

To the Graduate Council:

I am submitting herewith a thesis written by Bryan Keith Ary entitled "Data Mining over Mismatched Domains." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



Bruce Whitehead, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor
and Dean of the Graduate School

DATA MINING OVER MISMATCHED DOMAINS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Bryan Keith Ary

December 1997

Copyright © Bryan Keith Ary, 1997

All rights reserved

Dedication

This thesis is dedicated to my parents, who have provided me with endless opportunities to succeed in life.

Acknowledgments

Many thanks are due to my advisor, Dr. Bruce Whitehead, for the suggestions and tireless assistance he provided. I would also like to thank the other members of my thesis committee, Dr. Dinesh Mehta and Dr. Kenneth Kimble.

Abstract

Data mining is the discovery of non-obvious knowledge about a large set of related data. Pattern matching is one data mining technique often used to find non-obvious relationships and associations between data items. The problem addressed by this research is the reconciliation of two data sets of different origin that contain information about the same real-world entities. This research compares the effectiveness of the back-propagation neural network model and the least-squares multiple linear regression model by using each method to recognize when a record in one data set describes the same real-world entity as a record in the other data set. Results of this research indicate that back-propagation can easily over-fit mismatched data but does outperform least-squares approximations when the number of hidden layer neurons is carefully chosen.

TABLE OF CONTENTS

Chapter	Page
CHAPTER I: INTRODUCTION.....	1
CHAPTER II: LITERATURE REVIEW.....	6
A. Knowledge Discovery in Databases.....	6
A.1 Decision Support Systems.....	11
A.2 Data Warehouses.....	14
B. Artificial Neural Networks.....	20
B.1 Mathematical Basis.....	22
B.2 Backpropagation.....	28
B.3 Interpreting Results.....	37
C. Regression Analysis.....	40
C.1 Mathematical Basis.....	42
C.2 Method of Least-Squares.....	45
C.3 Interpreting Results.....	51
CHAPTER III: METHODS.....	53
A. Data Preparation.....	53
A.1 Enrichment.....	58
A.2 Representations.....	60
A.3 Quantity and Quality.....	68
B. Backpropagation Training.....	71
C. Least-Squares Execution.....	75
CHAPTER IV: RESULTS AND CONCLUSIONS.....	77
A. Results.....	77
B. Conclusions.....	80
LIST OF REFERENCES.....	83
VITA.....	89

LIST OF FIGURES

Figure	Page
1. DIAGRAM OF AN M-P NEURON.....	23
2. PERCEPTRON WITH 3 INPUTS AND 2 OUTPUTS	24
3. LINEARLY SEPARABLE DATA.....	25
4. SIGMOIDAL THRESHOLD FUNCTION	27
5. BACKPROPAGATION NEURAL NETWORK WITH 5 INPUTS, 3 HIDDEN NEURONS, AND 2 OUTPUTS	29
6. GRADIENT DESCENT ON A PARABOLIC ERROR SURFACE	31
7. PLOT OF STROKE RELATED DEATHS AGAINST BLOOD PRESSURE	43
8. DEATHS DUE TO STROKE VERSUS BLOOD PRESSURE FOR DATA IN TABLE 3	48
9. STEPS IN A PURCHASE TRANSACTION BETWEEN THE DMS AND GL DATABASES.....	56
10. TRAINING POINTS FOR INPUT DIMENSIONS (x_6, x_7, x_8)	73
11. TRAINING THE 8x8x1 BP NEURAL NETWORK WITH INPUT DIMENSIONS (x_6, x_7, x_8)	74
12. RMSE FOR BP NEURAL NETWORKS DURING TRAINING.....	79
13. PERCENT CORRECT FOR EACH BP NETWORK AND LEAST-SQUARES	80

LIST OF TABLES

Table	Page
1. DATA THAT NEEDS TO BE CLEANED	7
2. DATA AFTER CLEANING IS PERFORMED	9
3. HYPOTHETICAL DATA RELATING THE DEATHS DUE TO STROKE AND BLOOD PRESSURE.....	47
4. VALUES USED TO FIND THE LINEAR LEAST-SQUARES APPROXIMATION FOR DATA IN TABLE 3	49
5. DATA FROM THE UNIVERSITY OF TENNESSEE AT KNOXVILLE GL DATABASE.....	54
6. DATA FROM THE UNIVERSITY OF TENNESSEE SPACE INSTITUTE DMS DATABASE.....	54
7. CODES USED FOR THE TRANSACTION TYPES IN THE ENRICHED DATA SETS.....	59
8. ENRICHED DATA FROM TABLE 5	60
9. ENRICHED DATA FROM TABLE 6	60
10. FIELD COMPARISON INPUT DATA FROM TABLES 8 AND 9	67
11. RESULTS FOR FIRST ATTEMPT AT FINDING DMS MATCHES	78
12. RESULTS FOR THE SECOND ATTEMPT AT FINDING DMS MATCHES	78

LIST OF EQUATIONS

Equation	Page
1. WEIGHTED SUM ADJUSTED FOR THRESHOLD	22
2. THE STEP ACTIVATION FUNCTION	22
3. THE SIMPLE LINEAR REGRESSION MODEL	42
4. LINEAR REGRESSION MODEL WITH TWO INPUTS	44
5. COLUMN VECTOR OF OUTPUTS FOR A LINEAR MODEL	44
6. COLUMN VECTOR OF UNKNOWN PARAMETERS FOR A LINEAR MODEL	45
7. COEFFICIENT MATRIX OF MULTIPLE INPUTS FOR MULTIPLE LINEAR REGRESSION	45
8. SQUARED ERROR FROM A SINGLE POINT i TO THE BEST FITTING LINE	46
9. SUM OF THE SQUARED ERROR FOR EACH POINT i	46
10. LEAST-SQUARES ERROR FOR THE DATA IN TABLE 3	47
11. "BEST" FITTING LINEAR SYSTEM OF EQUATIONS FOR THE DATA IN TABLE 3	47
12. EQUATION OF THE LINE THAT "BEST" FITS THE DATA FROM TABLE 3	48
13. NORMAL EQUATIONS FOR SIMPLE LINEAR REGRESSION	49
14. SOLUTIONS TO THE NORMAL EQUATIONS IN EQUATION 13	49
15. NORMAL EQUATIONS $\mathbf{A} \vec{\beta} = \vec{b}$ FOR LEAST-SQUARES PROBLEMS WITH MULTIPLE REGRESSORS AND MULTIPLE INPUTS	50
16. WEIGHTED SUM FORMULA USED FOR EACH FIELD COMPARISON RECORD	68
17. THE MEAN OF INPUT i	70
18. THE STANDARD DEVIATION OF INPUT i	70
19. FORMULA USED TO SCALE THE TRAINING AND TESTING INPUT VECTORS	70
20. ROOT MEAN SQUARED ERROR FOR Q TRAINING VECTORS	71
21. NORMALIZED ERROR FOR Q TRAINING VECTORS	71

LIST OF SYMBOLS

A	$(P + 1) \times (P + 1)$ normal equations coefficient matrix of X
$\bar{b}$	$(P + 1) \times 1$ normal equations column vector of observations
$d(t)$	Desired output at time t for a perceptron
E	Sum of the squared error
E_i^2	Squared error for point i to a regression line
$e(t)$	Error at time t for a perceptron
L	Number of input vectors $\bar{x}$ for scaling
m	Number of input layer neurons in a BP neural network
N	Number of observations (data points) in a linear model
n	Number of hidden layer neurons in a BP neural network
NE	Normalized error of a BP neural network
P	Number of unknowns in a linear model
p	Number of output layer neurons in a BP neural network
Q	Number of training vectors in a BP neural network
$RMSE$	Root of the mean squared error of a BP neural network
s_{x_i}	Standard deviation of input x_i
t	Time t
V	Weight matrix from input to hidden neurons in a BP neural network
$\bar{v}_j$	Column weight vector for hidden layer neuron j in a BP neural network
Δv_{jk}^t	Change in weight v from neuron j to neuron k at time t
W	Weight matrix from input to hidden neurons in a BP neural network
$\bar{w}_i$	Column weight vector for input layer neuron i in a BP neural network
w	“Importance” weight value used to manually reconcile training data sets
w_{ij}	Weight value from neuron i to neuron j in a perceptron

$\Delta w'_{ij}$	Change in weight w from neuron i to neuron j at time t
$w_i(t)$	Weight for neuron i at time t for a perceptron
$\mathbf{X}$	$N \times (P + 1)$ matrix of data points and regressors in a linear model
$\bar{x}$	An input column vector to a BP neural network or a linear model
$\bar{x}_i$	Mean of the input x_i
x_i	Single input value
$\bar{y}$	$N \times 1$ column vector of observations in a linear model
$\bar{y}^{error}$	Column vector of hidden layer neuron errors in a BP neural network
$\bar{y}^{out}$	Column vector of hidden layer neuron outputs in a BP neural network
y_i	Output of the i^{th} neuron; i^{th} observation in a linear model
$\bar{z}^{desired}$	Column vector of desired outputs in a BP neural network
$\bar{z}^{error}$	Column vector of output layer neuron errors in a BP neural network
$\bar{z}^{out}$	Column vector of actual outputs in a BP neural network
α	Momentum term for a BP neural network
$\bar{\beta}$	$(P + 1) \times 1$ column vector of unknown parameters in a linear model
β_j	Unknown parameter j in a linear model
η	Learning rate term for a perceptron and a BP neural network
θ_i	Threshold for the i^{th} neuron in a perceptron and a BP neural network

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
ASCII	American Standard Code for Information Interchange
BLN	Blanket Purchase Order
BP	Backpropagation
CSV	Comma Separated Values
DMS	Departmental Management System
DSB	Disbursement Voucher (DMS)
DSS	Decision Support System
DV	Disbursement Voucher (GL)
FP	Fiscal Period
FY	Fiscal Year
GL	General Ledger
INV	Invoice
KDD	Knowledge Discovery in Databases
LPO	Limited Purchase Order
LS	Least-Squares
NPO	Paper-less Purchase Order
OCR	Optical Character Recognition
OLAP	On Line Analytical Processing
OLTP	On Line Transaction Processing
PO	Purchase Order
RDBMS	Relational Database Management System
REQ	Requisition
RPO	Regular Purchase Order
SQL	Structured Query Language

UTK	University of Tennessee at Knoxville
UTSI	University of Tennessee Space Institute
VLDB	Very Large Databases

CHAPTER I

Introduction

Since the invention of electronic databases, many organizations have accumulated vast quantities of data related to, among other things, finances, customers, and the particular business needs of an organization. In many cases, these databases were built to fulfill an immediate business requirement without much thought as to how the actual data would be integrated into existing and future systems [2]. Likewise, the design of many of these databases did not include provisions suitable for the analysis of large sets of data. Instead, these databases were only used for the collection and storage of data. This sort of activity produced many disjoint data sets ranging from flat-files to relational and non-relational databases [12, 13, 44, 45, 51, 55].

After many years of building and populating databases, organizations usually determine that many valuable insights into the nature of their business, their customer's needs and wants, and their financial stability, or instability, is contained within the data that has accumulated throughout the lifetime of the organization. Unfortunately, it is often quite a chore to retrieve valuable information from such a large collection of sometimes disjoint data maintained under different operating requirements and different systems [2, 3, 4, 12, 19, 45]. Business managers may

spend hours developing complex queries and analyzing sets of data looking for the answers to general questions such as “how do I know when a data element from one database is the same data element in another database” or “what are the distinguishing characteristics of the consumers that buy our products and those that do not”, and many others. Obviously, answers to these types of questions can add to understanding and decision making and, thus, may improve the profitability of a business [2, 12, 13, 14, 55]. However, these answers may be difficult or even impossible to find by using traditional database query techniques. It is necessary to use a more advanced analysis technique.

In the past, the primary tools for analyzing data were based on statistical models that helped to identify trends and to find other interesting patterns in data [2, 3, 4]. For example, the regression model is a very popular model and has many variations including linear regression, non-linear regression, and two-pass logistic regression. These techniques are based on solid mathematical foundations and have been demonstrated to provide clarity and understanding about data. However, the patterns and trends found by statistical models are often subject to the understanding and experience of the persons selecting the data used by the model and also those interpreting the results. In some cases, this is acceptable; in others, hidden trends and patterns remain undetected due to the complicated attributes of the problem space as

well as the reader's experience and knowledge about the data being analyzed [3, 4, 15, 18, 46].

Neural networks provide more sophisticated pattern matching methods where the data selection by experts is of less importance due to the neural network's ability to sift through and classify the truly important characteristics in the data and ignore the trivial ones. In other words, a neural network will ignore those data inputs that are not important while at the same time paying much more attention to the inputs that are providing the most value [1, 4, 6, 15, 47, 52]. This ability negates, or at least may minimize, the need to rely on a human's intuition and experience when selecting the data. In addition, it is one of the goals of a neural network to identify hidden patterns and relationships in data that are not apparent by other means. Unfortunately, the end results of a neural network must still be interpreted by skilled individuals since a neural network can only work with numeric data [1, 4, 15, 52].

This thesis examines two databases maintained by The University of Tennessee at Knoxville and The University of Tennessee Space Institute which contain data about the same real-world business entities. Each database is maintained separately by different persons and departments. These databases are often reconciled manually to find answers to everyday business questions needed to make decisions. In addition, one database will often contain data that is months older than the other and data that is the same in both databases is troublesome to recognize. *The problem addressed by*

this thesis is that of identifying the records in a database which refer to the same real-world entity as records in the other database, so that decisions can be based upon the most current information available in the union of the two databases. At its root, this is a *data heterogeneity* problem, which is the structural or semantic incompatibilities that result from the same data being represented differently in separate databases [54]. Structural problems occur when the same data is depicted differently and semantic problems are found when the same data attributes have separate meanings [54]. Similar research into this area has been done by [44] and [54] using advanced additions to standard database *relational calculus* [10] for mismatched domains [44] and evidential reasoning methods for extended relations [54], respectively.

This research compares two methods to solve the data heterogeneity problem found in the two University of Tennessee databases. The first method uses artificial intelligence (AI), specifically a back-propagation neural network, and the second method uses a linear regression statistical model. Each method's results are examined and compared to determine the strengths, weaknesses, and similarities.

Chapter II reviews the current literature applicable to pattern matching in databases. This includes knowledge discovery in databases, data mining, neural networks, and linear regression modeling.

Chapter III presents the methods used to reconcile the two databases including data preparation, algorithm descriptions, and the pattern matching details and differences between the two algorithms.

Chapter IV describes the results and conclusions of both algorithms with a comparison of the good and bad points of each. Performance data is listed for each method.

CHAPTER II

Literature Review

A. Knowledge Discovery in Databases

Knowledge Discovery in Databases (KDD) is a process that has been described as the “non-trivial extraction of implicit, previously unknown and potentially useful knowledge from data” [3, 51, 55]. The KDD process consists of several phases that have generally been identified as data selection, cleansing, enrichment, coding, and data mining [2, 3, 4, 12, 17, 18, 55]. This section briefly describes each phase leading up to the data mining phase where the actual discovery of knowledge takes place.

Data selection is the procedure in which the source databases for an organization are analyzed to determine which data are necessary to find the answers to specific questions [2, 3, 4, 17, 55]. Usually, the selection process takes a substantial amount of time especially when the volume of data is large. Also, this step may have to be repeated if the selected data is missing information that may not have seemed important in the first attempt. For instance, in a marketing application, there may be many obscure relationships that identify which customers are more likely to buy products ranging from details such as whether or not a person owns a dog or drives a particular type of car [2, 13, 17]. Unusual details like these may be overlooked during

the first attempt at analyzing the source databases. Therefore, it may be necessary to repeat this step if the data mining doesn't find anything useful [2, 55].

Data cleansing occurs to fix any errant data such as duplicate records [2, 3, 4, 17, 55]. For example, in cases of insurance or other fraud detection, de-duplication of data may consist of identifying and correcting entries that were entered into the database correctly but that were purposely reported incorrectly by people attempting to obscure a fact about themselves [3, 15]. Also, duplicate records may have been entered as a result of misspelling a person's name, transposing a social security number, and many other causes [3, 4]. Data that needs to be cleaned is called *polluted* or is said to contain *pollution* [3].

Table 1 is an example of data that has been polluted by the misspelling of a person's last name in the records labeled 2001 and 2009. This brings up an interested point about data cleansing: even though we know that one of the records is wrong, which one is the correct one? For examples such as this, it may be necessary to manually correct the entry by looking back through other sources to find the correct

Table 1. Data that needs to be cleaned

Record Number	Last Name	First Name	Purchase Date	Social Security #
2001	Smith	Michael	04-30-1990	411-99-9182
2002	Jones	Elizabeth	11-11-1996	009-93-1256
2003	Koslowski	John	11-12-1993	988-12-3863
2005	Abbott	Craig	01-01-1901	111-34-9841
2009	Smyth	Michael	04-30-1990	411-99-9182

spelling if there are no other electronic sources containing the correct information. On the other hand, in examples of fraud, the last name may have been deliberately misspelled so it may be important for the mining of data to not change this record [2, 3, 4].

Another common cleaning process is the correcting of data items to provide for domain consistency [2, 3, 4, 17, 55]. For example, in Table 1, notice that record 2005 has a Purchase Date equal to January 1, 1901, which is probably an error since it is far out of the range of the other purchase dates [3]. Perhaps the data entry person did not know the purchase date and was forced by programmed restraints to enter a value. However, for the data mining process, this sort of data can be a problem as January 1, 1901 indicates a known value to the mining algorithm even though it is really intended to be an unknown value. One way of correcting this data would be to insert a NULL value for the date [3]. Table 2 shows the corrected data for this simple example.

The cleaning of data is comparable in many ways to data mining since many pattern matching techniques can be applied here. However, this is not the same as mining the data because the information being corrected is somewhat trivial in nature and may require a lot of manual effort to complete [2, 3].

In the enrichment phase, which is sometimes known as preprocessing of data, additional details may be added to each record which contains information obtained from other sources such as demographic data about customers or data collected from

customer surveys [2, 3, 4]. In addition, a common practice is to combine two or more data attributes into a new attribute. Many times, this new attribute is a calculated value [4]. Sums, averages, and other aggregate functions are some examples of information that can be created on a record by record basis that may have meaning during the discovery phase. The new attribute may either add to the data in a record or replace the fields that were used to arrive at the new value. In some instances, the enrichment of data may consist of breaking some fields in each record apart to better present the data for the coding phase. For example, it may be desirable to separate the parts of a date into three fields containing month, day, and year. This will give each date part a stronger value during the mining process and may make the coding phase easier to complete [3, 4]. An example of enrichment using the data from Table 2 might be the addition of income and credit history information about each person which would presumably be obtained by a purchased credit history report.

Table 2. Data after cleaning is performed

Record Number	Last Name	First Name	Purchase Date	Social Security #
2001	Smith	Michael	04-30-1990	411-99-9182
2002	Jones	Elizabeth	11-11-1996	009-93-1256
2003	Koslowski	John	11-12-1993	988-12-3863
2005	Abbott	Craig	NULL	111-34-9841

The coding phase consists of modifying the data to fit the algorithm that is to be used by the data mining algorithm [3, 4, 55]. This includes normalization of numeric data and the conversion of text data to a numeric form. As an example, a person's birth date may be converted to an age value to remove the date format [3, 4]. This encoding of data is necessary because mining algorithms typically work with discrete numbers rather than text or other complicated numeric data types such as dates [3, 4, 17]. Likewise, it may be determined during the coding phase that there are not enough details about a particular record to keep it in the data set to be mined. For example, in Table 2, it may be necessary to have an actual purchase date for the particular mining algorithm. Since there is not one, record 2005 may be removed from the data set [3].

Data selection, cleansing, enrichment, and coding are all part of the overall data preparation for the data mining phase. The discovery of knowledge is the primary objective of data mining. The term data mining comes from the analogy that data mining is used to find that nugget of valuable information in a mine full of data [2, 3, 4, 55]. It should be noted that the phases leading up to data mining are not necessarily taken in the listed order and some may have to be repeated before any real value is found by mining [2, 3, 55]. For example, if data currently being mined was coded incorrectly, then the data may need to be coded again in a different fashion before mining can discover relevant facts.

A.1 Decision Support Systems

Usually, KDD is brought about to benefit a decision support system (DSS) which is a computer system designed to give executive personnel and other business managers responsible for the current and future planning of an organization the ability to analyze current and historical data [2, 3, 4, 19, 55]. It is the extraction of such data by the discovery process that benefits business by predicting trends or finding unknown patterns which may lead to, among other things, increased profits or better cost savings [2, 4, 55]. Specifically, the information in a DSS is usually organized in such a way as to promote the use of drill-down capabilities, visualization, and other useful graphical ways of presenting data so that it can be quickly interpreted and, thus, provide value. Based on current literature, data about business entities seems to be the prevalent type of information used by decision support systems today. It is, therefore, no coincidence that the interest in the KDD process has arisen because of several factors related to business processing. However, any data that can be identified and organized by the KDD phases can be mined for new information and may provide valuable insights into the specific problem space represented by the data [2, 3, 4, 15, 19, 55].

In the 1980's, most organizations accumulated gigabytes of data about their customers, business requirements, scientific findings, etc. These very large databases (VLDB) usually contain a wealth of information [2, 16]. However, extracting hidden

information contained in these VLDB systems using standard query techniques, such as Structured Query Language (SQL) can be at best improbable and more likely practically impossible [2, 3, 4, 16, 46, 51]. Although SQL is a valuable tool, it is not a data mining tool. Instead, SQL usually presents what is known as *shallow* data, which simply means you must have some idea of what you are looking for before you can extract it; there is no discovery of non-obvious data. However, SQL and other similar tools are necessary since they may be able to extract as much as 80% of what is needed and also because of the aggregate and grouping functionality built in to the language [3, 14, 46, 51, 55].

Another factor leading to the popularity of decision systems using KDD is the improvements in machine learning techniques available. This has added merit to the idea of using AI for decision support data mining instead of a more statistical approach [3, 4, 46, 51, 55]. The research done in [34] is an example of using AI to solve a fraud detection problem where traditional statistical methods proved inadequate.

Likewise, the advances in parallel processing are directly linked to the success of using AI systems. As the hardware capabilities improve and the algorithms become more accurate, the use of KDD continues to increase [2, 3, 18, 51, 55]. Many computer chips now exist that contain AI algorithms, such as neural networks, that are built directly into the circuitry. An example of this type of chip is introduced in [35].

Finally, the merger of networking and client-server development has led to the capability of connecting many different databases together for the purposes of using data from both to make decisions [2, 3, 12, 13, 14, 19, 44, 54, 55]. Today, networks are available that can easily support a three tier client-server system that can communicate with many different server database systems. For example, a hospital may decide to connect its patient medical history and disease information databases with the intention of mining the combined information to better detect the unknown causes of certain diseases such as diabetes [4]. In the years before the prevalence of client-server development, this joining of databases may not have been possible due to the restrictions placed by the different systems managing each database. But, even if communication between systems was possible, in most cases it would not have been practical both from a technical and a cost standpoint to search through the data for hidden relationships. Client-server systems and the more advanced networking architecture have solved this problem by distributing the processing and data across different servers and have reduced the need for a mainframe class of machine to support decision support needs and other data mining applications [19]. In fact, most data mining and DSS software today are used to query a server-based data warehouse, which is the topic of the next section [2, 12, 13, 14, 55].

A.2 Data Warehouses

A data warehouse is a database system specifically designed to support query only types of operations [2, 3, 12, 13, 14, 55]. In other words, data in warehouses is usually not allowed to be updated. For the purposes of this research, we will only consider Relational Database Management Systems (RDBMS) as a data warehousing platform [10]. This section touches upon several key points about data warehouses and their benefits for data mining.

Unlike relational databases or other industry defined terms, a data warehouse has no official definition [2]. Nevertheless, most sources agree that a data warehouse is a database that stores a collection of data obtained both from operational systems and external sources. Data warehouses exist to support business decisions and not to support business operations [2, 12, 13, 14, 55]. In other words, a data warehouse's sole purpose is to support data mining and other query only systems such as a DSS. In fact, a DSS is often built by using data from a data warehouse [2, 3, 14, 55].

Designing a data warehouse is not a trivial task. Unlike most transaction based databases, the data warehouse may never actually be complete since the users of the warehouse will most likely find new things to do with the data and need other changes to support their findings [19]. However, as with most systems, the initial content of the warehouse and reporting capabilities it will support must be defined. It may take several months of discussion with the primary users and managers of legacy data

systems before a plan can be created that describes what data to include and exclude as well as the types of applications the warehouse will initially support. Likewise, in many organizations, finding someone who actually remembers what the legacy data in older systems means can be a quite a task. However, once a plan is decided upon, then decisions about how to get the legacy data into a warehouse must be made [2, 12, 13, 14, 46]. This usually involves many hours of programming and data extraction as well as interpreting older database designs and other such things. In addition, if a company is planning to mine its data warehouse, then consideration to the type of mining should be given when designing the data warehouse so that it naturally accommodates the selected data mining algorithms [2, 3, 14, 55].

The data in a warehouse is arranged in such a manner as to speed up the selection of subsets of data so that searching for values and performing other intensive operations, such as data mining, are handled in an efficient manner [2, 12, 13, 14]. Typical database optimizations such as clustered indexes, division of data among several files physically located on different volumes, and the addition of *metadata*, which is data describing the warehoused data, are examples of procedures that can be implemented inside a data warehouse to increase the efficiency of query-only databases [12, 13, 45, 55]. There may also be some duplicate storage of data, but not duplicate records. For instance, data about an entity may need to be represented in two or more different formats for use by different reporting mechanisms [2, 3, 18, 55].

Also, even within a single data set, duplicate information may be necessary to speed up query operations. This information is generated through the denormalization of the database tables [2, 10, 12, 13, 14].

A properly designed data warehouse also minimizes or, in some cases, eliminates, the need for clients to query the on-line transaction processing (OLTP) systems that supply the data in a real-time environment. The data warehouse thus reduces the load on OLTP systems in addition to adding additional reporting functionality. Databases supplying the data warehouse can therefore concentrate on the day-to-day operations without being interfered with by large queries [2, 13, 14, 18]. Likewise, the data warehouse is not affected by the day-to-day transaction processing.

Data warehouses are typically populated on a regular interval by data from other databases. This interval may be each day, each week, or possibly each month and is selected based on the specific reporting requirements [2, 12, 13, 14, 19]. For instance, in a financial accounting database, information about certain accounts may not change in the source database except once a month. Therefore, it is not necessary to load the budget data nightly to provide for reporting. Also, intelligent data extraction tools can be written to only refresh data that has changed since the last time it was loaded. Likewise, aggregate values related to the data may be calculated and stored inside the detail records so that simple queries can produce the desired value

rather than recalculated the aggregate each time. Obviously, this enhances the speed of data retrieval when aggregates are involved. The only caveat to using a data warehouse is the delay of information; a data warehouse is not the right environment to support the needs of on-line, up to the minute types of queries or systems such as a stock market ticker. However, it is certainly the right place to analyze the stock market data to predict the future value of stocks, for instance [2, 3, 4].

Any erroneous data discovered in a warehouse should be first corrected at the source of the data and then rolled forward into the data warehouse at the next refresh interval. This insures a single point of update and does not allow more errors to be introduced. In some cases, an automated attendant can handle the updates to both the source and the data warehouse to keep the reporting up to date. This is known as *back-flushing* the changes made in a warehouse back to the source of the data [2]. It is not an industry practice, however, to allow data in a warehouse to be updated. But, for data that is not reloaded frequently, changes may have to be made in both the operational databases and the data warehouse to provide for correct reporting [2].

When building warehouses, the data preparation procedure of KDD can be used to populate the warehouse with the correct interpretation of data needed by the mining algorithm. In fact, a properly designed data warehouse should contain exactly what a data mining algorithm needs to handle the discovery of new information [3, 4, 55]. If it doesn't, then some changes to the way the data is extracted and stored may

need to be made before data mining can continue. The general accepted procedure for populating a data warehouse involves extracting data from the source databases, enhancing the data with pre-calculated aggregate values and any other additional information, and then importing the data into the warehouse [2, 3, 12, 13, 14].

Although very similar to the KDD data preparation process, it is not exactly the same since the data warehouse may be supporting many other applications other than data mining. However, the KDD process is certainly useful for helping to populate the warehouse.

The primary reasons for not linking the source databases to the data warehouse for loading the data is the inefficiency of maintaining the indexes in the warehouse and the increased network traffic. It is much faster to recreate an index from scratch than to have to maintain it when adding several thousand rows [2]. However, there is no reason that the source databases cannot be linked together and have the data mined in this manner. In some cases, the data is linked together as a first step in the creation of a data warehouse. It is usually not as efficient to do this, but it depends on the specific needs for data mining and other reporting requirements. Operational systems connected in this manner are called a virtual data warehouse, but, in most cases, this is not done since the efficiency would not be the same as with an actual, physical data warehouse [19, 56].

Once data is in the warehouse, it is then used by on-line analytical processing (OLAP) tools, data mining, and other reporting programs. Since the warehouse is designed for fast query access, data can be mined from the warehouse at a speed much greater than by accessing either flat files or the source databases. However, flat-files and the source databases could certainly be used by mining algorithms [2, 3, 12, 13, 14, 19, 55].

Any new data discovered by data mining algorithms can be stored in data mining metadata tables in the warehouse so it does not have to be completely redone at a later time. Also, most data discovery programs are smart enough to not reduplicate the effort needed to discover data if it already exists in the metadata or does not need to be recreated [2, 12, 13, 14, 49, 51]. As an alternative, the processes used to discover new data may be redone if and when the data used by these processes is refreshed. Once again, it depends on the specific data mining requirements.

This section has been a brief introduction to data warehousing. Many more details have been omitted due to the complexity and length of such a discussion. As far as data mining is concerned, data warehouses are the primary sources for data being mined if the amount or complexity of data is large enough to justify the need for additional databases, staff, and the hardware to support them [2, 12, 13, 14]. The data mining algorithms do not depend on a data warehouse for any other reason and can

certainly work with data from extracted flat-files or the original source databases [3, 4].

B. Artificial Neural Networks

An artificial neural network (ANN) is a system designed in such a way as to mimic the human brain's ability to learn, recall, and generalize from training patterns of data. The ANN is made up of many individual nodes called, among other things, neurons, processing elements, or units [1, 4, 5, 6, 7, 8, 9]. Each neuron typically performs a relatively simple sum operation over all its inputs and, if the sum is greater than some threshold, the neuron is said to "fire" an output. The power of an ANN comes from the fact that all processing elements are interconnected such that the output of a single neuron is sent as the input to many other neurons, and so forth. In more complex neural networking architectures, the output of a neuron may also become part of that same neuron's input [1, 5, 6, 7, 8, 9, 42].

The architecture of an ANN is parallel in nature whereas most algorithms are linear and usually perform a series of steps in an iterative process and must be given specific instructions about what to do at each step. The ANN can be used to find a regression mapping for non-linear relationships such as those related to pattern matching, forecasting, classification, optimization, and data clustering [1, 5, 6, 7, 8, 9]. A typical computer program is poorly suited to these types of tasks due to the inherent

lack of parallelism of such systems. However, the typical computer system is much better at performing calculations involving complex computations and arithmetic operations [1, 7]. In other words, just as the brain may have difficulty in calculating a complex value, the ANN is not designed to solve computational arithmetic problems.

Most ANN systems contain neurons interconnected into layers. A minimum of three layers are necessary for the mapping of non-linear relationships although more may be added for various reasons [1, 5, 6, 7, 8, 9, 39, 50]. Each layer can have many neurons and no two layers are required to have the same number of neurons. The amount of neurons in the first layer is determined by the number of inputs to the network. Likewise, the amount of neurons in the outer layer is determined by the number of output values generated by the network. Therefore, both the number of input and output layer neurons are specific to the problem being solved. Any layers found between the input layer and the output layer are called hidden layers. The number of neurons in each hidden layer is arbitrary and is decided based upon how well the network learns the non-linear function it is attempting to find [1, 5, 6, 7, 39, 50]. Typically, data is processed in a feed-forward manner starting with the input neurons, passing through the hidden layer neurons, and then through the output neurons.

B.1 Mathematical Basis

A single neuron performs an operation on its given inputs called a *summation function*. This is typically just a weighted sum of all inputs. After the value is created, it is compared against a certain *threshold* using an *activation*, or *transfer*, function. If the value is greater than the threshold, then the neuron outputs a 1; otherwise, a 0 is output. This type of neuron was originally proposed by McCulloch and Pitts in 1943 and is sometimes referred to as an *M-P neuron* [7]. Figure 1 shows an M-P neuron. The linear summation function $f()$, shown in Equation 1, does a weighted sum of the inputs at time t multiplied by the weights for each input. Equation 2 shows the activation function, $a()$, which is a step function, which takes the output from $f()$ and compares it to the threshold value θ_i . The activation function causes the neuron to

$$y_i(t + 1) = f \left(\sum_{j=1}^m w_{ij} x_j(t) - \theta_i \right)$$

Equation 1. Weighted sum adjusted for threshold

$$a(f) = \begin{cases} 1, & f \geq 0 \\ 0, & f < 0 \end{cases}$$

Equation 2. The step activation function

“fire” a 1 if $f()$ is greater than θ_i or a 0 if not [1, 7]. The threshold value is really just adding an offset to the weighted sum. It can be stored as the first weight with the first

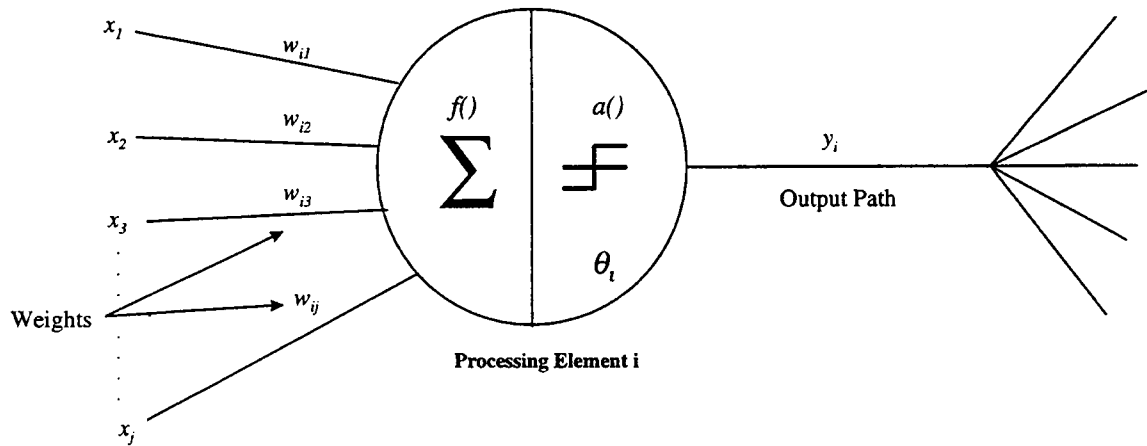


Figure 1. Diagram of an M-P neuron

input as 1. This is called a *bias* [1]. An alternative activation function produces *bipolar* output of -1 or +1 instead of 0 or 1 [1].

In 1962, Frank Rosenblatt coined the term *perceptron* to describe a neural network consisting of interconnected M-P neurons [1]. A simple perceptron may only have an input and an output layer of neurons. Figure 2 shows a simple perceptron with three input layer neurons and two output layer neurons.

The perceptron learning rule is used to adjust the weights between each layer. By adjusting the weights, the network is in effect making some connections stronger than others. Therefore, the input that is strongest will have more “weight” during the summation function causing the neuron to fire. The opposite holds true, as well, because there will be inputs that have a weaker connection, i.e., lower weight, which

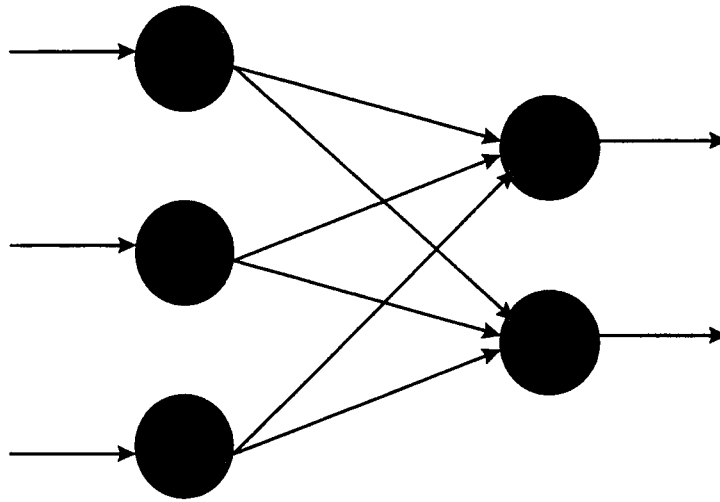


Figure 2. Perceptron with 3 inputs and 2 outputs

has little effect on the neuron firing. The repetitive process of adjusting weights is how the network “learns” to identify patterns in the data presented to it as inputs and is called *training* the network [1, 4, 5, 6, 7].

A basic shortcoming of a two-layer perceptron is that it is unable to learn data unless it is linearly separable. Data that is linearly separable can be separated into groups by a hyperplane. To learn non-linearly separable data, at least three layers must be present in the perceptron [1, 8]. Figure 3 is an example of linear separable data. The X values represent one group and the Y values the other. W , the weight vector, is perpendicular to the hyperplane separating the two groups.

The training of a perceptron is done by adjusting the hyperplane until it fits

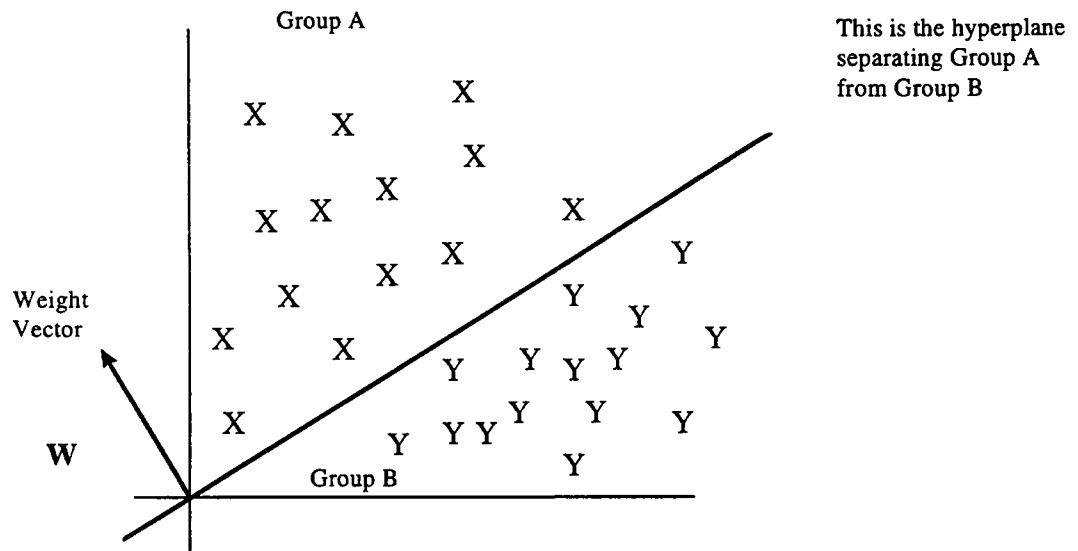


Figure 3. Linearly separable data

between the groups of data. This is done in an iterative process that calculates the activations of all neurons and adjusts the weights by an error term. The error term measures how far off the hyperplane is from being in the correct place. Therefore, the perceptron is minimizing the error between the actual and expected outputs [1]. After training for some specified amount of time or iterations, the network is ready to accept inputs and classify them into the groups.

Other additional items can be introduced that are used to control the change in the weights such as the *learning rate*. The learning rate is a number less than one that is multiplied to the weight adaption term and has the effect of slowing down the weight changes [1].

The algorithm used to train a perceptron follows [1].

(1) Initialize the weights and threshold.

Set $w_i(t)$, $1 \leq i \leq N$, to be the weight from input i at time t , θ to be the threshold value in the output node, η to be the learning rate, $0 \leq \eta \leq 1$. Set $w_i(0)$ and the threshold to small random values. Let $e(t)$ be the error term.

(2) Present input values and the desired output.

Inputs are $x_1, x_2, \dots, x_N$ and the desired output is $d(t)$.

(3) Calculate the actual output using Equation 1.

(4) Adjust the weights using both the learning rate (η) and the Widrow-Hoff delta rule error term $e(t)$.

a) $e(t) \leftarrow y(t) - d(t)$

b) $w_i(t+1) \leftarrow w_i(t) - \eta e(t) x_i(t)$

(5) Repeat steps 2, 3, and 4 until $e(t)$ is minimized or a specified number iterations has taken place.

A shortcoming of the perceptron rule is that for a multi-layered perceptron, the training algorithm may have problems adjusting the weights due to the on/off nature of the threshold function. This is known as a *credit assignment* or *credit apportionment* problem [42]. It is so named due to the difficulty the network has in assigning the proper credit to the weights in the middle that *actually* activated the neuron versus the

weights in the middle that *just* activated the neuron. In other words, there is no way for the network to determine the scale by which the weights in the middle layers should be adjusted [1].

The solution is to replace the on/off threshold function with either an “S”-shaped, or sigmoidal, type of function or a linear threshold between the limits of 0 and 1. The sloping region in the middle has the effect of producing an output close to 1 when the weighted sum is very high above the threshold, a 0 when it is well below, and, most importantly, a value somewhere between 0 and 1 when the weighted sum is close to the threshold. This solution allows the network to relate each neuron’s output back to the inputs [1]. Figure 4 shows an example of the sigmoidal transfer function.

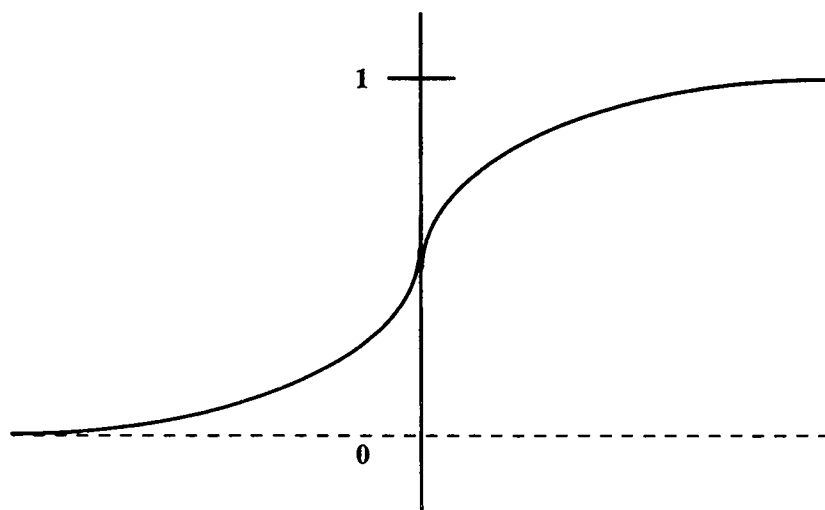


Figure 4. Sigmoidal threshold function

B.2 Backpropagation

Backpropagation (BP) is a rule used to train a multi-layered perceptron consisting of at least 3 layers of neurons. The BP algorithm, also called the *generalized delta rule*, became popular in 1986 after research done by Rumelhart, McClelland, and Williams [1]. It is the most widely used form of neural network today and can solve problems such as classification, pattern matching, financial forecasting, and medical diagnosis [4, 5, 7]. Most BP networks are three layer, feed-forward networks where each layer is fully connected to the next. In other words, all the output values from the neurons in the input layer are connected to each of the neurons in the middle layer which in turn has all of its outputs connected to the neurons in the output layer. Figure 5 is an example of a 3 layer network with 5 input neurons, 3 middle (hidden) neurons, and 2 output neurons that could be used for BP training.

The BP algorithm gets its name from the process of back propagating error terms in the reverse direction of the input flow. The error term calculated by the output neurons is sent back to the middle layer neurons and used to adjust the weights between the middle and output layers. Then, the error calculated by the output layer is used with the weights between the middle and output layers to adjust the weights between the input and middle layers [1, 4, 5, 6, 7].

BP is a *supervised learning* training algorithm. To put it another way, the

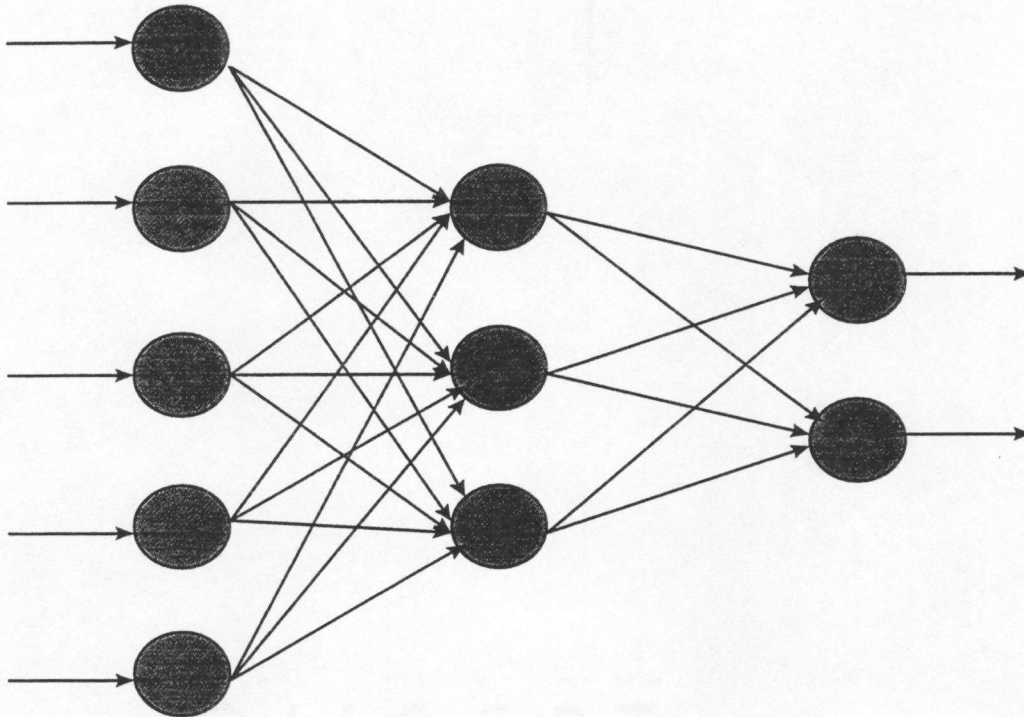


Figure 5. Backpropagation neural network with 5 inputs, 3 hidden neurons, and 2 outputs

network must be trained from a set of data that contains the correct answers. This insures that the error calculations and weight adjustments can be completed correctly while training because the network has the answer. To contrast, in an *unsupervised learning* training algorithm, the network does not have the correct answers and must attempt to learn without any outside help [1].

The training algorithm for a BP network is similar to that of a perceptron, but with some added additions to account for the hidden layer neurons. Basically, the BP network is a multi-layer perceptron and must use a differentiable transfer function to converge correctly due to the credit assignment problem. Usually, the function chosen for the activation is $a(x) = \frac{1}{(1 + e^{-x})}$ not only because it is differentiable but also because it has a simple derivative $a'(x) = a(x)(1 - a(x))$. The derivative is used by the new learning rule in the error calculation for backpropagation and a simple one is usually preferred to speed up the arithmetic [1].

The process for training consists of iterating through the input data until the mean squared error of the output is minimized or a specific number of iterations have been reached. This process is known as a *gradient descent* algorithm because it is causing the weight vectors to move down a *delta* vector which represents the steepest path toward the global minimum of the function. During each cycle, or *epoch*, the input data is fed forward through the network and the errors back propagated in the reverse order of the input. Successful minimization of the mean squared error term means the network has adjusted its weights to be able to generalize to the input patterns specified and can now be used on an unknown set of inputs and give a fairly accurate answer [1, 23]. Figure 6 is a simplified example of gradient descent on a parabolic error surface.

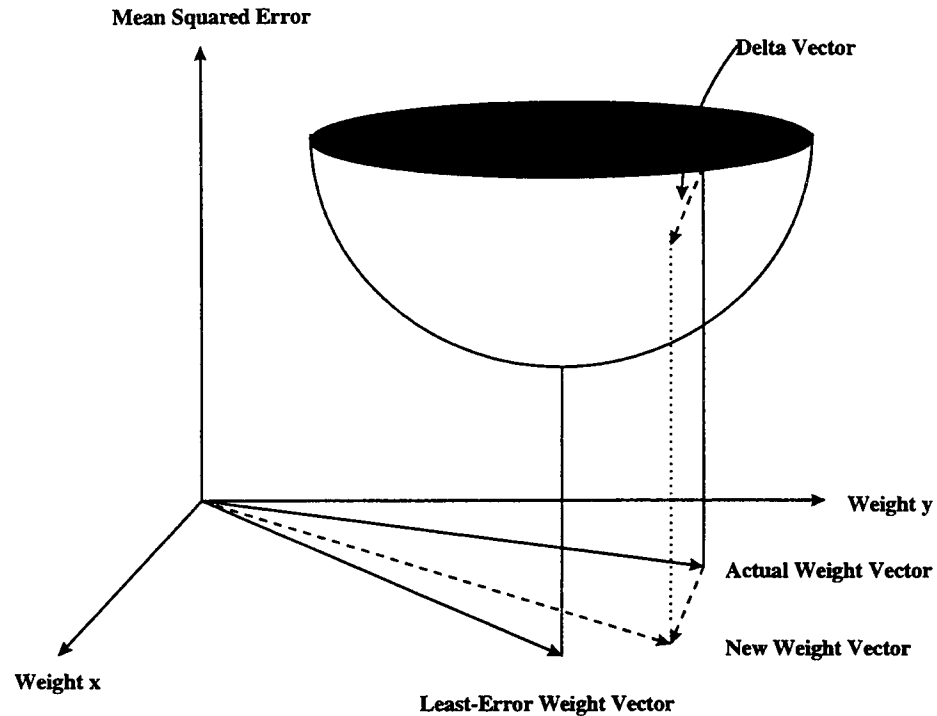


Figure 6. Gradient descent on a parabolic error surface

In some cases, the network may converge to a local minimum in the pattern space of the input data rather than to the absolute global minimum of the function. To attempt to avoid this result, another parameter called *momentum* was introduced into the training equation. The momentum term is a number between 0 and 1 and is used to force a big jump across local minimum early in the training but still allow the

network to remain sensitive to small changes in the function's mapping. This usually has the effect of swifter convergence of the network [1, 5, 6, 7, 8, 22].

In addition, if the network is having problems jumping over local minima, the addition of more hidden layer neurons may help the network to converge [1, 5, 6, 7, 8, 22]. The number of hidden layer neurons needed has been the topic of much research. It is a generally accepted practice to start out training a neural network with an abundance of hidden neurons. Then, after the network has been trained, reduce the number of hidden neurons until the generalization of the network is within the accepted tolerances and can still recognize the training set [4, 5, 6, 7, 8]. A reduction of hidden layer neurons while the network is training is called *pruning* the network [50].

The addition of threshold components to the hidden and output layers which in effect biases the activations. Also, each threshold must be adjusted during training based on the learning rate and the computed error for the appropriate layer of neurons. However, thresholds generally add little to the training or operation of a BP network [1, 5].

The algorithm used to calculate BP follows [1, 5, 23]. Vector notation is used to simplify the equations. Also, the dot product for any two column vectors $\vec{u}$ and $\vec{v}$ is written $\vec{u}^T \vec{v}$.

(1) Variables used by the BP algorithm.

- (a) Let m be the number of input layer neurons, n the number of hidden layer neurons, and p the number of output layer neurons.
- (b) Let $\mathbf{V}$ be the weight matrix between the input and hidden layers such that $\bar{v}_j$ is the column weight vector for hidden layer neuron j , $\mathbf{V} \in \mathbb{R}^{m \times n}$. Likewise, let $\mathbf{W}$ be the weight matrix between the hidden and output layers such that $\bar{w}_i$ is the column weight vector for output layer neuron i , $\mathbf{W} \in \mathbb{R}^{n \times p}$.
- (c) Let η be the learning rate such that $0 \leq \eta \leq 1$ and let α be the momentum term such that $0 \leq \alpha \leq 1$.
- (d) Let $\bar{x} \in \mathbb{R}^m$ be the input column vector to the input layer neurons, $\bar{y}^{out} \in \mathbb{R}^n$ be the output column vector for the hidden layer neurons, and $\bar{z}^{out} \in \mathbb{R}^p$ be the output column vector for the output layer neurons.
- (e) Let $\bar{y}^{error} \in \mathbb{R}^n$ be the column vector of errors for the hidden layer neurons and $\bar{z}^{error} \in \mathbb{R}^p$ be the column vector of errors for the output layer neurons.
- (f) Let $\bar{z}^{desired} \in \mathbb{R}^p$ be the column vector of desired output values corresponding to the input vector $\bar{x}$.
- (g) Let Δw_{ij}^t be the change in the weight w at time t from neuron i to neuron j and Δv_{jk}^t be the change in weight v at time t from neuron j to neuron k .

(h) Let $a(x) = \frac{1}{(1 + e^{-x})}$ be the activation function for each neuron where the

derivative of $a(x)$ is $a'(x) = a(x)(1 - a(x))$.

(2) Initialize weights.

Assign random values between 0 and 1 to the elements of matrices **V** and **W**.

(3) For each input pattern $(\tilde{x}, \tilde{z}^{desired})$,

(a) Compute the forward activations for the hidden and output layers:

Hidden layer:

for each j from 1 to n

$$y_j^{out} \leftarrow a(\tilde{x}^T \tilde{v}_j)$$

Output layer:

for each i from 1 to p

$$z_i^{out} \leftarrow a((\tilde{y}^{out})^T \tilde{w}_i)$$

(b) Calculate the output and hidden layer errors:

Output layer error:

for each i from 1 to p

$$z_i^{error} \leftarrow z_i^{out} (1 - z_i^{out}) (z_i^{desired} - z_i^{out})$$

Hidden layer error:

for each j from 1 to n

$$y_j^{error} \leftarrow y_j^{out} (1 - y_j^{out}) \sum_{i=1}^p w_{ji} z_i^{error}$$

(c) Adjust the weight matrices **W** and **V**:

Matrix **W**:

for each i from 1 to p

for each j from 1 to n

$$w_{ji} \leftarrow w_{ji} + \eta y_j^{out} z_i^{error} + \alpha \Delta w_{ji}^{t-1}$$

Matrix **V**:

for each j from 1 to n

for each k from 1 to m

$$v_{kj} \leftarrow v_{kj} + \eta x_k y_j^{error} + \alpha \Delta v_{kj}^{t-1}$$

(4) Repeat step 3 until all values of $\bar{z}^{error}$ are within a specified tolerance for each $(\bar{x}, \bar{z}^{desired})$ pair and for each neuron in each layer.

In step 3a, the input pattern is sent through the network in a feed-forward manner causing all neurons to perform the activation function. The output layer's activation also depends on the hidden layer's activation. This is the way in which the feed-forward of the input data occurs.

In step 3b, the error for the output layer is calculated by using $a'(x)$ along with subtracting the actual output from the desired output. Then, the hidden layer's error is

calculated by backpropagating the output layer's error and adjusting it using the weights between the hidden and output layers.

Finally, in step 3c, the weight vectors are adjusted using the learning rate and momentum terms. Also, the output layer's actual output and error is backpropagated to the weights to adjust them correctly. After this step, the BP network is said to have learned.

Successive repetitions of the BP algorithm subject to the constraints specified in step 4 will gradually cause each hidden neuron to become a *feature detector* that maps input patterns to the correct output patterns. This is how a BP neural network performs pattern matching [1].

The most common problem associated with the BP rule is the slow convergence. It may take many iterations through training data to get a BP network to converge correctly. This is most often due to an inappropriate initial set of weights which may cause the network to become *saturated* at a high error for several iterations during training [39]. Thus, a substantial amount of research has been conducted in hopes to avoid premature saturation and speed up the convergence of the BP algorithm by finding the appropriate initial settings of the weights of the network. One example of this research is using a weighted least-squares estimator to predict the initial values of the weight vectors [33].

Another problem that may occur during the training process of a BP network is called *overfitting* [1, 5, 6, 7]. Overfitting happens when the network memorizes rather than generalizes the input data. Then, when presented with an input pattern that it has not seen before, the network may return a very inaccurate answer [1, 5, 6, 7, 23]. One popular way of correcting overfitting is to reduce the number of hidden layer neurons so that the network is forced to generalize to the input patterns rather than memorizing the input patterns [1, 4, 50].

B.3 Interpreting Results

A neural network is considered to be a “black-box” algorithm because it doesn’t explain how it arrived at a conclusion. An input pattern is presented to the network and it returns an answer. Therefore, the output from a neural network must be carefully examined to determine what, if anything, the network has learned. Also, the weights used to arrive at a decision may need to be examined [1, 4, 48, 50, 52].

Since a neural network only works with numeric data, it is necessary to convert the output from the network into a usable format. In some cases, the numeric data can simply be de-normalized, or scaled, to arrive at the correct answer. For example, in a financial forecasting network, a single number indicating expected share price for a stock may be the network’s only output. Since the range of possible outputs of a sigmoid

unit is either between 0 and 1 or between -1 and +1, it must be scaled to fit the actual price. An output of 0.5 from the network could indicate 50 cents, 50 dollars, or 50,000 dollars a share [4]. The conversion of an output to a understandable form is very dependent upon the problem being addressed by the neural network.

In DSS uses, the fact that a neural network was able to generate the correct output value for a specified input pattern may not be as important as discovering what factors contributed the most in the network arriving at its decision. For instance, in a modeling or forecasting application, a neural network has learned to model a function, but there is no indication as to the formula that defines this function. However, by varying the input parameters, a lot can be understood about how the function acts. For example, suppose we are using a neural network that is predicting return on investment for a set of products. By holding all inputs but one constant, we can see the effect that the one input has on the expected return on investment. Likewise, if we then hold that one constant and vary another one, then we can see how the two are related. This can continue through all the inputs to determine their relationship to one another and to the output value generated. Determining the impact that an input neuron in a neural network has on the output result of a neural network is called *sensitivity analysis* [4]. The reader is referred to [43] for an excellent example of sensitivity analysis.

A more expensive way of performing sensitivity analysis with a BP network is to keep up with the error terms computed during the backpropagation step. By having all the error going back to the input layer, a measure of the degree to which each input contributes to the output error can be obtained. The accumulated errors over time can then be normalized to compute the contribution percentage that each input has on the output error [4].

Another way in which a neural networks results can be interpreted is to derive a set of rules in the form of a Boolean *decision tree* based on the raw data in a network. The process of mapping a neural network to a decision tree can be a chore since a neural network is using a non-linear equation and a decision tree is Boolean in nature. However, it is not impossible and some research has been done in this field [4].

Visualization techniques are by far the most widely used method for interpreting the networks results. Some simple examples of visualization include bar charts, histograms, and scatter plots. Other more advanced graphical models include those that actually draw the neural network and visually display the strongest connections and neurons in different colors or the plotting of clusters of data as identified by a neural network. Another widely used visual interpretation tool used with neural networks is the *Hinton* diagram. It is a display showing a collection of boxes whose size is comparable to the magnitude of the connection weight and whose

color depends on the sign [4]. By examining the magnitude and sign of the weights, the relative importance of each input can be determined.

C. Regression Analysis

There are many mathematical techniques used to fit a curve $g(x)$ to a set of data points in any dimension that exactly passes through each point once. This process, known as *interpolation*, can be used to determine the specific values of $g(x)$ for any value x in the data. The main assumption when interpolating data points is that each point in the set is free from all error. In other words, each point $(x_1, x_2, \dots, x_N)$ is known to be *exactly* on the interpolated curve [25, 26]. However, when dealing with experimental data, each point has associated with it a possible degree of error of sufficient magnitude that any interpolated curve will not correctly represent the data [25, 26].

Regression analysis seeks to overcome the effect of experimental error by finding the curve that “best” fits the data points so as to minimize the distance from the known data points to the curve [2, 21, 25, 26]. In other words, a regression line filters out the error in data and finds the underlying trend [31]. The term “best-fit” is used because the measure of degree of accuracy is arbitrary [25]. This curve is then

used to analyze any possible relationships between variables in the set of data with a measurable degree of error.

A relationship between variables may be either a functional relationship or some other *degree of association*. A functional relationship implies that the value for one variable can be used to *exactly* determine the value of the related variable. On the other hand, a degree of association means that although one variable does not exactly determine the value of the related variable, it has associated with it a range of values for the other variable, and vice-versa. As it turns out, after adding an error term, the methods for finding the “best” fit curve for either type of relationship are essentially the same [25].

Linear regression is used when the relationship between variables is linear in nature. That is, the value of the “best” fit curve is a polynomial of degree one which is simply a line. For non-linear regression, the curve will take the form of either a polynomial of degree greater than one or an exponential form. The variables for linear and non-linear regression are called *regressors*, *predictors*, or *independent variables*. In most cases, regressors represent a quantitative value, such as temperature, rather than a qualitative value, such as a department code, because it is most often more useful to analyze quantitative values [21].

C.1 Mathematical Basis

Regression analysis starts with a set of observations, or data points, which are obtained in either a designed experiment or in a random fashion on the data being analyzed. These observations all have measured responses, or outputs, which depend on one or more unknown parameters [21]. When linear relationships exist between the data and the responses, linear regression is used to find the “best” fitting line for the observation data by approximating the values of the unknown parameters.

When there is only one regressor and one output, the regression line has two unknown parameters: the intercept of the line and the slope of the line. Equation 3 is the simple linear regression model where x is the one regressor, y the one output, and β_0 and β_1 are the intercept and slope of the line, respectively.

$$y = \beta_0 + \beta_1 x$$

Equation 3. The simple linear regression model

As an example, consider a situation in which several data points exist about the number of deaths resulting from strokes as related to blood pressure for patients in various hospitals in the United States. As one would suspect, the number of deaths will increase as the blood pressure rises; that is, the number of deaths is a function of the blood pressure. Figure 7 is a plot of some hypothetical data points relating stroke

deaths to blood pressure. By simple observation, the number of data points tends to increase as the blood pressure increases in approximately a linear fashion. Equation 3 could be used to fit this data.

Equation 3 is useful for several reasons. First, it is useful for interpolating or extrapolating the y-values for a regressor at unknown values, inferences about the intercept and slope are easily analyzed, and it is intuitively useful to represent data in this manner [21]. For instance, notice that the assumption that $\beta_1 = 0$ is the same as saying that the number of deaths due to stroke have no relation to the blood pressure. Also, if the death rate at a specific blood pressure is desired, then it can be calculated simply by using Equation 3 [21].

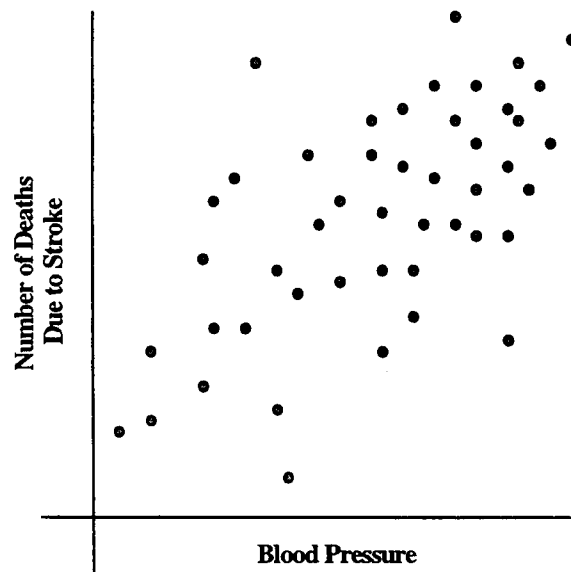


Figure 7. Plot of stroke related deaths against blood pressure

The regression model may also consist of more than one input. If multiple inputs are needed, then Equation 3 is no longer adequate. For the example above, assume that data about the cholesterol level of the patients was also known. Then the number of deaths due to stroke is now a function of both blood pressure and cholesterol level, or $y = f(x_1, x_2)$. Equation 4 shows the new linear regression model where x_2 represents the cholesterol level and β_2 is the unknown associated with x_2 .

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Equation 4. Linear regression model with two inputs

By further expanding Equation 3 into vector form, an equation can be produced that has multiple inputs and multiple regressors. Let N be the number of inputs and P the number of unknowns. Equation 5 is the column vector $\vec{y}$ of multiple outputs and Equation 6 is the column vector $\vec{\beta}$ of unknowns. The multiple regressors and the multiple inputs are shown in the $N \times (P + 1)$ matrix $\mathbf{X}$ of Equation 7. Using this

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Equation 5. Column vector of outputs for a linear model

$$\vec{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

Equation 6. Column vector of unknown parameters for a linear model

form, the best fitting regression line can be determined by solving the system of equations $\mathbf{X}\vec{\beta} = \vec{y}$.

$$\mathbf{X} = \begin{bmatrix} x_{10} & x_{11} & \cdots & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N0} & x_{N1} & \cdots & x_{Np} \end{bmatrix}$$

Equation 7. Coefficient matrix of multiple inputs for multiple linear regression

C.2 Method of Least-Squares

When regression analysis is applied to a set of data, the criteria for what curve “best” fits the data is somewhat arbitrary. However, a common method is to minimize the sum of the squared error from the data points to the curve where error is measured as the distance from the point to the generated curve. This is known as the method of *least-squares*. The sum of the squared error is known as the *least-squares error* [21, 25, 26, 27, 28, 29, 30, 31].

For simple linear regression, the error is the vertical distance from each (x_i, y_i) point to the line $y_i = \beta_0 + \beta_1 x_i$ that best fits the data points. Since a point on the line can be written $(x_i, \beta_0 + \beta_1 x_i)$, the vertical distance is simply the difference between the y-coordinates. This is presented in Equation 8, where the error variable is represented by E_i squared. Equation 9 is the sum of the squared error for each point (x_i, y_i) in the data set. In Equation 9, if all the points are exactly on the line, then the error is zero, i.e., $E = 0$ [31].

$$E_i^2 = (\beta_0 + \beta_1 x_i - y_i)^2$$

Equation 8. Squared error from a single point i to the best fitting line

$$E = \sum_{i=1}^N E_i^2$$

Equation 9. Sum of the squared error for each point i

In most situations, a line where the error is zero for a set of data points does not usually occur. Therefore, the line that “best” fits the data is the line where E is minimized. To do this, we must represent E as a function of the two parameters β_0 and β_1 . Let $E = r(\beta_0, \beta_1)$ represent this equation. The general least-squares problem is solved by taking the partial derivatives of $r(\beta_0, \beta_1)$ with respect to β_0 and β_1 , setting

Table 3. Hypothetical data relating the deaths due to stroke and blood pressure

Hospital	Number of Stoke Related Deaths	Average Blood Pressure
Atlanta General	66	66
Baptist Hospital	77	95
St. Mary's	100	89
Orange County Medical	110	120

each partial derivative equal to 0, and then solving the resulting least-squares system of equations [21, 25, 26, 27, 28, 29, 30, 31].

For instance, consider the example presented in Section C.1 relating the number of deaths from stroke to blood pressure. Table 3 shows some hypothetical data for patients in four hospitals. Using the data from Table 3, the resulting equation for the least-squares error is shown in Equation 10. Equation 11 shows the simplified partial derivatives with respect to β_0 and β_1 as a system of linear equations in matrix

$$E = (\beta_0 + 66\beta_1 - 66)^2 + (\beta_0 + 95\beta_1 - 77)^2 + (\beta_0 + 89\beta_1 - 100)^2 + (\beta_0 + 120\beta_1 - 110)^2$$

Equation 10. Least-squares error for the data in Table 3

$$\begin{bmatrix} 8 & 740 \\ 740 & 71404 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} 706 \\ 64542 \end{bmatrix}$$

Equation 11. "Best" fitting linear system of equations for the data in Table 3

form. The equation for the line that “best” fits the data is shown in Equation 12, where the solutions are $\beta_0 = 18.202$ and $\beta_1 = 0.757$, respectively. Finally, Figure 8 shows the line from Equation 12 plotted against the data points from Table 3.

$$y_i = 18.202 + 0.757x_i$$

Equation 12. Equation of the line that “best” fits the data from Table 3

As it turns out, the solution to a linear system of equations for a least-squares fitting line can be determined by calculating a few simple values based on the N data points. These calculations are shown in Table 4, and the resulting system of equations

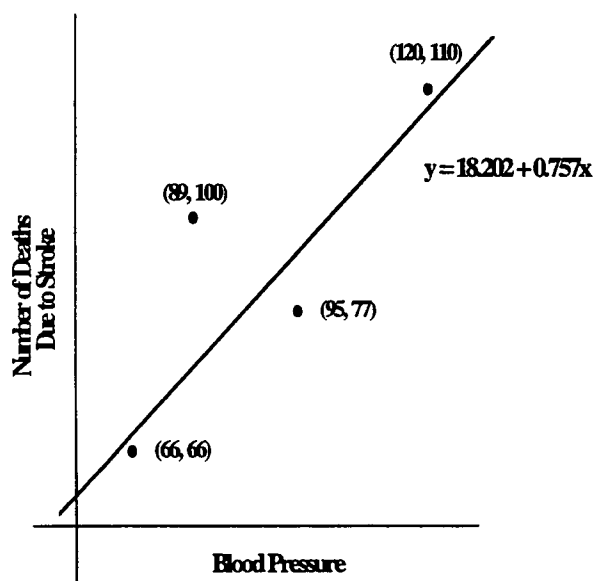


Figure 8. Deaths due to stroke versus blood pressure for data in Table 3

Table 4. Values used to find the linear least-squares approximation for data in Table 3

x_i	y_i	x_i^2	$x_i y_i$	y_i^2
66	66	4356	4356	4356
89	100	7921	8900	10000
95	77	9025	7315	5929
120	110	14400	13200	12100
$\sum 370$	$\sum 353$	$\sum 35702$	$\sum 33771$	$\sum 32385$

in matrix form, or the *normal equations*, used to find β_0 and β_1 are shown in Equation

13. Finally, Equation 14 is the solution to the normal equations. The same solutions obtained above in Equation 12 are easily found by simple substitution of the values

$$\begin{bmatrix} N & \sum_{j=1}^N x_1 \\ \sum_{j=1}^N x_1 & \sum_{j=1}^N x_1^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^N y_1 \\ \sum_{j=1}^N x_1 y_1 \end{bmatrix}$$

Equation 13. Normal equations for simple linear regression

$$\beta_0 = \frac{\sum_{j=1}^N x_1 y_1 - \frac{\left(\sum_{j=1}^N x_1 y_1 \right)}{N}}{\sum_{j=1}^N x_1^2 - \frac{\left(\sum_{j=1}^N x_1 \right)^2}{N}}, \quad \beta_1 = \sum_{j=1}^N y_1 - \beta_0 \frac{\left(\sum_{j=1}^N x_1 \right)}{N}$$

Equation 14. Solutions to the normal equations in Equation 13

from Table 4 into Equation 13 [25, 26, 28, 29, 30].

As would be expected, a set of normal equations similar to Equation 13 can also be generated for least-squares solutions to problems having multiple regressors and multiple inputs. The usual case for a least-squares problem of the form $\mathbf{X}\vec{\beta} = \vec{y}$ is that the number of observations is greater than the number of unknowns ($N > P$) which means that the system is expected to be inconsistent [28]. Furthermore, this means the normal equations $\mathbf{A}\vec{\beta} = \vec{b}$ can be generated by $\mathbf{X}^T \mathbf{X} \vec{\beta} = \mathbf{X}^T \vec{y}$, where $\mathbf{A} = \mathbf{X}^T \mathbf{X}$ and $\vec{b} = \mathbf{X}^T \vec{y}$ [28]. The general form for these equations is shown in Equation 15, where the coefficient matrix $\mathbf{A}$ has dimensions $(P+1) \times (P+1)$, unknown column vector $\vec{\beta}$ is $(P+1) \times 1$, and the output column vector $\vec{b}$ is $(P+1) \times 1$.

$$\begin{bmatrix} N & \sum_{j=1}^N x_1 & \cdots & \sum_{j=1}^N x_{(P-1)} & \sum_{j=1}^N x_P \\ \sum_{j=1}^N x_1 & \sum_{j=1}^N x_1^2 & \cdots & \sum_{j=1}^N x_1 x_{(P-1)} & \sum_{j=1}^N x_1 x_P \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum_{j=1}^N x_{(P-1)} & \sum_{j=1}^N x_{(P-1)} x_1 & \cdots & \sum_{j=1}^N x_{(P-1)}^2 & \sum_{j=1}^N x_{(P-1)} x_P \\ \sum_{j=1}^N x_P & \sum_{j=1}^N x_1 x_P & \cdots & \sum_{j=1}^N x_{(P-1)} x_P & \sum_{j=1}^N x_P^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{(P-1)} \\ \beta_P \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^N y_1 \\ \sum_{j=1}^N x_1 y_1 \\ \vdots \\ \sum_{j=1}^N x_{(P-1)} y_{(P-1)} \\ \sum_{j=1}^N x_P y_P \end{bmatrix}$$

Equation 15. Normal equations $\mathbf{A}\vec{\beta} = \vec{b}$ for least-squares problems with multiple regressors and multiple inputs

Finally, if the columns of $\mathbf{X}$ are linearly independent, then the unique, least-squares solution is given by $\bar{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \bar{y}$, or $\bar{\beta} = \mathbf{A}^{-1} \bar{b}$ [28].

C.3 Interpreting Results

For most cases, the results presented by regression analysis are somewhat straightforward to interpret since the actual equation used to generate the “best” fitting curve is known. Least-squares approximations are generally accepted as the most objective fit for a set of data when the error involved is Gaussian [29]. Therefore, as long as the error can either be assumed Gaussian or proved Gaussian, then the generated curve will give valid answers. Nevertheless, there are a few important things to consider when examining the results.

First, the regression techniques used for least-squares approximation are solely dependent upon the data points. If, for example, a few data points are in error, then the generated curve may skew in order to minimize the least-squares from these points. Therefore, it is extremely important that all data points be independent and represented by the same scale [29]. If the regression curve has only two or three dimensions, one method used to overcome this is simply to draw a plot of the regression curve on graph paper. For multiple linear regression, the plot of the plane may be more difficult to complete. Nevertheless, this should suffice to identify any outlying points that are obviously in error so that they may be discarded from the set of points used to find the

“best” fitting curve [29]. For large data sets, several iterations of removing outlying points in the data may be necessary to finally find the correct curve.

Many statistical calculations may also be used for analyzing the variances in the slope, intercepts, and inflection points of a “best” fitting curve [29]. Also, for point forecasting, it may be useful to find the *confidence interval estimate*, which is the measure of uncertainty each point has when estimated by the regression curve [30]. These measures are useful in describing the behavior of any new data using the regression curve as a basis.

The interpolation and extrapolation of points on the “best” fitting curve is simply a mathematical exercise using the equation of the generated “best” fitting curve. This is a relatively simple way to forecast results, find trends, or discover patterns in data that is often as accurate as more sophisticated techniques.

CHAPTER III

Methods

A. Data Preparation

The data for this research was taken from two real-world databases used by the University of Tennessee at Knoxville (UTK) and the University of Tennessee Space Institute (UTSI) over a period of 12 months. The database at UTK is a general ledger (GL) database and the UTSI database is a Departmental Management System (DMS) database which contains more detailed information about each department's purchase transactions.

This problem was selected due to the burdensome task of manually reconciling the records found in the DMS to the GL database to find the correct account balances for a department. The DMS database is designed to provide a more detailed audit trail on purchases whereas the GL database only shows transactions when purchased goods and services are either promised to be paid or actually paid. So, since a purchase may take a few months to clear the system, the GL information is not up to date with the DMS system information. In other words, once the DMS reflects a purchase, the UTSI staff has essentially spent the money. However, the official amount of money in each account must come from the GL system and not the DMS system. Currently, to obtain more accurate account balances from the DMS database, the matching of records

Table 5. Data from the University of Tennessee at Knoxville GL database

Transaction	Account	Object	Amount	Date	Description
PO 616323	R02400310	334	\$27.00	9/27/95	AMERICAN MATHEMATICAL SOC
DV 771693	E020450	391	\$85.00	11/16/95	MID STATE COMPUTER INC
DV 755738	E024352	612	\$6,866.48	11/1/95	COMARK GOVT-EDUCATION INC
PO 616303	R02400310	334	\$51.91	9/11/95	CRC PRESS INC
DV 771531	E020450	353	\$363.23	11/22/95	BELLSOUTH

Table 6. Data from the University of Tennessee Space Institute DMS database

Account	Encumbrances	FY	FP	Type1	Ref1	Sub Code	Description
E024352	\$6,750.00	96	04	INV	57	612	PENTIUM COMPUTERS (2)
E024352	\$499.00	96	01	REQ	9219055	612	16MB SIMM 70NS GOLD PIN
R02400310	\$51.91	96	03	RPO	616303	334	CRC PRESS INC
R02400310	\$31.78	96	03	MET	52	391	SURGE PROTECTOR PLUG OUTLET
R02400310	\$203.24	96	03	INV	51	334	RESEARCH BOOKS (6)

between the two databases must be completed manually at the end of each month.

Therefore, the data being discovered is hidden and non-obvious in nature, but necessary for business decisions. Tables 5 and 6 are examples of data taken from each database (fields not used by this research have been omitted).

The data from Table 5 is from a printout of transactions in the GL database. The columns in Table 5 consist of a transaction type and number, the account number, an object code used to identify the type of object in the transaction, the amount of the transaction, the date of the transaction, and the description.

For Table 6, the data from the DMS was taken from a printed *suspense report* generated every month. The columns are the account number, the encumbrances, or promised amount, for that transaction, the fiscal year, the fiscal period within the fiscal

year, type and reference fields which indicate the type of transaction and a referring transaction id, a sub-code number which is the type of object in the transaction, and the description. The fiscal year and fiscal period values are interpreted so that the fiscal year number represents the previous year, i.e., fiscal year 96 starts in 1995, and a fiscal periods start at July. So, 96-01 is July, 1995 and 96-12 is June, 1996, for instance.

The transactions in each database follow this entry sequence. First, a purchase requisition (REQ) is entered into the DMS. This is not an official promise for payment to a vendor so there is no corresponding entry in the GL database. Next, either a blanket purchase order (BLN), limited purchase order (LPO), paper-less purchase order (NPO), or a regular purchase order (RPO) is created in the DMS. Since this is an actual promise of money to a vendor, a corresponding purchase order (PO) is entered into the GL data at some point in time after the DMS has been updated. Then, after the vendor's good has been delivered or service has been rendered, the vendor sends an invoice (INV) which is entered into the DMS database. Once again, there is no entry in the GL database to support an invoice as this does not indicate a change in promised or real money. Finally, a disbursement voucher (DSB) is issued and the transaction is entered into the DMS database. A corresponding disbursement voucher (DV) is entered into the GL database to reflect a close to the purchase. Figure 9 illustrates the corresponding processing of purchasing transactions between the two databases.

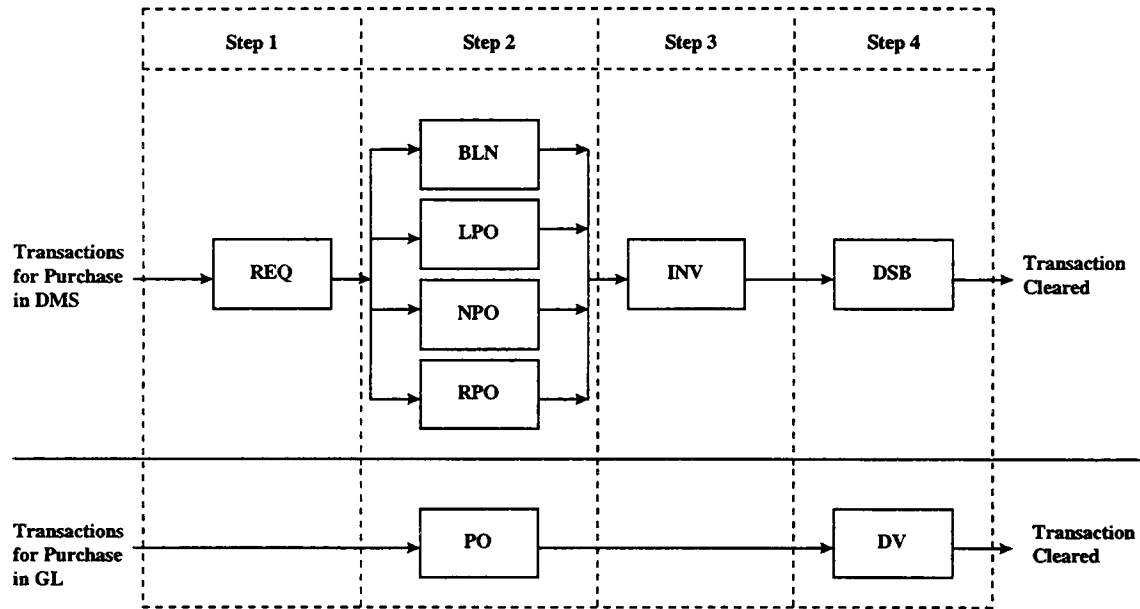


Figure 9. Steps in a purchase transaction between the DMS and GL databases

Data from each database was selected based on the criteria used to manually reconcile the two reports. As seen in Figure 9, any record from the GL database that has a transaction type of PO should have a corresponding entry in the DMS database with a Type1 code of BLN, RPO, NPO, or LPO. Likewise, a DV in the GL database should have both an INV and a DSB in the DMS database. Also, some columns, such as the Object and Sub-Code columns were added because of the uncertainty as to whether these fields contained any valuable information when compared with each other.

There are a number of other transaction types in the DMS and GL databases that were not related to purchases that were removed from the data. These transactions were related to salary transfers, budget revisions, and journal vouchers, for example.

The columns selected for the neural network and linear regression algorithms from the GL data (Table 5) are the transaction Type (text part of the Transaction field), Transaction number (numeric part of the Transaction field), Account, Object, Amount, Date, and Description. For the DMS data (Table 6), the columns used were the Account, Encumbrances, Fiscal Year (FY), Fiscal Period (FP), Type1, Ref1, Sub-Code and Description.

The cleansing of this data consisted first of obtaining the data presented in the reports in an electronic form. This was done by capturing the printouts of these reports in spool file format, cleaning the printer control characters from the files, and reorganizing the data to remove miscellaneous column and report headings.

Several attempts were made to use printed reports and optical character recognition programs (OCR) to convert a scanned image of the reports into a text format. However, this attempt was unsuccessful. For those reports that were only available in paper format, the records were manually entered into a text file. Finally, the data was imported into a Microsoft Access database where it was sorted and then

extracted to a flat file using a comma-separated values (CSV) format to allow easy interpretation.

A.1 Enrichment

The data in CSV format was moved to a Sun Sparc10 workstation where C++ programs were written to encode the data into a form usable by a back-propagation neural network and by a linear regression statistical model. Since this problem is looking for matching records, there was no need to enrich the data sets with any aggregate information such as the sum of the Amount fields, etc. Likewise, there was no external sources of data about either data set to add to the outcome.

First, each data set had added to it an identification number, or *surrogate key*, to each record for proper identification. The record numbers were not, however, used as inputs to either the BP neural network or the linear regression model.

The DSB transactions were dropped from the DMS data since the INV transactions in the DMS data are used to match DV transactions in the GL database during the manual reconciliation process. The remaining transaction type fields in the DMS database were converted to an integer code of 1-3 to indicate which step in the process a particular transaction represented (see Figure 9). Next, the transaction type from the GL database records was separated from the transaction numbers and assigned a corresponding code that would match the step from the DMS database. The

DV type from the GL database was assigned a code of 3 since it can be matched to an INV record from the DMS database. Table 7 shows the codes used for the DMS and GL transactions by type where a '-' indicates no valid entry was available for the corresponding step code.

Table 7. Codes used for the transaction types in the enriched data sets

Step	DMS Database	GL Database
1	REQ	-
2	BLN, LPO, NPO, RPO, MET, MXB	PO
3	INV	DV

Since the Account fields from both data sets were not used in any calculations, it was decided to append 0's onto the end of any value in which the length did not match the longest account number found in either data set. All accounts in the data start with either an 'E' or an 'R' followed by up to eight digits for a total account number length of nine. Every account number that had fewer than eight digits had zeros appended to the end to make its length a total of one letter plus 8 digits.

Finally, the FY/FP combination in the DMS database was converted to the corresponding MM/DD/YY format found in the GL database by using 01 as the day.

Tables 8 and 9 show the data that was originally presented in Tables 5 and 6 after selection, cleaning, and enrichment have been completed.

Table 8. Enriched data from Table 5

ID	Step	Transaction	Account	Object	Amount	Date	Description
1	2	616323	R02400310	334	\$27.00	9/27/95	AMERICAN MATHEMATICAL SOC
2	3	771693	E02045000	391	\$85.00	11/16/95	MID STATE COMPUTER INC
3	3	755738	E02435200	612	\$6,866.48	11/1/95	COMARK GOVT-EDUCATION INC
4	2	616303	R02400310	334	\$51.91	9/11/95	CRC PRESS INC
5	3	771531	E02045000	353	\$363.23	11/22/95	BELLSOUTH

Table 9. Enriched data from Table 6

ID	Account	Encumbrances	Date	Step	Ref	Sub_ Code	Description
1	E02435200	\$6,750.00	10/01/95	3	57	612	PENTIUM COMPUTERS (2)
2	E02435200	\$499.00	7/01/95	1	9219055	612	16MB SIMM 70NS GOLD PIN
3	R02400310	\$51.91	9/01/95	2	616303	334	CRC PRESS INC
4	R02400310	\$31.78	9/01/95	2	52	391	SURGE PROTECTOR PLUG OUTLET
5	R02400310	\$203.24	9/01/95	3	51	334	RESEARCH BOOKS (6)

A.2 Representations

To make the BP neural network and linear regression model work with the data sets, the inputs and outputs for each method had to be defined in a mathematical context. It was decided that a record from one data set would have a certain distance from any records in the other data set. By using a distance measure, both methods are predicting the distance of each record in the GL database from each record in the DMS database. We say that a record from the GL databases matches a record from the DMS database when both records refer to the same purchase. Therefore, records that are close together in distance are considered matching records. Specifically, a value between 0 and 1 was assigned for the distance where a number close to 1 means the

records probably do not match and a number closer to 0 means the records probably do match.

To define the distance measures, the enriched data set was carefully examined to look for proper ways of identifying records. In each case, a field from the GL database was compared with a field from the DMS database and a corresponding distance evaluation formula was generated for each field comparison. There are 9 field comparisons, or inputs, to the problem. The following algorithm describes the method used to generate each field comparison input value.

(1) Variables and functions used by the field comparison input generation algorithm.

- (a) Let (a, b) be a pair of records where $a \in GL$ database and $b \in DMS$ database and $a.fieldname$ (or $b.fieldname$) indicates a specific field within the record (the field names used are from Tables 8 and 9).
- (b) Let $\vec{x} = [x_1, x_2, x_3, \dots, x_9]^T$ be the column vector of field comparisons for a single comparison instance between records a and b .
- (c) Let $substr(str, startpos, len)$ be a function that returns the part of the string identified by str starting at the $startpos$ and of length len , where 1 is the first position in a string.
- (d) Let $julian(date_value)$ be a function that returns the Julian date for any date in MM/DD/YY format identified by $date_value$. The reader is referred to

[57] for a complete Julian date algorithm.

(e) Let $ascii(char)$ be a function that returns the integer ASCII value of the character identified by $char$.

(f) Let $soundex(word)$ be a function that returns the SoundEx value of a string identified by $word$, where a word is any sequence of characters separated by one or more spaces. The SoundEx value is the first letter of the word followed by exactly three digits which are assigned based on the remaining consonants in the word or which are 0 if the word is too short to have a lot of consonants. This value is intended to be equal for two words that sound alike even if they are not spelled exactly the same, i.e., Smith and Smyth have the same SoundEx value of S530. The reader is referred to [11] for a more complete description of the SoundEx algorithm.

(g) Let ‘&’ be used to indicate concatenation of strings such that $s_1 \& s_2 = s_1s_2$.

(h) Let $tonum(string)$ be a function that converts the ASCII string representation of a number to the integer representation of the number.

Likewise, let $tostr(num)$ be a function that converts the integer representation of a number to the string representation of a number.

(i) Let $wcount(str)$ be a function that returns the number of words in the string str , where a word is any sequence of characters separated by one or more

spaces. Likewise, let $word(str, n)$ be a function that returns the string that is the n^{th} word in string str .

(j) Let $ucase(str)$ be a function that returns the string identified by str in all upper case characters.

(k) Comments are enclosed between sets of `/* */`.

(2) For each pair of records (a, b) , $a \in GL$ database, $b \in DMS$ database, we define field comparison input values x_i , $1 \leq i \leq 9$:

(a) `/* Step code */`

$$x_1 = |a.step - b.step|$$

(b) `/* GL Transaction to DMS Reference */`

`IF a.transaction = b.ref THEN`

$$x_2 = 0$$

`ELSE` $x_2 = 1$

`ENDIF`

(c) `/* First letter equality comparison between the two Account numbers */`

`IF substr(a.account,1,1) = substr(b.account,1,1) THEN`

$$x_3 = 0$$

`ELSE` $x_3 = 1$

`ENDIF`

(d) `/* 5th-7th positions (4th-6th digits) of the two Account numbers */`

$$x_4 = |tonum(substr(a.account,5,3)) - tonum(substr(b.account,5,3))|$$

(e) `/* 8th-9th positions (7th-8th digits) of the two Account numbers */`

$$x_5 = |tonum(substr(a.account,8,2)) - tonum(substr(b.account,8,2))|$$

(f) `/* GL Object code to DMS Sub_Code */`

$$x_6 = |a.object - b.sub_code|$$

(g) /* Difference between GL Amount and DMS Encumbrances */

$$x_7 = |a.amount - b.encumbrances|$$

(h) /* Julian date difference between the two Date values */

$$x_8 = |julian(a.date) - julian(b.date)|$$

(i) LET SEWrd = ""
 LET SumA = 0
 LET SumB = 0
 LET CntAWrds = wcount(a.description)
 LET CntBWrds = wcount(b.description)

/* Sum the modified SoundEx values for the GL Description field */
 FOR i = 1 TO CntAWrds DO

/* Make each word UPPERCASE */

$$SEWrd = soundex(ucase(word(a.description, i)))$$

/* Convert the SoundEx value's first (and only) letter to an ASCII
 * number, concatenate the rest of the digits to this number, then
 * convert to an integer and record the sum.
 */

$$SumA += tonum(tostr(ascii(substr(SEWrd, 1, 1))) & substr(SEWrd, 2, 3))$$

NEXT i

/* Same loop as above, but use the Description from the DMS database */
 FOR i = 1 TO CntBWrds DO

$$SEWrd = soundex(ucase(word(b.description, i)))$$

$$SumB += tonum(tostr(ascii(substr(SEWrd, 1, 1))) & substr(SEWrd, 2, 3))$$

NEXT i

/* Find the difference between the mean of the converted SoundEx values
 * for the GL and DMS Descriptions using the number of words as the
 * denominator for the mean calculation.
 */

$$x_9 = |(SumA / CntAWrds) - (SumB / CntBWrd)|$$

The Account fields from both databases can be broken into parts with each part having a hierarchical meaning to the systems. For example, the account “R02400310” really means “R-02-40-03-10” with the beginning letter and each 2 digit combination having some hierarchical meaning to the systems. However, it was observed that the 1st - 3rd digits of the account number were always “024” (“R02400310” in the example above). Therefore, these were dropped from the account number field. The remaining account numbers are divided so that the third input uses a code of 0 if the first characters (‘R’ or ‘E’) match or 1 otherwise, the fourth uses the absolute value of the difference between the 4th-6th digits in each account, and the fifth uses the absolute value of the difference between the 7th-8th digits in each account.

The date fields were converted to Julian date values because a normal Gregorian date is more difficult to encode. Another format for the date that was considered consisted of splitting the date into three separate inputs with one input each for the month difference, the day difference, and the year difference, respectively. However, the Julian form was selected because the FY/FP data always had a day of 01 assigned to it after converted to a MM/DD/YY value, and it was decided that this enhancement would have less error effect inside a large Julian value than as a separate input value.

Finally, the SoundEx algorithm [11] was selected because the variances in descriptions between matching records could be caused by differing abbreviations of words or by misspellings. It was felt that a SoundEx solution would better handle these differences since the values would likely still sound alike. Also, the conversion of the first character to an ASCII value along with taking the average was developed to get a single value from the phrase-like descriptions. For example, the SoundEx value for the word 'Smith', or S530, would first be converted to the integer 83530 (ASCII value for 'S' is 83) and then averaged with the other converted SoundEx values for the remaining words in the description. Had the normal SoundEx of the entire description been used, the value returned would have ignored the right side words of the descriptions.

Table 10 shows the data from Tables 8 and 9 after it has been formed into a Cartesian product and had the field comparison algorithm applied to it. Notice from Table 10 that row (4, 3) has 0's for the input vector values except for x_7 , which is the Julian date field comparison. The reason for the value "10" is because a day of "01" was used when the FY/FP combination was converted into a Gregorian date during the enrichment stage. As is expected, record #4 from Table 8 is a match for record #3 in Table 9. Although record (4, 3) is the only occurrence of a match in the sample data from Tables 8 and 9, there are several values for various inputs that are equal or similar.

Table 10. Field comparison input data from Tables 8 and 9

(a, b)	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
(1, 1)	1	1	1	349	10	278	6723.00	4	1347
(1, 2)	1	1	1	349	10	278	472.00	88	2775
(1, 3)	0	1	0	0	0	0	24.91	26	1451
(1, 4)	0	1	0	0	0	57	4.78	26	5632
(1, 5)	1	1	0	0	0	0	176.24	26	1140
(2, 1)	0	1	0	98	0	221	6665.00	46	1391
(2, 2)	2	1	0	98	0	221	414.00	138	2731
(2, 3)	1	1	1	447	10	57	33.09	76	1495
(2, 4)	1	1	1	447	10	0	53.22	76	5588
(2, 5)	0	1	1	447	10	57	118.24	76	1184
(3, 1)	0	1	0	0	0	0	116.48	31	3286
(3, 2)	2	1	0	0	0	0	6367.48	123	7408
(3, 3)	1	1	1	349	10	278	6814.57	61	3182
(3, 4)	1	1	1	349	10	221	6834.70	61	10265
(3, 5)	0	1	1	349	10	278	6663.24	61	3493
(4, 1)	1	1	1	349	10	278	6698.09	20	104
(4, 2)	1	1	1	349	10	278	447.09	72	4226
(4, 3)	0	0	0	0	0	0	0.00	10	0
(4, 4)	0	1	0	0	0	57	20.13	10	7083
(4, 5)	1	1	0	0	0	0	151.33	10	311
(5, 1)	0	1	0	98	0	259	6386.77	52	7601
(5, 2)	2	1	0	98	0	259	135.77	144	11723
(5, 3)	1	1	1	447	10	19	311.32	82	7497
(5, 4)	1	1	1	447	10	38	331.45	82	14580
(5, 5)	0	1	1	447	10	19	159.99	82	7808

A.3 Quantity and Quality

The set of data used for training was taken from the first six fiscal periods of the 1996 fiscal year (July, 1995 - December, 1995). This made the training set approximately 36% of the entire set of data as most transactions occur in the later 6 fiscal periods of each fiscal year. The remaining 64% was used for testing purposes.

To create the correct outputs for the training set, it was necessary to manually search for the correct matches between the first 6 months of the DMS database and the GL database. For the sole purpose of aiding this manual search process, the “importance” weights given in Equation 16 were assigned to each of the nine field comparison values where a higher value indicated that the corresponding input was

$$w = 10x_1 + 100x_2 + 100x_3 + 10x_4 + 100x_5 + x_6 + 10x_7 + x_8$$

Equation 16. Weighted sum formula used for each field comparison record

deemed more important to the matching process than others with lower values. A value of 1 was used for inputs x_6 and x_8 since the subcode (object code) in the GL database can be arbitrarily reassigned by UT personnel if it is felt the DMS assigned number was incorrect, and the FY/FP combinations all have a day of 01 assigned by the field comparison algorithm presented in A.2. For inputs x_1 , x_4 , and x_7 , 10 was used since the step code is a generated value that has a small differences, digits 4-6 of the account number were observed to be the same in most cases, and the amount-

encumbrances differences were incorrect enough of the time to warrant a smaller weight for some field comparisons. 100 was used for x_2 , x_3 , and x_5 because matching transaction numbers as well as account types and account digits 7-8 are strong indicators of matching records between the two databases. Finally, input x_9 corresponding to the differences in the average SoundEx values of the descriptions was dropped because many of the GL records had no descriptions. However, the actual DMS and GL text descriptions were used when manually checking the training and testing results. Using these importance weights, a weighted sum was generated to aid the manual search for matches between the two databases. The lower the weighted sum value, the greater the chance that the two associated records from the GL and DMS databases matched. The lowest 10 weighted field comparison values for each DMS record in the training set were used as a starting point for the manual search.. Of these 10 values, the best match was determined manually and assigned an output of 0. Next, the first three non-matches were added to the training set with an output of 1. This made the training set exactly 25% matches and 75% non-matches.

For example, by applying Equation 16 to comparisons (4, 3) and (5, 2), we arrive at the values $w_{(4,3)} = 10.0$ and $w_{(5,2)} = 2680.7$. Obviously, comparison (5, 2) is not a match and (4, 3) is most likely a match. So, for the training set, (4, 3) will have an answer of 0 and (5, 2) will have an answer of 1.

Finally, the training and testing data sets of field comparisons (see Table 10) had to be scaled for the neural network and least-squares algorithms. To do this, the mean and standard deviations of each input for L vectors (where L is either the number of training vectors or the number of testing vectors) was found using Equations 17 and 18. Then, Equation 19 was used to get the final scaled vector.

$$\bar{x}_i = \frac{\sum_{j=1}^L x_{ij}}{L}, \quad 1 \leq i \leq 8$$

Equation 17. The mean of input i

$$s_i = \sqrt{\frac{\sum_{j=1}^L (x_{ij} - \bar{x}_i)^2}{L-1}}, \quad 1 \leq i \leq 8$$

Equation 18. The standard deviation of input i

$$\vec{x} = \begin{bmatrix} (x_1 - \bar{x}_1) / s_{x_1} \\ (x_2 - \bar{x}_2) / s_{x_2} \\ (x_3 - \bar{x}_3) / s_{x_3} \\ \vdots \\ (x_8 - \bar{x}_8) / s_{x_8} \end{bmatrix}$$

Equation 19. Formula used to scale the training and testing input vectors

B. Backpropagation Training

The training of the backpropagation neural networks was conducted on a Sparc Ultra station using Ebtide software available at UTSI. This software provided methods to display up to three dimensions of data as well as the hidden node hyperplanes used by each neural network while the network was training. Also, Ebtide provided many measurements used to determine the effectiveness of training like the root-mean-squared error (RMSE) and the normalized error (NE). Equation 20 is the formula for the RMSE and Equation 21 is the formula for the NE.

$$RMSE = \sqrt{\frac{\sum (z^{out} - z^{desired})^2}{Q}}$$

Equation 20. Root mean squared error for Q training vectors

$$NE = \frac{RMSE}{\sqrt{\frac{\sum (z^{desired} - \bar{z}^{desired})^2}{Q}}}$$

Equation 21. Normalized error for Q training vectors

Several different dimensions of data were displayed to get an idea about what the plotted training space would look like. A cutout picture of the data space for input

dimensions (x_6, x_7, x_8) is presented in Figure 10, which has a '+' and '-' signs representing the sign of the point. As can be seen, the data is very condensed into small regions of the input space making it difficult to distinguish between '+' and '-' labels. For this reason, the initial analysis was that a number of hidden nodes would be necessary to successfully train the BP neural network. Also, it was decided to allow a maximum of 100,000 epochs for training each BP neural network. However, the best state of each of the networks were recorded at intervals of 1,000 epochs and were used to test the network at later stages.

Next, the learning rate and momentum were decided upon. Based on the results of the research in [43] that also used BP neural networks, a learning rate ranging from 0.2 to 0.4 was chosen. Also, many local minima were suspected in the error space, a momentum in the range of 0.5 - 0.9 was chosen to help avoid getting stuck before finding the correct global minimum value.

Finally, several different configurations of BP neural networks were considered starting with 8 hidden nodes to match the 8 input values. Also chosen were BP neural networks with 10, 12, and 16 hidden nodes. Each configuration had the 8 inputs and only the one predicted distance output as described in this chapter. Figure 11 is the BP neural network with 8 hidden nodes during the training against plots of the data in the input dimensions (x_6, x_7, x_8).

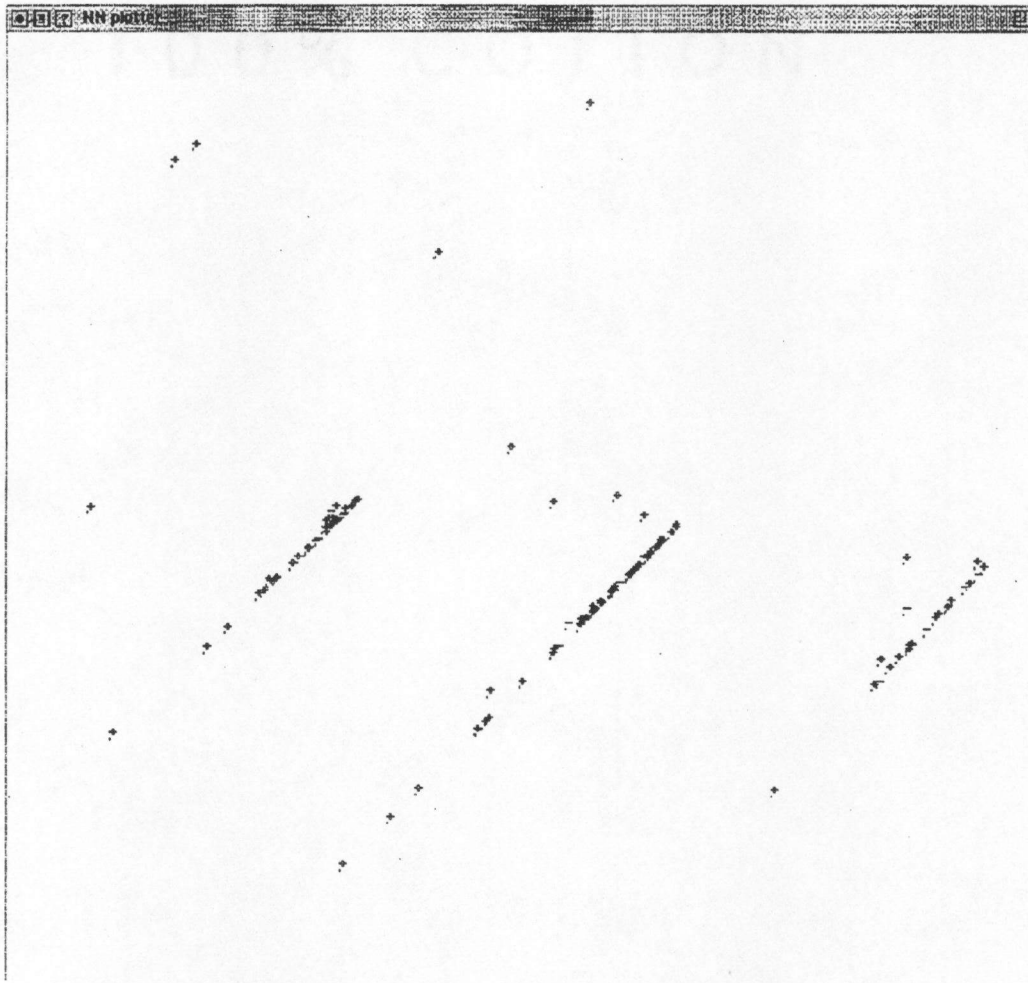


Figure 10. Training points for input dimensions (x_6, x_7, x_8)

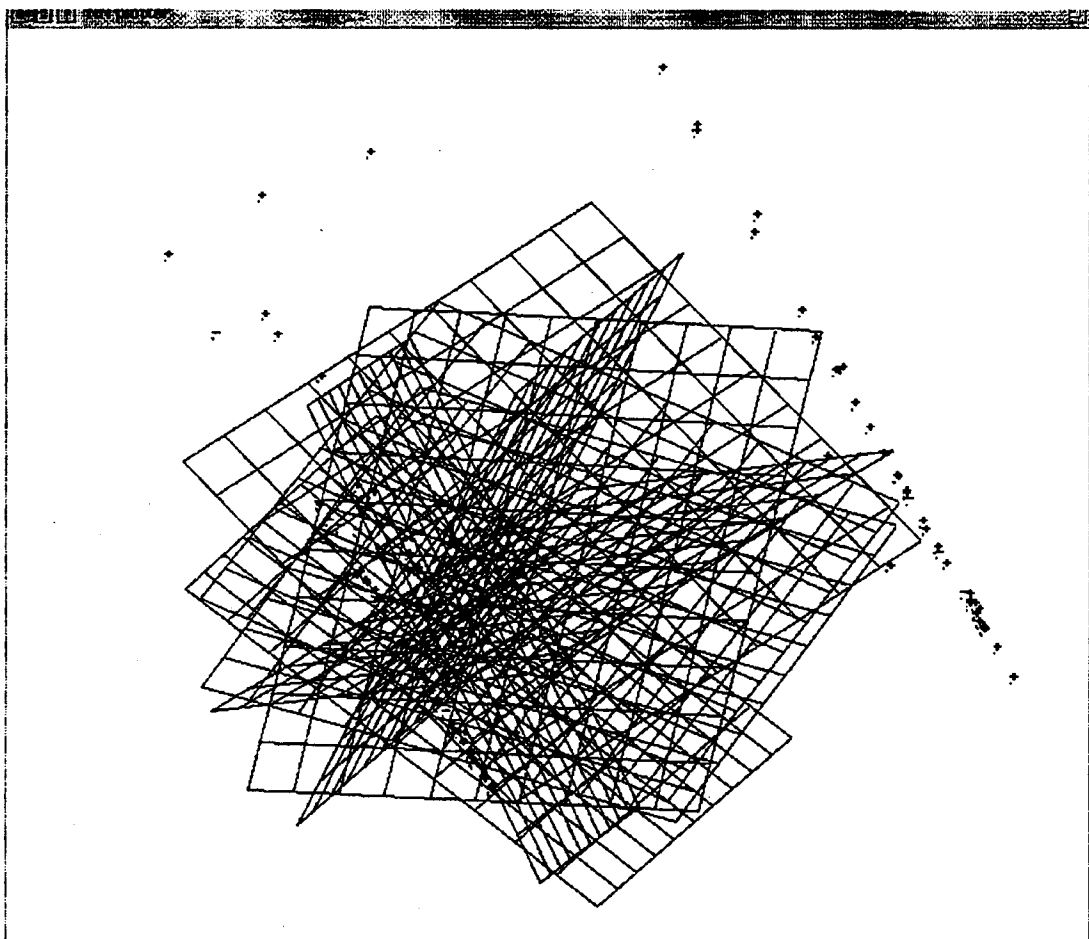


Figure 11. Training the 8x8x1 BP neural network with input dimensions (x_6, x_7, x_8)

After several iterations through various BP neural network architectures, a learning rate of 0.4 was used consistently as this seemed to provide the fastest and most consistent learning. Likewise, any momentum of less than 0.9 had very little positive effect and slowed training considerably. Therefore, 0.9 was used consistently for the momentum term.

Once 100,000 epochs were completed, the best states for each BP configuration were used with the test data set and the resulting outputs were merged with the generated GL and DMS ID numbers. Then, the minimum value for each DMS record was compared manually to determine the accuracy of the solution. There were exactly 119 DMS records to test, and the accuracy measurement was based solely on the percentage correct out of the 119 DMS records. Therefore, if 50 out of 119 records were correctly given by the BP neural network, then the percentage correct assigned was $\frac{50}{119} = 42.02\%$.

C. Least-Squares Execution

For least-squares regression, a C++ program was first written to produce the least-squares normal equations $A\vec{\beta} = \vec{b}$ (Equation 15) using the data in the training set. Then, MatLab software was written to read in the points from the testing set of data and solve the resulting system of equations using the column vector of solutions,

$\bar{\beta}$. The solutions were then merged with the generated GL and DMS ID numbers and the minimum value for each DMS record in the testing set were found exactly as they were for the BP neural networking configurations. The percentage correct was also used as an accuracy measurement for least-squares.

CHAPTER IV

Results and Conclusions

A. Results

The results as first generated by the BP neural network were not encouraging. The best state BP solutions were producing many badly matched records either from not distinguishing correctly between GL and DMS comparisons with matching account numbers or from overemphasizing the differences in amounts and dates. Many iterations using different numbers of hidden nodes and various guesses as to learning rates were tried without success.

The least squares solution initially produced results that were significantly better than the BP solutions. However, there were values in the solutions that had small negative numbers that were obviously due to error and scaling problems. Likewise, many of the values generated for matching GL and DMS records in the least-squares solution were much closer to 1 than those produced by the matches in the BP solutions.

Table 11 is the results of the first attempt at finding the matching records using least-squares (LS) and several BP neural networking configurations with the number of hidden nodes given in parentheses.

Table 11. Results for first attempt at finding DMS matches

Method	Number of Training Epochs	Minimum Training RMSE	Minimum Training NE	Percent of DMS Records Correctly Matched
LS	N/A	N/A	N/A	70/119 = 58.82%
BP (8)	100,000	0.169896	0.392359	24/119 = 20.16%
BP (10)	95,000	0.291307	0.126140	51/119 = 42.86%
BP (12)	31,000	0.085300	0.196992	30/119 = 25.21%
BP (16)	51,000	0.065628	0.151562	59/119 = 49.57%

Given the puzzling testing performance of the BP solutions, attention was given to why the neural networks were unable to learn the training data. Therefore, several more attempts with different configurations were trained and then tested. Table 12 contains the results of these new configurations. A learning rate of 0.4 and a momentum term of 0.9 continued to be used. Also, the training was allowed to go a

Table 12. Results for the second attempt at finding DMS matches

Method	Number of Training Epochs	Minimum Training RMSE	Minimum Training NE	Percent of DMS Records Correctly Matched
BP (2)	50,000	0.279763	0.646086	88/119 = 73.95%
BP (3)	2,000	0.261479	0.603859	59/119 = 49.58%
BP (4)	3,000	0.268949	0.621111	57/119 = 47.90%
BP (5)	18,000	0.221452	0.511422	53/119 = 44.54%
BP (6)	82,000	0.196472	0.453732	46/119 = 38.66%
BP (7)	6,000	0.204185	0.471545	57/119 = 47.90%

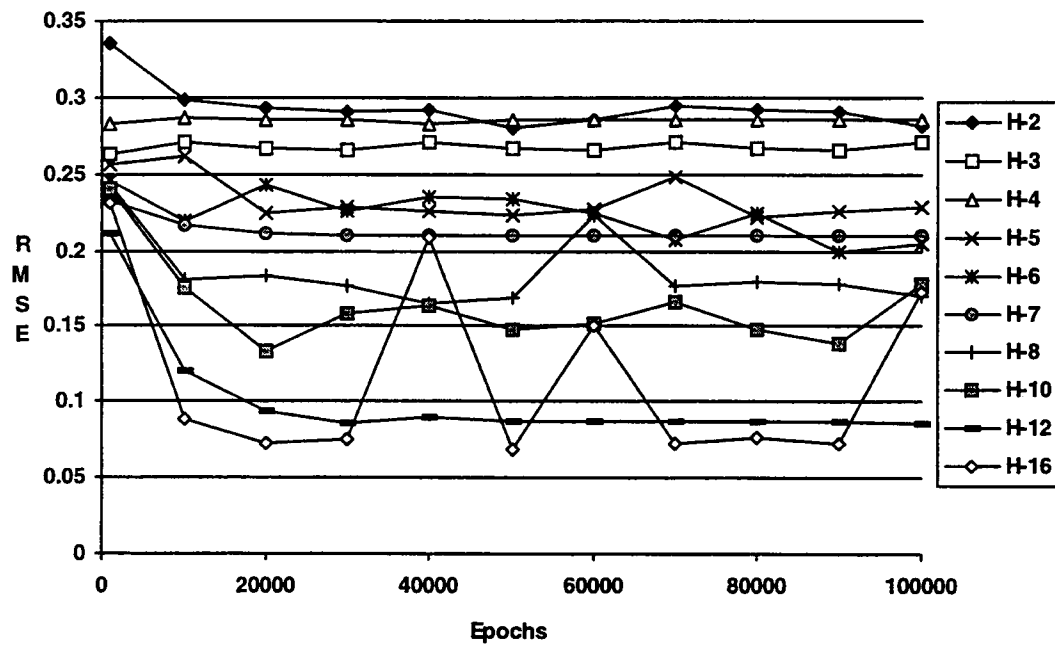


Figure 12. RMSE for BP neural networks during training

maximum of 100,000 epochs and the best state found during these epochs was used to generate the solution for the testing data.

Figure 12 shows the RMSE for all the BP neural network solutions during training for the specified number of epochs. The RMSE is much more constant for the BP solutions with few hidden nodes but begins to oscillate more as the number of hidden nodes increase.

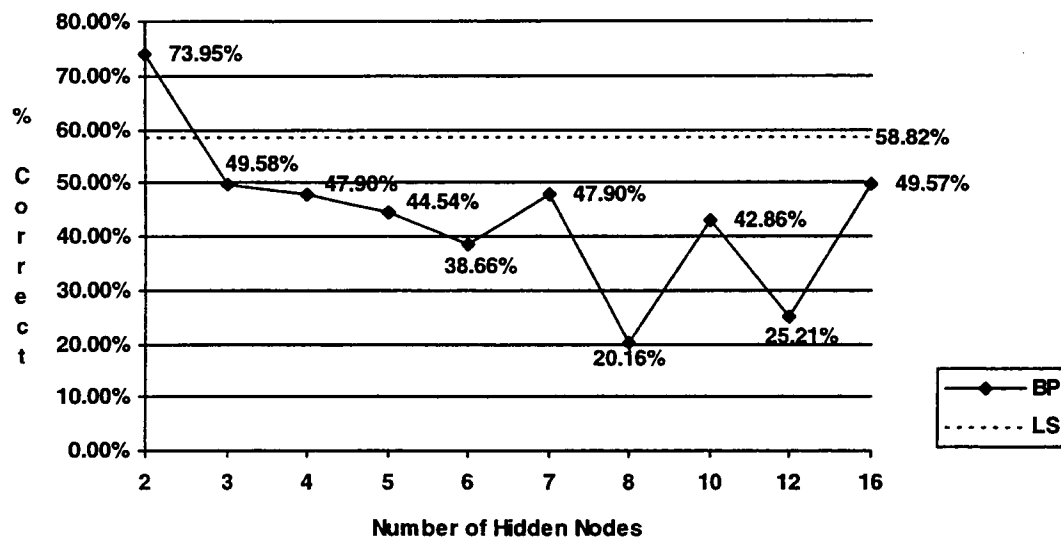


Figure 13. Percent correct for each BP network and Least-Squares

Figure 13 shows the percentage of DMS records correctly matched in the test data set for each BP neural network solution with the specified the number of hidden nodes. The least-squares line is also shown for comparison.

B. Conclusions

As can be seen from Table 12, the BP solution with only 2 hidden nodes outperformed all other BP solutions as well as the least-squares solution. This leads to

the assumption that the BP networks with a higher number of hidden nodes over-fitted the training data and thus did not generalize well when presented with the testing input patterns. This trend can be followed in Figure 13. Notice that the percentage correct from Figure 13 seems to drop steadily from the BP configuration with 2 hidden nodes to the one with 6 hidden nodes. After 6 hidden nodes, the percentage correct oscillates between a fairly good results and a very poor results.

The graph of RMSE values for each BP configuration shown in Figure 12 and the percentage correct shown in Figure 13 indicate that a lower RMSE value does not necessarily mean a better generalized network. In fact, the combination of the second highest RMSE and the highest NE produced the best results for the BP configuration with 2 hidden nodes. To contrast, the 10 hidden node configuration had a higher RMSE and the smallest NE, but only had 42.86% correct matches. These results seem to indicate that a low NE may mean a BP neural network has not generalized well to the training data.

Also, when using a BP neural network to predict a value that is dependent upon data from mismatched domains, the results tend to indicate that this type of data has many local minima that can prevent accurate learning. This experiment may point to the need for different types of data cleaning and data preparation for data across mismatched domains. It is possible that a different way of encoding the input training

vectors may reduce the number of local minima and increase the performance of the BP learning algorithm.

Furthermore, this particular problem had a very high dependence on the momentum term used when training the BP neural network. When the momentum was reduced from its constant 0.9 value, the RMSE frequently oscillated between high and low values. Given that a high momentum was needed to train, one would suspect that a lot of the local minima are perhaps close in value to the global minimum. It may be possible to increase the training accuracy of the BP solutions with greater than 2 hidden nodes and a smaller momentum term by first using some form of initial weight optimization in order to avoid the local minima present early in training.

Since the results of this research show approximately a $\frac{3}{4}$ percentage correct ratio, it is likely that the BP solution with 2 hidden nodes could substantially aid the current labor intensive process of reconciling records in the DMS database to corresponding records in the GL database on a consistent basis. Likewise, these results are encouraging for any similar type of process that must deal with data from mismatched domains. However, given the complex algorithm and manual efforts needed to clean and encode the data sets, an actual application would need to automate this process to be of any real value for UTSI personnel.

List Of References

1. Beale, R. and Jackson, T. Neural Computing: An Introduction. New York: Adam Hilger, 1990.
2. Mattison, Rob, Data Warehousing: Strategies, Technologies, and Techniques. New York: McGraw-Hill, 1996.
3. Adriaans, Pieter and Zantinge, Dolf, Data Mining. New York: Addison-Wesley, 1996.
4. Bigus, Joseph P. Data Mining with Neural Networks: Solving Business Problems – from Application Development to Decision Support. New York: McGraw-Hill, 1996.
5. Blum, Adam, Neural Networks in C++: An Object Oriented Framework for Building Connectionist Systems. New York: John Wiley and Sons, Inc., 1992.
6. Rao, Valluru and Rao, Hayagriva, C++ Neural Networks and Fuzzy Logic. New York: IS:Press, 1995.
7. Lin, Chin-Teng and Lee, C. S. George, Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems. New Jersey: Prentice-Hall, 1995.
8. Fuller, Sean Lane, Using a Cooperative Genetic Algorithm Population to Train Hidden Neural Network Weights. Tullahoma, Tennessee: The University of Tennessee Space Institute, May 1993.
9. Cozart, Michael Thomas, Evaluation of the 'Neural Gas' Network Vector Quantization and Approximation Components. Tullahoma, Tennessee: The University of Tennessee Space Institute, May 1996.
10. Date, C. J. An Introduction to Database Systems. Reading, Massachusetts: Addison-Wesley, 1991.
11. Knuth, Donald E. The Art of Computer Programming, Volume 3: Sorting and Searching. Reading, Massachusetts: Addison-Wesley, 1973.
12. Brackett, Michael H. The Data Warehouse Challenge: Taming Data Chaos. New York: John Wiley and Sons, Inc., 1996.
13. Gill, Harjinder S. and Rao, Prakash C. The Official Client/Server Computing Guide to Data Warehousing. Indianapolis, Indiana: Que Corporation, 1996.

14. Corey, Michael J. and Abbey, Michael, Oracle Data Warehousing. Berkeley, California: McGraw-Hill, 1997.
15. Brooks, Peter, "Data Mining Today." DBMS, February, 1997, pp. 59-60,64.
16. Brobst, Stephen and Rotertson, Owen, "Taming Data Giants: Getting a Grip on VLDB in the Real World." DBMS, February, 1997, pp. 38-47, 65.
17. Edelstein, Herb and Millenson, Janet, "Lessons from the Trenches: Knowledge, Discovery, and Data Mining." DBMS, February, 1997, pp. 62-63.
18. Mena, Jesus, "Mining the Warehouse." DM Review, January, 1997, pp. 41-45.
19. DeCori, Bob, "Data Delivery Middleware", DM Review, April, 1997, pp. 53-55.
20. Fukunaga, Keinosuke, Introduction to Statistical Pattern Recognition. San Diego, California: Academic Press, Inc., 1990.
21. Hocking, Ronald R. Methods and Applications of Linear Models: Regression and the Analysis of Variance. New York: John Wiley and Sons, Inc., 1996.
22. Ealy, Derek Michael, Adaptive Learning Rate Techniques for Neural Networks. Tullahoma, Tennessee: The University of Tennessee Space Institute, May, 1993.
23. Goldberg, David E. and Whitehead, Bruce, A Short Course on: Genetic Algorithms and Neural Networks. Tullahoma, Tennessee: The University of Tennessee Space Institute, May, 1993.
24. Whitehead, Bruce A. Genetic Evolution of Radial Basis Function Coverage Using Orthogonal Niches. Tullahoma, Tennessee: The University of Tennessee Space Institute, April, 1996.
25. Gerald, Curtis F. and Wheatley, Patrick O. Applied Numerical Analysis. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., June, 1989.
26. Burden, Richard L. and Faires, J. Douglas, Numerical Analysis. Boston, Massachusetts: PWS-Kent Publishing Company, 1988.

27. Johnson, Robert S. and Vinson, Jr., Thomas O. Elementary Linear Algebra. Orlando, Florida: Harcourt Brace Jovanovich, Inc., 1987.
28. Strang, Gilbert, Linear Algebra and Its Applications. New York, New York: Academic Press, Inc., 1976.
29. Taylor, John Keenan, Statistical Techniques for Data Analysis. Chelsea, Michigan: Lewis Publishers, Inc., 1990.
30. Hillier, Frederick S. and Lieberman, Gerald J. Introduction to Operations Research. New York, New York: McGraw-Hill, Inc., 1990.
31. Goldstein, Larry J., Lay, David C., and Schneider, David L. Brief Calculus and Its Applications. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1987.
32. Van Huffel, Sabine, and Vandewalle, Joos. The Total Least Squares Problem: Computational Aspects and Analysis. Philadelphia, Pennsylvania: The Society for Industrial and Applied Mathematics, 1991.
33. Chen, C. L. P. "A Rapid Supervised Learning Neural Network for Function Interpolation and Approximation." IEEE Transactions on Neural Networks, Volume 7, Number 5, 1996, pp. 1220-1230.
34. Kwon, T. M., and Feroz, E. M. "A Multilayered Perceptron Approach to Prediction of the SEC's Investigation Targets." IEEE Transactions on Neural Networks, Volume 7, Number 5, 1996, pp. 1286-1290.
35. Cichocki, A., and Unbehauen, R. "Simplified Neural Networks for Solving Linear Least Squares and Total Least Squares Problems in Real Time." IEEE Transactions on Neural Networks, Volume 5, Number 6, 1994, pp. 910-923.
36. Oja, Erkki, and Wang, Liuyue, "Robust Fitting by Nonlinear Neural Units." Neural Networks, Volume 9, Number 3, 1996, pp. 435-444.
37. Ishikawa, Masumi, "Structural Learning with Forgetting." Neural Networks, Volume 9, Number 3, 1996, pp. 509-521.
38. Mielniczuk, Jan, and Tyrcha, Joanna, "Consistency of Multilayer Perceptron Regression Estimators." Neural Networks, Volume 6, Number 7, 1993, pp. 1019-1022.

39. Lee, Youngjik, Oh, Sang-Hoon, and Kim, Myung Won, "An Analysis of Premature Saturation in Back Propagation Learning." Neural Networks, Volume 6, Number 5, 1993, pp. 719-728.
40. Tang, Zaiyong, and Koehler, Gary J. "Deterministic Global Optimal FNN Training Algorithms." Neural Networks, Volume 7, Number 2, 1994, pp. 301-311.
41. Thomas, Danny S. and Mitiche, Amar, "Asymptotic Optimality of Pattern Recognition by Regression Analysis." Neural Networks, Volume 7, Number 2, 1994, pp.313-320.
42. Smalz, Robert, and Conrad, Michael, "Combining Evolution With Credit Apportionment: A New Learning Algorithm for Neural Nets." Neural Networks, Volume 7, Number 2, 1994, pp. 341-351.
43. Refenes, Apostolos Nicholas, Zapranis, Achileas, and Francis, Gavin, "Stock Performance Modeling Using Neural Networks: A Comparative Study With Regression Models." Neural Networks, Volume 7, Number 2, 1994, pp. 375-388.
44. DeMichiel, Linda G. "Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains." IEEE Transactions on Knowledge and Data Engineering, Volume 1, Number 4, 1989, pp. 485-493.
45. Agrawal, Rakesh, Imielinski, Tomasz, and Swami, Arun, "Database Mining: A Performance Perspective." IEEE Transactions on Knowledge and Data Engineering, Volume 5, Number 6, 1993, pp.914-925.
46. Dhar, Vasant, and Tuzhilin, Alexander, "Abstract-Driven Pattern Discovery in Databases." IEEE Transactions on Knowledge and Data Engineering, Volume 5, Number 6, 1993, pp. 926-938.
47. Verveer, Peter J., and Duin, Robert P. W. "An Evaluation of Intrinsic Dimensionality Estimators." IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 17, Number 1, 1995, pp. 81-86.

48. Ney, Hermann, "On the Probabilistic Interpretation of Neural Network Classifiers and Discriminative Training Criteria." IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 17, Number 2, 1995, pp. 107-119.
49. Roddick, John F., Craske, Noel G., and Richards, Thomas J. "Handling Discovered Structure in Database Systems." IEEE Transactions on Knowledge and Data Engineering, Volume 8, Number 2, 1996, pp.227-240.
50. Looney, Carl G. "Advances in Feedforward Neural Networks: Demystifying Knowledge Acquiring Black Boxes." IEEE Transactions on Knowledge and Data Engineering, Volume 8, Number 2, 1996, pp. 211-226.
51. Han, Jiawei, Huang, Yue, Cercone, Nick, and Fu, Yongjian, "Intelligent Query Answering by Knowledge Discovery Techniques." IEEE Transactions on Knowledge and Data Engineering, Volume 8, Number 3, 1995, pp. 373-389.
52. Vai, Mankuan, and Xu, Zhimin, "Representing Knowledge by Neural Networks for Qualitative Analysis and Reasoning." IEEE Transactions on Knowledge and Data Engineering, Volume 7, Number 5, 1995, 683-690.
53. Parsons, Simon, "Current Approaches to Handling Imperfect Information in Data and Knowledge Bases." IEEE Transactions on Knowledge and Data Engineering, Volume 8, Number 3, 1996, pp. 353-372.
54. Lim, Ee-Peng, Srivastava, Jaideep, and Shekhar, Shashi, "An Evidential Reasoning Approach to Attribute Value Conflict Resolution in Database Integration." IEEE Transactions on Knowledge and Data Engineering, Volume 8, Number 5, 1996, pp. 707-723.
55. Fayyad, Usama, Piatetsky-Shapiro, Gregory, and Smyth, Padhraic, "From Data Mining to Knowledge Discovery in Databases." AI Magazine, Volume 17, Number 3, 1996, pp. 37-54.
56. Stedman, Craig, "Going 'Virtual' Eases Pain of Managing Multiple Data Marts." Computerworld, August 4, 1997, pp. 8.
57. Cornell, Gary, and Horstmann, Cay S. Core Java. New Jersey: SunSoft Press/Prentice-Hall, 1996.

Vita

Bryan Keith Ary was born in Centerville, Tennessee on November 14, 1969. He graduated from Perry County High School in 1988. He received an Associate of Science degree in Computer Information Systems from Columbia State Community College in 1990. He received the Bachelor of Science degree in Computer Science from Middle Tennessee State University in 1992. In 1993, he entered into graduate studies at the University of Tennessee Space Institute and received the Master of Science degree in Computer Science from The University of Tennessee in 1997. He is presently employed as a database administrator at ADT Automotive, Inc., in Nashville, Tennessee.