FUTURE HARDWARE REALIZATION OF SELF-ORGANIZING LEARNING ARRAY AND ITS SOFTWARE SIMULATION

A Thesis Presented to

The Faculty of the

Fritz J. and Dolores H. Russ

College of Engineering Technology

Ohio University

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

By

Tsun-Ho Liu

November, 2002

THIS THESIS ENTITLED

"FUTURE HARDWARE REALIZATION OF SELF-ORGANIZING LEARNING ARRAY AND ITS SOFTWARE SIMULATION"

by Tsun-Ho Liu

has been approved

for the School of Electrical Engineering and Computer Science

and the Russ College of Engineering and Technology

Janusz Starzyk, Professor

School of Electrical Engineering and Computer Science

Dennis Irwin, Dean

Fritz J. and Dolores H. Russ

College of Engineering and Technology

ACKNOWLEDGEMENT

First, I like to thank my parents who have been standing behind me through these years. This work will never be completed without their support.

I like to thank many of my friends. A special thank you to Zhen Zhu and Mingwei Ding for their valuable ideas and suggestions to my works. Thank you to Ivan Chang who helped us in developing the software code.

Finally, I think my advisor, Dr. Janusz. A. Starzyk, who proposed this self-organizing learning structure and gave me the opportunity to develop it. During this time, not only was he an advisor, but a friend. Thank you for teaching me how to play tennis.

CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

Chapter 1.
1. Introduction

In the recent computational history, problems have been analyzed and solved by powerful computational machines in order to achieve a good result in a short time interval. Although the performance of digital processors double yearly, solving more complex problems may still require more powerful machines and more complex software. In addition, our daily problems are usually presented by a relationship that is not well defined. Therefore, biologically inspired networks, which do not require software to operate, have been introduced.

One type of these networks is called the Artificial Neural Networks. Unlike the digital computer that is extremely effective at producing accurate answers to well-defined problems, the artificial neural network, which is modeled after the structure of the human brain, splits an ill-defined problem into many small pieces allowing each neuron in the network to solve its own task and gives an approximate output. It may perform better than other methods, especially in categorization and pattern recognition. Such a system classifies an object and processes it as one of the possible categories, which may result in a recommendation of an action. The processing speed of each neuron is not the main factor in the network since effective performance can

be achieved with parallel processing for real-time applications. This is an advantage with which no software based learning algorithms can compete, especially with a large dimensional training dataset. Due to their good performance, artificial neural networks have been adopted in many practical applications such as credit card approvals, potential customer analysis and pattern recognition. However, many existing artificial neural network designs require a huge number of connections between inputs, neurons, and outputs (Dayhoff, 1990, p. 3). It causes the area consumed by interconnections to be far greater than that of the processing units. This can result in an expensive hardware implementation, and ineffective Very Large Scale Integration (VLSI) circuit design. Thus, new artificial neural network design with less interconnections and better organization can result in a more effective network to solve complicated problems.

1.1 Research Objective

This thesis focuses on Future Hardware Realization of Self-Organizing Learning Array (SOLAR) and its Software Simulation. The structure of this network is similar to programmable arrays such as Field Programmable Gate Arrays (FPGA). The basic fabric of SOLAR is a fixed lattice of processing units acting as single neurons with programmable interconnections between them. In this thesis, a software version of SOLAR architecture is considered with the lattice size based on the number of inputs. The network is designed for nonspecific classification and for future hardware

realization. In SOLAR, a set of preprocessed training data, which well represents the learning space, is given to the network for learning. The network can then determine and self-organize the interconnections between inputs and outputs during the learning process. Each neuron can also select the best transformation function and threshold value for later classification using information index. After learning, SOLAR is prepared for any classification within the learning space.

1.2 Thesis Organization

The thesis is structured as follows:

Chapter 2 gives an overview of the biological neural networks. Architecture, functionalities, and organization of living neurons are discussed.

Chapter 3 discusses the organization of Self-Organizing Learning Array (SOLAR). Different artificial neural network organizations are described in this chapter. SOLAR organization and wiring concept are explained.

Chapter 4 explains the inputs and outputs of neurons. The basic structure of SOLAR neuron is shown, and details of inputs and outputs of SOLAR are discussed. Data pre-processing methods for missing data and symbolic values are also introduced.

Chapter 5 deals with the arithmetic operations. Possible operations are listed and demonstrated graphically.

Chapter 6 shows the self-organization principles. In this chapter, SOLAR self-organization and learning principles are demonstrated both verbally and graphically. In addition, information index calculation is explained in this chapter.

Chapter 7 shows final classification and voting. Weight function is introduced for each neuron participating in the final voting.

Chapter 8 demonstrates the software simulation. In this chapter, a two-dimensional sample data is used to illustrate the performance of SOLAR. Besides, two real world problems are used to compare the SOLAR performance to other algorithms.

Chapter 9 is the conclusion of this thesis. It concludes the SOLAR software project and gives the prospects for future works.

Chapter 2.
2. Overview of the Biological Neural Networks

In order to create an artificial neural network, one must study the structure and behavior of living neurons. Glia and neuron are the basic elements of the brain. Glias are supporting cells while the decision making processes are done within neurons. Jobs of neurons are receiving, integrating, and transmitting information. Organization of neurons is not homogeneous. Neurons located in different regions of the brain are shaped differently due to their responsibility and functions. However, each of them has the same basic elements.

2.1 Living Neuron Structure and Function

The structure of a neuron, shown in Figure 2-1, includes three basic elements: a nucleus, dendrites, and an axon. Dendrites act as receivers of a neuron while an axon acts as the transmitter. When a neuron communicates with another neuron, chemicals are fired from the axon terminals. These chemicals travel through a small synaptic gap and arrive at receptor sites of the dendrites. Dendrites are excited by the chemicals, and the potential of positively charged ions increases. When the potential

exceeds the threshold, the neuron fires to thousands of other neurons. This process is repeated throughout the network.

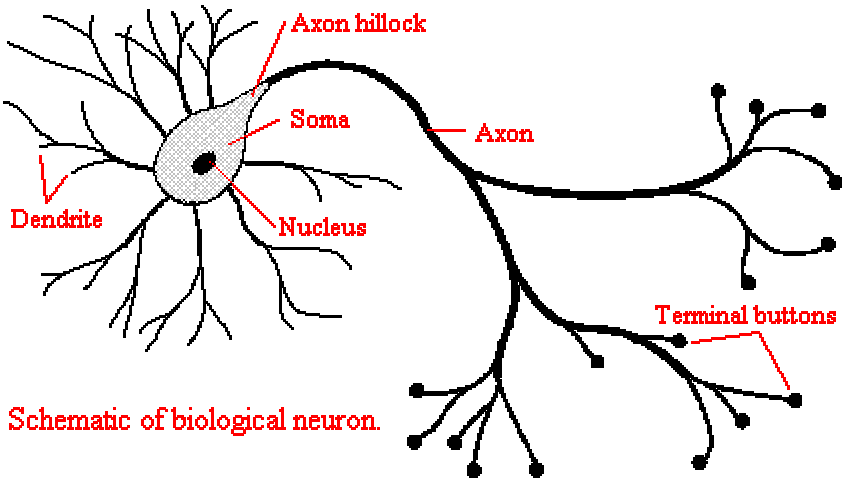

Figure 2-1 Structure of a Biological Neuron
(Fraser, 1998, September)

2.2 Biological Neuron Organization

Neurons, in general, can be classified into two categories: long-axon cells and short-axon cells (Dowling, 1998, p. 15). Long-axon cells are responsible for carrying information from one side of the brain to another. They have long axons and tend to communicate with neurons further away. Short-axon cells, on the other hand, are interconnected only with local neurons. They are mainly involved in integrating and processing information. In some regions of the brain, neurons function in continuous layers rather than in a random network. This layer organization is illustrated in Figure

2-2. In general, short-axon neurons tend to interact with neighboring neurons locally while long-axon neurons pass information from one local neighborhood to another.

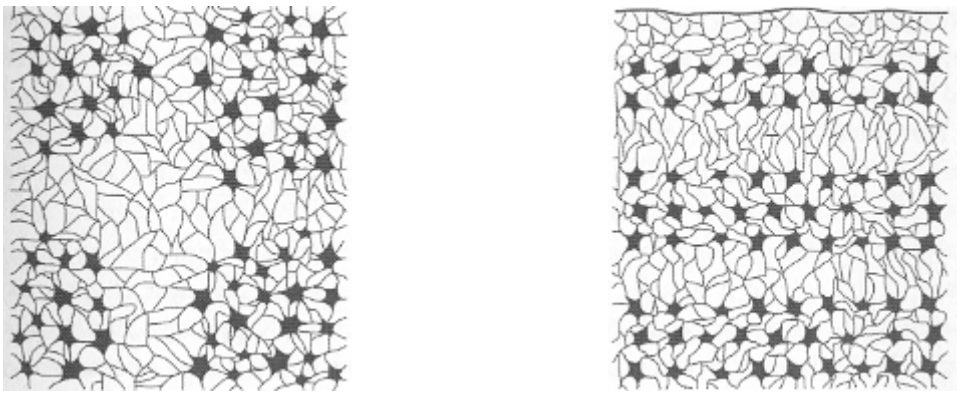

Figure 2-2 Neurons Organization in Groups and Layers
(Dowling, 1998, p. 17)

In addition, some neurons in some parts of the brain can grow quicker than other parts. When a person grows up, during learning, new branches are expanded from one neuron to another to form new connections. However, it is suggested that initial connections may be broken during development (Dowling, 1998, p. 133). Most synapses move away from less useful neurons (Purves, 1994, p. 60). These are the reasons neurons can self-organize to achieve a better performance.

The Self-Organizing Learning Array discussed in this thesis uses these distinct features of biological neural networks: local pseudo-random interconnections, selection of control and input signals by each neuron and parallel processing in

training and recognition. This work is based on the self-organizing learning array project developed at the Ohio University by Dr. Starzyk and partially described at http://www.ent.ohiou.edu/%7Ewebcad/proj/solar/index.html (Starzyk, 2000)

Chapter 3.
3. Structure of Self-Organizing Learning Array

SOLAR is simply an electronic model based on the biological brain structure. It has the capability to solve and analyze problems, such as pattern recognition and classification, which may be impossible for traditional digital computers. SOLAR includes three main components which are the inputs, process layers, and outputs. Just like the biological brain that can solve and analyze more complicated problems after years of learning, SOLAR requires learning before it can be put to any test. Before learning can take place, initial wiring is required. This wiring will be modified during the learning process.

Biological neural networks are constructed in three-dimensions from microscopic components. Billions of neuron interconnections can be broken or developed to achieve a better performance. This is not true for artificial neural networks built with integrated circuits on silicon. Artificial neural networks are limited to a two-dimensional plane whose hard-wiring interconnections cannot be replaced or changed once they are constructed. In addition to the inert, space on an integrated circuit is

limited. Artificial neurons can only learn within these limited conditions. Therefore, neurons organization and initial wiring are extremely important for a good performance.

3.1 Neural Network Organization

Most of the existing artificial neural network organizations can be classified into three main categories: cellular neural network (CNN), feed backward neural network (FBNN) and feed forward neural network (FFNN) (Cichocki & Unbehauen, 1993, p. 65).

CNN includes many identical cells (or processing units) that have local interconnections among each other, and only the nearest neighbors are connected. The neighboring cells can interact directly with each other, while other cells, not directly connected together, may still be affected indirectly due to the propagation effects from the dynamics of CNN. Applications of CNN are mainly in image processing, where they show a great performance in solving many complex image-processing tasks that cannot be solved using conventional approaches.

In FBNN organization, neurons are generated in parallel. Output of each neuron in the network can be connected backward or forward as inputs to other neurons. Networks having FBNN organization may become unstable if a positive feedback causes the

increase in the input signal values. One must ensure that the mean of the output of any neuron in the network must be less or equal to the mean of all the inputs.

In FFNN organization, on the other hand, structure of the network is constructed under the condition that all inputs of each neuron are connected to the input layer or the existing neuron outputs. Neurons are generated in parallel. The size of the network increases when new neurons are added. As a result, growing number of outputs from existing neurons can be used as inputs for new neurons generated at later stages. Outputs of subsequently generated neurons cannot be used as inputs for those neurons already generated in feed forward organized neuron network, so that the network is always stable. Therefore, feed forward organization is chosen for SOLAR.

3.2 Initial Wiring

Since this is a software simulation version of SOLAR, it can take any number of inputs. Based on the size of input, numbers of neurons are generated. It is assumed that a fixed number of neurons are added per each layer. Each neuron is then identified by its location, row and column as shown in Figure 3-1. Although this organization is not a requirement of SOLAR, it is better suited for VLSI design of SOLAR, where neurons are organized in a regular array to best utilize the available silicon area.

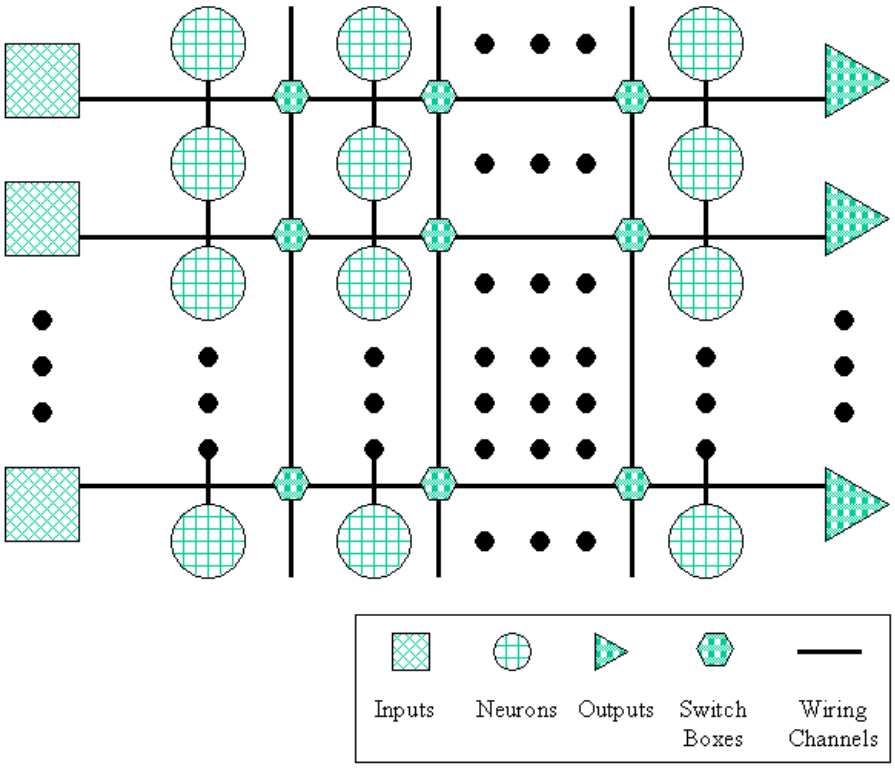

Figure 3-1 Basic SOLAR Structure

It is believed that a large number of biological neurons, which are responsible for processing information, tend to have local connections. There is a higher probability that one neuron should have connections to close neighboring neurons. Therefore, statistically determined Mahalanobis distance (Mahalanobis, 1936) is introduced in the determination of the initial wiring. With the feed-forward structure, new neurons are connected only to the previously generated neurons. As shown in Figure 3-2, the neuron located at a given row and column should always be connected to the one at the same row and the previous column. The next nearest neurons are those located at

two neurons away from the connecting neuron and so on. Similar to the short-axon cells, although local connecting is highly preferred, some random and further connections may also be allowed with smaller probability in the pre-wiring stage. This pseudorandom wiring organization applies to both the neuron's input signals as well as the neuron's control (input clock), which come from logical output (output clock) of other neurons.

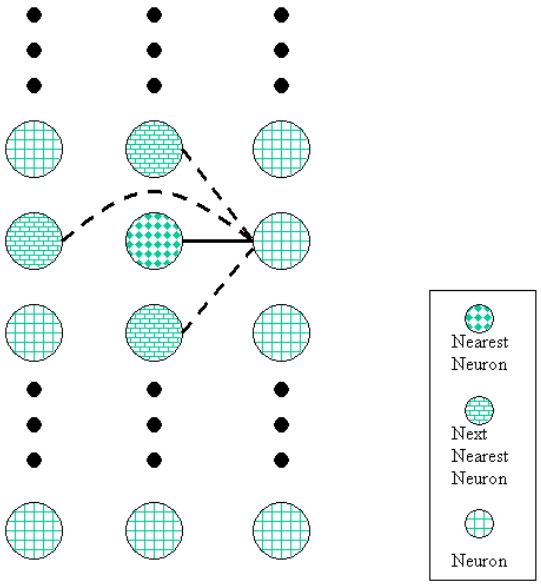

Figure 3-2 Example of Neurons' Initial Wiring

Chapter 4.
4. Neurons' Inputs and Outputs

The process layers are where the processing neurons are located. Preprocessed data is sent to different neurons through pre-wired interconnections. Self-organizing neural network is very similar to living neurons in terms of architecture. A neuron has many parallel inputs but only a single output, which may feed many other neurons as their inputs as shown in Figure 4-1.

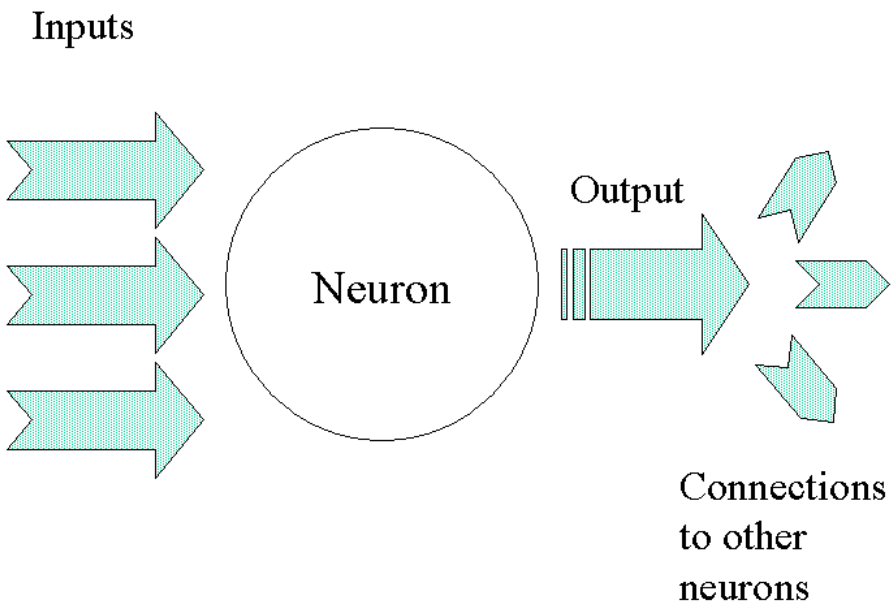

Figure 4-1 Neuron's Input and Output Signals

4.1 Basic Operation of a Neuron

Figure 4-2 shows all necessary inputs and outputs of a neuron. All input data are rescaled so that they are always within a specified range (for instance 0-255 for 8-bit digital hardware representation). Each neuron is able to select any inputs (or a single input) and perform different transformation operations. During the learning process, neurons learn in parallel one layer at a time. A neuron calculates the information index and selects a threshold for a combination of inputs (or a single input), a transformation function, and an input clock. Information index indicates the quality of learning, and it is discussed in detail in Chapter 6 Self-Organization Principles.

With the highest information index, the input combination, transformation function and threshold are stored and fixed in the neuron for later use during testing. The values of the output information deficiency indicate how much the selected subspace has been learned, and this is also described in details in Chapter 6 Self-Organization Principles. These output information deficiency values are saved. Calculated output (system output) is passed to other neurons as inputs. Output clocks are also generated and passed to other connected neurons with their output information deficiency.

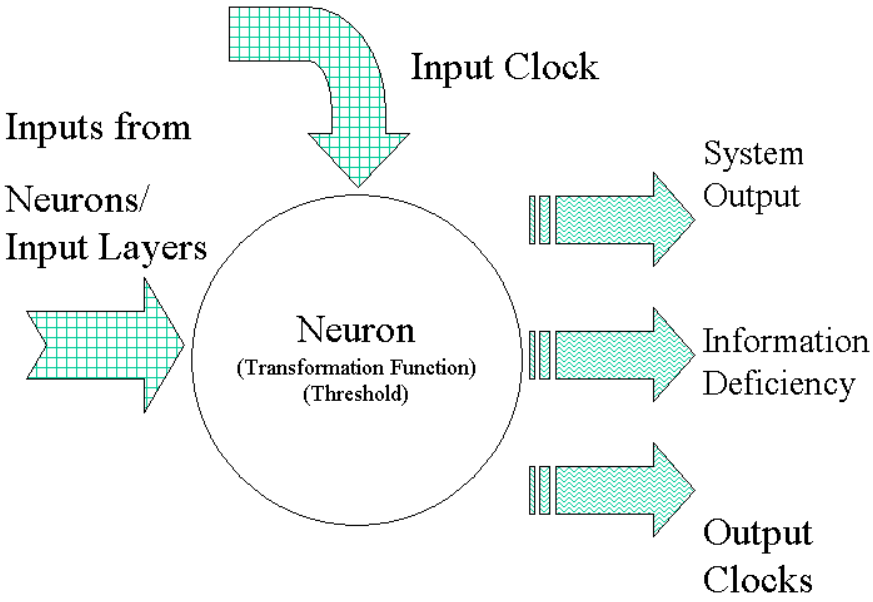

Figure 4-2 Neuron Inputs and Outputs

4.2 Neurons' Clock Inputs and Outputs

Similar to all sequential machines, every neuron has an input clock control. At each clock cycle, an input data is expected to arrive from the previous neurons or initial inputs. There is one more clock input in SOLAR called threshold-control-input (TCI). This particular clock is designed to control neuron operation, and it is obtained by multiplexing the output clocks. A neuron tries to learn more about a particular space based on the selected TCI. There are three types of output clocks per neuron, and they are listed as follows:

1. Threshold-control-output (TCO) is the original TCI of this neuron.

2. Threshold-control-output-thresholded (TCOT) is the original TCI multiplied with the logic value indication that the transformed data points passes the threshold. It can be done with an AND operation as shown in Figure 4-3.

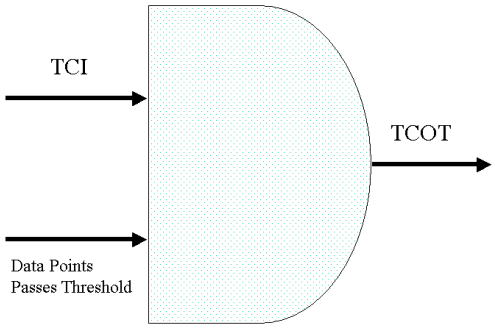

Figure 4-3 TCOT

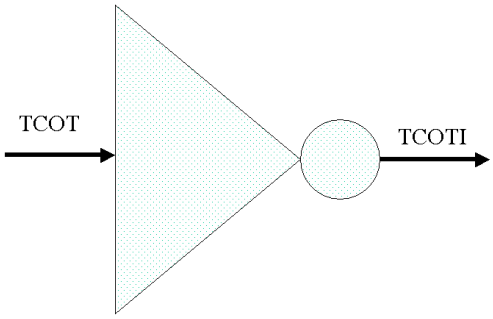

Figure 4-4 TCOTI

3. Threshold-control-output-thresholded-inverted (TCOTI) is the original TCI multiplied with the logic value indication that the transformed data points do not pass the threshold. It can be implemented with an inverse operation of TCOT as shown in Figure 4-4.

There can be more than one TCI for a neuron. One always connects to the output clock from the closest neuron (the one located at the same row and previous column) since local interconnection is believed to result in a better performance. Other TCIs can be selected with lower probability from other previously generated neurons. These n TCIs can be selected by a n-to-1 muliplexer by each neuron.

4.3 Neurons' Signal Inputs

Each neuron has more than one input pre-wired from the initial inputs or from outputs of previous neurons. One or two of these will be selected to perform arithmetic operation and to produce a system output that may be used as an input by other neurons. Similar to TCI, these n inputs can be selected by a n-to-1 or a n-to-2 muliplexers for each neuron based on the transformation function, its threshold value, and a selected information index. In general, neuron inputs are normalized to provide sufficient resolution for input-output functions.

4.3.1 Neurons' Input Data

Input data is presented to SOLAR as n dimensional feature vectors. Each feature represents one dimension of the whole input space. At each clock cycle, one set of n dimensional data is buffered to the input layer. The j^{th} input data with n features can be represented by a vector: $X^j=[x^j_1, x^j_2 \ldots x^j_n]$. As a result, the whole set of input data can be described by a matrix in (4-1).

$$\overline{X} = \begin{bmatrix} X^1 \\ ... \\ X^t \end{bmatrix} = \begin{bmatrix} x^1_1 x^1_2 ... x^1_n \\ .... \\ x^t_1 . x^t_2 ... x^t_n \end{bmatrix} \qquad \textbf{(4-1)}$$

where t is the length of the whole input data set.

The problem of databases containing missing data and symbolic values is very common. These incomplete data can cause problems for neuron operations that only take numerical inputs. Therefore, blanks and symbols must be replaced by meaningful numerical values with some transformations, which are discussed in 4.3.1.1 and 4.3.1.2.

4.3.1.1 Missing Data

There are three approaches to the missing values datasets (Ennett, Frize & Walker, 2001). First approach is to simply delete all cases with missing values. This can result in losing potential important information of the cases. Second approach is to find the mean value. Although it is easy to calculate, it can bias the dataset. The last approach is to replace missing data with statistically unbiased estimates that can improve the network performance. Mahalanobis distance is used in the procedure developed to normalize the missing values.

To define the Mahalanobis distance, mean value vector for a given class μ_c and covariance matrix for all training data from this class C_c are needed. Then, a given vector of training data $X \in \overline{X}$ with missing coordinates is represented as $X=[X_k, X_m]$ where X_k are known coordinates while the missing values X_m are

$$\mathrm{X}_m = \left\{ \tilde{\mathrm{X}}_m : d\left(\tilde{\mathrm{X}}_m \right) = d_{\min} \right\} \qquad \textbf{(4-2)}$$

where $d(\mathrm{X}) = (\mathrm{X} - \mu_c) C_c^{-1} (\mathrm{X} - \mu_c)^{\mathrm{T}}$

Since d(X) is a quadratic form of the unknown values X_m, the minimum can be obtained by setting its derivative to zero.

$$\left. \frac{\partial d(\mathrm{X})}{\partial \mathrm{X}_m} \right|_{\mathrm{X}_m = \tilde{\mathrm{X}}_m} = 0 \qquad \textbf{(4-3)}$$

The inverse of the covariance matrix C_c is divided according to partition of X into known and missing values parts.

$$\mathrm{C_c}^{-1} = \mathrm{D_c} = \begin{bmatrix} \mathrm{D_{kk}}, \mathrm{D_{km}} \\ \mathrm{D_{mk}}, \mathrm{D_{mm}} \end{bmatrix} \qquad \textbf{(4-4)}$$

Since C_c is symmetrical $\mathrm{D_{mk}} = \mathrm{D^T_{km}}$ and

$$\left[\frac{\partial \mathrm{d}}{\partial \mathrm{X}_1} \cdots \frac{\partial \mathrm{d}}{\partial \mathrm{X}_\mathrm{n}} \right] = 2\{ [\mathrm{X}_k \quad \mathrm{X}_m] - \mu_c \} \begin{bmatrix} \mathrm{D_{kk}}, \mathrm{D_{km}} \\ \mathrm{D_{mk}}, \mathrm{D_{mm}} \end{bmatrix} = 0 \qquad \textbf{(4-5)}$$

As a result vector X_m can be obtained from

$$X_m = -(X_k - \mu_{ck})D_{km}D^{-1}{}_{mm} + \mu_{cm} \quad \textbf{(4-6)}$$

where μ_{ck} is the part in μ_c corresponding to X_k while μ_{cm} represents the part corresponding to X_m.

If the matrix $Y = \overline{X} - \mu_c$ does not have the full column rank, Y can be first factorized using QR factorization.

$$YE = Q_Y R_Y = Q_Y \begin{bmatrix} R_{Y1} & R_{Y2} \\ 0 & 0 \end{bmatrix} \quad \textbf{(4-7)}$$

where E is the permutation matrix, and R_{Y1} is upper triangular. Therefore, Y can be represented as

$$Y = [Y_1 \quad Y_2] = Q_{\tilde{Y}}[R_{Y1} \quad R_{Y2}] = Q_{\tilde{Y}} R_{Y1}[I \quad R_{Y1}^{-1} \quad R_{Y2}] \quad \textbf{(4-8)}$$

where $Q_{\tilde{Y}}$ contains columns of Q_Y which are multiplied by R_{Y1} in (4-7)

Y_2 can be expressed as a linear combination of Y_1

$$Y_2 = Y_1(R_{Y1}^{-1} \quad R_{Y2}) = Y_1 C_r \quad \textbf{(4-9)}$$

Only elements of the matrix Y_1 need to be determined for the missing data. The reduced covariance matrix, which is based on the matrix Y_1 only, is defined as

$$C_R = \perp_k \mathrm{Y}_{1k}^{\mathrm{T}} \mathrm{Y}_{1k} \tag{4-10}$$

where $\mathrm{Y}_1 = \begin{bmatrix} \mathrm{Y}_{1k} \\ \mathrm{Y}_{1m} \end{bmatrix}$ and $\mathrm{Y} = \begin{bmatrix} \mathrm{Y}_{1k} & \mathrm{Y}_{2k} \\ \mathrm{Y}_{1m} & \mathrm{Y}_{2m} \end{bmatrix}$

The reduced Mahalanobis distance for vector X is

$$d(\mathrm{X}) = (\mathrm{X}_1 - \mu_{c1}) C_R^{-1} (\mathrm{X}_1 - \mu_{c1})^{\mathrm{T}} \tag{4-11}$$

where $X = [X_1\ X_2]$ and $m_c = [\mu_{c1}\ \mu_{c2}]$ according to partition of matrix $Y = [Y_1\ Y_2]$

Following derivation (4-3) to (4-6), missing data result can be generated as follows

$$\mathrm{X}_{\mathrm{m1}} = -(\mathrm{X}_{\mathrm{k1}} - \mu_{ck1}) \mathrm{D}_{\mathrm{kmR}} \mathrm{D}^{-1}{}_{\mathrm{mmR}} + \mu_{cm1} \tag{4-12}$$

where $\mathrm{C}_{\mathrm{R}}{}^{-1} = \begin{bmatrix} \mathrm{D}_{\mathrm{kkR}}, \mathrm{D}_{\mathrm{kmR}} \\ \mathrm{D}_{\mathrm{mkR}}, \mathrm{D}_{\mathrm{mmR}} \end{bmatrix}$

After missing data is recovered, the result can be obtained and pasted back to vector X_1.

$$X_1 = [X_{k1}\ X_{m1}] \quad \textbf{(4-13)}$$

After vector X_1 is obtained, X_2 can be determined from

$$X_2 = X_1 C_1 = X_1 \left(R_{Y1}^{-1} \quad R_{Y2} \right) \quad \textbf{(4-14)}$$

Thus, all missing data (independent and dependent) are recovered. This operation can be repeated sequentially for each data vector X with missing data or performed concurrently on all vectors with some missing data. However, separation of matrix $\overline{X}$ into missing and known values may be difficult or impossible.

4.3.1.1.1 Illustration of Missing Data Recovery

The following two examples are used to show the missing data recovery by applying Mahalanobis distance. The dataset (Abalone Database) of the first example was obtained from the University of California at Irvine (ICS, UCI, 1995, December). This dataset contains information from 4177 input data with 29 classes. In order to demonstrate the performance in a 2 dimensional plane, only two features, which are "height" and "weight", are used.

In this example, class 9 is chosen. There are 689 data points that belong to this class. Ten of them in feature “weight” have been randomly chosen, removed and become the missing data. Then, these data were recovered by the missing data recovery algorithm using the Mahalanobis distance. As the results shown in Table 4-1 and Figure 4-5, the recovered missing data are replaced with reasonable numerical values based on the distribution of this particular class.

Table 4-1 Original and Recovered Data Comparison

Original Weight	Recovered Weight
0.2800	0.4818
1.5100	1.4995
1.2945	1.1037
0.6995	0.7079
0.6880	0.8776
1.1000	0.9907
0.5780	0.9907
0.9070	0.7645
0.9615	0.8210
1.2960	0.9341

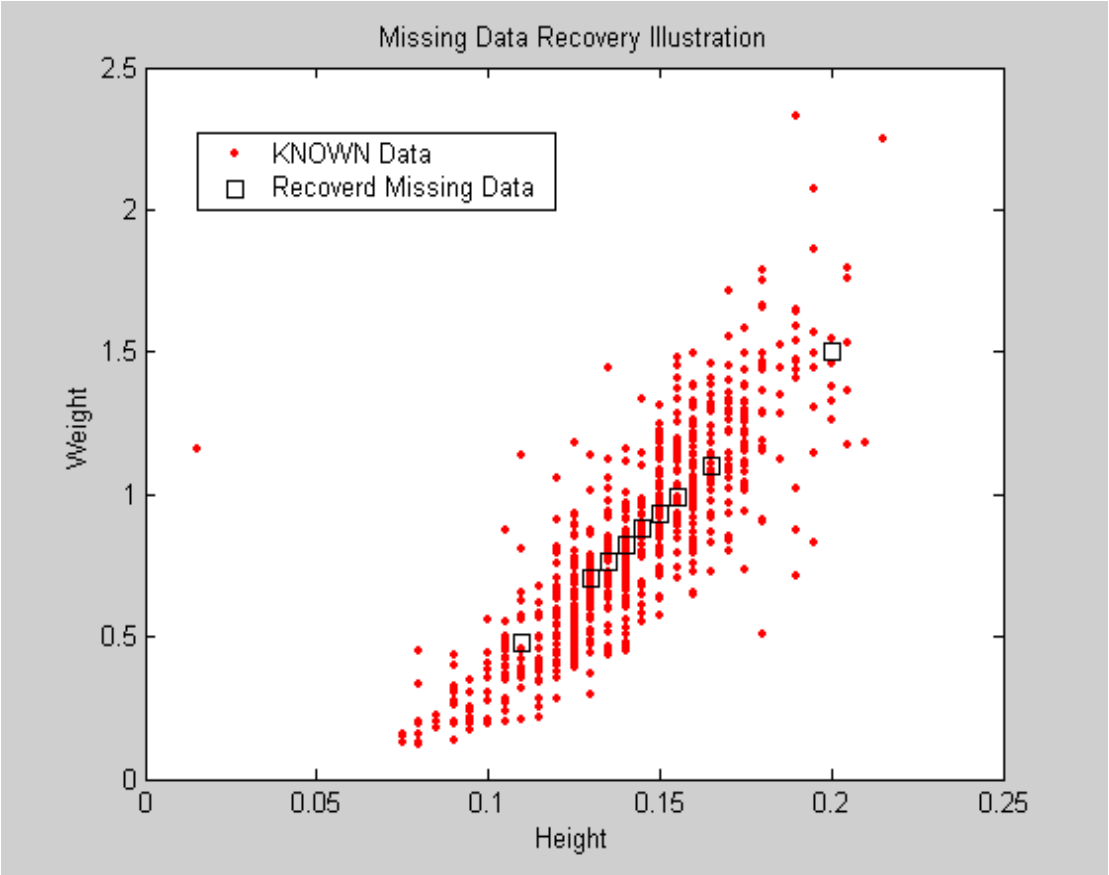

Figure 4-5 Missing Data Recovery Illustration

Sometimes, a dataset contains dependent information. This dependent features have to be identified and taken out when recovering the missing data because error can occur when the inverse covariance matrix is calculated during the process of missing data recovery. Dependent features can be calculated after the independent part of the dataset is recovered. The following example illustrates how singular matrix is handled. An input matrix with 2 classes is shown in Table 4-2, which contains 21 data points with 11 features. Two of the features in the input matrix are dependent, and the relationship is:

$$\text{Row } 1 = \text{Row } 2 * 1.03 + \text{Row } 4 * 1.02$$

$$\text{Row } 8 = \text{Row } 11 * 1.10 - \text{Row } 5 * 1.05$$

Table 4-2 Singular Input Matrix with Missing Data

#1	#2	#3	#4	#5	#6	#7
Class 1	Class 2	Class 2	Class 1	Class 1	Class 2	Class 1
579.2	?	5699.7	577.0	534.4	5708.9	538.5
184.0	?	1827.0	177.0	194.0	1727.0	193.0
294.0	2749.0	2843.0	255.0	223.0	2213.0	283.0
382.0	?	3743.0	387.0	328.0	3853.0	333.0
494.0	4360.0	4321.0	439.0	485.0	4996.0	442.0
578.0	5700.0	5495.0	523.0	596.0	5323.0	510.0
623.0	6210.0	6723.0	663.0	688.0	6232.0	693.0
518.6	5682.8	5718.3	564.3	494.0	4788.4	574.3
743.0	7410.0	7239.0	732.0	723.0	7221.0	734.0
842.0	8318.0	8372.0	898.0	839.0	8843.0	833.0
943.0	9328.0	9323.0	932.0	912.0	9122.0	944.0

#8	#9	#10	#11	#12	#13	#14
Class 1	Class 1	Class 2	Class 2	Class 2	Class 1	Class 1
431.4	?	5905.6	4636.7	4555.4	516.3	523.8
101.9	?	1923.0	1101.0	1231.0	121.0	154.0
240.0	291.0	2938.0	2302.0	2943.0	254.0	298.0
320.0	380.0	3848.0	3434.0	3223.0	384.0	358.0
?	490.0	4858.0	4324.0	4211.0	432.0	475.0
530.0	580.0	5959.0	5483.0	5321.0	549.0	552.0
639.0	619.0	6835.0	6859.0	6948.0	684.0	671.0
?	518.4	5263.3	5891.1	6101.0	583.7	593.6
853.0	?	7122.0	7473.0	7484.0	723.0	718.0
821.0	840.0	8235.0	8243.0	8873.0	824.0	892.0
?	939.0	9422.0	9483.0	9566.0	943.0	993.0

#15	#16	#17	#18	#19	#20	#21
Class 1	Class 1	Class 1	Class 2	Class 2	Class 1	Class 1
466.2	513.2	532.1	5865.5	5724.2	433.8	582.2
112.0	111.0	172.0	1983.0	1837.0	101.3	183.0
238.0	219.0	232.0	2837.0	2744.0	243.0	254.0
344.0	391.0	348.0	3748.0	3757.0	323.0	386.0
483.0	438.0	495.0	4983.0	4372.0	443.0	493.0
594.0	512.0	538.0	5848.0	5748.0	535.0	573.0
673.0	611.0	695.0	6382.0	6223.0	646.0	654.0
541.2	567.5	568.2	4912.0	56746	550.2	497.7
732.0	739.0	724.0	7223.0	7434.0	754.0	786.0
844.0	800.0	832.0	8332.0	8321.0	823.0	834.0
953.0	934.0	989.0	9222.0	9332.0	923.0	923.0

After missing data of the independent feature is recovered with equation (4-12), dependent part of the missing data can be generated by equation (4-14). Results are demonstrated in Table 4-3. Recovered data seems reasonable and fits in the whole matrix.

Table 4-3 Result of a Singular Input Matrix with Missing Data

#2	#8	#9
Class 2	Class 1	Class 1
5772.1	431.4	555.3
1919.9	101.9	162.2
2749.0	240.0	291.0
3720.2	320.0	380.6
4360.0	454.7	490.0
5700.0	530.0	580.0
6210.0	639.0	619.0
4447.2	491.7	499.8
7410.0	853.0	757.3
8318.0	821.0	840.0
9328.0	881.1	939.0

4.3.1.2 Symbolic Values

If the input matrix $\overline{X}$ contains symbolic (non-numerical) data, this data can be assigned a numerical value so that they are best correlated to the existing data. This can be accomplished with minimization of the determinant of the resulting covariance matrix.

$$\overline{X} = \left[X_r \quad \tilde{X}_s \right]_{t \times n} \tag{4-15}$$

where $\tilde{X}_s$ is a sub-matrix or a vector with all symbolic values

X_r is a sub-matrix or a vector with all numerical values

t is the number of samples

n is the number of features

In order to minimize $\det[Cov(\overline{X})]$, value X_s can be selected to minimize the rank of $\overline{X}$. First, one should consider a single symbolic vector $\tilde{X}_s$ to which numerical values should be assigned so that the numerical vector X_s is a linear combination of vectors X_r.

$$X_s = X_r * \alpha \quad , \quad X_s \in \tilde{\tilde{X}}_s \tag{4-16}$$

where α is a nonzero linear combination vector.

Since this problem may not have an exact solution, the norm of error vector E is minimized, where

$$E = \mathrm{X}_s - \mathrm{X}_r * \alpha \tag{4-17}$$

X_s can be replaced by the product of a binary matrix A and a vector of all symbols H.

$$\mathrm{X}_s = \mathrm{AH} \tag{4-18}$$

A final form of the error vector is obtained.

$$E = \mathrm{AH} - \mathrm{X}_r \alpha \tag{4-19}$$

Since the objective is to minimize the error (E=0), values of H can be obtained applying pseudo-inverse of A

$$\mathrm{H} = pinv(\mathrm{A})\mathrm{X}_r \alpha \tag{4-20}$$

This is a desired solution with α=1 if X_r has only a single column. If X_r has more than one column, H can be achieved by minimizing the norm of the error function and setting its derivatives to zero.

$$\left|E\right|^2 = E^{\mathrm{T}}E \geq 0 \tag{4-21}$$

$$\frac{\partial\left|\mathrm{E}\right|^2}{\partial \mathrm{H}} = \mathrm{A}^{\mathrm{T}}\left[\mathrm{AX}_r\right] \begin{bmatrix} \mathrm{H} \\ -\alpha \end{bmatrix} = 0 \tag{4-22}$$

$$\frac{\partial\left|\mathrm{E}\right|^2}{\partial \alpha} = \mathrm{X}^{\mathrm{T}}\left[\mathrm{AX}_r\right] \begin{bmatrix} \mathrm{H} \\ -\alpha \end{bmatrix} = 0 \tag{4-23}$$

Let us define matrix B as below and partition it into symbolic and numerical parts B_s and B_r.

$$\mathrm{B} = \begin{bmatrix} \mathrm{A}^{\mathrm{T}} \\ \mathrm{X}_r^{\mathrm{T}} \end{bmatrix} \left[\mathrm{AX}_r\right] = \begin{bmatrix} \mathrm{A}^{\mathrm{T}}\mathrm{A} & \mathrm{A}^{\mathrm{T}}\mathrm{X}_r \\ \mathrm{X}_r^{\mathrm{T}}\mathrm{A} & \mathrm{X}_r^{\mathrm{T}}\mathrm{X}_r \end{bmatrix} = \begin{bmatrix} \mathrm{B}_s & \mathrm{B}_r \end{bmatrix} \tag{4-24}$$

The minimum error norm is obtained by solving the following equation

$$\begin{bmatrix} B_s & B_r \end{bmatrix} \begin{bmatrix} \mathrm{H} \\ -\alpha \end{bmatrix} = 0 \tag{4-25}$$

B_r can be factorized by using QR factorization and its orthogonal matrix Q will be divided according to the rank of its upper triangular matrix R.

$$\mathrm{B}_r = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 & R_2 \\ 0 & 0 \end{bmatrix} \tag{4-26}$$

$$\begin{bmatrix}\begin{bmatrix}Q_1^T \mathrm{B}_s\\ Q_2^T \mathrm{B}_s\end{bmatrix} & \begin{bmatrix}R_1 & R_2\\ 0 & 0\end{bmatrix}\end{bmatrix} \begin{bmatrix}\mathrm{H}\\ -\alpha\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix} \quad \textbf{(4-27)}$$

Equation (4-25) will change to and can be separated to two equations (4-28) and (4-29).

$$\begin{cases} Q_1^{\mathrm{T}} \mathrm{B}_s \mathrm{H} + \begin{bmatrix}R_1 & R_2\end{bmatrix} \begin{bmatrix}\alpha_1\\ \alpha_2\end{bmatrix} = 0 & \textbf{(4-28)}\\ Q_2^{\mathrm{T}} \mathrm{B}_s \mathrm{H} = 0 & \textbf{(4-29)} \end{cases}$$

Values of H can be solved by using equation (4-29) since it does not depend on α. However, H is always zero if $Q^{\mathrm{T}}{}_2 B_s$ is a full rank matrix. A single variable in H has to be set, such as H_1=1

$$\begin{pmatrix} Q_2^{\mathrm{T}} & \mathrm{B}_s \end{pmatrix} \begin{bmatrix}\mathrm{H}_1\\ \mathrm{H}_{\tilde{s}}\end{bmatrix} = \begin{bmatrix}C_1 & C_{\tilde{s}}\end{bmatrix} \begin{bmatrix}1\\ \mathrm{H}_{\tilde{s}}\end{bmatrix} = 0 \quad \textbf{(4-30)}$$

where C_1 is the first column of $Q^{\mathrm{T}}{}_2\, B_s$

and $\mathrm{H}_{\tilde{s}}$ can be determined from

$$C_{\tilde{s}} \mathrm{H}_{\tilde{s}} = -C_1 \quad \textbf{(4-31)}$$

After applying pseudo-inverse of $C_{\tilde{s}}$, H can be solved as follows.

$$\mathrm{H} = \begin{bmatrix} 1 \\ -\,pinv(C_{\tilde{s}})C_1 \end{bmatrix} \tag{4-32}$$

Equation (4-32) requires that $C_{\tilde{s}}$ has full column rank. If it does not, it can be divided into independent and dependent parts as follows

$$C_{\tilde{s}} = \begin{bmatrix} C_{\tilde{s}1} & C_{\tilde{s}2} \end{bmatrix} = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \begin{bmatrix} \mathrm{I} & R_1^{-1} & R_2 \end{bmatrix} \tag{4-33}$$

where $C_{\tilde{s}1}$ and $C_{\tilde{s}2}$ can be determined using QR factorization of $C_{\tilde{s}}$.

$\mathrm{H}_{\tilde{s}}$ is partitioned accordingly to $\mathrm{H}_{\tilde{s}} = \begin{bmatrix} \mathrm{H}_{\tilde{s}1} & \mathrm{H}_{\tilde{s}2} \end{bmatrix}$. Instead of (4-31), the following equation will be solved.

$$C_{\tilde{s}1}\left(\mathrm{H}_{\tilde{s}1} + R_1^{-1}R_2\mathrm{H}_{\tilde{s}2}\right) = -C_1 \tag{4-34}$$

Instead of solving for parameters of H, a combined vector H_c can be solved as follows

$$\mathrm{H}_c = \mathrm{H}_{\tilde{s}1} + R_1^{-1}R_2\mathrm{H}_{\tilde{s}2} = -\,pinv(C_{\tilde{s}1})C_1 \tag{4-35}$$

Since $\mathrm{H}_{\tilde{s}1}$ and $\mathrm{H}_{\tilde{s}2}$ cannot be uniquely defined, one can either set $\mathrm{H}_{\tilde{s}2}$ to zero and $\mathrm{H}_{\tilde{s}1}$=$\mathrm{H_c}$ or introduce another constraint for elements of $\mathrm{H}_{\tilde{s}1}$ and $\mathrm{H}_{\tilde{s}2}$, for instance minimize the norm of H under constraint defined by (4-35). The constraint minimization problem can be formulated using a Lagrangian function. The objective function is:

$$F = \sum_{h_{\tilde{s}i} \in \mathrm{H}_{\tilde{s}}} h_{\tilde{s}i}^2 \qquad \textbf{(4-36)}$$

with constraints

$$e = \mathrm{H}_{\tilde{s}1} + D\mathrm{H}_{\tilde{s}2} + pinv\left(C_{\tilde{s}1}\right)C_1 = 0 \qquad \textbf{(4-37)}$$

where $D = R_1^{-1}R_2$

The Lagrangian function is defined as follows:

$$L\left(\mathrm{H}_{\tilde{s}}, \lambda\right) = F - \sum_{j=1}^{\overline{\mathrm{H}}_c} \lambda_j e_j \qquad e_j \in e \qquad \textbf{(4-38)}$$

where $\overline{\mathrm{H}}_c$ is the cardinality of $\mathrm{H_c}$

In order to locate the optimum of the constrained minimization of $\|\mathrm{H}_{\tilde{s}}\|$ the derivatives of $L(\mathrm{H}_{\tilde{s}},\lambda)$ with respect to H and λ are set to zero as shown in (4-39) and (4-40).

$$\frac{\partial L}{\partial h_{\tilde{s}i}} = \nabla F - \mathrm{N}\lambda = 0 \qquad \textbf{(4-39)}$$

$$\frac{\partial L}{\partial \lambda} = -e_j\left(\mathrm{H}_{\tilde{s}}\right) = 0 \qquad \textbf{(4-40)}$$

where $\nabla F = 2\begin{bmatrix} h_{\tilde{s}1} \\ \dots \\ h_{\tilde{s}n} \end{bmatrix}$ and $\mathrm{N} = \begin{bmatrix} \nabla e_1 & \dots & \nabla e_{\overline{\mathrm{H}}_c} \end{bmatrix} = \begin{bmatrix} \mathrm{I} \\ D^{\mathrm{T}} \end{bmatrix}$, $\mathrm{n} = \overline{\mathrm{H}}_s$

After determining derivatives of $L(\mathrm{H}_{\tilde{s}},\lambda)$, equation (4-41) is obtained from (4-39).

$$2\begin{bmatrix} \mathrm{H}_{\tilde{s}1} \\ \mathrm{H}_{\tilde{s}2} \end{bmatrix} - \begin{bmatrix} \mathrm{I} \\ D^{\mathrm{T}} \end{bmatrix}\lambda = 0 \qquad \textbf{(4-41)}$$

Equations (4-42) and (4-43) are obtained from (4-41).

$$2\mathrm{H}_{\tilde{s}1} - \lambda = 0 \qquad \textbf{(4-42)}$$

$$2\mathrm{H}_{\tilde{s}2} - D^{\mathrm{T}}\lambda = 0 \qquad \textbf{(4-43)}$$

λ and $\mathrm{H}_{\tilde{s}2}$ are solved from (4-42) and (4-43) and shown as the follows:

$$\lambda = 2\mathrm{H}_{\tilde{s}1} \tag{4-44}$$

$$\mathrm{H}_{\tilde{s}2} = D^{\mathrm{T}}\mathrm{H}_{\tilde{s}1} \tag{4-45}$$

After substituting (4-45) in (4-37), a new equation is obtained as follows:

$$\mathrm{H}_{\tilde{s}1} + DD^{\mathrm{T}}\mathrm{H}_{\tilde{s}1} + pinv\left(C_{\tilde{s}1}\right)C_1 = 0 \tag{4-46}$$

From which a unique solution for $\mathrm{H}_{\tilde{s}1}$ can be obtained

$$\mathrm{H}_{\tilde{s}1} = -\left(1 + DD^{\mathrm{T}}\right)^{-1} pinv\left(C_{\tilde{s}1}\right)C_1 \tag{4-47}$$

Thus, the minimum norm solution of (4-35) is

$$\mathrm{H}_s = \begin{bmatrix} \mathrm{H}_{\tilde{s}1} \\ D^{\mathrm{T}}\mathrm{H}_{\tilde{s}1} \end{bmatrix} \tag{4-48}$$

4.3.1.2.1 Illustration of Symbolic Values Assignment

The following two examples are used to show the symbolic values assignment performance. An example is for input matrix containing only one feature vector of numerical values while the second has more than one column of numerical values.

The following example illustrates the assignment of symbolic values when numerical sub-matrix X_r is a vector. The input matrix is given as $\overline{X}$,

$$\overline{X} = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 8 & 9 & 8 & 9 & 10 \\ e & a & a & b & b & d & d & c & c & c \end{bmatrix}^T \quad \textbf{(4-49)}$$

The norm of the error vector in equation (4-19) must be minimized. To do so, a binary matrix A, which locates the symbolic values in $\overline{X}$, is obtained first.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \quad \textbf{(4-50)}$$

X_r can be obtained from $\overline{X}$ by removing the symbolic values.

$$X_r = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 8 & 9 & 8 & 9 & 10 \end{bmatrix}^T \quad \textbf{(4-51)}$$

Since X_r has only one column, H can be computed by applying equation (4-20). The results are shown in (4-52), and symbolic values X_s can be replaced by numerical values by multiplying matrix A and H as shown in (4-53).

$$\begin{matrix} H = [3.0 & 3.5 & 9.0 & 8.5 & 1.0]^T \\ a & b & c & d & e \end{matrix} \quad \textbf{(4-52)}$$

$$X_s = AH = [1.0 \quad 3.0 \quad 3.0 \quad 3.5 \quad 3.5 \quad 8.5 \quad 8.5 \quad 9.0 \quad 9.0 \quad 9.0]^T \quad \textbf{(4-53)}$$

The correlation coefficient between numerical values and evaluated symbolic values is calculated using (4-54).

$$r = \frac{\sigma_{X_r H}}{\sigma_{X_r} \sigma_H} \quad \textbf{(4-54)}$$

where $\sigma_{X_r H}$ is covariance between the two vectors and (σ_{X_r}, σ_H) are standard deviations of X_r and H.

The calculated correlation coefficient, shown in (4-55), and Figure 4-6 illustrate that the solution values H are well correlated with the numerical values. Calculated correlation coefficient is

$$r = 0.9746 \quad \textbf{(4-55)}$$

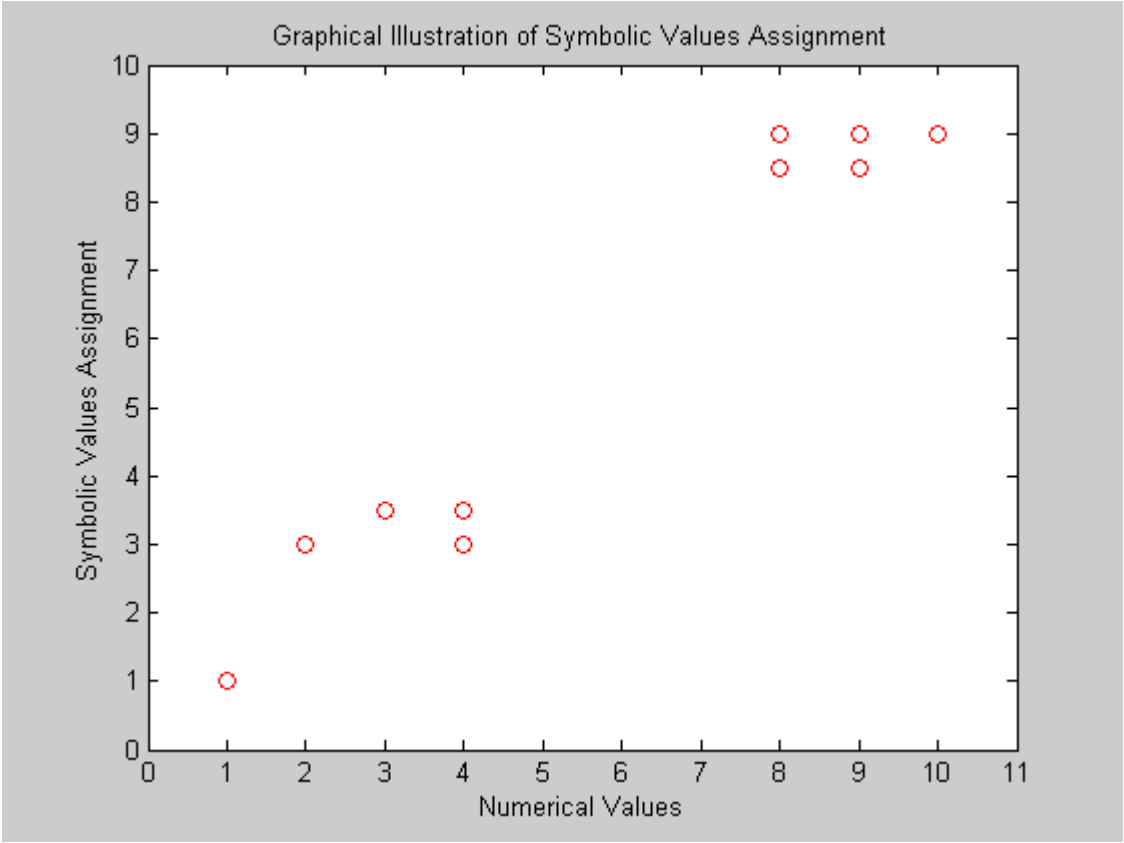

Figure 4-6 Graphical Illustration of Symbolic Values Assignment

The following is another example illustrating the assignment of symbolic values when numerical sub-matrix X_r has more than one row. The input matrix is given as $\overline{X}$,

$$\overline{X} = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 8 & 9 & 8 & 9 & 10 \\ e & a & a & b & b & d & d & c & c & c \\ 1 & 2 & 2 & 0 & 4 & 2 & -4 & 2 & 2 & -1 \end{bmatrix}^T \quad \textbf{(4-56)}$$

A binary matrix A which represents symbolic values in $\overline{X}$ and X_r which contains all numerical values are

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \quad \textbf{(4-57)}$$

and

$$X_r = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 8 & 9 & 8 & 9 & 10 \\ 1 & 2 & 2 & 0 & 4 & 2 & -4 & 2 & 2 & -1 \end{bmatrix}^T \quad \textbf{(4-58)}$$

Since X_r has more than one column, H can be determined by minimizing the norm of the error function and setting its derivatives to zero as shown in equation (4-22) to (4-23). To minimize the error norm, the matrix B can be obtained as defined in (4-24).

$$B = \left[\begin{array}{ccccc:cc} 2 & 0 & 0 & 0 & 0 & 6 & 4 \\ 0 & 2 & 0 & 0 & 0 & 7 & 4 \\ 0 & 0 & 3 & 0 & 0 & 27 & 3 \\ 0 & 0 & 0 & 2 & 0 & 17 & -2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 6 & 7 & 27 & 17 & 1 & 436 & 33 \\ 4 & 4 & 3 & -2 & 1 & 33 & 54 \end{array}\right] \quad \textbf{(4-59)}$$

B_s B_r

Then, Q_2^{T} is obtained using QR factorization on B_r and $Q_2^{\mathrm{T}} B_s$ is equal to

$$Q_2^{\mathrm{T}} B_s = \left[\begin{array}{c:cccc} -0.5359 & -0.4929 & 1.3394 & -0.9888 & -0.0735 \\ -0.0893 & 0.0747 & -1.0066 & 1.1191 & 0.0184 \\ -0.0766 & -0.1062 & -0.0704 & 0.0224 & 0.9825 \\ -1.6685 & 0.5709 & 0.3389 & 0.0064 & 0.0704 \\ 0.1332 & -1.5848 & 0.4663 & 0.1345 & 0.0731 \end{array}\right] \quad \textbf{(4-60)}$$

C_1 $\qquad$ $C_{\tilde{s}}$

H, the normalized vector of symbolic values, can be now computed by applying pseudo-inverse of $C_{\tilde{s}}$ (4-32).

$$\mathrm{H} = \begin{bmatrix} 1.0000 & 1.1495 & 2.7683 & 2.5424 & 0.3511 \end{bmatrix}^{\mathrm{T}} \quad \textbf{(4-61)}$$

To compare the previous result, H is scaled by multiplying all values by $\frac{1}{0.3511}$, H becomes

$$\mathrm{H} = \begin{bmatrix} 2.8480 & 3.2738 & 7.8840 & 7.2406 & 1.0000 \end{bmatrix}^{\mathrm{T}} \quad \textbf{(4-62)}$$

a $\quad$ b $\quad$ c $\quad$ d $\quad$ e

Symbolic values X_s can be replaced by numerical values by multiplying matrix A and H as shown in (4-63).

$$X_s = [1.0 \quad 2.85 \quad 2.85 \quad 3.274 \quad 3.274 \quad 7.241 \quad 7.241 \quad 7.884 \quad 7.884 \quad 7.884]^T \quad \textbf{(4-63)}$$

The correlation coefficients between different columns of numerical values and the evaluated symbolic values are calculated using (4-54). The calculated correlation coefficients are shown in Table 4-4.

Table 4-4 Correlation Coefficient Between Numerical and Symbolic Values

Symbolic values are correlated with	Correlation coefficient (r)
the 1st column of X_r	0.9739
the 2nd column of X_r	-0.3102

The calculated correlation coefficient results, and Figures 4-7 and 4-8 illustrate that the solution values H are well correlated with the first column of numerical values while they are not well correlated with the second column of X_r. It suggests that this set of symbolic values have much more dependence on the first column of numerical values than the second one.

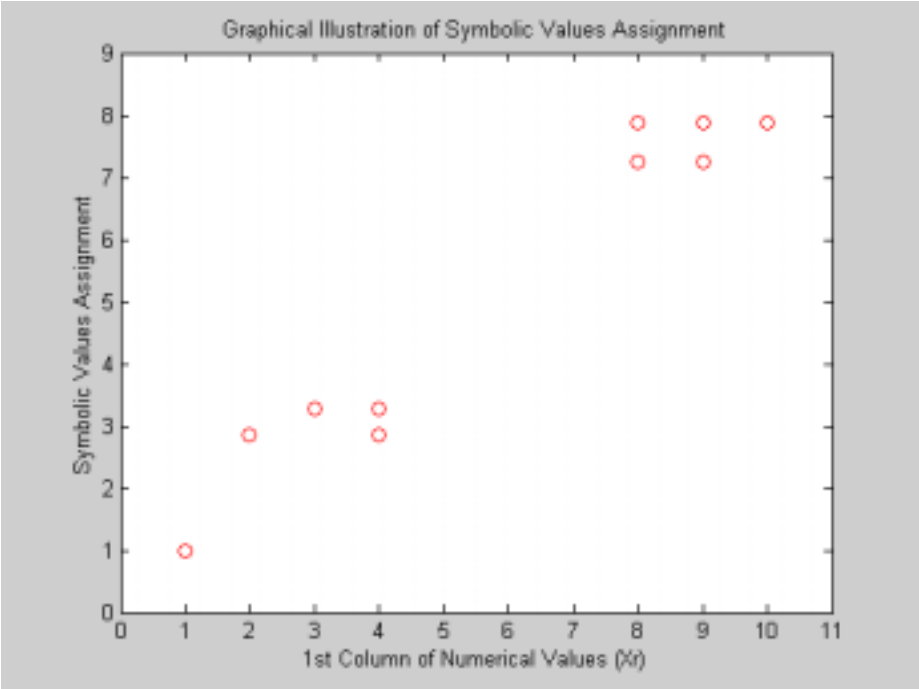

Figure 4-7 Symbolic Values Assignment Using 1st Column of Numerical Values

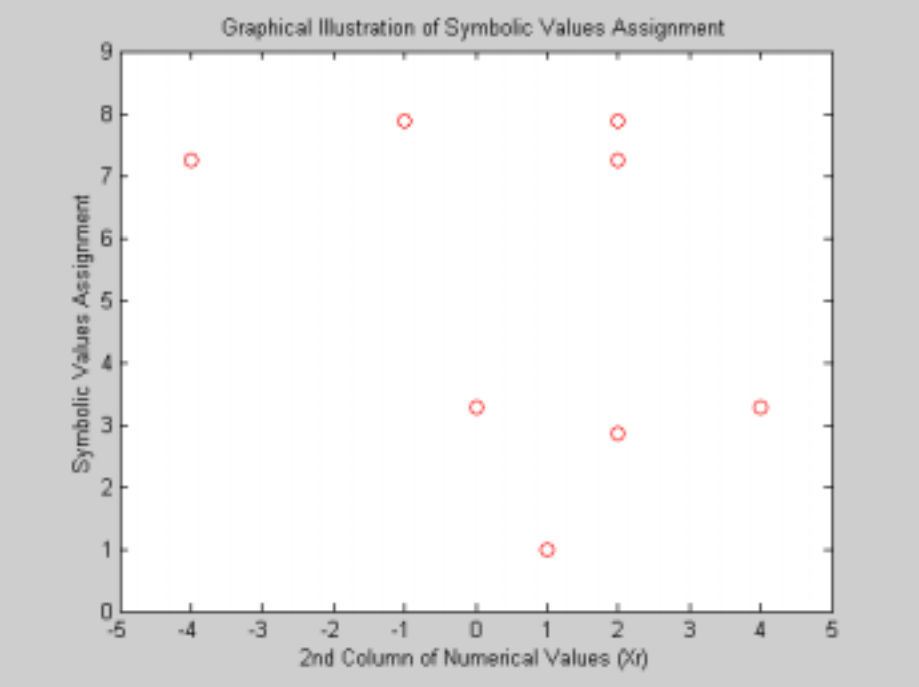

Figure 4-8 Symbolic Values Assignment Using 2nd Column of Numerical Values

Figure 4-9 gives an overview of how well the features are correlated in a three dimensional space. Although the calculated symbolic values do not have much dependence on the 2nd column of X_r alone, all features fit well in a three dimensional space.

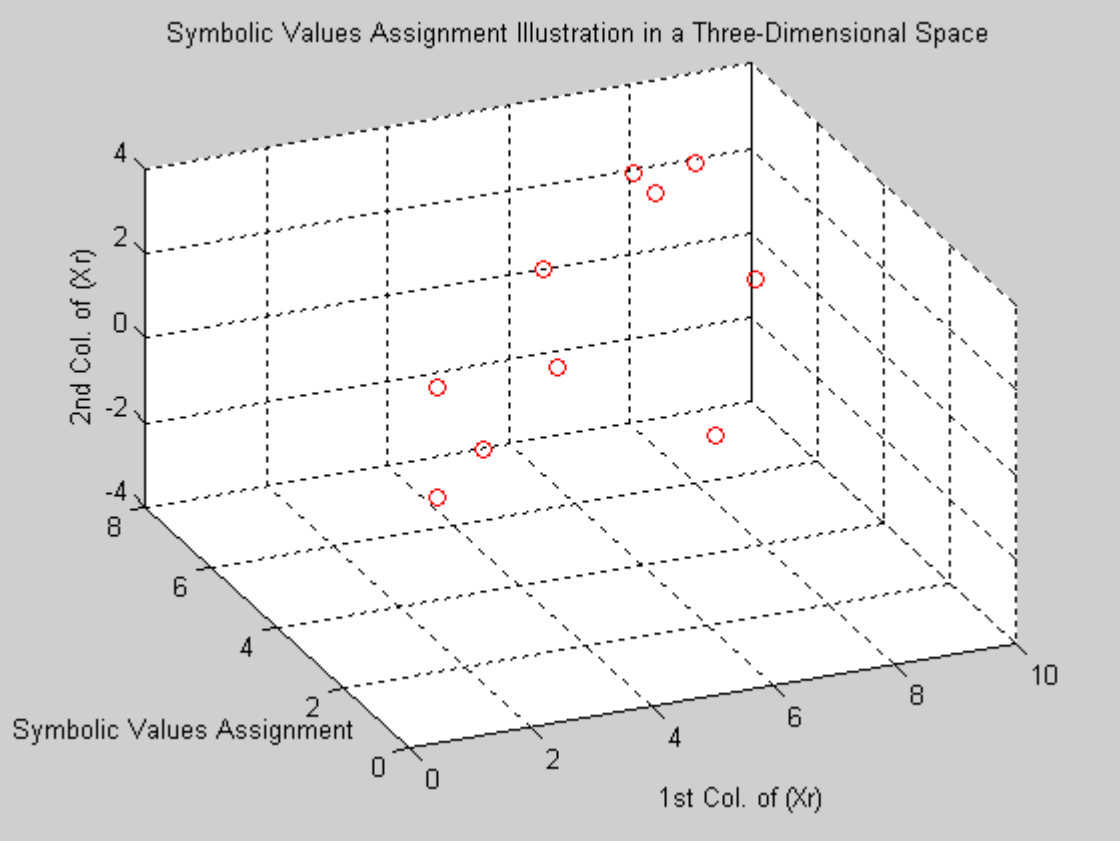

Figure 4-9 Symbolic Values Assignment in a Three-Dimensional Space

The following illustrates that evaluated symbolic values with better representation can be achieved using all numerical values rather than taking only one vector of numerical values. The comparison is done by obtaining determinants of the resulting covariance

matrixes of $\overline{X}$ in (4-56). If symbolic values assignments of matrix $\overline{X}$ are obtained by applying equation (4-20) and by selecting one vector of numerical values at a time, two solutions are produced as shown in Table 4-5. The correlation coefficients between different columns of numerical values and their evaluated symbolic values are calculated using (4-54). The calculated correlation coefficients are illustrated in Table 4-6, and the symbolic values assignment corresponding to the highest correlation coefficient is selected.

Table 4-5 Two Sets of Evaluated Symbolic Values

the 1st column of X_r	the 2nd column of X_r
3.0000	2.0000
3.5000	2.0000
9.0000	1.0000
8.5000	-1.0000
1.0000	1.0000

Table 4-6 Correlation Coefficient Between Numerical and Symbolic Values

the 1st column of X_r	the 2nd column of X_r
0.9746	0.5222

Since the solution obtained from the 1st column of X_r has the larger correlation coefficient, it is used to replace all symbolic values. Therefore, the matrix $\overline{X}$ (4-56) becomes $\overline{X}_a$ (4-64) where the calculated symbolic values are obtained from only one

vector of numerical values, and becomes $\overline{\mathrm{X}}_b$ (4-65) when the symbolic values are obtained from all numerical values.

$$\overline{\mathrm{X}}_a = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 8 & 9 & 8 & 9 & 10 \\ 1 & 3 & 3 & 3.5 & 3.5 & 8.5 & 8.5 & 9 & 9 & 9 \\ 1 & 2 & 2 & 0 & 4 & 2 & -4 & 2 & 2 & -1 \end{bmatrix}^{\mathrm{T}} \quad \textbf{(4-64)}$$

$$\overline{\mathrm{X}}_b = \begin{bmatrix} 1 & 2 & 4 & 3 & 4 & 8 & 9 & 8 & 9 & 10 \\ 1 & 2.85 & 2.85 & 3.27 & 3.27 & 7.24 & 7.24 & 7.88 & 7.88 & 7.88 \\ 1 & 2 & 2 & 0 & 4 & 2 & -4 & 2 & 2 & -1 \end{bmatrix}^{\mathrm{T}} \quad \textbf{(4-65)}$$

The covariance is defined as the average of the products of the deviations of feature values from their means in a closed sphere. This sphere can be normalized to obtain a unit volume, and each element of the covariance matrix can be correspondingly multiplied by (n-1), where (n x n) is the size of the covariance matrix. Determinant of such normalized covariance matrix gives an overview of how well all features are correlated within a matrix in a multidimensional space. Ideally, the determinant of such normalized covariance D_c is one if all features in the matrix are totally independent, and it is zero if all features are perfectly correlated. Determinants of resulting covariance matrices of $\overline{X}_a$ and $\overline{X}_b$ are shown in Table 4-7. The evaluated symbolic values obtained with all numerical values give a better representation since the determinant is smaller.

Table 4-7 Determinants of Resulting Covariance Matrices

	Determinant of Covariance of $\overline{X}_a$	Determinant of Covariance of $\overline{X}_b$
D_c	0.3455	0.3444

The following example illustrates how H is obtained if $C_{\tilde{s}}$ does not have full rank. Suppose that the coefficient matrix $C_{\tilde{s}}$ and C_1 are as follows:

$$C_{\tilde{s}} = \begin{bmatrix} -0.49 & 1.34 & -0.0621 & -2.0913 \\ 0.07 & -1.0 & -0.3469 & 1.4631 \\ -0.11 & -0.07 & -0.1771 & 0.0645 \\ 0.57 & 0.34 & 0.9077 & -0.3015 \\ -1.58 & 0.47 & -1.8946 & -1.1982 \end{bmatrix} \quad \textbf{(4-66)}$$

$$C_1 = \begin{bmatrix} -0.54 \\ -0.09 \\ -0.08 \\ -1.67 \\ 0.13 \end{bmatrix} \quad \textbf{(4-67)}$$

If a straightforward solution is used with equation (4-35) by setting $H_{\tilde{s}2} = 0$, $H_{\tilde{s}1}$ becomes

$$H_{\tilde{s}1} = -pinv(C_{\tilde{s}1})C_1 = \begin{bmatrix} -0.5256 \\ -0.5741 \end{bmatrix} \quad \textbf{(4-68)}$$

and its norm is

$$\left\| H_{\tilde{s}1} \right\| = 0.7784 \quad \textbf{(4-69)}$$

However, if $\mathrm{H}_{\tilde{s}1}$ is calculated using equation (4-47), it becomes

$$\mathrm{H}_{\tilde{s}1} = -\left(1 + DD^{\mathrm{T}}\right)^{-1} pinv\left(C_{\tilde{s}1}\right)C_1 = \begin{bmatrix} 0.1761 \\ 0.1698 \end{bmatrix} \qquad \textbf{(4-70)}$$

where $D = R_1^{-1}R_2 = \begin{bmatrix} 1.33 & 0.33 \\ 0.44 & -1.44 \end{bmatrix}$

Instead of setting $\mathrm{H}_{\tilde{s}2}$ to zero, it can be calculated using equation (4-45).

$$\mathrm{H}_{\tilde{s}2} = D^{\mathrm{T}}\mathrm{H}_{\tilde{s}1} = \begin{bmatrix} 0.309 \\ -0.1864 \end{bmatrix} \qquad \textbf{(4-71)}$$

The resulting symbolic vector $\mathrm{H_s} = \left[\mathrm{H}_{\tilde{s}1}^{\mathrm{T}} \quad \mathrm{H}_{\tilde{s}2}^{\mathrm{T}}\right]^{\mathrm{T}} = [0.1761 \quad 0.1698 \quad 0.309 \quad -0.1864]^{\mathrm{T}}$, which has the norm $\|\mathrm{H_s}\| = 0.4360$. Obviously, the obtained solution satisfies equation (4-37) with the minimum norm.

4.3.1.3 Other Approach for Missing and Symbolic Data

Instead of using the same covariance matrix for all points from a given class, local covariance matrixes obtained from clusters of points in a given class should be used. A clustering algorithm has to be used first to obtain these clusters and their covariance matrixes. Then, their missing and symbolic values problems can be solved. When a combination of symbolic and missing data exists, then the symbolic values problem should be solved first using samples without missing data, after which all missing data should be recovered.

4.4 Neurons' Output

In a process layer, after a neuron has selected inputs (or a single input), transformation function, threshold value, and the TCI, the neuron's output is generated. This output will become an input to other neurons which are wired to the present neuron. These outputs are combined with the results of the transformation functions from previous layers of neurons with neuron's own transformation creating complex partitions of the input space. Final, logical outputs are generated based on the threshold separation results from the process layers.

Chapter 5.
5. Arithmetic Operations

Each neuron processes its input data by selecting one of the operations. Since artificial neuron networks are designed for real time processing, operations must be simple in order to result a small physical area and fast processing time. Moreover, neurons are capable to perform liner and nonlinear mathematical operations. The processor performing these operations is a reduced instruction-set processor (RISP). This processor is able to select one of the operations and perform it on the inputs. It is assumed that the processor is designed to work with 8-bit input data.

All linear and nonlinear transformations can be derived from adding, subtracting, averaging, and shifting. Nonnegative results and inputs are expected. All results from arithmetic operations must be scaled to full range from 0 – 255 for the 8-bit numbers in order to maintain the full resolution.

5.1 Basic Arithmetic Operations

Since both linear and nonlinear arithmetic operations can be obtained using add, subtract, compare, and shift, the RISP processor only needs to perform such operations. In addition, two single bit operations are defined in Table 5-1 and Table 5-2:

L(a) returns the location (starting from 0) of the most significant bit position of its argument a, while E(a) is the inverse of L(a).

Table 5-1 L(a) Function

a	0	1	2	4	8	16	32	64	128
L(a)	0	1	2	3	4	5	6	7	8

Table 5-2 E(a) Function

a	0	1	2	3	4	5	6	7	8
E(a)	0	1	2	4	8	16	32	64	128

The property which relates L(a) and E(a) is follows:

$$L[E(a)] = E[L(a)] = a \qquad \textbf{(5-1)}$$

Potential transformation operations within 0-255 are defined in Table 5-3.

Table 5-3 Simple Arithmetic Operations

Identical (X): X	Half (X): $\frac{X}{2}$
Addition (a, b): $\frac{(a+b)}{2}$	Subtraction (a, b): $\begin{cases} a-b & if \ \ a>b \\ 0 & if \ \ a<b \end{cases}$
Multiplication (a, b): $E\{Sub[L(a)+L(b),B]\}$	Exponent (a): $E\langle E\{Sub[L(a),L(32)]]\}\rangle$
Square root (a): $E\left\{\frac{L(a)}{2}+\frac{B}{2}\right\}$	Inverse (a): $E\{Sub[B,L(a)]\}$
Square (a): $E\{Sub[2*L(a),B]\}$	Logarithm (a): $E\{L[L(a)]+5\}$

Figure 5-1 shows the behaviors of one-argument operations: exponent, square, square root, logarithm, and inverse functions in an 8-bits space.

X feature input array = 1, 2, 3, … 255

Y feature output array = 1, 2, 3, … 255

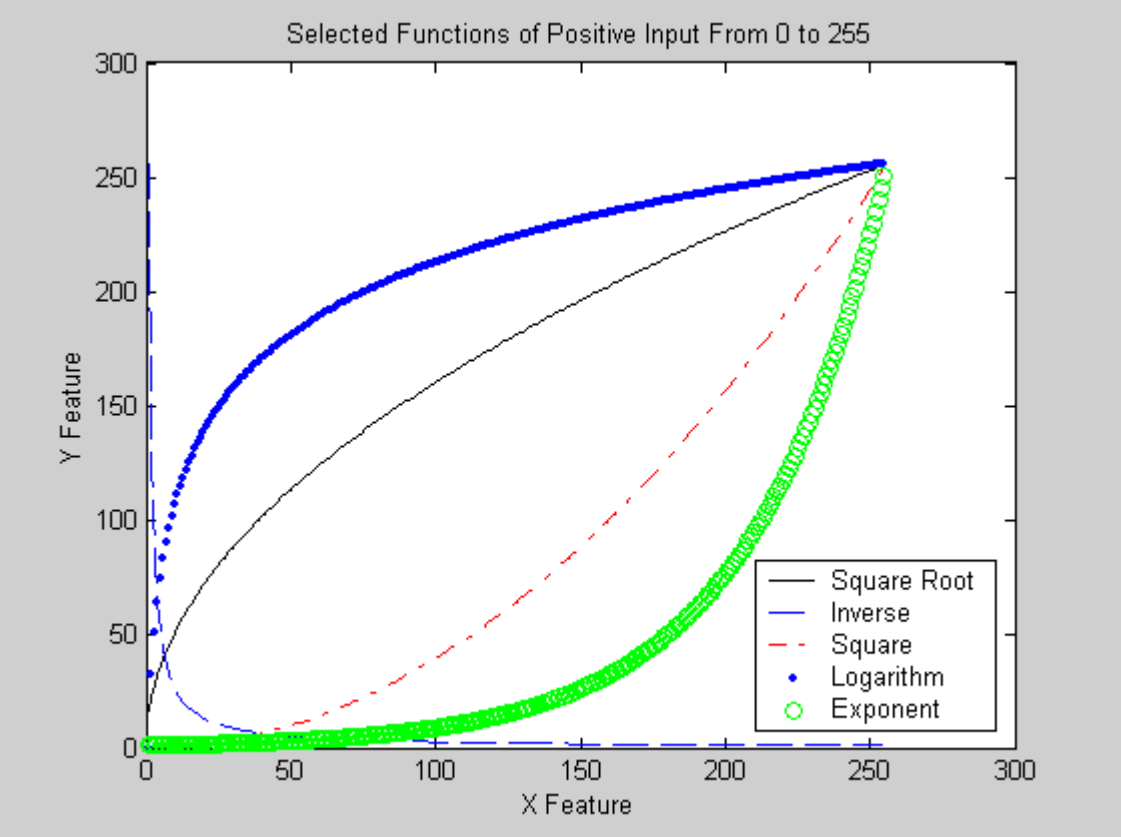

Figure 5-1 Exponent, Square, Square root, Logarithm, and Inverse Function

5.2 Multiple functions

Besides regular operations, multiple functions, which combine the different arithmetic operations, can also be used to generate more complicated expressions. These functions are useful in separating different classes from the local input space. In fact, with these operations, more complex transformation functions corresponding to the original input space grow along with increasing numbers of neuron's layers. After several layers, a neuron may be able to generate a complex transformation function based on its own basic operations with input data that may have been processed with many prior operations. Figure 5-2 demonstrates how multiple simple functions combined together become a complex function as the one described by equation (5-1).

X feature input array = 1, 2, 3, … 255

Y feature input array = 1, 2, 3, … 255

$$\log(Y) - \left[\frac{X + 2\log(Y)}{8}\right] - \log(X) = 424.374 \qquad \textbf{(5-1)}$$

where 424.374 was the equivalent threshold value in the input space.

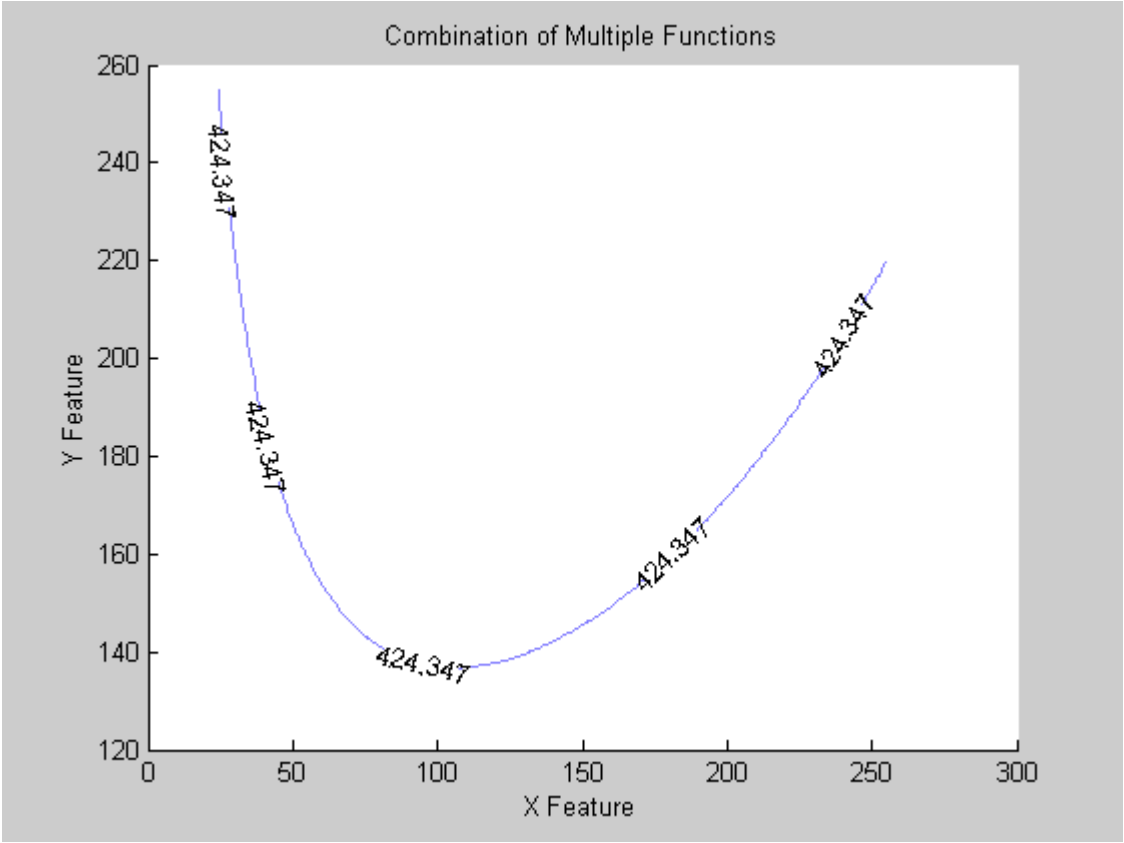

Figure 5-2 Combination of Multiple Functions

Chapter 6
6. Self-Organization Principles

Once a network has been designed, it is ready for training. There are two types of learning approaches in artificial neural network design: supervised and unsupervised (Hassoun, 1995, p. 57). Supervised learning requires desired outputs and inputs. Outputs of the network are compared with the desired outputs, and the differences are propagated back to the system. The network has to adjust its weights to match the network outputs. This process continues until the network is able to produce output similar to the desired one. However, if the network cannot solve the problem, all the parameters, such as weights, connections, number of layers, etc, must be revised and adjusted.

The other approach is called unsupervised learning. With such learning, only the input signal is provided to the network with no other influence. Unsupervised learning in literature refers almost exclusively to self-organization of the training data. It helps with clustering and data representation.

In this thesis, self-organization is applied to the learning hardware and, in general, can be either supervised or unsupervised. However, the examples illustrating its use in

this thesis are based on the supervised training. Unlike neural networks which have well defined organization of interconnection and neuron functions, SOLAR involves its connections, neuron control, the transformation function, and the threshold value to achieve the best performance during the learning phase. Since it does not require an outside help and is able to organize itself, it is also referred to as self-organizing network.

6.1 Neuron Self-Organizing and Learning

During learning, a neuron counts the total amount of training data n_t. This can be done simply by counting the impulses of its system clock input. Similar to any sequential machines, each neuron performs an operation on the selected inputs (or single input) at the rising edge of the system clock. The result may become the system output or an input to other neurons. If the TCI associated with a particular input data is high, the result of this operation is compared against a set threshold value. This means that this input data is within the subspace where the current neuron is learning. If TCI is zero, on the other hand, no comparison takes place since this particular input data is outside of the subspace where the neuron is learning. Counters in each neuron controlled by its TCI count three sets of numbers.

- Amount of data that satisfy the threshold value: n_s
- Amount of data belonging to a class that satisfy the threshold value: n_{sc}

- Amount of data belonging to a class that does not satisfy the threshold value: n_{sic}

By doing so, threshold value divides the neuron's input space into two subspaces. The quality of learning of each neuron can be calculated statistically by computing the information index.

In order to calculate information index, finding the probabilities of training data which fall into each subspace is required.

- Probability of a class satisfying threshold: $P_{sc} = \frac{n_{sc}}{n_t}$ **(6-1)**

- Probability of a class not satisfying threshold: $P_{sic} = \frac{n_{sic}}{n_t}$ **(6-2)**

- Subspace probability (pass threshold): $P_s = \frac{n_s}{n_t}$ **(6-3)**

- Complementary subspace probability (does not pass threshold): $P_{si} = 1 - P_s$ **(6-4)**

- Class probability – $P_c = \frac{n_c}{n_t}$ **(6-5)**

With these calculated probabilities, information index can be obtained from (6-6).

$$I = 1 - \frac{\Delta E_s}{E_{max}} = 1 - \frac{\left[\sum_{sc} P_{sc} \log(P_{sc}) - P_s \log(P_s)\right] + \left[\sum_{sic} P_{sic} \log(P_{sic}) - P_{si} \log(P_{si})\right]}{\sum_{c} P_c \log(P_c)} \quad \textbf{(6-6)}$$

Different combinations of inputs, transformation operations and TCI can result in different information index values. Neurons perform information index calculation for different combinations, and the maximized result is obtained in order to provide an optimum separation of the input training data. When the index value becomes "1", it indicates that the neuron has solved its problem completely. However, it does not mean that any test data can be classified correctly all the time.

Figure 6-1 and Figure 6-2 show that different transformation functions can result in the different information index values. These graphs show the information index computation by using addition and multiplication. Having the same inputs and TCI, multiplication of neuron's inputs can produce a higher information index than addition. If the value remains the highest among all combinations, multiplication is selected by this neuron as its transformation function. These input connections (or single connection) and threshold value are also stored.

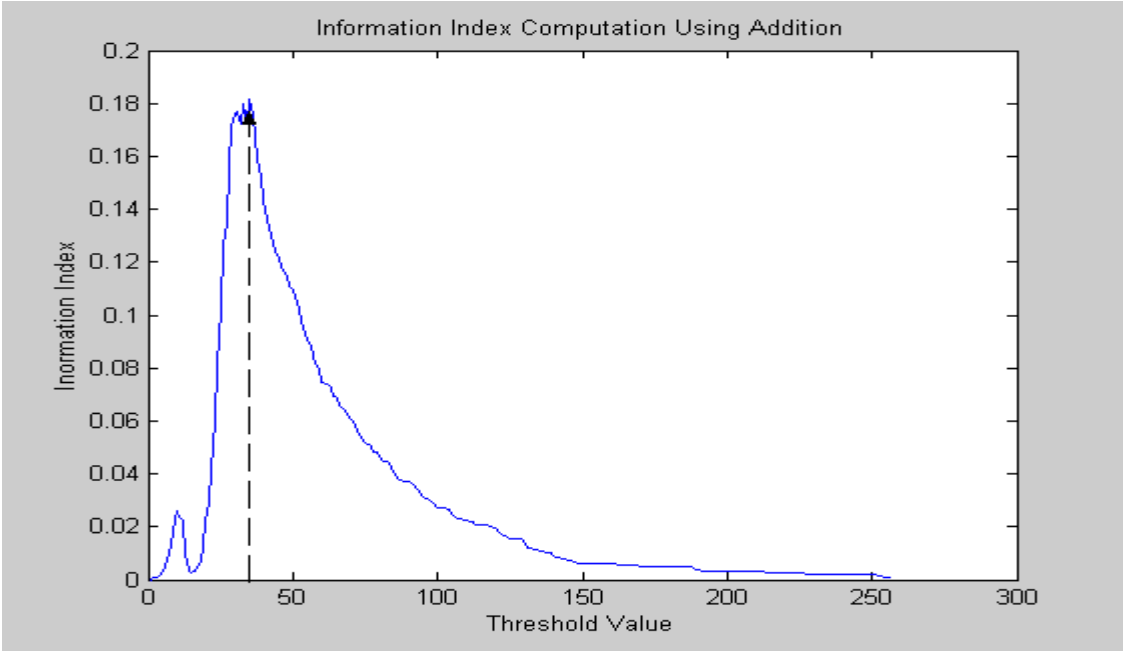

Figure 6-1 Finding Information Index Using Addition

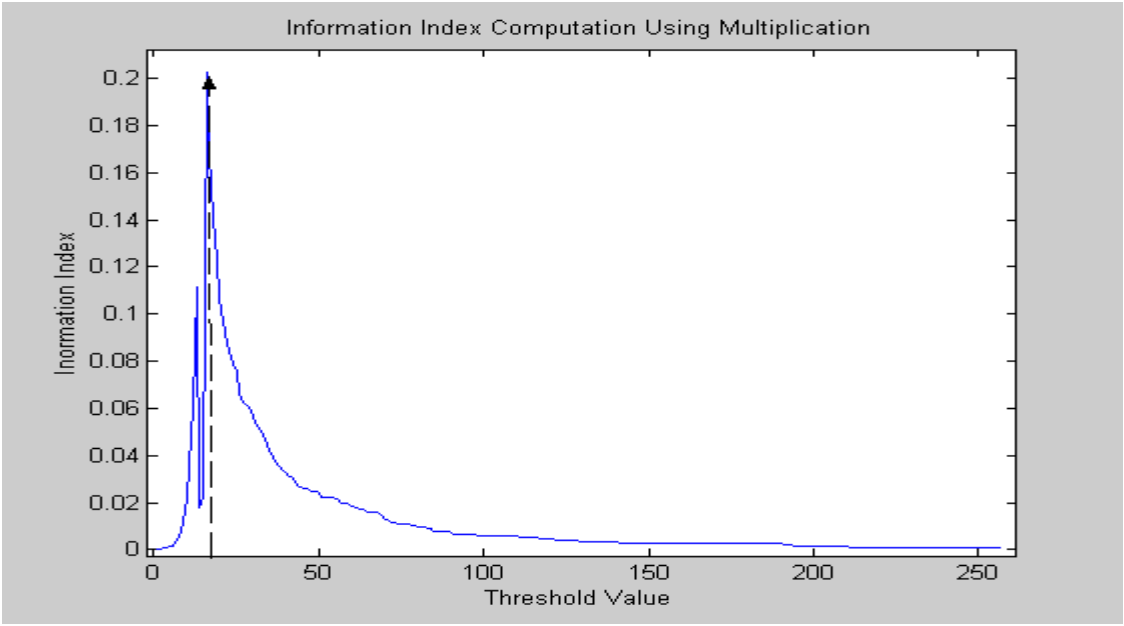

Figure 6-2 Finding Information Index Using Multiplication

Threshold value is used to separate the input space. Figure 6-3 shows how subtraction with a threshold divides a space into two subspaces and separates the two classes.

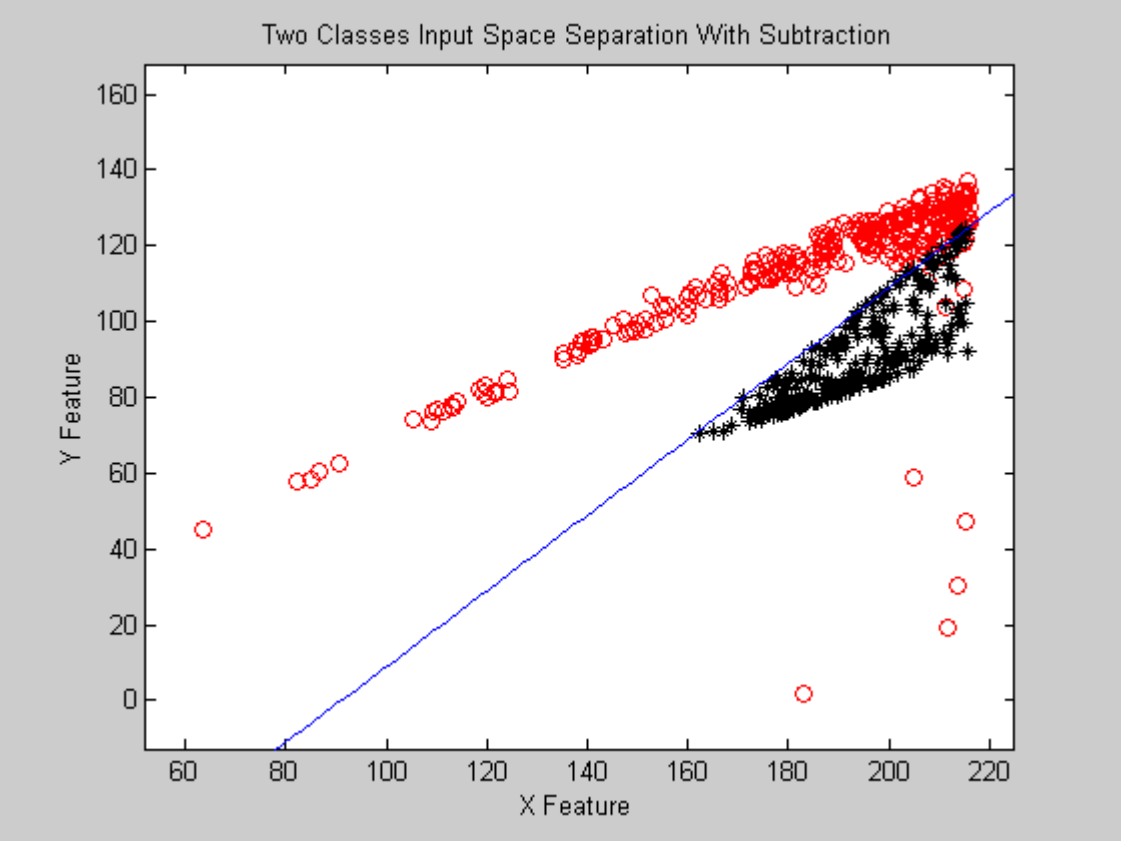

Figure 6-3 Input space Separation Using Subtraction

6.2 Subspace Learning

Information deficiency is simply a normalized relative subspace entropy. It indicates the amount of knowledge that must be learned to solve a classification problem in a given subspace. Subspace s information deficiency is defined in equation (6-7).

$$\delta_s = \frac{\Delta E_s}{E\max} = \frac{\sum_{sc} P_{sc} \log(P_{sc}) - P_s \log(P_s)}{\sum_{c} P_c \log(P_c)} \qquad \textbf{(6-7)}$$

One space can be divided into many subspaces during learning. At the first layer of neurons, it is assumed that the input information deficiency is one. The relationship of the information index and the information deficiencies are shown in equation (6-8)

$$1 - I = \sum_{s} \delta_s \qquad \textbf{(6-8)}$$

Each subspace can be learned by minimizing the information deficiency. If the information deficiency becomes zero, it means that there is nothing left to be learned by a neuron. The frequency of subdividing a subspace is based on the probability of each neuron selecting a TCOT or TCOTI as its TCI. If TCI of a neuron is connected to TCO from the previous neurons, it has a greater chance in subdividing an input subspace.

Once local features are selected based on the maximum local information index, output information deficiencies for TCOT and TCOTI are obtained, which are defined as the product of subspace information deficiencies and input information deficiency. On the other hand, the information deficiency for TCO remains the same as its input information deficiency. These output information deficiencies are carried out to the neurons where they are connected to the threshold-control-clock (TCO, TCOTI, TCOT) for the next stage of learning and become the input information deficiencies of the next stage neurons.

6.2.1 Termination of Learning

The computed output information deficiency allows the next neuron to know if its corresponding selected subspace has been learned enough. If the incoming information deficiency is less than or equal to the chosen information deficiency threshold (IDT), it indicates that not much information can be gained by further dividing the selected input space. This neuron stops learning the selected subspace and moves on to other selected subspaces. If the incoming information deficiencies are all low enough, this neuron stops learning and will not participate in voting during the testing stage.

Chapter 7.
7. Final Classifications

Data which needs to be classified is sent to the network, and each neuron performs classification based on their learning results. Finally during the voting process, all participating neurons vote for the class to which they believe the data is categorized. The voting layer gathers all the information and decides which class the input really belongs to using a weighting function.

7.1 Voting Neurons

Each neuron identifies itself whether it is qualified to participate in the final classification after training. This identification is done by comparing neurons' output information deficiencies of TCOT and TCOTI with the voting threshold. There are two flags in each neuron. One flag is select-output-passed threshold (SOT) which indicates that the neuron is capable to vote when its calculated output passes its threshold. It is set only if the neuron's output information deficiency of TCOT is less than or equal to a voting threshold. The other flag, select-output-passed threshold-inverse (SOTI), indicates that the neuron is qualified to participate in voting when the transformed output does not pass its threshold. This flag is set if the neuron's output

information deficiency of TCOTI is smaller than or equal to a voting threshold. If both flags are set, a neuron can vote for either its transformed output data passes its threshold or fails it. Table 7-1 shows all the conditions for setting flags SOT and SOTI required for a neuron to vote.

Table 7-1 SOT and SOTI Flag Set Condition

SOT	SOTI	δ_{output} of TCOT	δ_{output} of TCOTI
0	0	> (1-DIT)	> (1-DIT)
0	1	> (1-DIT)	<= (1-DIT)
1	0	<= (1-DIT)	> (1-DIT)
1	1	<= (1-DIT)	<= (1-DIT)

7.2 Weighting Function

Unlike other artificial neural networks using the “winner takes all” approach, a neuron having more knowledge about an input data is weighted heavier, while other neurons weights are lower and have less influence in the final voting. Neurons of SOLAR first check whether they are capable to vote by checking their flags for a given input. Then, they internally stored probabilities of correct classification, based on whether the input data passes or does not pass threshold values, are used to calculate the weighting function which determines the classification of input data.

During learning, each neuron counts and stores three numbers. One is the total amount of data that satisfies threshold $n_{s,}$ another is the total number of data belonging

to a class that satisfies threshold n_{sc} and finally an amount of data belonging to a class that does not satisfy threshold n_{sic}.

The probability of correct classification P_{cc} can be computed as follows:

- Probability of correct classification: $P_{cc} = \frac{n_a}{n_b}$ **(7-1)**

where $n_a = \begin{cases} n_{sc} & \text{(pass threshold)} \\ n_{sic} & \text{(does not pass threshold)} \end{cases}$

$n_b = \begin{cases} n_s & \text{(pass threshold)} \\ n_{si} & \text{(does not pass threshold)} \end{cases}$

$n_{si} = n_t - n_s$

n_s = amount of data that satisfy the threshold value

n_{sc} = amount of data belonging to a class that satisfy the threshold value

n_{sic} = mount of data belonging to a class that does not satisfy the threshold value

n_t = total among training data

After the transformed signals are calculated and checked against threshold, neurons set their self-organizing neural network output (SONNO) to notify the voting layer if they are voting for this incoming data. Their classification probabilities are also sent to the voting layer for final classification calculation.

Once the voting layer receives the information from all participating neurons, it performs a final classification calculation using a weight function which is described in equation (7-3).

$$W_c = 1 - \frac{P_{cc_{MAX}} + \left\{ \left[\sum_{i=1}^{n} \left(1 - P_{cc_i}\right) \right] \left(1 - P_{cc_{MAX}}\right) \right\}}{\sum_{i=1}^{n} \left(\frac{1}{1 - P_{cc_i} + \varepsilon} \right)} \qquad \textbf{(7-3)}$$

where $P_{cc_{MAX}}$ = maximum P_{cc} of all SONNOs' "voting" for class c

P_{cc_i} = P_{cc} of each "vote" for class c

n = number of "votes" for class c

ε = small number preventing division by zero

7.2.1 Example of Weighting Function Calculation

In order to illustrate the property of this weight function, an example is provided. Assume that there are five neurons participating in voting for a particular input data with three classes, and that the correct classification probabilities of each neuron corresponding to the threshold value are shown in Table 7-2. This information is obtained from each neuron after a learning process.

Table 7-2 Probabilities of Correct Classification

	Neuron Number				
	1	2	3	4	5
Class 1	0	0.293	0.179	0.671	0.015
Class 2	0.833	0.632	0.325	0.329	0.985
Class 3	0.167	0.075	0.496	0	0

As $\varepsilon = 0.001$, the weights of different classes for this particular input data are calculated with equation (7-3), and they are shown in Table 7-3. Class 2 is classified by the network for this particular input data since it has the largest weight value.

Table 7-3 Voting Weight for Different Classes

Class 1	Class 2	Class 3
0.7478	0.9863	0.5774

Chapter 8
8. Software Simulations

In order to show the results of the network learning, two sets of real datasets are selected from the University of California at Irvine (ICS, UCI, 1995, December) and fed to SOLAR as inputs. These datasets represent real world application problems. One is a credit card approval problem from Australia, and the other one is a personal income classification. In addition, a two-dimensional data example, based on synthetic data, is used to illustrate the SOLAR performance. Figure 8-1 and Figure 8-2 are flow charts indicating the logic flow of SOLAR simulation program during learning and testing.

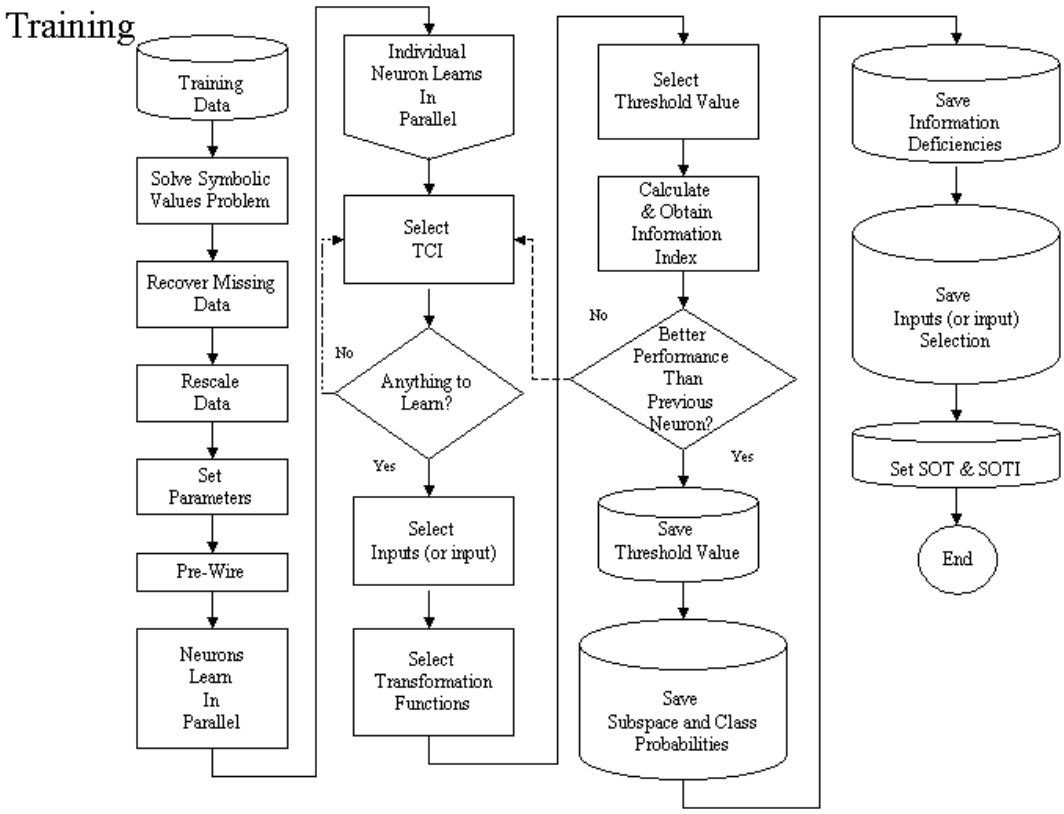

Figure 8-1 Flow Chart of SOLAR Software Program in Learning

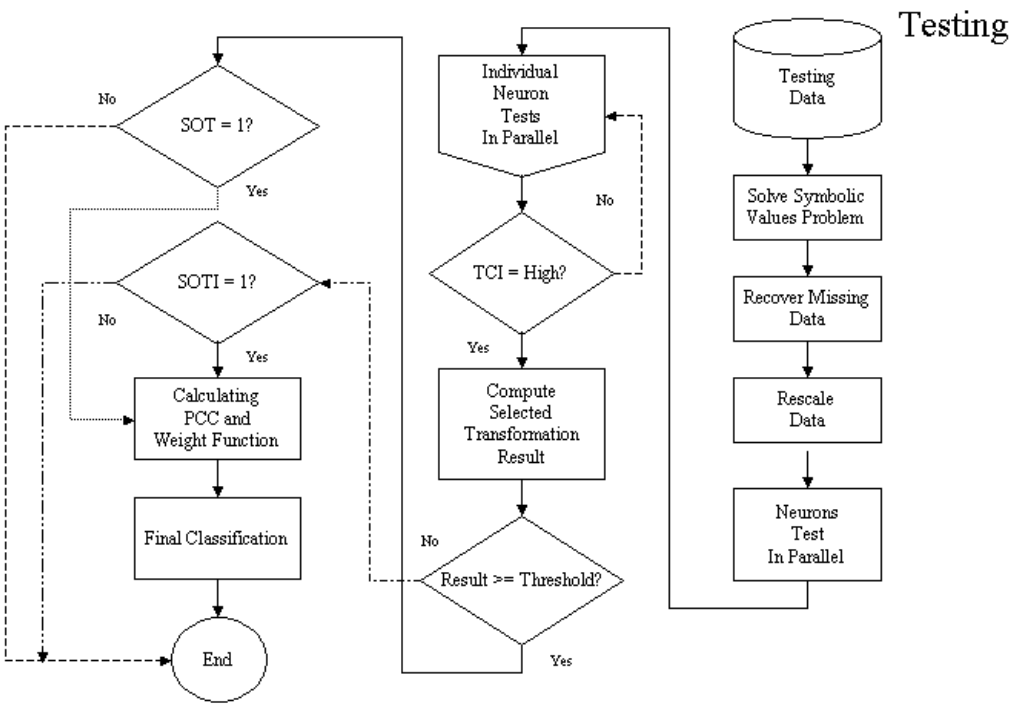

Figure 8-2 Flow Chart of SOLAR Software Program in Testing

8.1 Two Dimensional Data Illustration

This two-dimensional training dataset was statistically generated, and it is not a real-world application dataset. There are five classes in this training dataset as shown in Table 8-1 and Figure 8-3.

Table 8-1 Classes of Two Dimensional Training Data

	Class 1	Class2	Class 3	Class 4	Class 5
Number of points	503	429	190	682	542

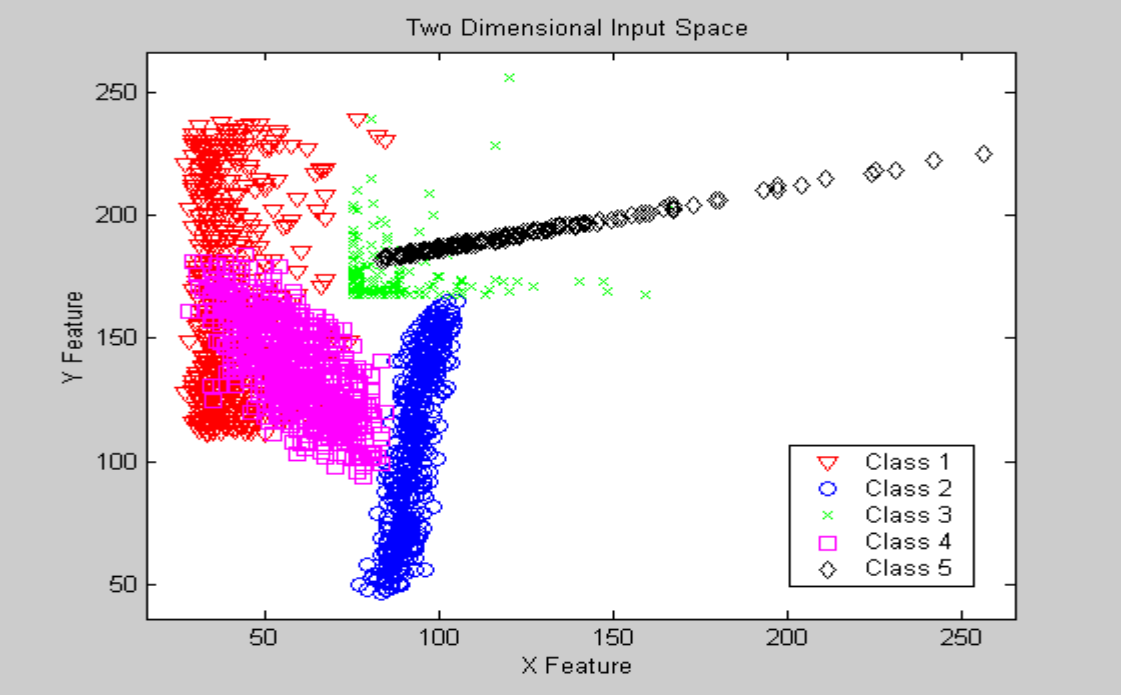

Figure 8-3 Two Dimensional Input Space

8.1.1 Network Parameters

Before generating the network, design parameters are required to be set. These parameters determine the size of the input, input control clock, probability of selecting a subspace, and information deficiency threshold. For example, majority of the TCI (90%) is connected to the TCO while 5% of the TCI connects to either TCOT or TCOTI randomly, when the subspace selection probability is set to 0.1. These parameters are important because they can have significant effect on the network performance.

- Input parameters:
    1. Number of input(s) from the nearest neighbors = 1
    2. Number of input(s) from the next nearest neighbors = 1
    3. Number of input(s) from remote neighbors = 1
- Number of connection(s) to TCI = 3
- Voting threshold = 0.9
- Subspace selection probability = 0.1
- Information deficiency threshold = 0.1
- Number of layers = 15
- Number of neurons per layer = 2

8.1.2 Initial Wiring

As discussed in Chapter 3, statistically generated Mahalanobis Distance is applied in initial wiring for neurons' inputs. According to the design parameter setting, there are three inputs for each neuron. The nearest neuron, which is located at the previous column and the same row, is always connected. Since more than one neuron are considered as the next nearest, one of them is connected based on a random generator. The last input is connected totally based on the random generator.

After wiring the input signals, threshold-control-inputs (TCI) are connected. Because local learning is believed to result in a better performance, one of the TCI is always connected to the nearest neuron while the other two are connected to the previous neurons based on the random generator. The final initial wiring of the network is shown in Figure 8-4.

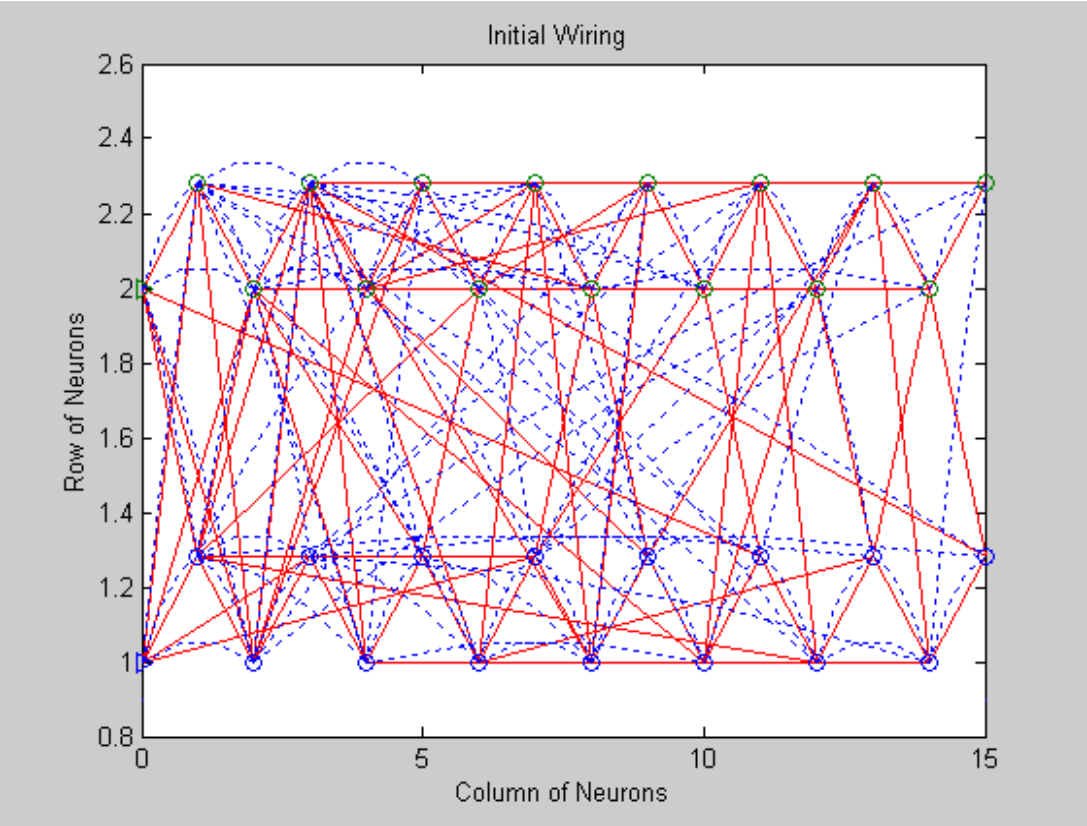

Figure 8-4 Initial Wiring of SOLAR for Two Dimensional Dataset

8.1.3 Functions

Each neuron in SOLAR has pre-defined set of operations, and they are all the same for each neuron throughout the network. For faster operation, few simple operations are included in each neuron. These operations (discussed in Chapter 5 – Arithmetic Operations) can be classified into two groups. One group is called “unary kernels ” while another one is called “binary kernels”.

"Unary kernels" include operations such as identity, half, logarithm and exponential.

- Identity function: $Y = IDENT(X) = X$ **(8-1)**
- Half function: $Y = HALF(X) = \frac{X}{2}$ **(8-2)**
- Logarithm function: $Y = NLOG2(X) = 2^{\{\log 2[\log 2(X)]+5\}}$ **(8-3)**
- Exponential function: $Y = NEXP2(X) = 2^{\left(\frac{X}{32}\right)}$ **(8-4)**

"Binary kernels" include operations such as addition and subtraction.

- Addition function: $Y = NADD(X, Y) = \frac{X+Y}{2}$ **(8-5)**
- Subtraction function: $Y = NSUB(X, Y) = X - Y$ **(8-6)**

A combination of "unary kernels" and "binary kernels" operation is encouraged since multiple transformation functions can result in a more complicated curve that may be used in dividing local input space.

8.1.4 Neuron Learning

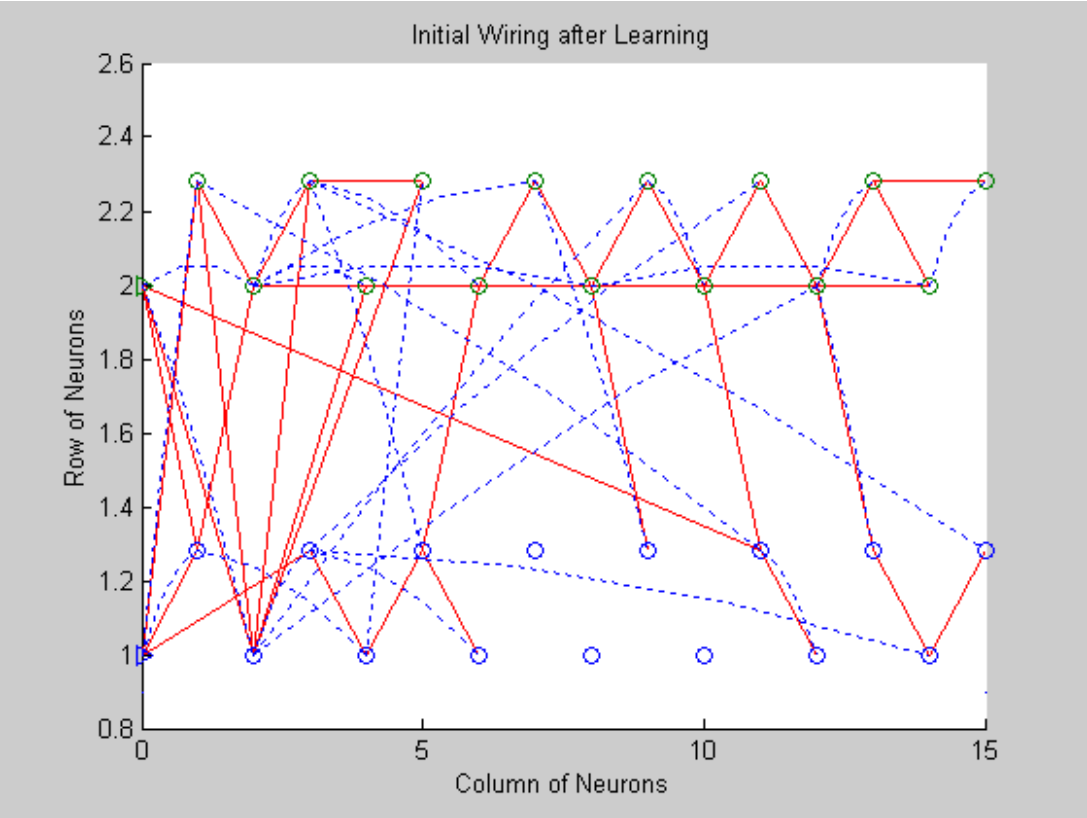

Figure 8-5 Wiring of SOLAR after Learning Process

Figure 8-5 shows the wiring of the network after learning. Each neuron connects to a maximum of two inputs, and only one output clock is selected for its TCI. Neurons, in Figure 8-5 which have no wires connected, do not have any effect on the final classification since they did not learn during the training process. It is because there is nothing left for this neuron to learn from the input space.

8.1.4.1 Local Space

Figure 8-6 and Figure 8-7 are examples showing a neuron (28) that divides a local input space by applying a combination of "unary kernels" and "binary kernels" operations. The combination is:

$$\text{Output} = \text{NADD}\,[\text{NHALF}\,(X), \text{NLOG2}\,(Y)] \qquad \textbf{(8-7)}$$

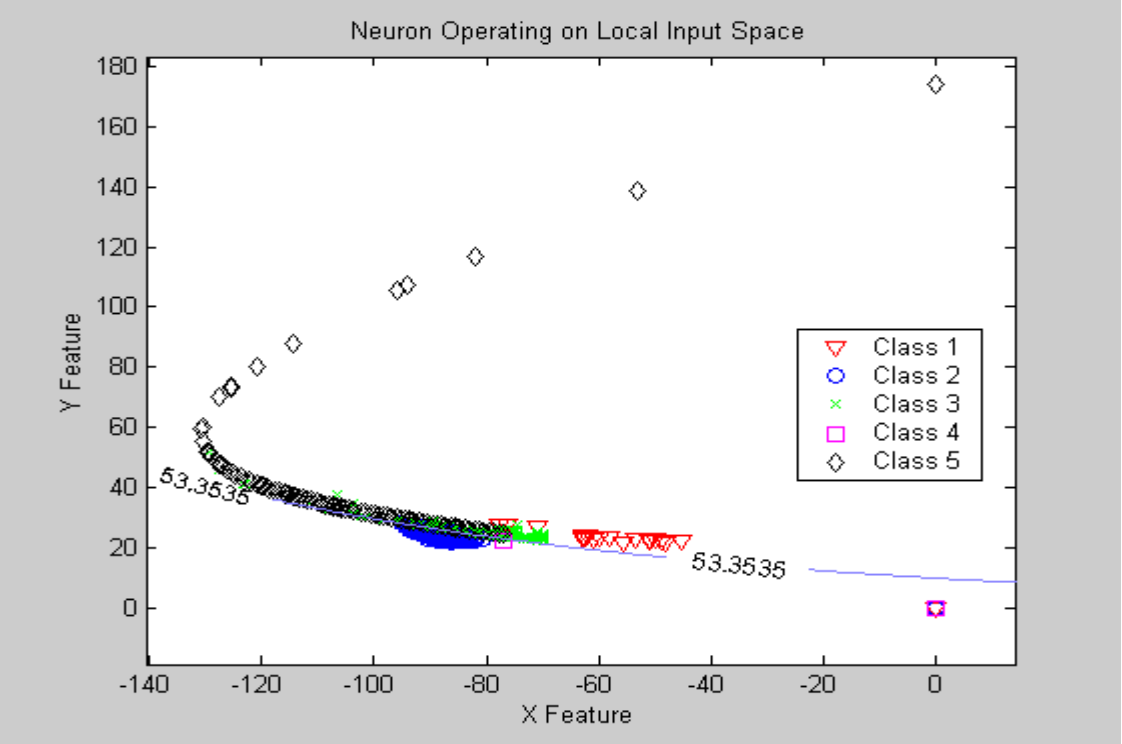

Figure 8-6 Neuron Dividing Local Input Space

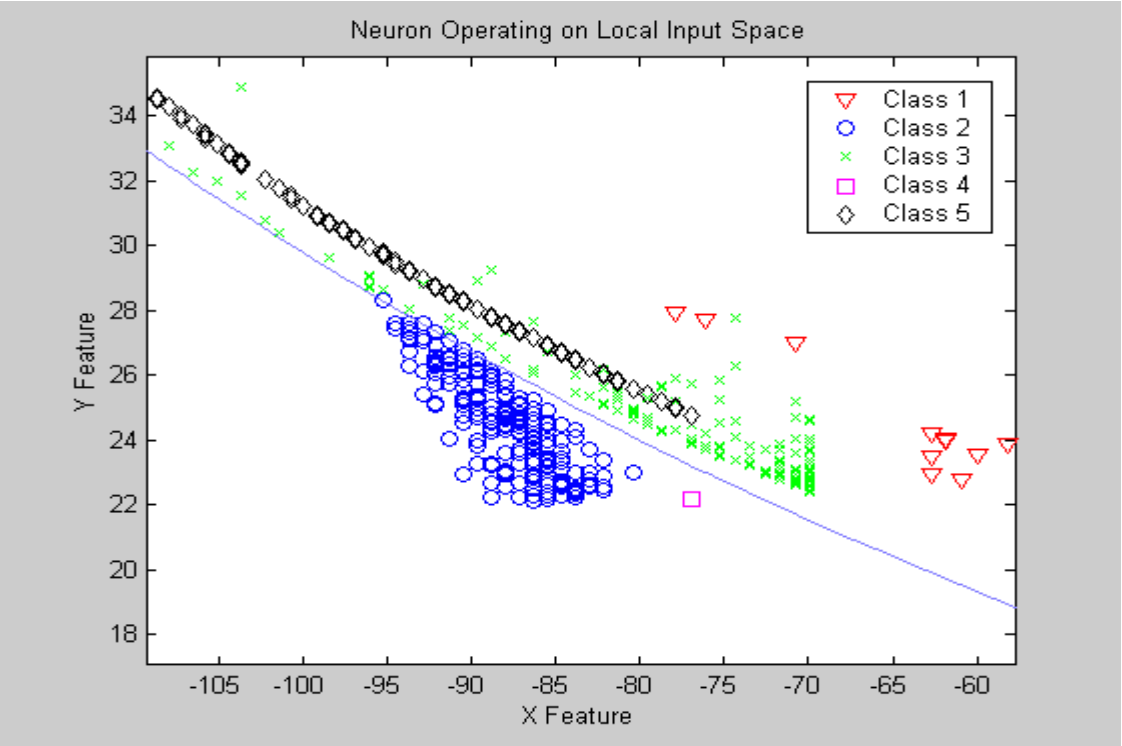

Figure 8-7 Neuron Dividing Local Input Space (Zoom In)

Since both information deficiencies of neuron (28) are less than voting threshold as shown in Table 8-2, both SOT and SOTI are set. This neuron is qualified to participate in the final classification for a testing data which either passes or fails its threshold. Probabilities of the correct classification are calculated, and they are shown in Table 8-3.

Table 8-2 Output Information Deficiencies of Neuron (28)

Output Information Deficiency (Passing Threshold)	Output Information Deficiency (Not Passing Threshold)
0.1425	0.0017

Table 8-3 Correct Classification Probabilities of Neuron (28)

	Class 1	Class 2	Class 3	Class 4	Class 5
Pass Threshold	0.0253	0.0013	0.2527	0	0.7207
Does Not Pass Threshold	0	0.9950	0	0.0050	0

Table 8-2 and Table 8-3 illustrate the close relationship between an output information deficiencies and the probabilities of correct classification. As the output information deficiency decreases, it indicates how much this neuron knows about its local input space. For neuron (28), the output information deficiency of data that did not pass threshold is 0.0017, and it has 0.995 probability of a correct classification for Class 2. This is also demonstrated graphically in Figure 8-6 and Figure 8-7 where Class 2 is efficiently separated from the rest of the classes by the neuron's threshold line. Therefore, at later testing classification, a data, which belongs to neuron (28)'s local input space and does not satisfy its threshold value, is voted with high probability as being Class 2.

8.1.4.2 Original Space

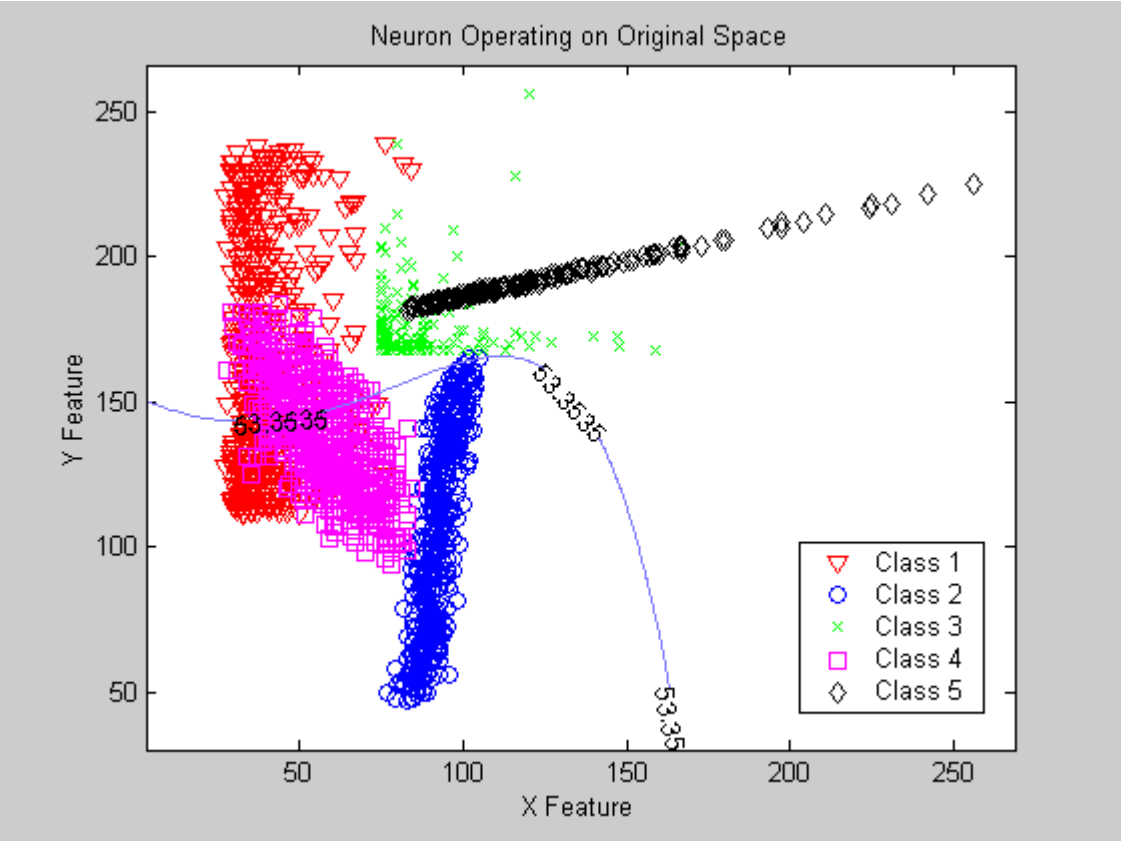

Figure 8-8 Cutting the Original Input Space

More and more complex transformation functions grow along with the increasing layers of neurons. As discussed in Chapter 5, a neuron applies its different transformation functions to the local input space. These inputs may have been subjected to many transformations of the original input space after several neurons processing. For example, neuron (28) cuts the original space as shown in Figure 8-8. It basically makes the transformation function more complicated which corresponds to

the original space by applying its own operation on the two inputs. The relationship between final transformation of this neuron and the original space at Figure 8-8 can be obtained by tracing back all the operations that have been applied to the local inputs (or a single input). The output of neuron (28) expressed by input variables in the original space is as follows:

$$\text{Output (28)} = \frac{(\exp X) - X}{4} + \log\left\{\frac{2X + Y + 8\exp X}{32}\right\} \qquad \textbf{(8-7)}$$

Figures 8-9 and 8-10 show that having help from another neuron, most of Class 2 can be divided from the rest of the classes. Classification is achieved if all neurons work together.

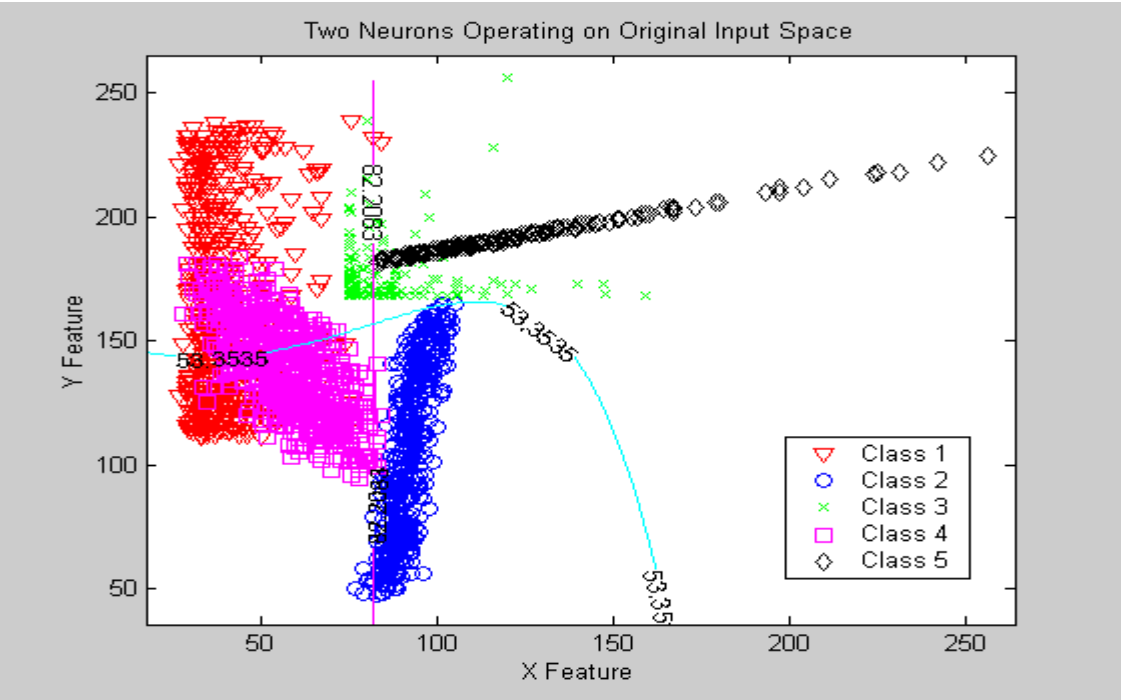

Figure 8-9 Two Neurons Separating Class 2

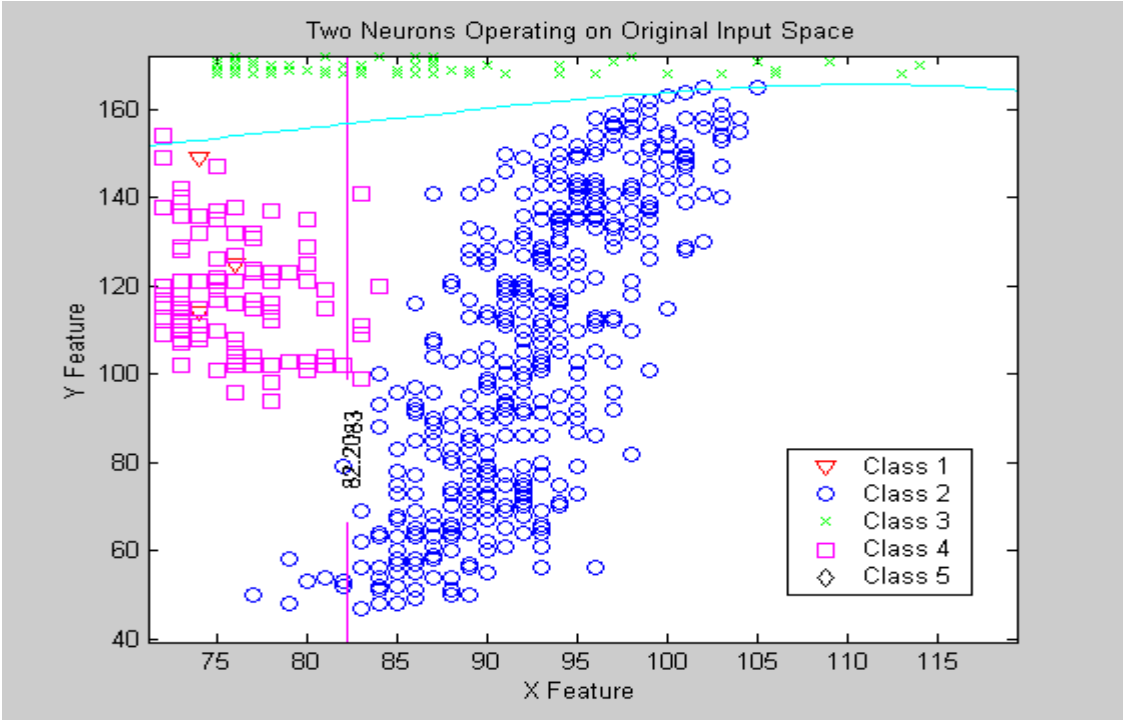

Figure 8-10 Two Neurons Separating Class 2 (Zoom In)

8.1.5 Neuron Testing

After learning, SOLAR is prepared to classify any data that is within the learning space. A testing data is chosen from the training data to demonstrate how the network performs the final classification based on their probabilities of correct classification. The chosen test data is one of the training data from Class 2. Values of its input features are

X = 097

Y = 138

According to the TCI, SOT, and SOTI, neurons that participate in voting are

Neuron (1)
Neuron (2)
Neuron (3)
Neuron (4)
Neuron (5)
Neuron (8)
Neuron (10)
Neuron (16)
Neuron (21)
Neuron (24)
Neuron (28)

Correct classification probabilities of each neuron corresponding to their threshold values are illustrated in Table 8-4.

Table 8-4 Probabilities of Correct Classification

	Neuron Number					
	1	2	3	4	5	8
Class 1	0.0352	0.0055	0.2399	0.0199	0	0
Class 2	0.1586	0.3341	0.2809	0.2109	0.9850	0.6073
Class 3	0.2093	0.1480	0	0.1994	0	0.3897
Class 4	0	0.0903	0.4792	0.0010	0.0150	0.0030
Class 5	0.5969	0.4221	0	0.5687	0	0

	Neuron Number				
	10	16	21	24	28
Class 1	0	0.0012	0.0073	0	0
Class 2	0.5106	0.2442	0.7806	0.9850	0.9950
Class 3	0.4894	0.0948	0	0	0
Class 4	0	0.0012	0.2121	0.0150	0.0050
Class 5	0	0.6586	0	0	0

As discussed in Chapter 7 – Final Classification, probability of being a class of a testing data can be calculated with a weight function as described in equation (8-8).

$$W_c = 1 - \frac{P_{cc_{MAX}} + \left\{ \left[\sum_{i=1}^{n} \left(1 - P_{cc_i}\right) \right] \left(1 - P_{cc_{MAX}}\right) \right\}}{\sum_{i=1}^{n} \left(\frac{1}{1 - P_{cc_i} + \varepsilon} \right)} \qquad \textbf{(8-8)}$$

where $P_{cc_{MAX}}$ = maximum P_{cc} of all SONNOs' "voting" for class c

P_{cc_i} = P_{cc} of each "vote" for class c

n = number of "votes" for class c

ε = small number preventing division by zero

As ε = 0.001, the weights of different classes for this particular testing data are calculated; and they are shown in Table 8-5. The final voting result suggests that this testing data belongs to Class 2, which is a correct classification.

Table 8-5 Probability Estimates for Different Classes

Class 1	Class 2	Class 3	Class 4	Class 5
0.2645	0.9967	0.6018	0.5206	0.7780

As all the training data were used to test the performance of SOLAR, probabilities of classification for different classes are shown in Table 8-6. Table 8-6 suggests that SOLAR is more confident to classify Class 2, Class 3, and Class 5 compared to others.

Table 8-6 Probabilities of Classification

	Data Classified as				
	Class 1	Class 2	Class 3	Class 4	Class 5
Data from Class 1	0.8171	0	0.0040	0.1769	0.0020
Data from Class 2	0	0.9977	0.0023	0	0
Data from Class 3	0	0	0.9263	0	0.0737
Data from Class 4	0.1481	0.0103	0	0.8416	0
Data from Class 5	0	0	0	0	1

The reason SOLAR has inefficient performance in classifying Class 1 and Class 4 is because Class 1 has large overlapping area with Class 4 as illustrated in Figure 8-11. Information deficiency calculation can result in a good separation between two groups. After a few layers of neurons, different classes should be identified by the network. However, if a large number of different members from different classes overlap, SOLAR can never accurately classify these members even with many layers of neurons because they are statically non-separable, and the probabilities of correct classification of these classes are always low. The algorithm performance in such case is limited by Bayesian probabilities.

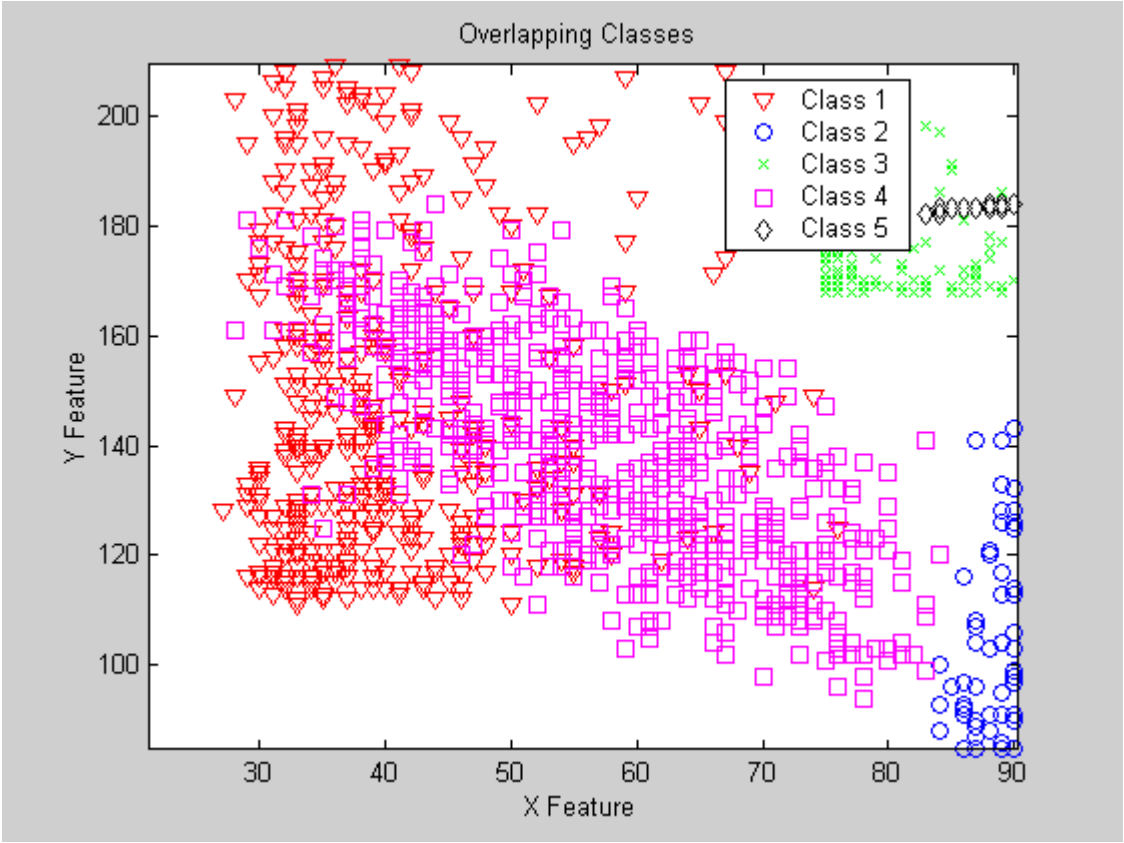

Figure 8-11 Overlapping Classes

8.2 Credit Card Dataset

Credit cards have been widely used and have become very popular around the world. According to Master Card International, the number of credit cards in Asia/Pacific region alone grew from 30.9 million to 72.6 million between 1990 and 1998. The increasing number of applications create a huge task for processing them, which is impossible to handle by hand. Artificial neural networks can be used to facilitate this task.

8.2.1 Dataset Background

The credit card approval data in Australia (Credit Screening Database) was acquired from the University of California at Irvine (ICS, UCI, 1995, December). The dataset has 690 instances, 16 features including class attribute, and it is divided into 2 classes, which are approve and reject represented by "+" and "–". The dataset contains numbers of credit card applications, which feature names and values have been replaced by symbols to protect the individuals' privacy of the data. The content of the dataset is described in Table 8-7. Missing values and class distribution are presented in Table 8-8.

Table 8-7 Credit Card Dataset Information

Number of instances	690
Number of attributes	15 + Class attribute
Attribute information	
Attribute 1	b,a
Attribute 2	Continuous values (13.75-80.25)
Attribute 3	Continuous values (0-28)
Attribute 4	u,y,l,t
Attribute 5	g,p,gg
Attribute 6	c,d,cc,i,j,k,m,r,q,w,x,e,aa,ff
Attribute 7	v,h,bb,j,n,z,dd,ff,o
Attribute 8	Continuous values (0-28.5)
Attribute 9	t,f
Attribute 10	t,f
Attribute 11	Continuous values (0-67)
Attribute 12	t,f
Attribute 13	g,p,s
Attribute 14	Continuous values (0-2000)
Attribute 15	Continuous values (0-100000)
Attribute 16	+,- (Class attribute)

Table 8-8 Missing Data and Class Distribution of Credit Card Dataset

Missing attribute	37 cases (5%) has one or more missing values
Attribute 1	12
Attribute 2	12
Attribute 4	6
Attribute 5	6
Attribute 6	9
Attribute 7	9
Attribute 14	13
Class distribution	
+	307 (44.5%)
-	383 (55.5%)

The ranges of numerical values are listed in Table 8-7, and their distributions are shown in Figure 8-12. All plots are illustrated in their original scales except for Attribute 15 because the majority of its samples are extremely small. Applying logarithm scale helps to present its distribution.

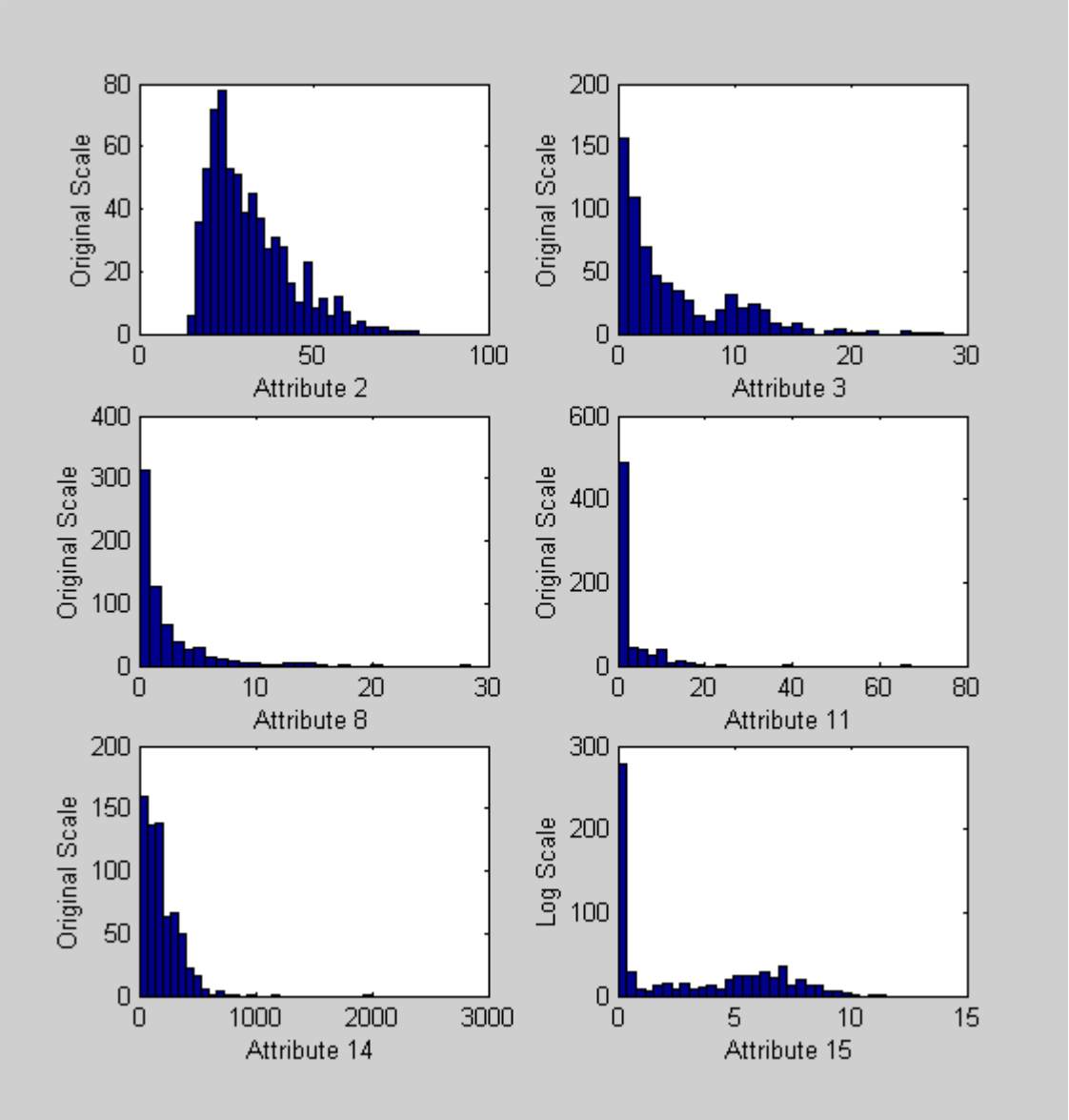

Figure 8-12 Numerical Values Distributions of Credit Card Dataset

8.2.2 Missing Data and Symbolic Values

There are missing data and symbolic values presented in this credit card dataset due to the protection of privacy of the individuals and the nature of the dataset. These data and values must be replaced by meaningful numerical values so that SOLAR can perform transformation functions on the data. The replacement values can be obtained by applying the methods discussed in section 4.3.1.1 and 4.3.1.2. The result of assigning numerical values to symbolic values are shown in Table 8-9.

Table 8-9 Symbolic Values Assignment for Credit Card Dataset

b, a	0.9189, 1.0000
u, y, l, t	1.0000, 1.0081, 12.6678, 0
g, p, gg	0.0784, 1.0081, 12.6678
c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff	1.0000, 1.1453, 1.1053, 1.1479, 1.4554, 1.1103 1.2030, 2.1342, 1.0121, 1.0602, 1.2539, 1.5053, 1.0444, 1.3118
v, h, bb, j, n, z, dd, ff, o	1.0000, 1.0997, 1.3572, 1.6518, 1.8488, 2.9303, 1.6381, 1.3340, 6.2384
t, f	1.0000, 0.8646
t, f	1.0000, 0.8520
t, f	1.0000, 0.9135
g, p, s	1.0000, 15.7278, 1.2447

8.2.3 Network Parameters

All parameters are set the same as for the two-dimensional problem discussed in section 8.1 except number of neurons per layer and the number of layers since this problem has more input features. Since the increased number of features raises the complexity of the problem, more neurons are required in generating a reliable result. All parameters are shown as follows:

- Input parameters:
    1. Number of input(s) from the nearest neighbors = 1
    2. Number of input(s) from the next nearest neighbors = 1
    3. Number of input(s) from remote neighbors = 1
- Number of connection(s) to TCI = 3
- Voting threshold = 0.9
- Subspace selection probability = 0.1
- Information deficiency threshold = 0.1
- Number of layers = 17
- Number of neurons per layer = 15

8.2.4 Simulation Results

In order to compare a result from SOLAR with previous works, the same experimental setup as used in previous experiments (Michie, Spiegelhalter & Taylor, 1994) was introduced in this simulation. The setup of the previous experiments used cross-validation technique (Ston, 1974) to divide the dataset randomly into n mutually exclusive data groups with equal size. The number of times of training and testing process is based on the number of data groups n. During each training and testing process, one of the groups is selected as testing data, while the rest (n-1) of the groups are training data. The same testing group will not be selected as testing data again if it has been chosen before, and each data group will be selected as testing data group only once. The error rate is the average error rate of the n groups. This can eliminate the statistical biases, and the error rate can be estimated efficiently. Similar to the previous experiments, n was set to 10 in this simulation.

Since SOLAR is a self-organizing network, each network with different pre-wiring can result in a different performance. In order to observe and estimate the average performance of SOLAR, nine identical networks with different pre-wiring were generated. As shown in Figure 8-13, these nine networks were assumed working in parallel, and a final majority voting was performed. Table 8-10 demonstrates the result of each network while Table 8-11 shows the result of the average performance after the majority voting.

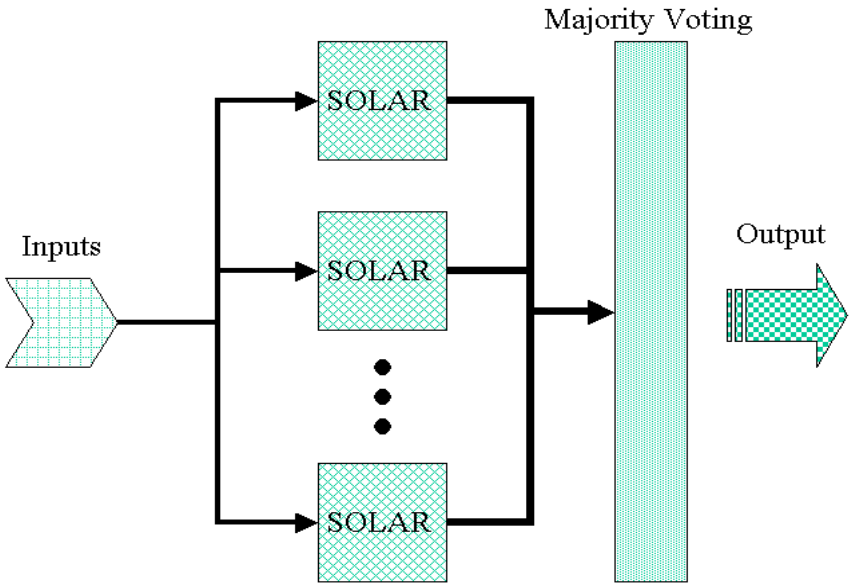

Figure 8-13 Majority Voting with Parallel SOLARs

Table 8-10 Performance of Each SOLAR

	SOLAR Number								
	N1	N2	N3	N4	N5	N6	N7	N8	N9
Mean	0.8609	0.8609	0.8594	0.8565	0.8580	0.8551	0.8261	0.8696	0.8594
SD	0.0717	0.1025	0.0820	0.0922	0.0717	0.0717	0.3689	0.0820	0.0820
Min	0.7971	0.7971	0.7971	0.7826	0.7971	0.7971	0.4203	0.8116	0.7971
Max	0.8986	0.9420	0.9130	0.9130	0.8986	0.8986	0.9420	0.9275	0.9130

Table 8-11 Average Performance after Majority Voting (Credit Card)

Mean	0.8638
SD	0.0820
Min	0.7971
Max	0.9130

Setting different values to the voting threshold can help to optimize the voting result. Figure 8-14 and Table 8-12 demonstrate the process of searching for the optimized voting threshold value for this particular dataset.

Table 8-12 Voting Thresholds and Error Rates (Credit Card)

Voting Threshold	0.1	0.15	0.2	0.5	0.9
Error Rate	0.1420	0.1333	0.1362	0.1377	0.1362

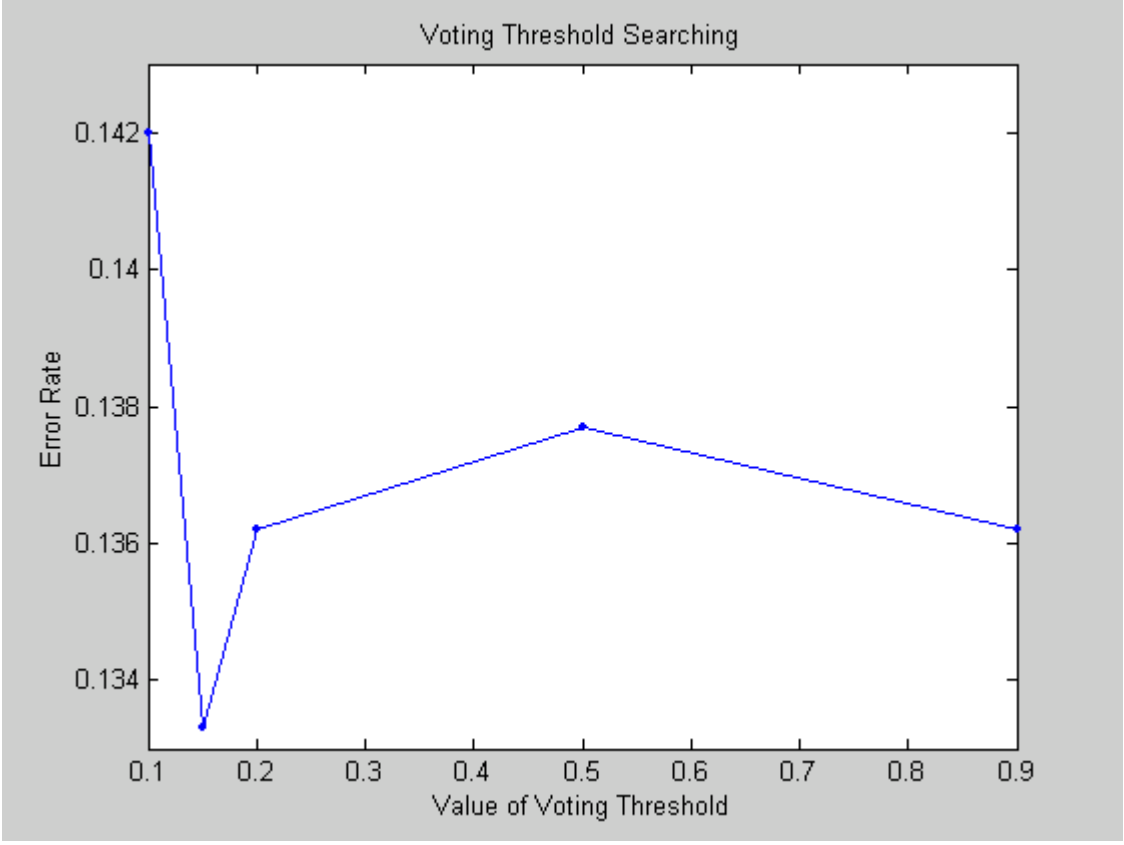

Figure 8-14 Voting Threshold Searching (Credit Card)

Figure 8-15 illustrates the self-organized SOLAR for this credit card problem. The confusion matrix of the better result (voting value = 0.15) is shown in Table 8-13. This result is compared with other algorithms, and Table 8-14 shows the comparison. SOLAR performed fairly well among all the algorithms. Although it does not compete with the decision tree algorithm CAL5, it has the best performance among all artificial neural networks, which are highlighted in Table 8-14.

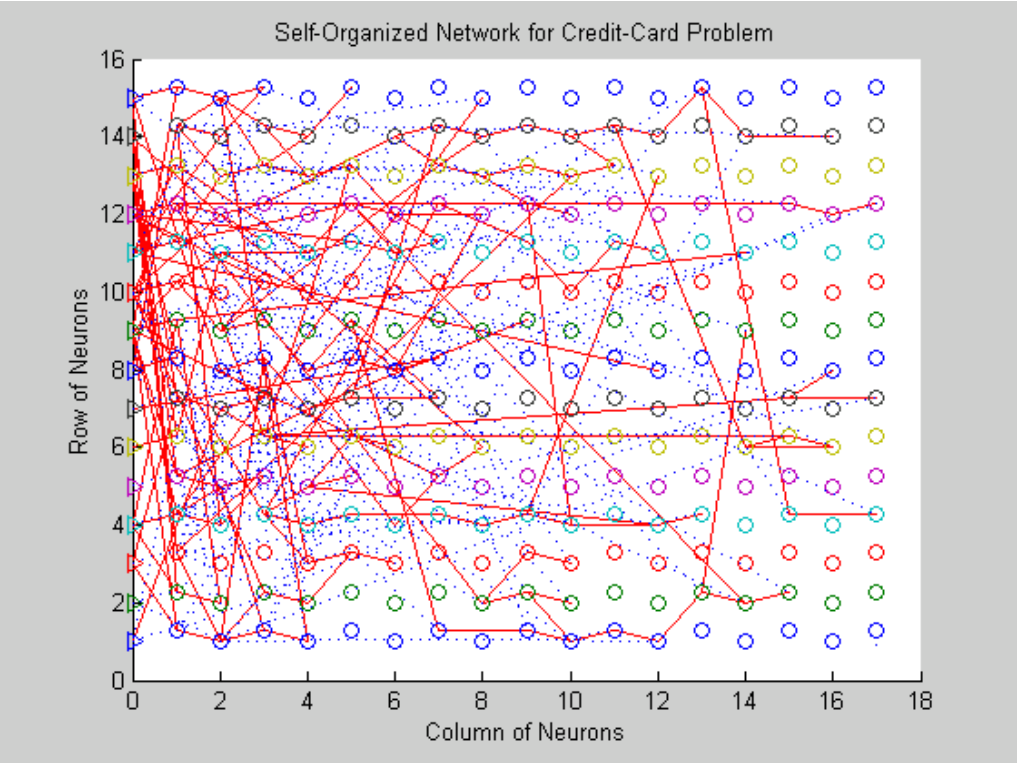

Figure 8-15 Self-Organized Network Structure for Credit Card Problem

Table 8-13 Probabilities of Classification (Credit Card)

	Data Classified as	
	Class 1	Class 2
Data from Class 1	0.9349	0.0651
Data from Class 2	0.1880	0.8120

Table 8-14 Comparison Result for Credit Card Approval Dataset

Algorithm	Error Rate
Cal5	0.131
SOLAR	0.1333
Itrule	0.137
Discrim	0.141
Logdisc	0.141
DIPOL92	0.141
CART	0.145
RBF	0.145
CASTLE	0.148
NaiveBay	0.151
IndCART	0.152
Backprop	0.154
C4.5	0.155
SMART	0.158
Baytree	0.171
k-NN	0.181
NewID	0.181
AC^2	0.181
LVQ	0.197
ALLOC80	0.201
CN2	0.204
Quadisc	0.207
Default	0.440
Kohonen	Failed

8.3 Adult Income Dataset

Besides credit card approval, potential customer analysis is an example of another real world application to which banks or financial companies can apply artificial neural networks. These analyses help companies understand their current or potential customers and to react properly. Personal income certainly is one of the information a company is interested to investigate. According to The Hong Kong and Shanghai Banking Corporation Limited (HSBC) 2001 annual review, their new mortgage loans increased by 56 percent in value in UK while there was a 46 percent new mortgage business volume increase in Hong Kong, most of which were related to refinancing. Knowing the income information for current or potential customers can help banks or financial companies to provide "right" loan packages to target customers before other competitors do.

8.3.1 Dataset Background

This adult income dataset (Adult Database) was obtained from the University of California at Irvine (ICS, UCI, 1995, December). The dataset contains two sets of data. One is training data, which has 32561 instances while another one is testing data, which has 16281 instances. Both have 15 features including class attribute, and they are also divided into 2 classes. The dataset contains both symbolic values such as gender, race, etc., and missing data. The content of the dataset is described in Table 8-15.

Table 8-15 Adult Income Dataset Information

Number of instances (Training data)	32561
Number of instances (Testing data)	16281
Number of attributes	14 + Class attribute
Number of missing values	7%
Attribute information	
Age	Continuous values (17-90)
Work-Class	Symbolic values (8)
Fnlwgt	Continuous values (12285-1490400)
Education	Symbolic values (16)
Education-Num	Continuous values (1-16)
Marital-Status	Symbolic values (7)
Occupation	Symbolic values (14)
Relationship	Symbolic values (6)
Race	Symbolic values (5)
Sex	Symbolic values (2)
Capital-Gain	Continuous values (0-99999)
Capital-Loss	Continuous values (0-4356)
Hours-Per-week	Continuous values (1-99)
Native-Country	Symbolic values (41)
Class	>50K, <=50K (Class attribute)
Class distribution	
>50K	23.93%
<=50K	76.07%

The ranges of numerical values are listed in Table 8-15, and their distributions are illustrated in Figure 8-16. All features are plotted with all samples in their original scales except for Capital-Gain and Capital-Loss. Because the majority (more than 40,000) samples of both features are zero, only non-zeros samples are shown in order to obtain better representations of their distributions.

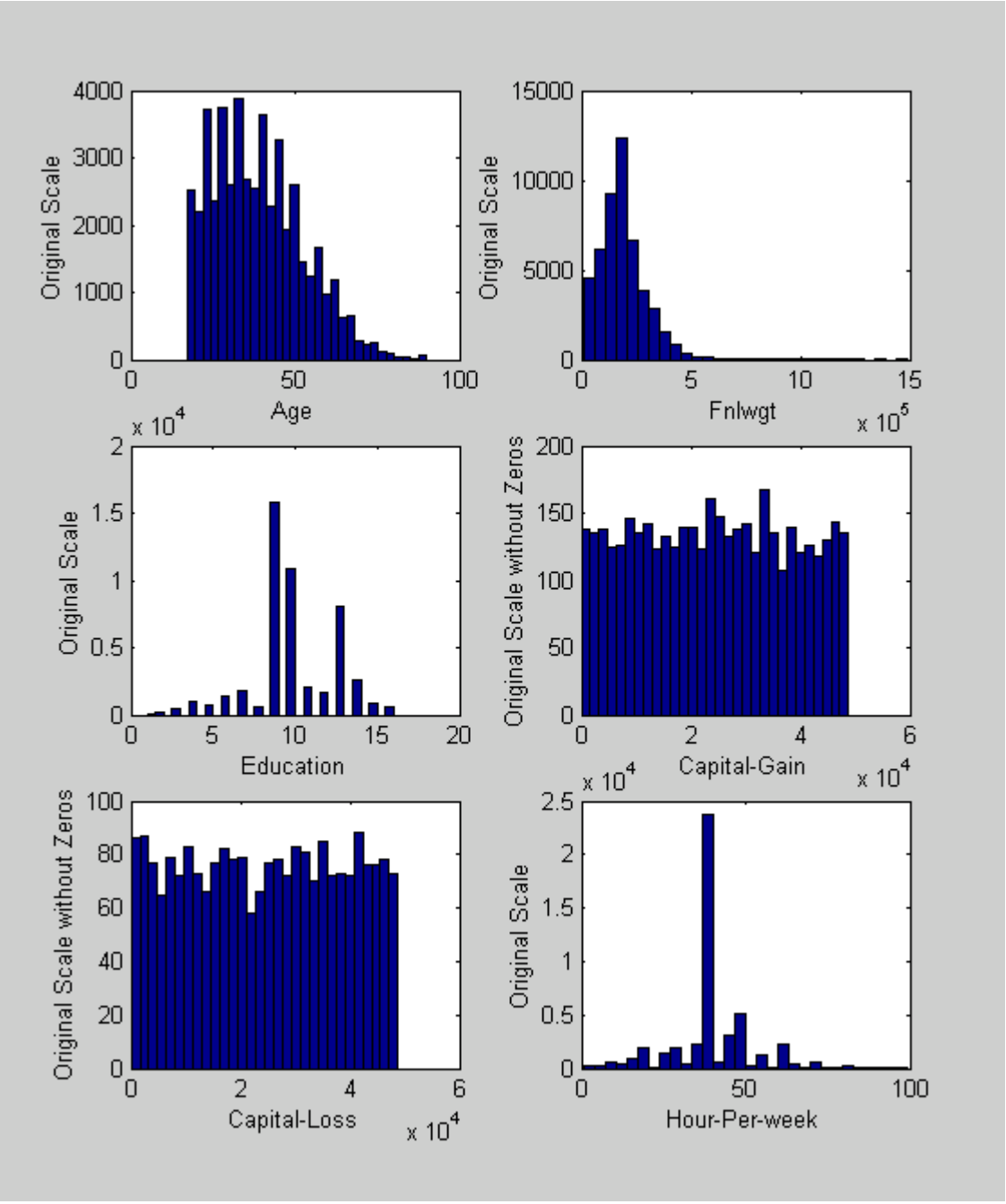

Figure 8-16 Numerical Values Distributions of Adult Income Dataset

8.3.2 Missing Data and Symbolic Values

There are missing data and symbolic values present in this adult income dataset due to the nature of the dataset. These data and values must be replaced by meaningful numerical values so that SOLAR can perform transformation functions on the data. The replacement values can be obtained by applying the methods discussed in section 4.3.1.1 and 4.3.1.2. The result of symbolic values assignment is shown in Table 8-16.

Table 8-16 Symbolic Values Assignment for Adult Income Dataset

Work-Class	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked	1.0000, 1.1865, 1.4286, 1.3889, 1.2063, 1.2778, 20.5992, 33.7778
Education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool	1.0000, 0.7691, 0.5385, 0.6924, 1.1538, 0.9232, 0.8462, 0.3847, 0.3076, 0.6153, 1.0771, 0.1538, 0.4615, 1.2309, 0.2309, 0.0771
Marital-Status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse	1.0000, 0.9977, 0.8499, 1.0271, 1.1443, 1.1865, 5.1813
Occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical,	1.0000, 0.8775, 0.8076, 0.9156, 1.0025, 1.0432, 0.8216, 0.8483, 0.8908,

	Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces	0.9086, 0.8971, 1.0025, 0.9829, 5.4006
Relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried	1.0000, 0.7693, 0.9608, 0.9157, 0.9106, 0.9177
Race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black	1.0000, 1.2086, 1.4746, 1.5877, 1.0382
Sex	Female, Male	1.0000, 1.0300
Native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands	1.0000, 2.1185, 1.3416, 1.0854, 1.2397, 1.2066, 2.4105, 1.3058, 1.4160, 1.6804, 1.3554, 1.3196, 1.2810, 1.7052, 2.3471, 1.1157, 1.2727, 1.3802, 1.2672, 1.2617, 0.8788, 1.2893, 1.7934, 2.0551, 1.1653, 2.3085, 1.6171, 1.7355, 1.3361, 1.3471, 2.9614, 1.1212, 1.5758, 2.6198, 2.1625, 2.2948, 1.0083, 2.0606, 1.6804, 2.0799, 25.4160

8.3.3 Network Parameters

Almost all parameters are set the same as in the two-dimensions problem discussed in section 8.1 except number of layers, number of neurons per layer, and number of TCI connections. Different numbers of layers with two numbers of TCI are chosen to demonstrate different performances of SOLAR. All parameters are shown as follows:

- Input parameters:
    1. Number of input(s) from the nearest neighbors = 1
    2. Number of input(s) from the next nearest neighbors = 1
    3. Number of input(s) from remote neighbors = 1
- Number of connection(s) to TCI = 3 and 5
- Voting Threshold = 0.9
- Subspace selection probability = 0.1
- Information deficiency threshold = 0.1
- Number of layers = 16, 30, 50, and 70.
- Number of neurons per layer = 14

8.3.4 Simulation Results

Since SOLAR is a self-organizing network, each network with different pre-wiring can result in a different performance. In order to observe and estimate the average performance of SOLAR, nine identical networks with different pre-wiring were

generated. As shown in Figure 8-11, these nine networks were assumed working in parallel, and final majority voting was performed. Table 8-17 demonstrates the result of each network while Table 8-18 shows the result of average performance after the majority voting.

Table 8-17 Performance of Each SOLAR

TCI = 3				
	16 Layers	30 Layers	50 Layers	70 Layers
N1	0.7877	0.7777	0.8178	0.8270
N2	0.8012	0.8300	0.7952	0.8399
N3	0.8199	0.8304	0.8230	0.8357
N4	0.8030	0.8309	0.8082	0.8248
N5	0.8055	0.7925	0.8434	0.8300
N6	0.7925	0.8207	0.8366	0.8142
N8	0.8027	0.8194	0.8058	0.8318
N8	0.8070	0.8176	0.8379	0.8203
N9	0.8111	0.8296	0.8313	0.8292
TCI = 5				
	16 Layers	30 Layers	50 Layers	70 Layers
N1	0.8031	0.8241	0.8073	0.8304
N2	0.8025	0.8110	0.8272	0.8318
N3	0.8027	0.8242	0.8181	0.8388
N4	0.8054	0.8360	0.8065	0.8369
N5	0.8012	0.8229	0.8423	0.8297
N6	0.8022	0.8311	0.8439	0.8119
N8	0.8098	0.8045	0.8388	0.8249
N8	0.8028	0.8071	0.8165	0.8159
N9	0.8125	0.8273	0.8125	0.8371

Table 8-18 Average Performance after Majority Voting (Adult Income)

TCI = 3			
16 Layers	30 Layers	50 Layers	70 Layers
0.8036	0.8206	0.8294	0.8297
TCI = 5			
16 Layers	30 Layers	50 Layers	70 Layers
0.8040	0.8225	0.8313	0.8331

The result in Figure 8-17 suggests that SOLAR with more layers performs better. In fact, SOLAR with more layers and more TCI inputs results in a classification improvement for this particular dataset. These outcomes are expected since increasing number of neurons raises the chance of applying different transformations on the inputs. Moreover, with more TCI inputs, a neuron can check more subspaces and select the one in which the neuron has a better reduction of information deficiency. In addition, Figure 8-16 also suggests that little improvement is gained as the number of layers keeps increasing after 50 layers while the number of TCI inputs remains the same.

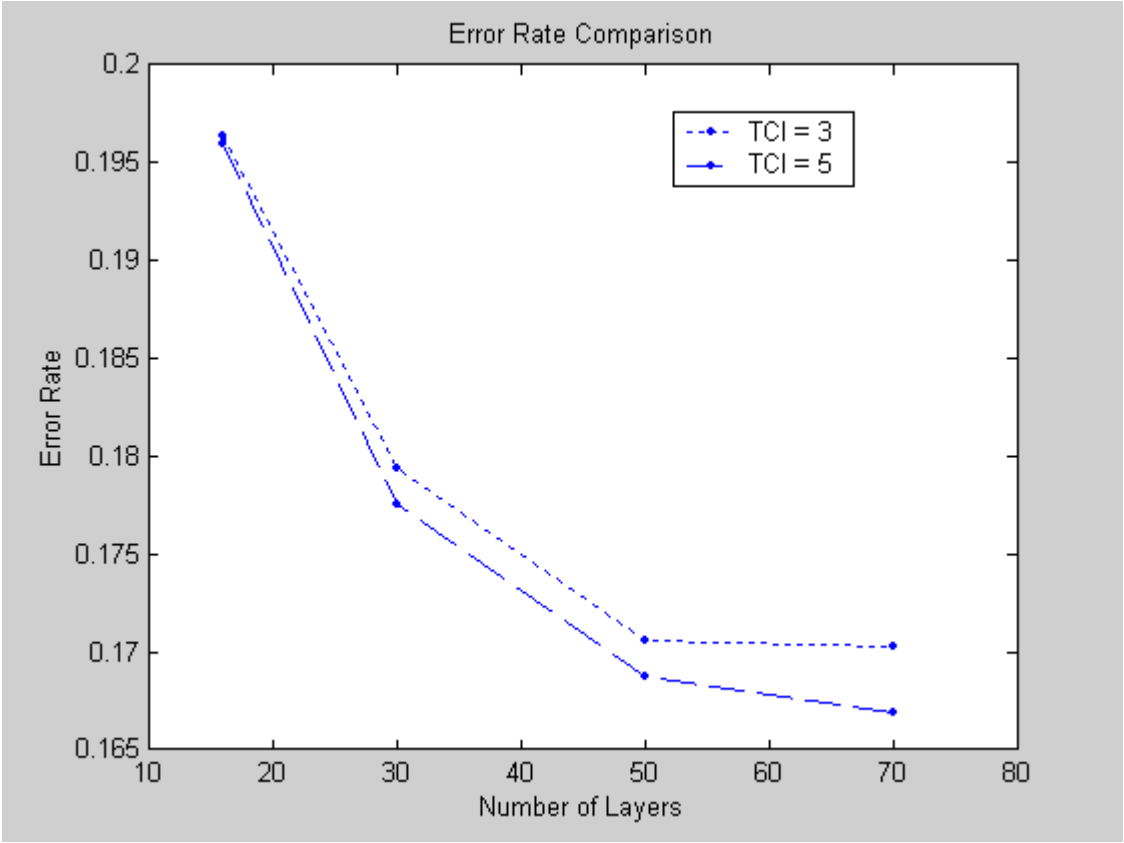

Figure 8-17 Error Rate Comparison with Number of Layers and TCI Inputs

Setting different values to the voting threshold can optimize the voting result. Figure 8-18 and Table 8-19 demonstrate the process of searching for the optimized voting threshold value with only 17 layers of neurons for this particular dataset.

Table 8-19 Voting Thresholds and Error Rates (Adult Income)

Voting Threshold	0.1	0.15	0.2	0.3	0.5	0.9
Error Rate	0.1671	0.1482	0.1514	0.1537	0.1665	0.1964

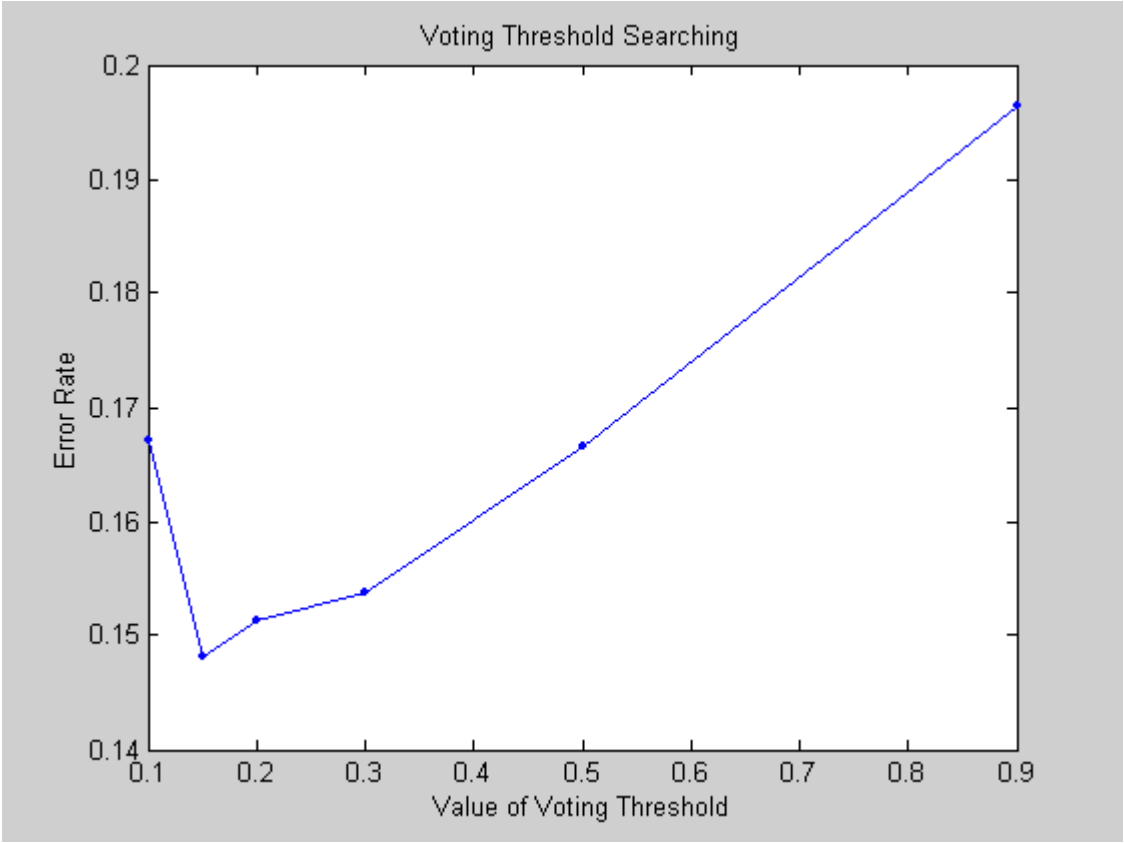

Figure 8-18 Voting Threshold Searching (Adult Income)

Figure 8-19 illustrates the self-organized SOLAR with 17 layers for this adult income problem. The confusion matrix of the better result is shown in Table 8-20. This result is compared with other algorithms and Table 8-21 shows the comparison. Although SOLAR does not perform as well as the best algorithms, it is the only artificial neural network on the list, and it was not designed for any specific classification and recognition tasks.

Figure 8-19 Self-Organized Network Structure for Credit Card Problem

Table 8-20 Probabilities of Classification (Adult Income)

	Data Classified as	
	Class 1	Class 2
Data from Class 1	0.5785	0.4215
Data from Class 2	0.0637	0.9363

Table 8-21 Comparison Result for Adult Income Dataset

Algorithm	Error Rate
FSS Naïve Bayes	0.1405
NBTree	0.1410
C4.5-auto	0.1446
IDTM (Decision table)	0.1446
HOODG / SOLAR	0.1482
C4.5 rules	0.1494
OC1	0.1504
C4.5	0.1554
Voted ID3 (0.6)	0.1564
CN2	0.1600
Naïve-Bayes	0.1612
Voted ID3 (0.8)	0.1647
T2	0.1687
1R	0.1954
Nearest-neighbor (3)	0.2035
Nearest-neighbor (1)	0.2142
Pebls	Crashed

Chapter 9
9. Conclusion and Future Work

9.1 Conclusion

This thesis demonstrates the MATLAB software simulation of Self-Organizing Learning Array (SOLAR), which introduces a new method in machine learning design. This software design is aimed for future hardware realization, which will be eventually implemented in a Very Large Scale Integration (VLSI) circuit. It is mainly used to test and design the future hardware structure.

The first part of the thesis explains the biological neural network structure, where processing cells are usually locally connected. This idea was implemented in SOLAR organization and pre-wiring. Then, different inputs and outputs were discussed, and threshold clock input (TCI) was introduced. Methods for computing missing data and symbolic values were presented. Potential arithmetic operations were shown and also demonstrated graphically. Applying multiple functions was suggested since it could result in a more complicated cutting of the input space. Learning and self-organizing principles were then illustrated by introducing information index. This was followed by the final voting with a weight function. SOLAR was simulated with two real world

problems, credit card approval and adult income analysis. Although SOLAR did not perform the best among all algorithms, it shows its abilities in classifying while it was not designed particularly for any classification or recognition, and has better performances compared to all other artificial neural network algorithms. In summary, the performance of SOLAR was satisfactory, and this thesis demonstrated its ability to self organize and learn.

9.2 Future Work

The implemented weighting function was based on the estimation of probability of correct classification (7-1). This estimates true values of probabilities with the confidence interval which is a function of the number of training samples in a given subspace. When the number of points in a subspace is small, the error resulting from the confidence interval is large, and the weighting function may wrongly select a less reliable result. Another weighting function based on interval analysis should be investigated as an alternative to (7-1). Additional discussion of this issue is on Appendix A.

This thesis only covers the MATLAB software design and simulation of SOLAR. The SOLAR project will be carried on to the next level, which is Very High Speed Integrated Circuit Hardware Description Language (VHDL) simulation and hardware realization. Before any further hardware implementation is done, SOLAR must be

simulated using VHDL in order to address hardware design problems and other difficulties. It then can be downloaded on FPGA chips for further simulation and prototyping. Since resources of a single FPGA chip are limited, an FPGA machine, which is specially designed and built with multi-FPGA chips, may be required so that enough resources are guarantied. VLSI circuit design of SOLAR and chip fabrication will be the last state of the project.

Reference

1. Cichocki, A., and Unbehauen, R., (1993), Neural Networks for Optimization and Signal Processing, John Wiley & Sons, Inc., New York.
2. Dayhoff, J. E. (1990), Neural Network Architectures : An Introduction, Van Nostrand Reinhold, New York.
3. Dowling, J. E. (1998), Creating Mind : How the Brain Works, W.W.Norton & Company, Inc., New York.
4. Ennett, C. M., Frize, M., and Walker, C. R. (2001), "Influence of Missing Values on Artificial Neural Network Performance", *Medinfo*, Vol. 10, pp. 449-53.
5. Fraser, N. (1998, September), The Biological Neuron, Available http://vv.carleton.ca/~neil/neural/neuron-a.html
6. Hassoun, M. H. (1995), Fundamentals of Artificial Neural Networks, The Massachusetts Institute of Technology Press, Massachusetts.
7. Information & Computer Science (ICS), University of California at Irvine (UCI). (1995, December), Machine Learning Repository, Available FTP: Hostname: ftp.ics.uci.edu Directory: /pub/machine-learning-databases/

8. Mahalanobis, P. C. (1936), “On the generalized distance in statistics”, *Proceedings National Institute of Science of India*, 2, 49-55.
9. Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994), Machine Learning, Neural and Statistical Classification, Ellis Horwood Limited, London, U.K.
10. Newcombe, R. (1998), “Two-Sided Confidence Intervals for the Single Proportion: Comparison of Seven Methods”, *Statistics In Medicine*, Vol. 17, pp. 857-872.
11. Purves, D. (1994), Neural Activity and the Growth of the Brain, Cambridge University Press, Cambridge, Great Britain.
12. Starzyk, J. (2000), SOLAR Project, Available http://www.ent.ohiou.edu/%7Ewebcad/proj/solar/index.html
13. Stone, M. (1974), “Cross-validatory choice and assessment of statistical predictions”, *Journal of the Royal Statistical Society*, 36, 111-147.

Appendix A

Confidence Interval Discussion

When the number of points in a subspace is small, the error resulting from the confidence interval is large, and the weighting function may wrongly select a less reliable result. Confidence interval calculation should be introduced to improve the reliability of the weighting results. The following example illustrates how confidence interval analysis improves the correct classification result by calculating the mean value of the interval and using this value to decide to which class a particular incoming data belongs.

Let us denote the true class probability of a voting neuron by P_x. This probability is an unknown, and it is estimated based on proportion P_c. In order to estimate the unknown probability P_x under the observation P_c, a statistical experiment was conducted. In the experiment, probability P_x was set to a specified value and 10 points were generated 2000 times. A class A_x with probability P_x contained all uniformly generated points from [0,1] interval whose values were larger than $1-P_x$. At each run (of 2000), the number of points out of 10 points that were generated and belonged to the class A_x was counted. If the count was equal to P_c (in this case 9), then the count

$n_{x|c}$ was increased by one. P_x value was iterated from 0 to 1 using 100 steps (s in general). The probability density function for P_x under observation P_c was then estimated as follows:

$$pdf\left(\frac{i}{s}\right) = \frac{n_{x|c}}{s * w} \quad , \quad x = \frac{i}{s} \qquad \textbf{(A-1)}$$

where *w* was chosen such that $\sum_{i=0}^{s} pdf\left(\frac{i}{s}\right) = 1$

Figure A-1 shows the probability density function of $P(P_x \mid P_c)$ where P_c is set to 0.9. After setting the confidence level to 95%, the Low and High limits are obtained as 0.59 and 0.98. The area under the curve between the low limit and high limit is equal to 0.95 (a constant). Based on pdf(x), the mean value of P_x in this interval is 0.86. Thus, probability of correct classification 0.9 under observed proportion should be replaced by 0.86 in the voting procedure.

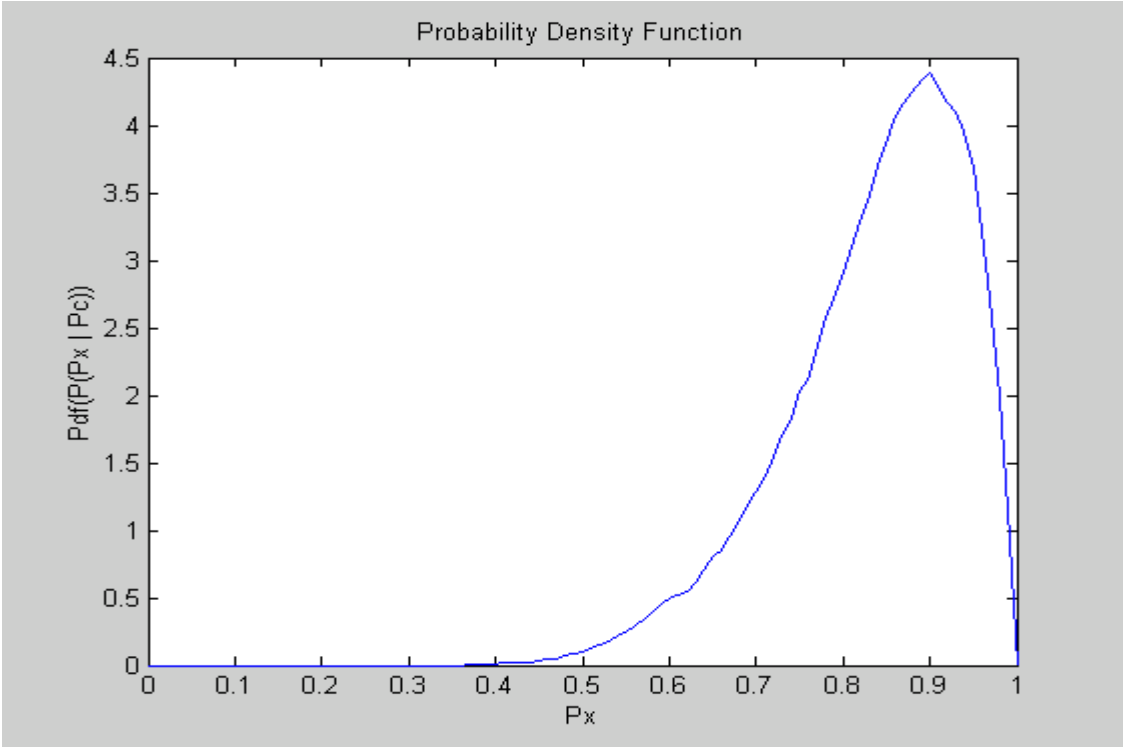

Figure A-1 Probability Density Function of P(Px | Pc) – pdf(x)

However, it is expensive to obtain the real density function if an estimated probability density function produces a reasonable result. The estimated probability density function is obtained using only three points, which are the Low limit, the P_c, and the High limit, to calculate the mean value of the unit triangle. In this case, High and Low limits were obtained from the estimation of the proportion confidence interval from literature (Newcombe, 1998).

$$\text{Low Limit} = \frac{2nP_c + Z_{\alpha/2}^2 - Z\sqrt{\left(Z^2 + 4nP_c\left(1-P_c\right)\right)}}{2\left(n+Z^2\right)} \qquad \textbf{(A-2)}$$

$$\text{High Limit} = \frac{2nP_c + Z_{\alpha/2}^2 + Z\sqrt{\left(Z^2 + 4nP_c\left(1-P_c\right)\right)}}{2\left(n+Z^2\right)} \qquad \textbf{(A-3)}$$

where n = total number of samples

$Z_{\alpha/2}$ = value Z > 0 so that the area to the right of Z under the standard normal distribution (with zero mean and unit standard deviation) is $\alpha/2$

After setting confidence level to 95%, $Z_{\alpha/2}$ was obtained as 1.96. The Low and High limits of confidence interval were calculated using (A-2) to (A-3) as 0.5958 and 0.9821. The calculated results agree with the results of the experiment extremely well. Since P_c (0.9) is greater than 0.5*(High-Low), the mean value is calculated using equation (A-4). The result is 0.8382, which is very close to the experiment result 0.86.

$$Mean = \sqrt{\frac{1}{2}(P_c - Low)(High - Low)} + Low \quad , \quad \text{for } P_c \geq \frac{High - Low}{2} \qquad \textbf{(A-4)}$$

$$Mean = High - \sqrt{\frac{1}{2}(High - P_c)(High - Low)} \quad , \quad \text{for } P_c < \frac{High - Low}{2} \qquad \textbf{(A-5)}$$

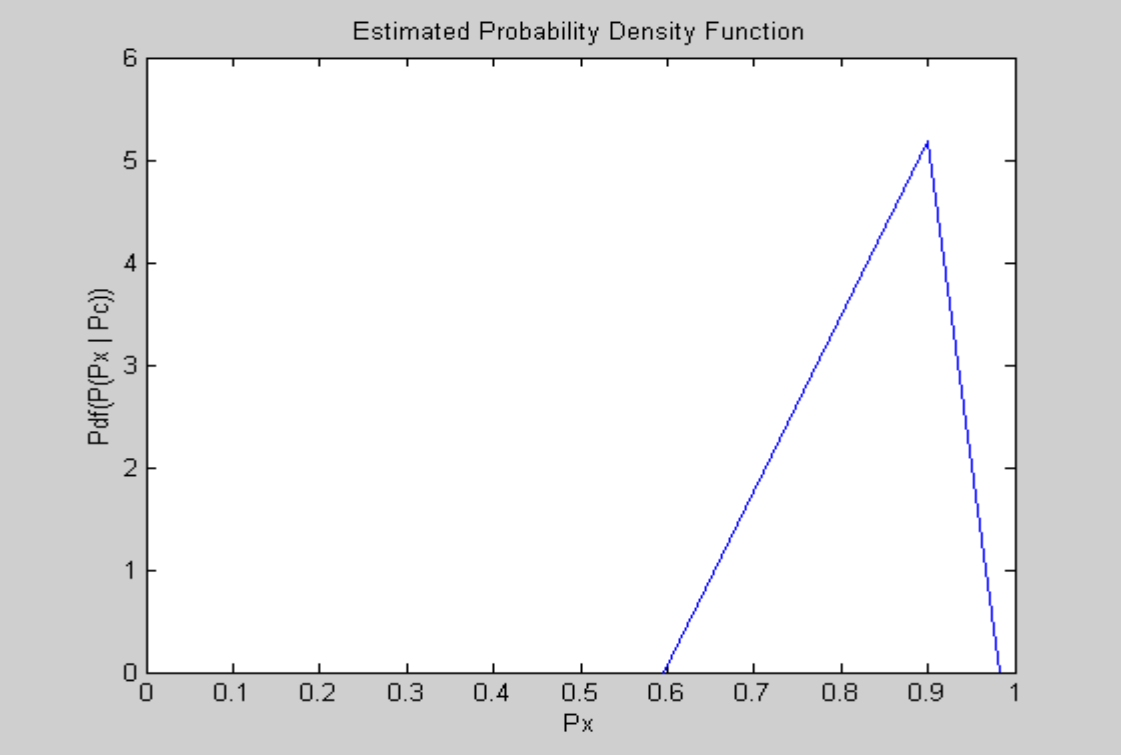

Figure A-2 Estimated Probability Density Function of P(Px | Pc)

The following example demonstrates how interval analysis improves the correct classification result. Suppose there are two classes, and six voting neurons. The probability of correct classification, number of samples in the subspace, and limits of 95% confidence interval are listed in Table A-1.

Table A-1 Probabilities of Correct Classification and Calculated Mean Value

	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6
Number of Samples	1	205	10	5	200	300
P_c for Class 1	1.00	0.60	0.71	0.08	0.12	0.25
Low Limit for Class 1	0.2065	0.5317	0.4057	0.007	0.0820	0.2044
High Limit for Class 1	1.00	0.6646	0.8978	0.5180	0.1723	0.3020
Calculated Mean for Class 1	0.7676	0.5991	0.6793	0.1835	0.1237	0.2516
P_c for Class 2	0.00	0.40	0.29	0.92	0.88	0.75
Low Limit for Class 2	0.00	0.3354	0.1022	0.4820	0.8277	0.6980
High Limit for Class 2	0.7935	0.4683	0.5943	0.9930	0.9180	0.7956
Calculated Mean for Class 2	0.2324	0.4009	0.3207	0.8165	0.8763	0.7484

Table A-2 contains two conditions for calculating the weighting function (7-3). Condition A uses only the probability of correct classification estimated directly from proportion to obtain the result using (7-3). Condition B uses the mean value of class probability each neuron calculated by equation (A-4) or (A-5) estimate probability of correct classification using (7-3). While under condition A, class 1 will be declared. Under condition B, the classification result points toward Class 2.

Table A-2 Weights Comparison

	Condition A	Condition B
Weight of Class 1	0.9999	0.8851
Weight of Class 2	0.9605	0.9454

In conclusion, the final result obtained from the weight function (7-3) using estimates for correct classification probabilities based on proportions can be not reliable when the number of input points is small. Therefore, applying the confidence interval analysis can improve the reliability for the final classification.

Appendix B

Matlab Code for Missing Data Recovery

```
%%%%%%%%%%%%%%%%%%%
%Name: Tsun-Ho Liu
%Date: 15th Sep. 2002
%
%This is a missing data recovery program
%It takes the KNOWN data to recover the UNKNOWN
%by calculating the Mahalanobis Distance
%
%"features" is an NxM input matrix.
%N is the number of features.
%M is the number of of input data.
%"classid" is a vector of classes describing the input matrix.
%(it should contain M elements)
%
%This program is designed for any number of classes!
%%%%%%%%%%%%%%%%%%%

clear all

%loading a file containing "features" and "classid" (Singular)
load missing_ill.mat

[Frow,Fcol]=size(features); %determine the size of the input matrix

%%%%%%%%%%%%%%%%%%%
%this loop searches for locations of missing values
TempB=[];
for(i=1:Frow)
    TempA=find(features(i,:)==0);
    if (size(TempA,2)~=0)
        TempB=[TempB TempA];
    end
end

%%%%%%%%%%%%%%%%%%%
%since one input-data can contain more than one missing value
%this loop deletes unnecessary information
stick=size(TempB,2);
stickP=TempB(1);
```

```
for(ck=2:stick)
    P=TempB(ck);
    if (size(find(stickP==P))<1)
        stickP=[stickP P];
    end
end

%%%%%%%%%%%%%%%%%%
%Check if the input matrix is singular or not
%by applying QR Factorization
checkQR=features;
checkQR(:,stickP)=[]; %take all the input data with missing values
                      %out from matrix
[q,r,p]=qr(checkQR');

r(find(r< 1e-6 & r >-1e-6))=0; %make those very small values to zero.

[r_row r_col]=size(r);
r_state=0;
for(i=1:r_col)
    tempZero=size(find(r(:,i)==0),1);
    numOFr=(r_row-tempZero);
    if(numOFr>r_state)
        r_state=numOFr;
    end
end

%%%%%%%%%%%%%%%%%%
%if the input matrix is singular, set flag = 1, otherwise flag = 0
if(r_state==r_col)
    flag=0;
else
    flag=1;
end

%%%%%%%%%%%%%%%%%%
%determine which feature is dependent and take it out
if(flag==1)
    R1=r(1:r_state,1:r_state);
    R2=r(1:r_state,r_state+1:r_col);
    R1=inv(R1);

    delCol=p(:,r_state+1:r_col);
    del_loc=[];
    for(i=1:(r_col-r_state))
        temp_del=find(p(:,r_state+i)==1);
        del_loc=[del_loc temp_del];
    end

    features(del_loc,:)=[];
end
```

```
trainingdata=features;
cid=classid;

stick=size(stickP,2)

trainingdata(:,stickP)=[]; %take all the input data with missing
                           %values out from matrix
classid(:,stickP)=[];

classid=classid';
trainingdata=trainingdata';

col=size(trainingdata,2);
row=size(trainingdata,1);
maxcid=max(classid); %check how many classes are included in matrix

%%%%%%%%%%%%%%%%%%
%this loop picks up all the input-data with missing values
test=[];
cid_test=[];
for (k=1:stick)
    F=features(:,stickP(k));
    F=F';
    test=[test;F];
    cid_test=[cid_test cid(stickP(k))];
end

%Calculating the Mahalanobis Distance
for(classSearch=1:maxcid)
    c_loc=find(classid==classSearch)'; %search for locations for this
                                       %class
    meanc=mean(trainingdata(c_loc,:))'; %calculate the mean of KNOWN
                                        %values for this class
    classified=trainingdata(c_loc,:); %find the KNOWN values for this
                                      %class

    for i=1:stick
       NOW=i;
       CLASSNOW=cid_test(NOW);
       if (CLASSNOW==classSearch)
           tempClass=classified;
           A=find(test(NOW,:)==0); %find all the missing values
                                   %location for this particular
                                   % input data

           S=size(A,2);

           T=size(test,2);

           tempClassA=tempClass(:,A);
           tempClass(:,A)=[];
```

```
            tempClass=[tempClass tempClassA]; %remove all the KNOWN
                                              %features based on the
                                              %missing values location
                                              %to the end of the KNOWN
                                              %matrix

             cc=cov(tempClass); %take the reverse covariance matrix
             dc=cc^-1;

             dcmm=dc(T-S+1:T,T-S+1:T);
             dcmk=dc(T-S+1:T,1:T-S);
             mc=meanc;

             m2=[mc(A)];
             m=mc;
             m(A)=[];

             HERE=test(NOW,:);
             HERE(A)=[];
             ANSWER=abs((-(HERE-m')*dcmk'*dcmm^-1)+m2')%the calculated
                                                       %missing values
             aaa=size(ANSWER',1);
             for(x=1:aaa)
                 features(A(x),stickP(i))=ANSWER(x);%put the
                                                    %calculated
                                                    %missing values
                                                    %back to the
                                                    %input matrix
             end
         end
     end
end

%If the input matrix is singular
%recover the depentant features
if(flag==1)
    tempF=features;
    [temp_row,temp_col]=size(tempF);
    features=zeros(Frow,Fcol);
    addRow=size(del_loc,2);
    org_del_loc=del_loc;
    del_loc=sort(del_loc);

    j=1;
    k=1;
    for (i=1:Frow)
        if(i==del_loc(j))
```

```
            add_here=zeros(1,Fcol);
            if(j<addRow)
                j=j+1;
            end
        else
            add_here=tempF(k,:);
            if(k<temp_row)
                k=k+1;
            end
        end
        features(i,:)=add_here;
    end

    R=R1*R2;
    [r_row, r_col]=size(R);
    R(find(R< 1e-6 & R >-1e-6))=0; %make those very small values
                                   %to zero.
    for(i=1:r_col)
        find_row_on_p=find(R(:,i)~=0);
        SizeDepen=size(find_row_on_p,1);
        row=[];
        for(z=1:SizeDepen)
            take_row=find(p(:,find_row_on_p(z))==1);
            value=R(find_row_on_p(z),i);
            row=[row; features(take_row,:)*value];
        end
        row=sum(row);
        features(org_del_loc(i),:)=row;
    end
end
```

Appendix C

Matlab Code for Symbolic Value Assignment

```
clear all

%   define numerical and symbolic columns
numcols=[1 3];
symcols=[2];

%   define maximum number of symbols in any coordinate
syms a b c d e;

%   read the mixed type data matrix
%   it contains both numeric and symbolic values
%   each column of data matrix is uniform and contains only numeric
%   or symbolic data
data=[1 e 1;2 a 2;4 a 2; 3 b 0; 4 b 4; . . .
      8 d 2;9 d -4;8 c 2; 9 c 2;10 c -1]
%   number of samples in the data array
nsamples=size(data,1);

%   matrix of numerical values
ndata=data(:,numcols);

%   solve symbolic value assignment one symbolic vector at a time
for k=1:size(symcols,2)
    sdata=data(:,symcols(k));

    %   vector of symbolic values
    symvector=[a b c d e];
    symnum=size(symvector,2);

    %   get numerical values matrix
    C=eval(ndata);

    %   formulate symbolic location matrix
    A=zeros(symnum,nsamples);
    for i=1:symnum
        loc=find(sdata==symvector(i));
        A(i,loc)=1;
    end;
```

```
    % A=[0 1 1 0 0 0 0 0 0 0;
    %    0 0 0 1 1 0 0 0 0 0;
    %    0 0 0 0 0 0 0 1 1 1;
    %    0 0 0 0 0 1 1 0 0 0;
    %    1 0 0 0 0 0 0 0 0 0];

    A=A';

    %special case if C is a single numerical column
    %and this is the answer
    coord1=pinv(A)*C

    %if C is non-single numerical column
    %Find B and divide B into B1 and Br
    B=[A'*A A'*C;C'*A C'*C];
    B1=[A'*A; C'*A];
    Br=[A'*C; C'*C];

    %Perform QR factorization
    %x=Q
    %y=R
    [x y]=qr(Br);
    x=x';

    %make those very small values to zero.
    y(find(y< 1e-6 & y >-1e-6))=0;

    %Search for independent columns
    [r_row r_col]=size(y);
    r_state=0;
    for(i=1:r_col)
        tempZero=size(find(y(:,i)==0),1);
        numOFr=(r_row-tempZero);
        if(numOFr>r_state)
            r_state=numOFr;
        end
    end

    %Find Q2 base on numbers of dependent columns
    q2=x(r_state+1:r_row,:);

    %Find C1 and Cs
    %Perform pseudoinverse of Cs and calculate answers
    Bx=q2*B1;
    c1=Bx(:,1);
    cs=Bx(:,2:size(B1,2));
    ar=-pinv(cs)*c1;
    ar=[1; ar];

end
```