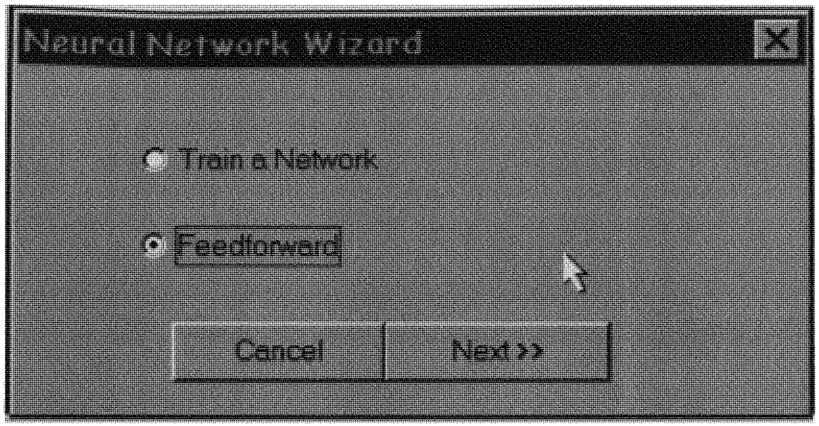
APPENDIX E

Excel Macro for Training and Testing Files

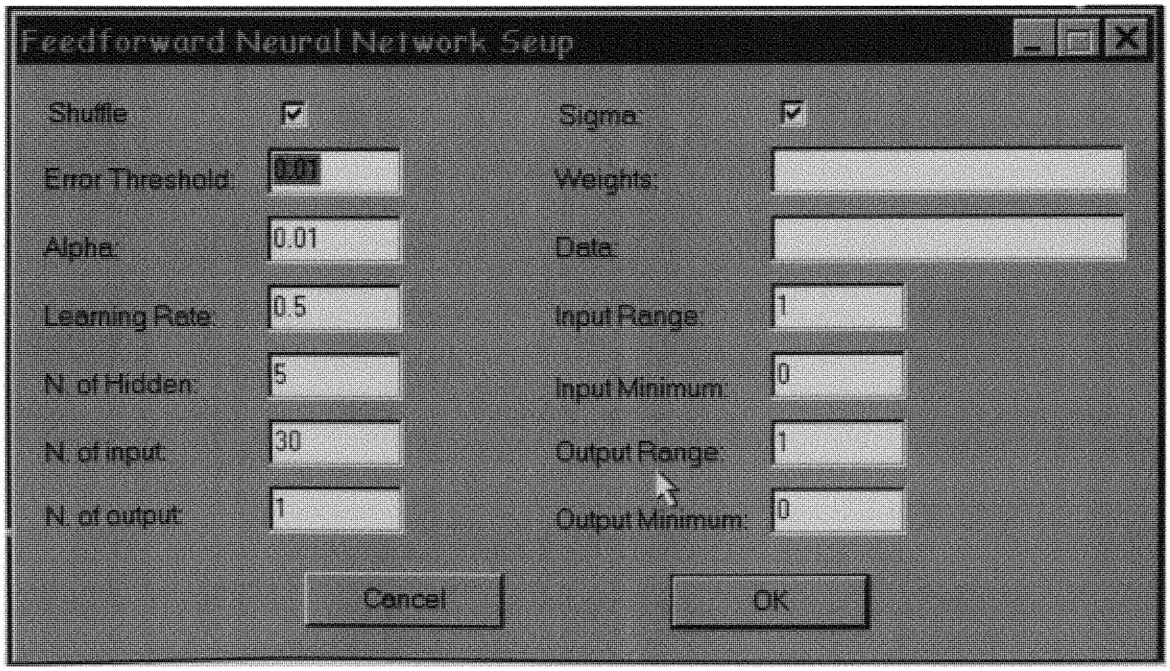
```
Sub NData4Training()
   Dim I As Integer
      Sheets("Norm").Select
      For I = 1 To 294
         Range(Cells(I * 2, 2), Cells(I * 2, 36)).Select
         Selection.Copy
         Sheets("Sheet6").Select
         Cells(I * 35 - 34, 1).Select
         Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone, _
            SkipBlanks:=False, Transpose:=True
         Sheets("Norm").Select
         Application.CutCopyMode = False
      Next I
End Sub

Sub NData4Testing()
   Dim I As Integer
      Sheets("Norm").Select
      For I = 1 To 50
         Range(Cells(I * 11, 2), Cells(I * 11, 36)).Select
         Selection.Copy
         Sheets("Sheet7").Select
         Cells(I * 35 - 34, 1).Select
         Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone, _
            SkipBlanks:=False, Transpose:=True
         Sheets("Norm").Select
         Application.CutCopyMode = False
      Next I
End Sub
```

APPENDIX F

Graphical Interface



Train a Network [✓ Feedforward]



Shuffle	✓	Sigma:	✓
Error Threshold:	0.01	Weights:	
Alpha:	0.01	Data:	
Learning Rate:	0.5	Input Range:	1
N. of Hidden:	5	Input Minimum:	0
N. of input:	30	Output Range:	1
N. of output:	1	Output Minimum:	0