A STUDY ON DATA COMPRESSION BY DYNAMICAL SYSTEM APPROACH

LAM FUNG-YEE

M. PHIL.

THE HONG KONG

POLYTECHNIC UNIVERSITY

2001

Abstract of thesis entitled 'A Study on Data Compression by Dynamical System Approach' submitted by Lam Fung-yee for the degree of Master of Philosophy at The Hong Kong Polytechnic University in October 2001.

The importance and demands for data compression have been increasing rapidly, especially with the growing popularity of Internet access and multimedia personal entertainment. The ratio and the quality of compression data are main concern when judging the compression algorithm. Recently, data compression techniques are dominated by the fast Fourier and wavelet transforms, which approximate the given sequence as a linear sum of the basis function, by retaining a finite number of coefficients to achieve the goal of compression.

In this project, we introduce a dynamical system approach, which compresses data in a totally different way from the ones mentioned above. Taking advantage of the fact that Leaky-integrator model recurrent neural net can approximate arbitrary finite sequence, we demonstrate in this thesis how to compress UV and IR spectrum by a discrete-time recurrent neural net. As this is an initial valued problem, the information we need to store is the parameters of the system and the initial states. Compression ratio is also discussed in this thesis.

Acknowledgements

I would like to express my deep gratitude to Dr. L.K. Li, my supervisor, for his support to my graduate study and research at the Department of Applied Mathematics in the Hong Kong Polytechnic University. The work for this thesis could not have been finished without Dr. Li's supervision.

I am very grateful to Dr. S.P. Yung for his help while I was doing the research on this thesis. Professor Chau of the Applied Biology and Chemical Technology Department, The Hong Kong Polytechnic University provided the chemical data sets and made many suggestions and feedback for the results.

I gratefully acknowledge the financial support and administration from the Departmental Research Committee and the Research Office of the Hong Kong Polytechnic University grants.

I would like to thank Dr. L.F. Cheung, Dr. K.F.C. Yiu and Miss Novend Chung for their support and valuable comments while I was doing the research on this thesis.

My thanks also go to all the people who work in the Department of Applied Mathematics.

Finally, and most importantly, I wish to thank all my family members for their help and understanding.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We are living in an information dependent society. Efficient delivery of data, audio and video signals via fixed and mobile networks is becoming increasingly important. Network transmission time is directly proportional to the file size. Since new audio and video applications are usually large in size, new and improved signal compression techniques are needed. In many cases, the exact information is not necessary and an approximation of the original data can be transmitted instead, which can be achieved by some compression techniques.

1.1 Fourier Transform Vs. Wavelet Transform

The conventional Fourier transforms and other orthogonal transforms provide the frequency information over the entire spectrum of a signal. The irregularities of a signal are not localized in a Fourier transform domain. Multi-resolution decomposition is a convenient tool to study the time-domain, scale-domain and frequency-domain aspects of a signal simultaneously. The conventional Fourier transform-based approaches have a block structure, while the pyramid decomposition and other multi-resolution-based approaches possess a pyramid or tree structure. A wavelet decomposition is a multi-resolution decomposition in which a discrete or a continuous signal at resolution 2^j is decomposed into a low-pass (called the approximate signal) at a resolution 2^{j-1} and a high-pass (called the detail signal) version at the same resolution 2^{j-1}. The low-pass version of the original signal is again decomposed into a low-pass at lower resolution and a high pass with the same resolution as the low-pass.

1.2 Scope and Objectives

The investigation, design and analysis of algorithms for lossless one-dimensional data compression using dynamical system approaches forms the primary objective of this study. A lot of work has been done in the area of lossy data compression. However, very little work has been done in Recurrent Neural Network Dynamics method. Since many signals are records of some dynamic behaviors, in this study, we propose to use a dynamical system approach to approximate a given data sequence.

Signal types included in this study were results from a chemical experiment. We demonstrate our approach by compressing two sets of data obtained from Professor Chau of the Applied Biology and Chemical Technology Department, at The Hong Kong Polytechnic University. These data were obtained from an experiment which had been described in Chau [6]. In chemical analysis, signal compression is very important, especially in setting up digitized spectral library, to minimize the size of the original database and to reduce the time for spectral searching.

In chemical studies, neural network has been applied successfully to solve problems in classification, optimization, modeling and mapping. But, to our knowledge, this mathematical technique has not been used for spectral compression. It is because the feedforward neural net which is commonly used in chemistry consists of a large number of hidden neurons and parameters. This leads to a common belief that neural nets are not suitable for compression purpose. Instead of using the popular feedforward networks, a small size recurrent neural net with no hidden unit involved was proposed in this study and applied to compress both synthetic and experimental spectra successfully.

1.3 Contributions of the thesis

In this project, we propose to use a dynamical system approach to approximate a given data sequence. This method is proposed by Li [13, 14, 15]. In this thesis, focus is directed towards the error analysis of this RNN method and the development of a new learning algorithm.

1.4 Thesis organization

The thesis is divided into nine chapters. In the first two chapters, we look at the most popular algorithms, i.e. Fast Fourier Transform and Wavelet Transform. Fast Fourier Transform and Wavelet Transform approximate the original signal by a linear sum of basis function. Then the background theory and merits are given when we apply the Recurrent Neural Network Dynamics method as the compression algorithm, which is a dynamical system approach.

In Chapter [3], the methodology of the dynamical system technique for data compression is discussed. The error estimation for different techniques is also discussed so that readers will have an idea of how close the approximated signals are to the original ones.

In Chapter [4], we look at the actual equations used for programming in this study.

In Chapters [5] and [6], the results from the Ultra Violet signals and Infra Red signals are given respectively.

In Chapter [7], the findings of this project and some concluding statements about the project as a whole are given. Since this project is at the inception for exploiting the Recurrent Neural Networks as a new compression methodology, remarks and suggestions are made for further research work in this area.

Finally, the programs for carrying out data compression in this project are appended.

Chapter 2

Literature Review

In this chapter, we describe previous work in the data compression as well as the classification of data compression.

2.1 Classification of Data Compression

Data compression can be divided into two main categories: lossless and lossy compression.

2.1.1 Lossless compression (or information preserving)

Lossless compression techniques involve no tolerance of distortion in the information; thus, if data have been losslessly compressed, the original data can be recovered exactly from the compressed data. They are used mainly for compressing database records, spreadsheets or word processing files, where exact replications of the originals are essential. However, the main shortcoming of lossless compression is that the amount of compression is very limited. Nowadays, most popular algorithms are based on adaptive Huffman, arithmetic coding, or Lempel-Ziv (LZ) methods.

2.1.2 Lossy compression

Lossy compression techniques allow a loss of accuracy in representing the information, and the reconstructed data are close to, but need not be the same as, the original one. Instead, decompression produces an approximation dependent on the compression ratio. The deformation of data makes it possible to have compression ratios that are typically greater than lossless compression ratios. In fact, these compression techniques are to construct an approximation to the original information. You can achieve more compression by allowing the algorithm to control this compression against quality trade-off.

Lossy compression is more effective when being used to compress graphic images and digitized voice, where information is in waveform data and losses outside visual or aural perception can be tolerated. Most lossy compression techniques can be adjusted to different quality levels, gaining higher accuracy in exchange for less effective compression.

The most popular compression algorithms, such as Fast Fourier Transform and Wavelet Transform, fall into the lossy compression category. Similarly, the dynamical system method that has been proposed in this project also falls into this category.

2.2 Conventional Data compression

The late 40's were the early years of Information Theory, when the idea of developing efficient new coding methods was just starting to be fleshed out. Ideas of entropy, information content and redundancy were explored. One popular notion is that if the probability of symbols in a message were known, there ought to be a way to encode the symbols so that the message would take up less space.

The first well-known method for the compression of digital signals is now known as Shannon-Fano coding. Shannon and Fano simultaneously developed the algorithm, which assigns binary-coded words to unique symbols that appear within a given data file. While Shannon-Fano coding was a great leap forward, it had the misfortune to be quickly superseded by an even more efficient coding system: Huffman coding.

Huffman coding [1952] shares all characteristics of Shannon-Fano coding; however Huffman coding could perform effective data compression by reducing the amount of redundancy in the coding of symbols. It has been proven to be the most efficient fixed-length coding method available.

In the last fifteen years, Huffman coding has been replaced by arithmetic coding. Arithmetic coding bypasses the idea of replacing an input symbol with a specific code. It replaces a stream of input symbols with a single floating-point output number. More bits are needed in the output number for longer, complex messages.

Dictionary-based compression algorithms use a completely different method to compress data. They encode variable-length strings of symbols as single tokens. The tokens form an index to a phrase dictionary. If the tokens are smaller than the phrases, they replace the phrases and compression occurs.

Two dictionary-based compression techniques called LZ77(Lempel-Ziv [1977]) and LZ78(Lempel-Ziv [1978]) have been developed. LZ77 is a "sliding window" technique in which the dictionary consists of a set of fixed-length phrases found in a "window" in the previously seen text. LZ78 takes a completely different approach to building a dictionary. Instead of moving fixed-length phrases from a window into the text, LZ78 builds phrases up one symbol at a time, adding a new symbol to an existing phrase when a match occurs.

Recently, many techniques have been developed to compress a given signal (a set of data). The usual ways include the well-known transformation methods, such as Fast Fourier Transform and Wavelet Transform that are based on approximating an arbitrary function.

Using the Stone Weierstrass' Theorem, we can approximate a given function or sequence by a linear sum of the basis functions. These methods integrate the given signal (data set) over the interval and store the information in terms of a linear sum of basis transform functions and their corresponding coefficients. The basis functions are usually orthogonal. Compression methods use the computation time in exchange for the storage space.

Given a signal $z(t)$ continuous on $[a,b]$ and supposing we have a countable basis of functions $f_i(t)$ in $L_2[a,b]$. Then, the difference $\left\| z(t) - \sum_{i=1}^{N} c_i f_i(t) \right\|_2 \to 0$ as $N \to \infty$ and $|c_i f_i| \to 0$, so that the difference $\left\| z(t) - \sum_{i=1}^{N} c_i f_i(t) \right\|_2$ will smaller than original pre-assigned tolerance. As we fix the set of basis $\{f_i(t)\}$ and ignore the coefficients c_i

with 'small' magnitude, such that the error $\left\| z(t) - \sum_{i=1}^{N} c_i f_i \right\|_2$ is within our tolerance, then an approximation of the original signal $z(t)$ is formed. Since $f_i(t)$'s are already specified, we store the information i and c_i instead of the original signal $z(t)$. Thus, we may consider these compression techniques as function approximation methods.

In the following sections, brief descriptions of the algorithms on data compression will be given by using Fast Fourier Transform. As pointed out by Zaknich and Attikiouzel (1995), most signal processing problems are related to a time or spatial data series, so we suppose the given sequence is quite smooth in the sense that $z(t)$ has some intrinsic property that follows some dynamical system locally. We will discuss the dynamical system approach in the second section.

2.3 Fast Fourier Transforms

The Fourier Transform is based on the discovery that it is possible to take any periodic function of time $x(t)$ and resolve it into an equivalent infinite summation of sine waves and cosine waves with frequencies that start at 0 and increase in integer multiples of a base frequency $f_0 = \frac{1}{T}$, where T is the period of $x(t)$. This is what the expansion looks like:

$$x(t) = a_0 + \sum_{k=1}^{\infty} \left(a_k \cos(2\pi k f_0 t) + b_k \sin(2\pi k f_0 t)\right)$$

An expression of the form of the right hand side of this equation is called a Fourier Series. The job of a Fourier Transform is to figure out all the a_k and b_k values to produce a Fourier Series, given the base frequency and the function $x(t)$.

As it is unrealistic to do an infinite summation, a finite set of sines and cosines is usually employed. Once the coefficients have been resolved, the corresponding approximate data can be reproduced. For most compression applications, the most significant coefficients in the Fourier Series are retained. As for the number of coefficients used, it depends on the precision required for the restored signals.

Fourier Transforms are one of the fundamental operations in signal processing. In digital computations, Discrete Fourier Transforms (DFT) are used to describe, represent and analyze discrete-time signals. However, direct implementation of DFT is computationally inefficient. Of the various available high-speed algorithms commonly used to compute DFT, the Cooley-Tukey algorithm is the simplest. The efficient algorithms to compute DFTs are called Fast Fourier Transforms (FFTs) which are the algorithm for computing the discrete Fourier Transforms in a faster way and reducing the number of computations from approximately $O(N^2)$ to $O(N \log N)$.

FFTs were first discussed by Cooley and Tukey (1965), although Gauss had actually described the critical factorization step as early as 1805 (Gergkand 1969, Strang 1993). A DFT can be computed using an FFT by means of the Danielson-Lanczos Lemma if the number of points N is a power of two. If the number of points N is not a power of two, a transform can be performed on sets of points corresponding to the prime factors of N, which is slightly slower. An efficient real Fourier transform algorithm or a fast Hartley Transform (Bracewell 1999) gives a further increase in speed approximately by a factor of two. Base-4 and base-8 fast Fourier transforms use optimized code, and can be 20-30% faster than base-2 fast Fourier transforms. Prime factorization is slow when the factors are large, but DFT can be made fast for

$N = 2, 3, 4, 5, 7, 8, 11, 13$, and 16 using the Winograd Transform Algorithm.

The FFT is a DFT Algorithm which reduces the number of computations needed for N points from $2N^2$ to $2N \lg N$, where lg is the base-2 logarithm. If the function to be transformed is not harmonically related to the sampling frequency, the response of an FFT looks like a sine function (although the integrated power is still correct). Aliasing (leakage) can be reduced by apodization using a tapering function. However, aliasing reduction is at the expense of broadening the spectral response.

2.4 Wavelet Transforms

Wavelets are versatile tools for harmonic analysis. Because of their many uses, the word 'wavelet' comes with different connotations (and different promises) to users in different fields. Therefore it is important to state the limitation of our approach at the beginning, so that the user will not be disappointed by unfulfilled expectations.

Wavelets are mathematical functions that divide data into different frequency components, and then study each component with a resolution matched to its scale. They have advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes. The fundamental idea behind wavelets is to analyze according to scale. Indeed, some researchers in the wavelet field feel that, by using wavelets, one is adopting a whole new mindset or perspective in processing data.

Wavelets are functions that satisfy certain mathematical requirements and are used in representing data or other functions. Approximation using superposition of functions has existed since the early 1800's, when Joseph Fourier discovered that he could superpose sines and cosines to represent other functions. However, in wavelet analysis, the scale that we use to look at data plays a special role. Wavelet algorithms process data on different scales or at different resolutions. If we look at a signal with a large "window," we notice gross features. Similarly, if we look at a signal with a small "window," we notice small features. The result in wavelet analysis to see both the forest and the trees.

The wavelet analysis procedure is to adopt a wavelet prototype function called an analyzing wavelet or mother wavelet. Temporal analysis is performed with a contracted, high-frequency version of the prototype wavelet, while frequency analysis is performed with a dilated, low-frequency version of the same wavelet. Because the original signal or function can be represented in terms of a wavelet expansion (using coefficients in a linear combination of the wavelet functions), data operations can be performed using only the corresponding wavelet coefficients. If we further choose the best wavelets adapted to our data, or truncate the coefficients below a threshold, our data are sparsely represented. This sparse coding makes wavelets an excellent tool in data compression.

To sum up, both of the transform algorithms express the signal as a linear sum of the basic functions. Compression of signals are achieved by retaining the finite number of coefficients, which reproduce a signal to approximate the original signal.

2.5 Neural Network

2.5.1 Background

Artificial neural network models ("ANN") or simply 'neural nets' are known with different names such as connectionist model, parallel distributed models, and neuromorphic systems. Whatever the name, it can be viewed as circuits of highly interconnected parallel processing units called 'neurons'. Much of the current algorithms for adjusting interconnection weights adaptively so as to perform the desired classification. There are two main types of models of ANN, namely the feed-forward neural network and the feed-back neural network. The following two sections will give each a simple illustration.

2.5.2 Feed-forward Neural Networks

In feed-forward networks, the data flow from input to output units is strictly feed-forward. The data processing can be extended over multiple (layers of) units, but no feedback connections (loops) are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers. This type of organization is also referred to as bottom-up or bottom-down. A feed-forward neural network with single hidden layer is denoted as $I \times H \times O$, where I, H and O represent the number of input units, the number of hidden units, and the number of output units, respectively. Generally there may be multiple hidden layers between the input and output layers. Figure 2.5 gives a typical fully connected 2-layer feed-forward NN, which the input layer does not count by convention, with a $2 \times 3 \times 2$ structure.

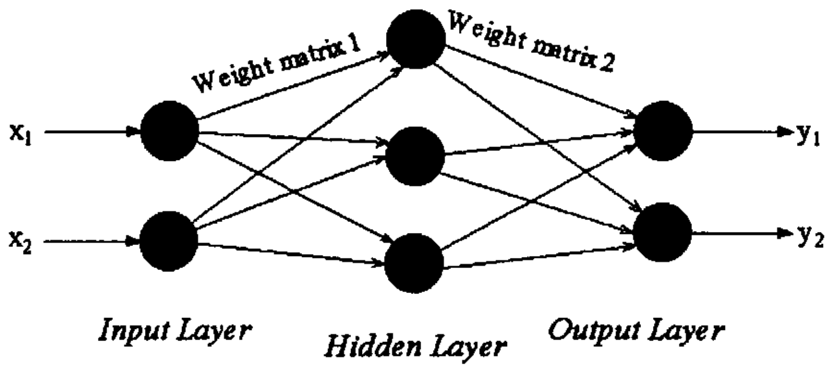


Figure 2-1: A $2 \times 3 \times 2$ feed-forward NN

The input units simply pass on the input vector x. The units in the hidden layer and output layer are processing units.

Feed-forward neural network work by training the connection weights with given examples. A sample pattern (x_i, y_i) is taken from the training set of known data $\{(x_i, y_i)|_{i=1,2,\ldots,n}\}$ and x_i is fed into the input layer of the network. After computing, an output vector $\mathbf{O}_i$ based on either none, one or more hidden layer output, then we compare the output vector $\mathbf{O}_i$ with the target value y_i. We adjust the weight matrices so as to minimize the difference between the output vector $\mathbf{O}_i$ and the target value y_i, and this process is called learning.

2.5.3 Feed-back (Recurrent) Neural Networks

In feed-back neural network, it contained feedback connections. It is very powerful and may be extremely complicated. Contrary to feed-forward networks, the dynamical properties of such networks are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications, the change of the activation values of the output neurons are significant, such that the dynamical behaviour constitutes the output of the network. Feedback architectures are also referred to as interactive or recurrent. In figure 2.6, it shows a configuration of a feed-back NN.

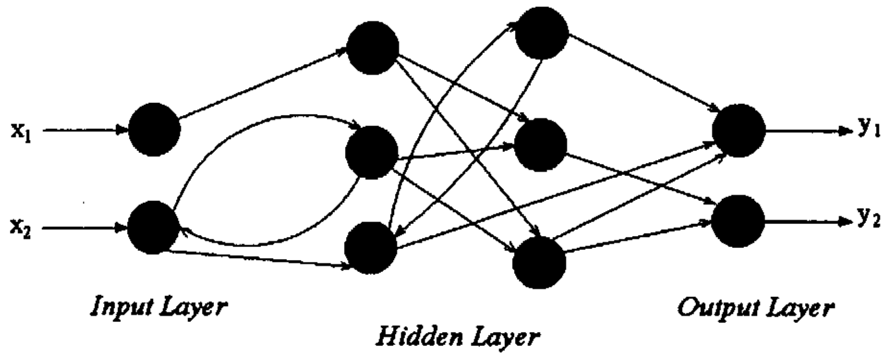


Figure 2-2: A feedback NN

2.6 Recurrent Neural Networks("RNN") in Data Compression

A simple network has a feed-forward structure: signal flows from inputs, forwards through any hidden units, eventually reaching the output units. Such a structure has stable behaviour. However, if the network is recurrent (contains connections back from later to earlier neurons) the state of the neurons may be unstable, and have very complex dynamics.

Recurrent networks contain feedback connections. Contrary to feed-forward networks, which is independent of time, the dynamical properties of the recurrent network are important in signal compression.

2.6.1 Neural Network for Signal Processing

According to the Li [15] "*Approximation Theory and Recurrent Network*", a given trajectory sequence with the corresponding time steps can be represented by a discrete-time fully connected recurrent neural net. Then the paper had also generalized the result to approximation of a twice differentiable trajectory on a compact time interval and show that recurrent nets can be universal approximators of trajectory in $\Re^n$.

Further explanation of how the method can applied to perform data compression in this study will be given in the following.

2.6.2 Compression Method

Given signal $\{z_t\}$, where $1 \leq t \leq m$ as a one-dimensional finite sequence, and suppose that z_t are quite smooth. The sequence can then be divided into n segments with p data in each segment. Place n segments into n row to form a $n \times p$ matrix with p column of data. In the following, we use $\mathbf{x}(k)$, where $1 \leq k \leq p$, to represent the column vectors of the $n \times p$ matrix. To begin with, consider the continuous neural network dynamic equation:

$$\frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{W}\sigma[\mathbf{x}(t) + \theta] + \mathbf{J} \quad \text{......(2.1)}$$

Using Euler's method, the equation is expressed as a function of past time function:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + h\frac{d\mathbf{x}(k)}{dt} \quad \text{......(2.2)}$$

Comparing both equations, the following equation is arrived:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + h\{-\mathbf{x}(k) + \mathbf{W}\sigma[\mathbf{x}(k) + \theta] + \mathbf{J}\} \quad \text{......(2.3)}$$

where

$\mathbf{x}(k)$	-	states of the system at time $k \in (-1,1)^m$,
h	-	step size,
$\mathbf{W}$	-	the $n \times n$ synaptic connection strength matrix,
σ	-	the artificial activation function of the system which is bounded, monotonic increasing and continuously differentiable arbitrarily many times,
$\boldsymbol{\theta}$	-	the bias or threshold vector, $n \times 1$ matrix,
$\mathbf{J}$	-	external input to the system, $n \times 1$ matrix.

A schematic diagram of a 2-neuron RNN is given in Figure 2.3.

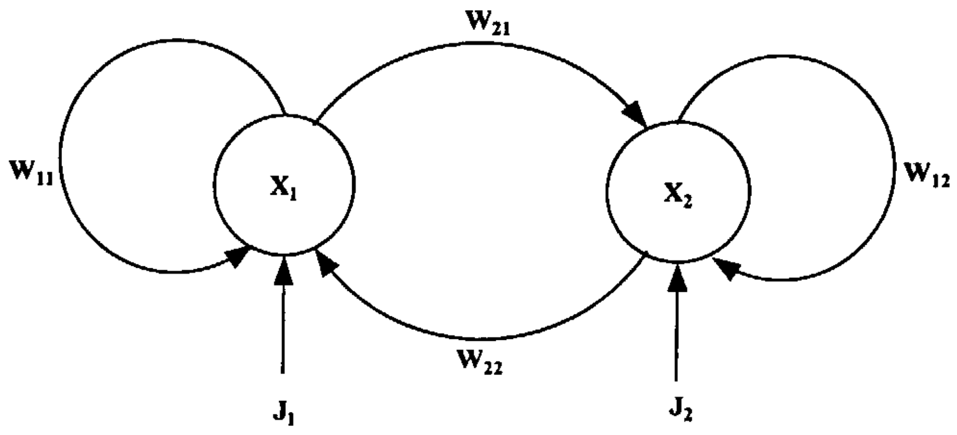


Figure 2-3: The 2-neuron recurrent neural net with input (J_1, J_2).

In this RNN scheme, $X(t)$ represents the two neurons $x_1(t)$ and $x_2(t)$ which changes with the time variable where $t = 1, 2, ...$ As an example in applying Equation (2.3), let the step size h be 0.5 and $\boldsymbol{\theta}$ be the zero vector,

$$\mathbf{J} = \begin{pmatrix} J_1 \\ J_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}, \qquad (2.4)$$

and the initial state $X(1)$ having values of

$$X(1) = \begin{pmatrix} x_1(1) \\ x_2(1) \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \qquad (2.5)$$

The next state $X(2)$ can be generated by Equation (2.3) as,

$$X(2) = \begin{pmatrix} x_1(2) \\ x_2(2) \end{pmatrix}$$
$$= \begin{pmatrix} 2 \\ -1 \end{pmatrix} + 0.5 \left\{ - \begin{pmatrix} 2 \\ -1 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix} \sigma \left[\begin{pmatrix} 2 \\ -1 \end{pmatrix} \right] + \begin{pmatrix} -1 \\ 1 \end{pmatrix} \right\} \qquad (2.6)$$

Similarly, other states $X(3)$, $X(4)$, ...$X(p)$ can be generated by using Equation (2.3) repeatedly. In this 2-neuron net, data needed to be stored are the system quantities $X(1)$ and the RNN parameters, h, $\mathbf{W}$, $\boldsymbol{\theta}$ and $\mathbf{J}$ vectors and their numbers of data involved are 2, 1, 4, 2 and 2, respectively. The universal approximation properties of recurrent networks as mentioned above are important for compressing any sequence or signal in this investigation. Li (1992) has shown that , for any arbitrary discrete finite sequence or signal $\{z(t) \in \Re\}_{1 \leq t \leq m}$, there always exists a discrete-time RNN as defined by Equation (2.3) to general $z(t)$ accurately.

Further explanation of how the equations are applied to perform data compression in this study will be given in the following chapter.

Chapter 3

Methodology

In the early sections of this chapter, the detailed methodology for asserting the compression capability of recurrent neural network will be shown systematically. The detailed steps leading to finding the compression parameters will be given as well.

In the following section of this chapter, some error analysis and estimation of the discrete recurrent neural network method will be discussed.

3.1 RNN Algorithm

Consider the task of compressing a given sequence $z(t)$ of length m by a fully connected RNN of network size n, i.e. n neurons, with iterative dynamics as given in Equation (2.3). We define here that a dynamical system is *exactly capable* for compression if there exists some neural parameters $\mathbf{W}$, $\boldsymbol{\theta}$, $\mathbf{J}$ and h such that the error between the system output $x(t)$ and $z(t)$ is zero. If the least square error between the output of the system and the smoothed sequence is less than a pre-assigned tolerance, e.g. $\varepsilon = 10^{-4}$, then the network is considered to be good for compression.

The RNN algorithm, as proposed in this project, are carried out in three steps as described below. In the first stage, if the signal is noisy, it is smoothened by using the moving average method or other pre-processing methods to deliver a better performance. For convenience, instead of $z(t)$, $y(t)$ will be adapted to be a smoothened sequence in the following discussion.

After smoothing, $y(t)$ is normalized within the range of -1 to 1 (instead of 0 to 1) and with this, we can assume that the parameters $\boldsymbol{\theta}$ and $\mathbf{J}$ are zero vectors, hence, save memory space. According to the Li (1992) "Approximation Theory and Recurrent Network", this setting has very little effect on the compression ratio and error of data compression.

In the second stage, $y(t)$ of length m is divided into n equal subsequences or segments having length p each (i.e., $m = np$). Then n neurons of $x_1(t)$, $x_2(t)$, ..., $x_n(t)$ are used in the RNN treatment with a network size n. It is assumed that m is divisible by n. (If not, the remaining data will be retained and will be not used in the RNN computation.) In the UV spectra under study, the remaining data are on the high energy side and contain no spectral information. Hence, neglect of these data do not affect the quality of the regenerated spectra. This is justified by the results obtained.

For an n-neuron RNN, the first neuron $x_1(t)$ of $X(t)$ (see the example Equation (2.5)) is related to the first subsequence or segment of $x_1(1)$, $x_1(2)$, ..., $x_1(p)$, the second one $x_2(t)$ to the second subsequence of $x_2(p+1)$, $x_2(p+2)$, ..., $x_2(2p)$, and so on. Before RNN treatment, the initial states $x_1(1)$, $x_2(1)$ and $x_n(1)$ of $X(1)$ are assigned to have the same values as $z(1)$, $z(p+1)$, $z((n-1)p+1)$ of the original signal, respectively. Then $\{x_1(2), ..., x_1(p)\}$, $\{x_2(2), ..., x_2(p)\}$, ... and $\{x_n(2), ..., x_n(p)\}$ can be generated via Equation (2.3) with given values of the RNN parameters h, $\mathbf{W}$, $\boldsymbol{\theta}$ and $\mathbf{J}$. By varying these RNN quantities systemically, one may obtained the calculated output (or the regenerated spectrum) $X(t)(= \{x_1(1), ..., x_1(p), ..., x_n(1), \ldots, x_n(p)\})$ to have the smallest discrepancies with respect to the original spectrum. Then, the optimized RNN quantities together with $X(1)$ are achieved as the compressed data set $\mathbf{R}$ and will be used to regenerate the original signal. The number of data involved in h, $\mathbf{W}$, $\boldsymbol{\theta}$ and $\mathbf{J}$ and $X(1)$ are 1, n^2, n, n and n respectively. Thus, the total number of data retained is $(n^2 + 3n + 1)$ compared to m (or np) of the original spectrum. If the number of the original data is not divisible by n and the number of remaining data r not being used in the RNN computation is non-zero, then the total number of data retained is equal to $(n^2 + 3n + 1 + r)$ compared to $(np + r)$.

Once the network size n is fixed, the goal in the third stage is to optimize the set of RNN parameters h, $\mathbf{W}$, $\boldsymbol{\theta}$ and $\mathbf{J}$ so that the discrepancy between $z(t)$ and $X(t)$ is minimized. As usual, the root mean square error (RMSE) was used in this investigation as an indicator of the discrepancy. RMSE is defined as follows:

$$RMSE = \left[\sum_{i=1}^{n} \sum_{t=1}^{p} \frac{1}{m} \{x_i(t) - z((i-1)p + t)\}^2\right]^{\frac{1}{2}}.$$

To adjust the RNN parameters so as to minimize the RMSE, an iterative process called learning algorithm was adopted. Many researchers applied the learning algorithm based on gradient descent method derived by William and Zipser. Here, we use a different algorithm because the RNN involved has no hidden neuron. Learning the optimal values of the RNN parameters were obtained using the method described in

Li (1996) for finding the fixed points of Equation (2.3). Like other gradient descent methods, there is no guarantee for obtaining the global minimum of the RMSE because the error function may contain a lot of local minimums. Further, we are not sure we can get the global minimum if $RMSE \neq 0$ as the problem is highly non-linear and the capability of RNN is still under-develop. However, the compression results obtained from our RNN study are good enough when compare to those results by Wavelet Transform, and in some cases, even better.

Once the optimized RNN parameters h, $\mathbf{W}$, $\boldsymbol{\theta}$ and J are obtained, these quantities together with the initial state of the system $X(1)$ are collected as the compressed data **R** as mentioned previously. In order to increase the compression efficiency further, the parameters $\boldsymbol{\theta}$ and **J** are assumed to be zero vectors. Thus, the total number of compressed data is reduced to $(n^2 + n + 1 + r)$. For instance, a UV spectrum of length 1024 being treated by an 8-neuron RNN, the parameters n, m, p and r have values of 8, 1024, 128 and 0, respectively, and the total number of data retained is $(8^2 + 8 + 1 + 0) = 73$ compared to 1024 of the original sequence. In regenerating the whole spectrum $X(t)$, the compressed set R is used together with Equation (2.3).

3.1.1 Pre-processing

In order to make a neural network produces accurate fitting curve, the selected raw data must be pre-processed. Two widely used pre-processing methods which are known as "Transformation" and "Normalization".

Transformation is used to manipulate one or more raw data inputs to generate a single network input. Normalization is a transformation which will distributes data evenly and scales it into an acceptable range for network usage. Decisions made during this phase are:

- What transformations should be applied to the data?
- Should these transforms include standard technical analysis indicators?
- How should the data be normalized?

As with the selection of raw data inputs, domain knowledge is critical to the choice of pre-processing methods. Pre-processing are needed so as to remove the noise embedded in the signal so that the signals are smoother and to normalize the signal range within operable range of neural network. The method of pre-processing depends very much on the success of the approximation of the signal.

Transformation

Smoothing Techniques

To reduce the influence of noise in the data set and improve the compression performance when the original data set is chaotic or has a lot of vibrations, three techniques are used in this project. They are Cyclic Transformation, Leveling and Bounding.

Cyclic Transformation The Cyclic Transformation techniques is used to 'transform' the data set to become a cycle that is the first and the last input data are the same.

The formula for transforming each data value x_i which represents the actual data set $(1 \leq i \leq N)$, to an input value y_i which represents the processed data set $(1 \leq i \leq N)$, is:

$$x_{i+1} = x_{i+1} - i \times \frac{x_N - x_1}{N - 1} \qquad (3.1)$$

where x_N and $x_1$are the last and the first raw input data respectively.

Leveling This technique is to make the data set having the mathematical behavior of summing up to zero.

The formula of leveling for each data value x_i which represents that the actual data $(1 \leq i \leq N)$, is given by:

$$x_i = x_i - x_{mean} \qquad (3.2)$$

where x_{mean} is the mean of the data set.

Bounding It helps to make the data set having the mathematical behavior of being bounded by one.

The formula of bounding for each data value $x_i > 0$ which represents that actual data set $(1 \leq i \leq N)$, is given by:

$$x_i = \frac{x_i}{x_{\max}} \qquad (3.3)$$

where $x_{\max}$ is the maximum of the data set.

Normalization

The goal of the normalization is to ensure that the distribution of values for each net input and output is in a roughly uniform. If this is not done, and input with, say, a normal distribution and small variance is used, then the net will only see a small number of occurrences of facts away from the central tendency. Such a net will not perform well on such data in the future as the fluctuating property will still affect too much in the dynamical system. The values should also be scaled to match the range of the input neurons. Therefore, in addition to any other transformations (weighted moving average) performed on network inputs, each should be normalized. Besides, according to the "*Approximation Theory of Continuous-time Recurrent Networks*" (Li, 1992), the data set is re-scaled from $\Re$ to $(-1, 1)$.

The formula for transforming each data value x_i which represents the actual data set $(1 \leq i \leq m)$, to an input value $\overline{x_i}$ which represents the processed data set $(1 \leq i \leq m)$, is:

$$\overline{x_i} = \left(\frac{x_i - x_{\min}}{x_{\max} - x_{\min}} - x_{\min}\right) \times amplitude \qquad (3.4)$$

where $x_{\min}$, $x_{\max}$ and *amplitude* are minimum. maximum and new range amplitude respectively. This method of normalization can scale input data into the new range, but does not increase its uniformity. Let us call this method as simple linear scaling normalization here.

In addition, the sigmoid function $\sigma = \tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$ is chosen as the unipolar activation function for this project. One can easily see that the sigmoid function σ is just the logistic function transformed by stretching the x and y axes by a factor of two and sliding the resulting curve down so that the working range for compression must be equal to or less than $[-1, 1]$.

In this project, range is chosen randomly and the data were normalized between $[-0.5, 0.5]$. Therefore, it is convenient to set the working range of the sigmoid function in $[-0.5, 0.5]$, so that the data are normalized between $[-0.5, 0.5]$.

3.1.2 Segmentation

Given a one-dimensional finite signal sequence $\{z_t\}$ of length m, let $x_1(1) = z_1$, $x_2(1) = z_{p+1}$, $x_3(1) = z_{2p+1}$, and $x_n(1) = z_{(n-1)p+1}$, where $1 \leq t \leq m$. If the sequence was divided into n segments with p data in each segment, then the system (2.3) iterates p times to generate an approximation of $\{z_t\}$. Here the initial state is the exact values z_t of at the corresponding t's.

If the *Root Mean Square Error* (l_2 error) between the output of the system and the smoothed sequence is less than a pre-decided tolerance, then the dynamical system is said to be capable of compression. It will be exactly achieved if the error is zero for some neural parameters $\mathbf{W}$, $\boldsymbol{\theta}$, $\mathbf{J}$ and h. As the system state is exactly equal to the initial state and need not be the desired sequence after iterations, it is reasonable to assume that the error between the actual trajectory and the system state accumulates with the number of iterations. With an aim of reducing the error, further subdividing each subsequence into shorter sub-segments, says length k, and restarting the system at an exact initial state after $(k-1)$ iterations were done. The cost of the multiple segmentation increases the storage of the corresponding initial states.

3.1.3 The Neural Network Dynamic Equation

The following vector differential equation is used for the purpose of compression. The states of the equation is the column vectors of the one dimensional signal $(n \times 1)$ after dividing into $n \times p$ matrix, where $p = \dfrac{m}{n}$. The applicability of the equation to compression is possible if:

1. the segments are correlated, i.e. there exists some kind of relationships between the segments, and
2. the signals are non-chaotic, in other words, the signals have some intrinsic properties that follow some dynamical system locally.

According to the Li [15] "*Approximation Theory and Recurrent Network*" and Li [14] "*Data Compression By Recurrent Neural Network Dynamics*", the recurrent neural nets can be universal approximators of trajectory in $\Re^n$.

In order to compress the given data, in particular, we consider the Leaky-integrator model:

$$\frac{d\mathbf{x}(t)}{dt} = -a\mathbf{x}(t) + b\mathbf{W}\sigma[\mathbf{x}(t) + \boldsymbol{\theta}] + \mathbf{J} \qquad (3.5)$$

where

h	-	step size,
$\mathbf{x}(t)$	-	states of the system at time $t \in (-1,1)^n$
$\mathbf{W}$	-	$n \times n$ synaptic correction weight matrix,
$\boldsymbol{\theta}$	-	bias or threshold, $n \times 1$ matrix,
$\mathbf{J}$	-	external input to the system, $n \times 1$ matrix,
σ	-	sigmoid (activation), which is a bounded, monotonic increasing differentiable function. $\{\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}}$ is chosen as activation function.$\}$

For simplicity, we take a, b to be the identity matrices in the project. Then equation (2.1) was obtained.

By Euler's forward approximation, the next states could be expanded in the following equation:

$$\mathbf{x}(t+1) = \mathbf{x}(t) + h\frac{d\mathbf{x}(t)}{dt} \quad (3.6)$$

where h is the step size.

Then, putting the differential equation into equation (2.1) and get:

$$\mathbf{x}(t+1) = \mathbf{x}(t) + h\{-\mathbf{x}(t) + \mathbf{W}\sigma[\mathbf{x}(t)+\theta] + \mathbf{J}\} \quad (3.7)$$

Now, consider the case when the state of the equation is the column vectors of the one-dimensional signal $(n \times 1)$ after dividing into $n \times p$ matrix, where $p = \frac{m}{n}$.

Besides, rearranging the equation (3.7) such that the unknowns ($\mathbf{W}$, $\boldsymbol{\theta}$, and h) are grouped on the left hand side:

$$\mathbf{W}\sigma[\mathbf{x}(t)+\boldsymbol{\theta}] + \mathbf{J} = \frac{\mathbf{x}(t+1)-\mathbf{x}(t)}{h} + \mathbf{x}(t) \quad (3.8)$$

For simplify, let $\boldsymbol{\theta} = \mathbf{0}$, $\mathbf{J} = \mathbf{0}$, as these two parameters are not dominating factors for the output [1]. Then, the above equation becomes:

$$\mathbf{W}\sigma[\mathbf{x}(t)] = \frac{\mathbf{x}(t+1)-\mathbf{x}(t)}{h} + \mathbf{x}(t) \quad (3.9)$$

If the new variables and are introduced,

$$\mathbf{V}_t = \frac{\mathbf{x}(t+1)-\mathbf{x}(t)}{h} + \mathbf{x}(t) \quad (3.10)$$

and

$$\mathbf{U}_t = \sigma[\mathbf{x}(t)] \quad (3.11)$$

Hence, the above equation can be written as:

$$\mathbf{V} = [V_1 \cdots V_{p-1}], \qquad \mathbf{U} = [U_1 \cdots U_{p-1}]$$

$$\mathbf{WU} = \mathbf{V} \quad (3.12)$$

[1]The parameters can be used to refine the results of compression. As closing to zeros as they are, as higher compression ratio as they can be obtained.

and

$$\mathbf{W} = \mathbf{V}\mathbf{U}^T \left(\mathbf{U}\mathbf{U}^T\right)^{-1} \qquad (3.13)$$

Since there is no guarantee that the inverse matrix will exist, so in practical, if $\left[\mathbf{U}\mathbf{U}^T\right]^{-1}$ does not exist, then we can consider to substitute the inverse matrix $\left[\mathbf{U}\mathbf{U}^T\right]^{-1}$ by a positive definite matrix $\left[c \cdot \mathbf{I} + \mathbf{U}\mathbf{U}^T\right]^{-1}$ for some constant $c > 0$, e.g. $c = 10^{-10}$.

3.1.4 Learning Algorithm

Given a one dimensional finite sequence $y(s)$ as our data set, then we divide the given sequence into n segments with p data in each segment (for simplicity, we assume that $m = np$). Then, w place n segments into n rows to form an $n \times p$ matrix with p columns of data (i.e. $x_i(t)$, $1 \leq t \leq p$, $1 \leq i \leq n$). Assume that for n-neuron RNN algorithm (as mention before), when given the intial value $x_1(t)$ and appropriate $\mathbf{J}$, $\boldsymbol{\theta}$ and step size h, there exists a matrix $\mathbf{W}$ that can exactly re-generate the original data set $y(s)$ if $p = n + 1$. If $p > n + 1$, we approximate the given sequence by choosing suitable parameters with our learning algorithm.

After the network size has been chosen, the most important thing is to find the optimal neural parameters $\mathbf{W}$, $\mathbf{J}$, $\boldsymbol{\theta}$ and h with dimensions $n \times n$, n, n and 1 respectively. Thus, we set up an error function (*Root Mean Square Error* (l_2 error)) so as to measure the performance of the RNN algorithm. The problem becomes a non-linear optimization problem.

In practice, as we have fixed the step size h and pre-processed the data within the range of ± 1, we can neglect the parameter $\mathbf{J}$ and $\boldsymbol{\theta}$ so as to reduce the number of variables. In that case, $\mathbf{J}$ and $\boldsymbol{\theta}$ are zero vectors. Then, after proceeding the data set $y(s)$ with an n-neuron RNN algorithm, we get a neural parameter $\mathbf{W}^*$ and a set of approximate data set $z(s)$ which is the approximation trajectory to the original data set $y(s)$. After that we evaluate the RMSE error between $y(s)$ and $z(s)$ and then apply a learning process so as to improve the result, the details are as follows.

Recently, many researchers use William and Zipser (1989) algorithm or other gradient descent based learning algorithm for discrete RNN. These algorithms need to compute the derivative along the trajectory which costing and time consuming. There are a lot of learning algorithms in neural networks because minimize the error function is a non-linear optimization problem and there is no simple way to guarantee to obtain the minimum. Our learning algorithm involves convex linear combination of the original sequence and the system output, so that no derivative is computed. In addition, there is no hidden neuron used in all our examples of this project, we use a new learning algorithm to train our RNN algorithm.

In our approach, if there exist a neural parameter $\mathbf{W}$ that can exactly regenerate the original trajectory $y(s) \in [-1,1]^m$ by RNN dynamic, we will find the neural parameter $\mathbf{W}$ at the first trial. If not, our learning algorithm is to construct a convex linear combination of original trajectory $y(s)$ and the approximation trajectory $z(s)$ to form a new data set $y'(s) \in [-1,1]^m$, where $z(s)$ is generated by the RNN dynamic and $y'(s)$ is the data sequence to be approximated. Geometrically, we try to find a convex combination of these two sequences, such that the new sequences $y'(s)$ can also be generated by the RNN dynamic with a new $\mathbf{W}$. And that approximation trajectory will be around some neighborhood of the original trajectory $y(s)$ that can be generated by the discrete dynamic of an RNN for some $\mathbf{W}$.

This learning algorithm divided into 2 stages:

1. We find the W that satisfies the equation (3.13).

2. After that, we generate the sequence by the RNN dynamic, and then adjust the sequence by the convex linear combination as follows:

$$y'(s) = (1-\alpha)y(s) + \alpha z(s) \dots\dots\dots (*)$$

Instead of approximate $y(s)$ directly, we approximate $y'(s)$ in the next iteration. Hence, we repeat all the steps above with the new data set $y'(s)$ and get another new neural parameter $\widetilde{\mathbf{W}}$ and a set of approximate data set $z'(s)$ which is also the approximation trajectory to the original data set $y(s)$.

Compute and record the RMSE error again. The learning algorithm repeated the iterations by using the original data set $y(s)$ and the new approximation trajectory. The error was computed in each iteration and only the best solution was stored. However, this algorithm has just been in the initial stage, the proper fraction α and the necessarily parameter h is obtained by trial and error method. Further analysis such as convergence, error bound of this learning algorithm is still under development.

3.2 The Error Analysis and Estimation

In this section, the error analysis, the errors and compression ratio estimations will be discussed.

3.2.1 Dynamic System Method

Consider

$le = n \cdot p$	:	number of original data, given that le is divisible by n;
n	:	number of segments divided;
ε	:	maximum one-directional propagation error;
$E_{i,j}$	:	sum of squared error when using i segments, j initial points;
$R_{i,j}$	:	compression ratio when using i segments, j initial points;

The compression ratio is defined as following:

$$R_{i,j} = \frac{\textit{total number of original data}}{\textit{number of data that required for regenerating the approximate data}}$$

In view of Iterativ ; Approach

Suppose that there is a original signal sequence $\{z_t\}$, $1 \leq t \leq le = n \cdot p$, where

n	:	number of segments divided;
p	:	length of each segment;

Hence, after segmentation, the original sequence becomes:

$$(z_1 \cdots\cdots z_{np})_{1\times np} \quad \xrightarrow{segmentation} \quad \begin{pmatrix} z_1 & z_2 & \cdots & z_p \\ z_{p+1} & z_{p+2} & \cdots & z_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ z_{(n-1)p+1} & z_{(n-1)p+2} & \cdots & z_{n\cdot p} \end{pmatrix}_{n\times p}$$

Error Bounds

Consider the case when there is a sequence $\{y_t\}$, $1 \le t \le n \cdot p = le$, which is an approximation of the original signal sequence $\{z_t\}$. Define $|z_{t+1} - z_t| = \varepsilon_t$ where $t = 1, 2, 3, \ldots, n{\cdot}p$, and $Max\,|\varepsilon_t| = \varepsilon$. Then the worst case occurs when the maximum error ε accumulates uniformly with the iteration, i.e. $\varepsilon_t = t \cdot \varepsilon$.

Due to the segmentation method shown above, there are a total of p iterations for each segment and the total of n segments. Therefore, the accumulated error for a single segment is:

$$E_i = \sum_{t=1}^{p} \varepsilon_t^2 \le \sum_{t=1}^{p} (t \cdot \varepsilon)^2 \ldots\ldots \quad (3.14)$$

where $i = 1, 2, \ldots\ldots, n$.

The total error is bounded as a function of n, p, and ϵ because

$$\sum_{i=1}^{n} E_i = \sum_{i=1}^{n}\sum_{t=1}^{p} (t \cdot \varepsilon)^2 \le n\varepsilon^2 \sum_{t=1}^{p} t^2 = n\varepsilon^2 \frac{p\,(p+1)\,(2p+1)}{6} \ldots\ldots \quad (3.15)$$

Thus, the RMSE becomes:

$$RMSE = \left[\frac{1}{np}\sum_{i=1}^{n} E_i\right]^{\frac{1}{2}} \le \left[\frac{1}{np} n\varepsilon^2 \frac{p\,(p+1)\,(2p+1)}{6}\right]^{\frac{1}{2}}$$

$$= \left[\frac{(p+1)\,(2p+1)}{6}\varepsilon^2\right]^{\frac{1}{2}}$$

3.2.2 Recurrent Neural Network System Method

The number of data required for regenerating the original data contains the following variables:

W - which is an $n \times n$ matrix	:	n^2
$\boldsymbol{\theta}$ - which is an $n \times 1$ matrix	:	n
J - which is an $n \times 1$ matrix	:	n
h - step size of the different equation	:	1
initial value - which is an $n \times 1$ matrix	:	n
		$n^2 + 3n + 1$

Hence, the compression ratio becomes:

$$R_{n,1} = \frac{np}{n^2 + 3n + 1} \qquad (3.16)$$

In addition, the number of propagation is:

$$s = \left(\frac{np}{n} - 1\right) = p - 1 \qquad (3.17)$$

Consider there is a sequence $\{y_t\}$, $1 \leq t \leq np$ which is the approximation of the original signal sequence $\{z_t\}$. Assume that $\varepsilon = \max|y_t - z_t|$, $\forall t \in [1, np]$, and the error is in uniform propagation that $\varepsilon_k = k\varepsilon$, then the sum of all square of errors is bounded by $E_{n,1}$, i.e.

$$E_{n,1} = \sum_{i=1}^{n} \sum_{k=1}^{s} (k\varepsilon)^2 = \frac{s(s+1)(2s+1)}{6} n\varepsilon^2 \qquad (3.18)$$

Improvement as the network size increases

Increasing the network size from n to $(n+1)$

Consider the case of $n+1$, the compression ratio becomes:

$$R_{n+1,1} = \frac{np}{(n+1)^2 + 3(n+1) + 1} = \frac{np}{n^2 + 5n + 5} \qquad (3.19)$$

Hence, the estimated error bound becomes:

$$E_{n+1,1} = \frac{\frac{sn}{n+1}\left(\frac{sn}{n+1}+1\right)\left(\frac{2sn}{n+1}+1\right)}{6}(n+1)\varepsilon^2 \qquad (3.20)$$

Then the improvement in estimated error bound relate to E_n is:

$$\frac{E_{n+1,1}}{E_{n,1}} = \frac{\frac{\frac{sn}{n+1}\left(\frac{sn}{n+1}+1\right)\left(\frac{2sn}{n+1}+1\right)}{6}(n+1)\varepsilon^2}{\frac{s(s+1)(2s+1)}{6}n\varepsilon^2}$$
$$= \frac{(sn+n+1)(2sn+n+1)}{(s+1)(2s+1)(n+1)^2} \qquad (3.21)$$

Since $s = p-1$, then after substitution it becomes:

$$\frac{E_{n+1,1}}{E_{n,1}} = \frac{(pn+1)(2pn-n+1)}{p(2p-1)(n+1)^2} = \frac{(pn+1)(2pn-n+1)}{(pn+n)(2pn-n)}\frac{n^2}{(n+1)^2}$$
$$\approx \left(\frac{n}{n+1}\right)^2 \qquad (3.22)$$

The number of data required $= \left[(n+1)^2 + 3(n+1) + 1\right] = n^2 + 5n + 5$, which is $(2n+4)$ data more than that in using n segment. Roughly speaking, when one more segment is used, the number of data required for compression in two cases have no significant difference. However, the error will reduce by approximately $\left(\frac{n}{n+1}\right)^2$ compared to the case of using n-segment.

Double the number of segments

Now, consider the case of $2n$, the estimated error bound then becomes:

$$E_{2n,1} = \frac{\frac{s}{2}\left(\frac{s}{2}+1\right)\left(\frac{2s}{2}+1\right)}{6}2n\varepsilon^2 \approx \frac{1}{4}E_{n,1} \qquad (3.23)$$

i.e. the error reduces by approximately 75% compared to the case before segment doubling.

Hence, the compression ratio becomes:

$$R_{2n,1} = \frac{np}{(2n)^2 + 3(2n) + 1} = \frac{np}{4n^2 + 6n + 1} \approx \frac{1}{4}R_{n,1} \qquad (3.24)$$

Doubling the segment will not only reduce the error, but also reduce the compression ratio by approximately to a quarter.

The number of data required $= \left[(2n)^2 + 3(2n) + 1\right] = 4n^2 + 6n + 1$, which is $(3n^2 + 3n)$ more when comparing to simplest n-segment method.

Chapter 4

Equation and Techniques for programming

In order to demonstrate our compressing technique, we apply our technique through computer simulation and compress the UV and IR spectra data provided by Professor Chau of the Applied Biology and Chemical Technology Department, at The Hong Kong Polytechnic University.

In this project, we use the software "Matlab for Windows" to handle all the programming matters.

4.1 The neural network equations for programming

Recall the discrete recurrent neural network equation in chapter [3].

$$\mathbf{x}(n+1) = \mathbf{x}(n) + h\left\{-\mathbf{x}(n) + \mathbf{W}\sigma\left[\mathbf{x}(n) + \boldsymbol{\theta}\right] + \mathbf{J}\right\} \quad \ldots\ldots\ldots\ldots\ldots (4.1)$$

where $\mathbf{x}(n)$ represents the data sequence after preprocessing. If the sequence is divided into n segments and is defined the n-dimensional sequence as $\mathbf{x}_s$, then the above equation becomes:

$$\mathbf{x}_s(n+1) = \mathbf{x}_s(n) + h\left\{-\mathbf{x}_s(n) + \mathbf{W}\sigma\left[\mathbf{x}_s(n) + \boldsymbol{\theta}\right] + \mathbf{J}\right\} \ldots\ldots\ldots\ldots (4.2)$$

In this equation $\mathbf{x}_s$ is already known, and h can be estimated by trial and error. The remaining neural parameters required to be calculated are $\mathbf{W}$, $\mathbf{J}$ and $\boldsymbol{\theta}$.

With some simple rearrangements, and letting $\mathbf{S} = \sigma\left(\mathbf{x}_s(n) + \boldsymbol{\theta}\right)$, then the following equations can be used for the purpose of programming ;

$$\mathbf{W} = \left\{\left[\frac{\mathbf{x}_s(n+1) - \mathbf{x}_s(n)}{h} + \mathbf{x}_s(n) - \mathbf{J}\right]\mathbf{S}^t\right\}\left\{\mathbf{S}\mathbf{S}^t\right\}^{-1} \ldots\ldots\ldots\ldots (4.3)$$

where $n = 1$ to m, which is the total number of patterns.

When the weight matrix $\mathbf{W}$ is found, then $\mathbf{J}$ and $\boldsymbol{\theta}$ can be calculated with the following equations:

$$\mathbf{J} = \frac{1}{p-1}\sum_{n=1}^{p}\left\{\frac{\mathbf{x}_s(n+1) - \mathbf{x}_s(n)}{h} + \mathbf{x}_s(n) - \mathbf{W}\sigma\left[\mathbf{x}_s(n) + \boldsymbol{\theta}\right]\right\} \ldots (4.4)$$

$$\boldsymbol{\theta} = \sum_{n=1}^{p}\left\{\sigma^{-1}\left[\mathbf{W}^{-1}\left(\frac{\mathbf{x}_s(n+1) - \mathbf{x}_s(n)}{h} - \mathbf{J}\right)\right] - \mathbf{x}_s(n)\right\} \ldots\ldots\ldots\ldots (4.5)$$

In the equations, the finding of $\mathbf{W}$, $\mathbf{J}$ and $\boldsymbol{\theta}$ are still unknown. There are several schemes can be applied for finding those parameters.

- set $\boldsymbol{\theta} = \mathbf{0}$ and $\mathbf{J} = \mathbf{0}$, then find $\mathbf{W}$, then $\boldsymbol{\theta}$, then $\mathbf{J}$, and then $\mathbf{W}$. The cycle goes for several times.

- set $\boldsymbol{\theta} = \alpha\,(median) + (1-\alpha)\,mean$ and $\mathbf{J} = \mathbf{0}$, then find $\mathbf{W}$, then $\boldsymbol{\theta}$, then $\mathbf{J}$, and then $\mathbf{W}$. Also the cycle goes for several times.

- fix $\boldsymbol{\theta} = \frac{median}{mean}$ and J = 0, then find **W**, then $\boldsymbol{\theta}$, then J, and then **W**. This cycle also goes for several times.

To implement, the parameters $\boldsymbol{\theta}$ and J were set to be *zero* in this project, i.e. $\boldsymbol{\theta} = 0$ and $\mathrm{J} = 0$.

4.2 Techniques for Learning Algorithm

For the purpose of minimizing the timing for learning, we only may apply the learning algorithm in the recurrent neural network. It means that pre-processing techniques such as normalization, cyclic transformation, leveling and bounding techniques will be applied before learning algorithm.

On the basis of the learning algorithm equation mentioned in previous chapter, let $y(s)$ be the original data segment after pre-processing and segmentation. Then putting this original data set into the process of the recurrent neural network, the approximating data set $z(s)$ will be generated. Hence, with the aim of training the weight matrix **W**, using the following equation to produce a new data set $y'(s)$ and putting it into the process of the network again until the acceptable tolerance *Root Mean Square Error* ("RMSE")(l_2 error) while compared to the actual original data segments have been achieved.

$$y'(s) = (1-\alpha)y(s) + \alpha z(s) \quad (4.6)$$

where

α	-	the trial and error proper fraction for the original and approximating data,
$y(s)$	-	the original data set,
$z(s)$	-	the approximating data set by the recurrent neural network,
$y'(s)$	-	the new original data set.

The proper fraction value α was different in different chemical combination, i.e. different UV spectra have different α. Since there is no particular method to find α, trial and error method has been used in this project.

The equations and techniques have been discussed thoroughly in this chapter and also in Chapter [3], it is time to implement the algorithms with computer programs. In order to increase the compression efficiency further, the parameters $\boldsymbol{\theta}$ and $\mathbf{J}$ were assumed to be zero vectors in this project.

Hence the compression ratio becomes:

$$R_{n,1} = \frac{le}{n^2+n+1} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4.7)$$

The lists of the program are given in Chapter [9] Appendices. The results of UV and IR spectra will be presented as diagrams in the subsequent two chapters, i.e. Chapter [5] and Chapter [6]. The errors calculated will be tabulated for easy comparison between the chemicals and different techniques used. As there are different pre-processing techniques are used together to find out the best results, different flow charts will also be given for easy following.

Chapter 5

Results from Ultra Violet Spectra

In the first section of this chapter, we will discuss the results of the original data set and data sets with different noise added by one-directional propagation. In the second section, comments on the results will be given.

5.1 Results - UV Spectra using Learning algorithm

The following diagrams show the result of original data sets and approximated data sets when learning algorithm ("L") with several network being used. Besides, cyclic transformation, leveling, bounding and normalization were used as the pre-processing techniques. The flow chart of the program for those results of 5.1 is shown in Figure 5.1.

In order to compare the data more easily, the value of 0.3 was added to the original data. In the diagrams, the *upper* curves are the *original curves* which were printed in red, and the *lower* curves are the *approximation curves* which were printed in blue.

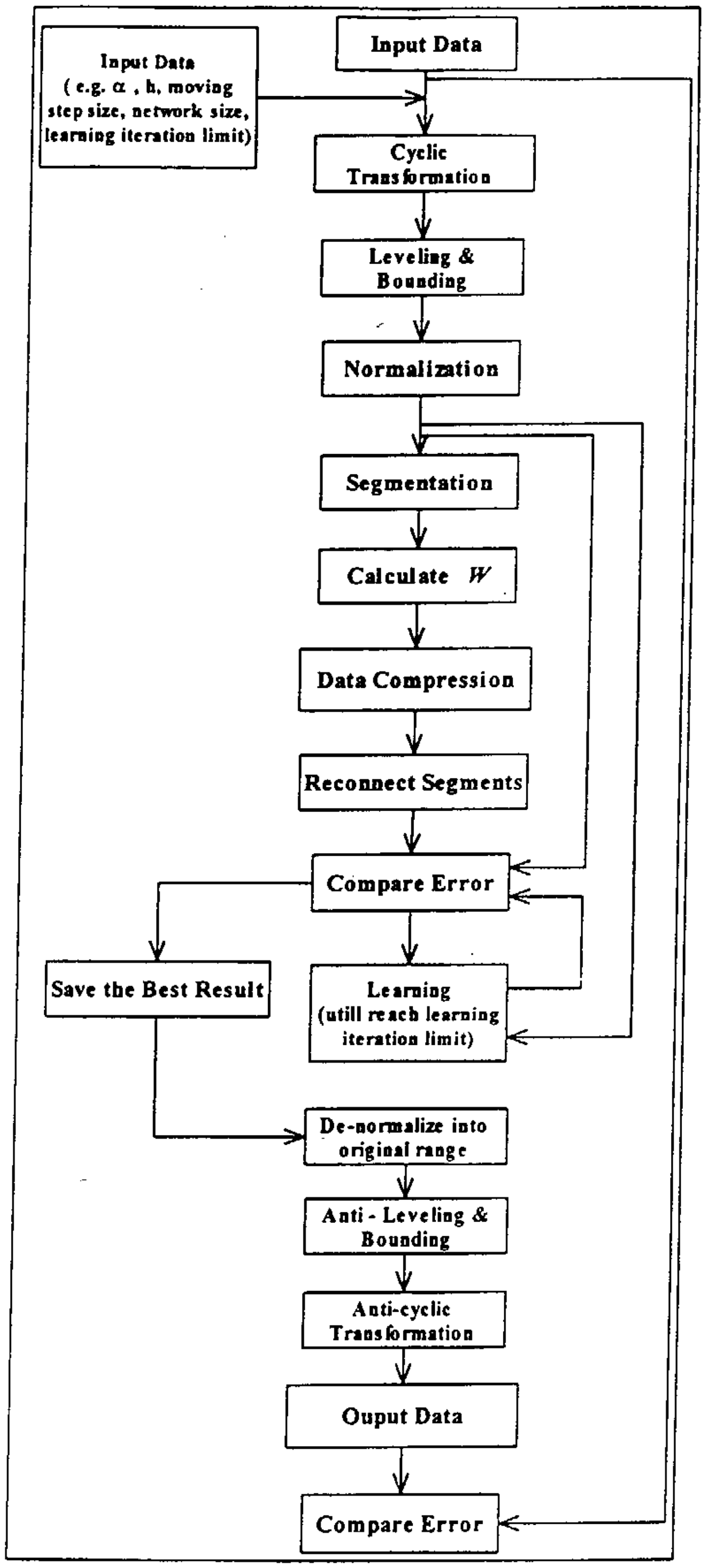

Input Data
Input Data (e.g. α , h, moving step size, network size, learning iteration limit)
Cyclic Transformation
Leveling & Bounding
Normalization
Segmentation
Calculate W
Data Compression
Reconnect Segments
Compare Error
Save the Best Result
Learning (utill reach learning iteration limit)
De-normalize into original range
Anti - Leveling & Bounding
Anti-cyclic Transformation
Ouput Data
Compare Error

Figure 5-1: Flow chart for the results of Section 5.1.

The corresponding maximum absolute error(l_∞ error), mean absolute error(l_1 error), root mean square error ("RMSE") (l_2 error) and their relative error percentages(%) are calculated and tabulated in the tables shown below.

In addition, the compression ratio:

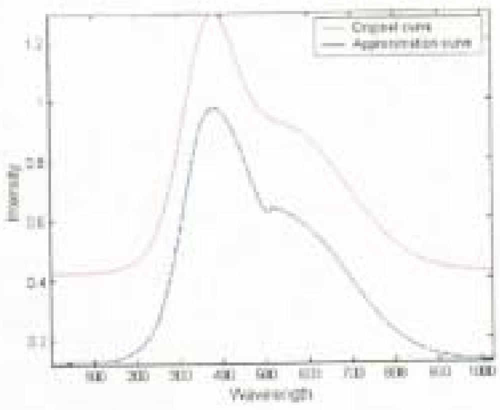
$$R_{i,j} = \frac{\textit{total number of original data}}{\textit{number of data that required for regenerating the approximate data}}$$

where i - number of segments;

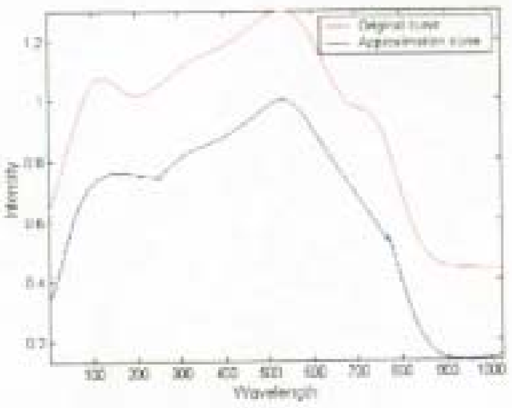
j - number of initial point.

have also been calculated.

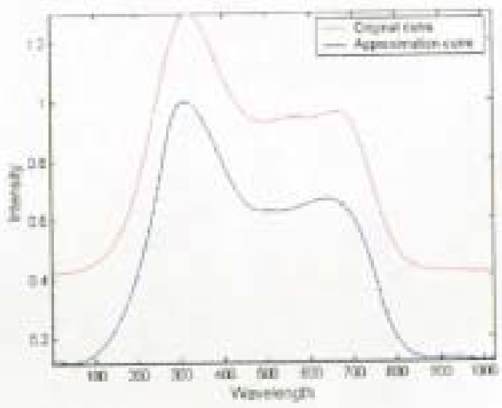
5.1.1 Single initial values using 4 segments



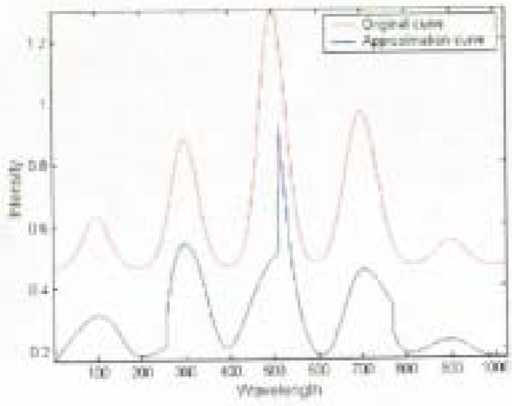
UV in a, 4 segments



UV in b, 4 segments



UV in c, 4 segments

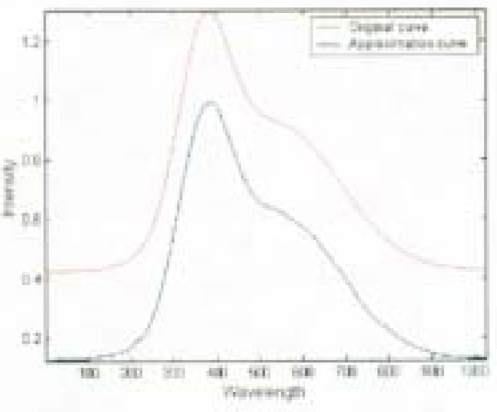


UV in d, 4 segments

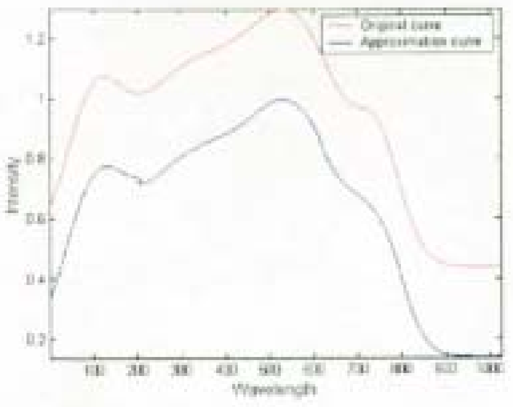
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	1.4407e-002	2.0867e-003	3.8845e-003	1.6442	0.2382	0.4433
UV in b	2.4379e-002	5.0021e-003	8.0416e-003	2.8201	0.5788	0.9305
UV in c	1.9960e-002	3.6669e-003	5.0481e-003	2.2657	0.4162	0.5730
UV in d	1.9977e-001	2.7976e-002	4.3723e-002	23.9736	3.35729	5.2470

Table 5.1: Errors of the UV spectra when 4 segments and learning is used. ($R_{4,1} = \dfrac{1024}{4^2+4+1} = 48.7619$)

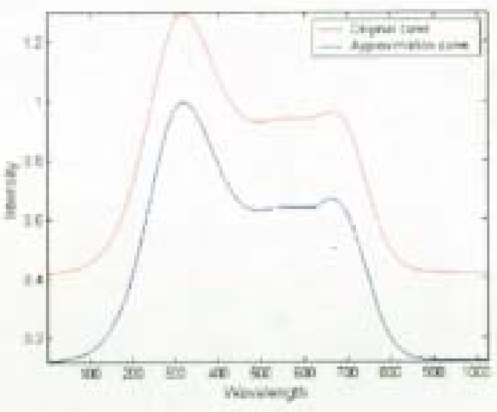
5.1.2 Single initial values using 5 segments



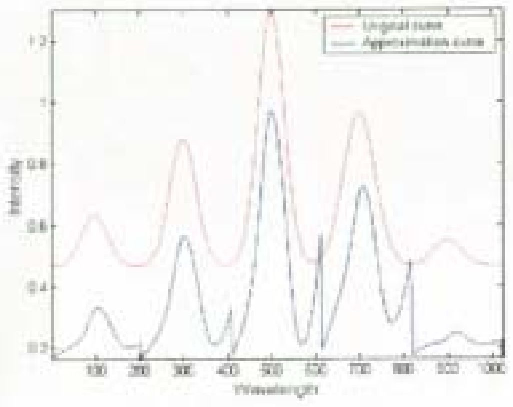
UV in a, 5 segments



UV in b, 5 segments



Uv in c, 5 segments

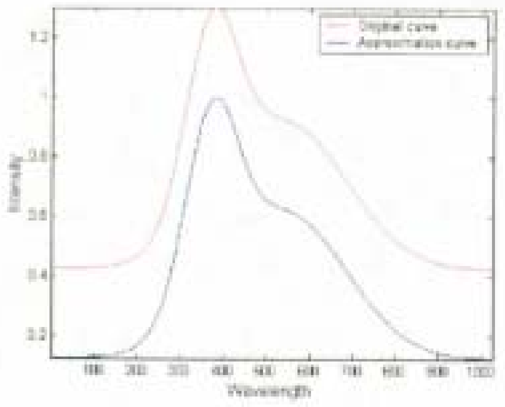


UV in d, 5 segments

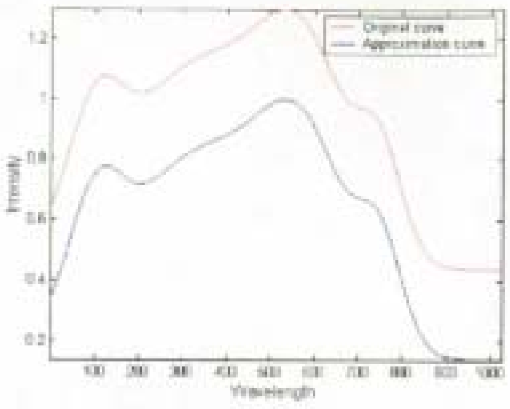
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	1.2865e-002	1.5325e-003	2.3840e-003	1.4683	0.1749	0.2721
UV in b	1.1069e-002	1.9349e-003	3.2344e-003	1.2807	0.2239	0.3742
UV in c	3.5037e-003	6.1355e-004	9.3037e-004	0.3977	0.0696	0.0930
UV in d	1.8824e-001	1.9409e-002	3.2059e-002	22.5900	2.3292	3.8473

Table 5.2: Errors of the UV spectra when 5 segments and learning is used. ($R_{5,1} = \dfrac{1020+4}{5^2+5+1+4} = 29.2571$)

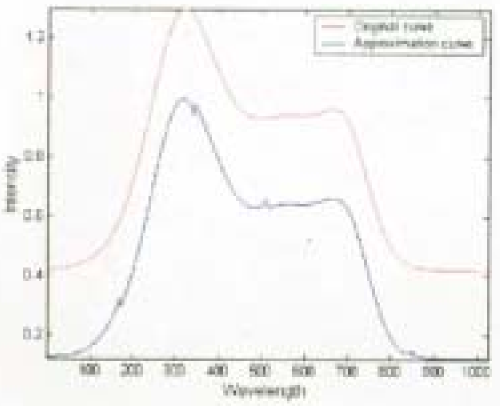
5.1.3 Single initial values using 6 segments



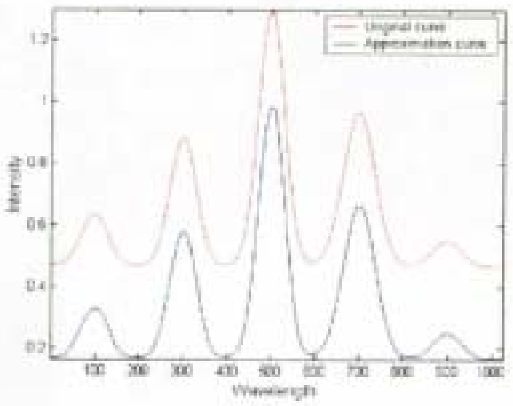
UV in a, 6 segments



UV in b, 6 segments



UV in c, 6 segments

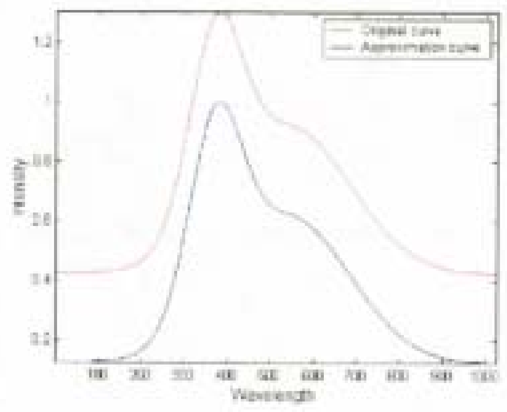


UV in d, 6 segments

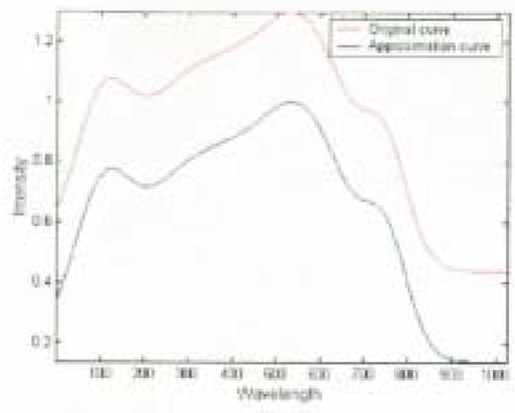
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	3.0069e-003	2.8125e-004	6.0098e-004	0.3432	0.0321	0.0686
UV in b	6.1990e-004	1.3183e-004	1.8103e-004	0.0717	0.0153	0.0209
UV in c	4.2469e-003	4.4959e-004	7.0619e-004	0.4821	0.5103	0.0802
UV in d	9.0120e-003	1.2776e-003	1.8963e-003	1.0230	0.1450	0.2153

Table 5.3: Errors of the UV spectra when 6 segments and learning is used. ($R_{6,1} = \frac{1020+4}{6^2+6+1+4} = 21.7872$)

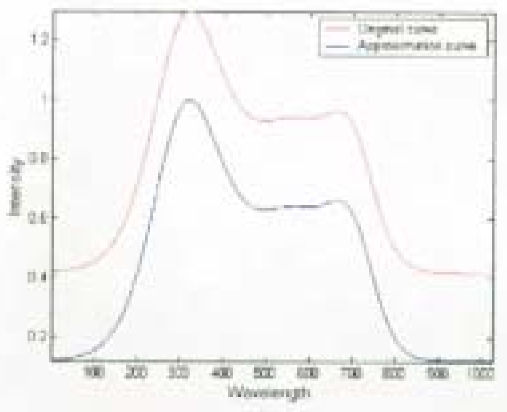
5.1.4 Single initial values using 7 segments



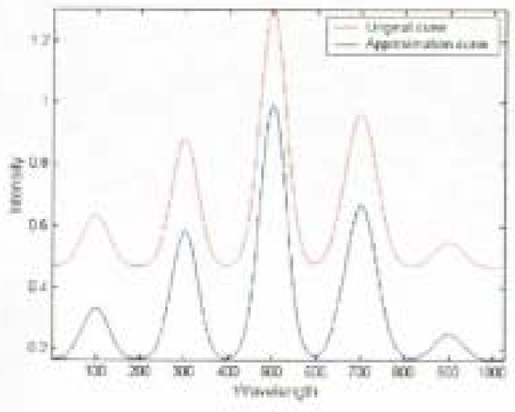
UV in a, 7 segments



UV in b, 7 segments



UV in c, 7 segments

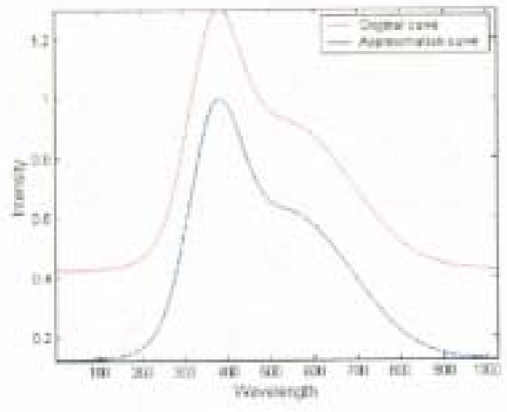


UV in d, 7 segments

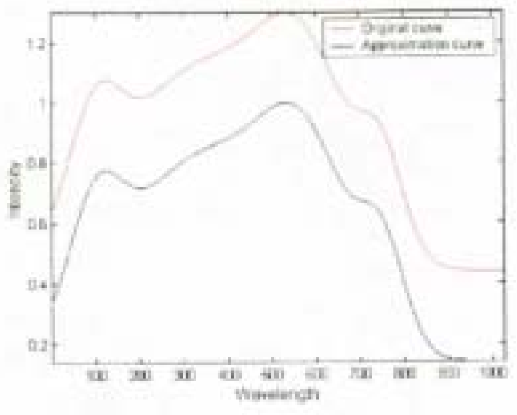
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	1.6181e-004	2.2523e-005	4.0897e-005	0.0185	0.0026	0.0047
UV in b	2.3447e-005	6.6385e-006	8.7570e-006	0.0027	0.0008	0.0010
UV in c	1.1461e-004	3.1241e-005	4.4637e-005	0.0130	0.0035	0.0051
UV in d	4.3512e-003	5.6796e-004	9.1934e-004	0.5222	0.0682	0.1103

Table 5.4: Errors of the UV spectra when 7 segments and learning is used. ($R_{7,1} = \dfrac{1022+2}{7^2+7+1+2} = 17.3559$)

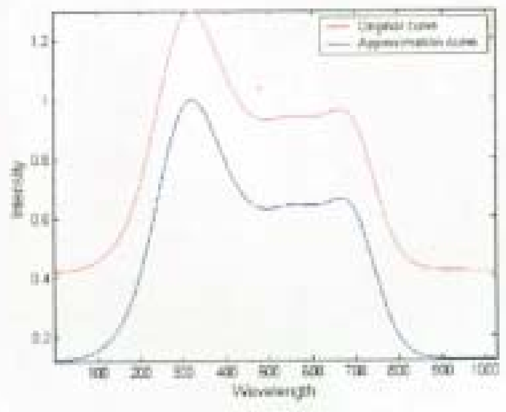
5.1.5 Single initial values using 8 segments



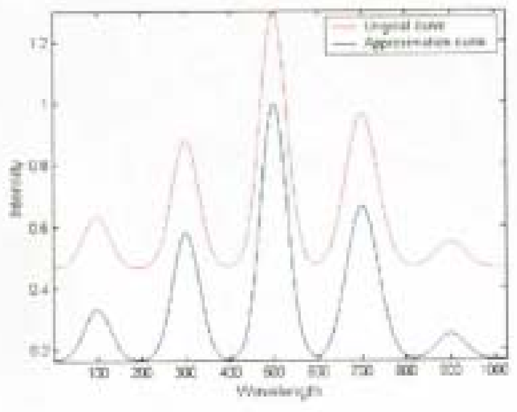
UV in a, 8 segments



UV in b, 8 segments



UV in c, 8 segments

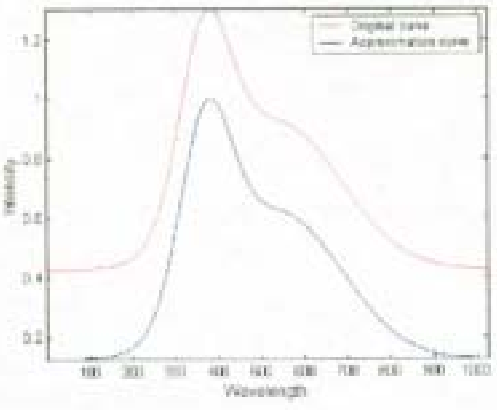


UV in d, 8 segments

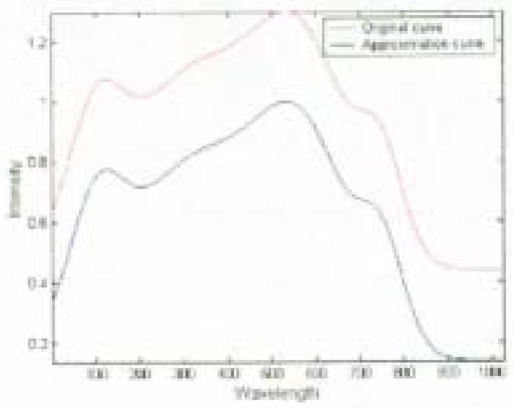
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	2.8837e-005	3.4813e-006	7.1887e-006	0.0033	0.0004	0.0008
UV in b	1.1254e-005	2.5167e-006	3.7524e-006	0.0013	0.0003	0.0004
UV in c	5.8094e-005	8.2003e-006	1.4500e-005	0.0066	0.0009	0.0016
UV in d	1.5034e-003	3.7903e-004	5.0780e-004	0.1804	0.0455	0.0609

Table 5.5: Errors of the UV spectra when 8 segments and learning is used. ($R_{8,1} = \frac{1024}{8^2+8+1} = 14.027$)

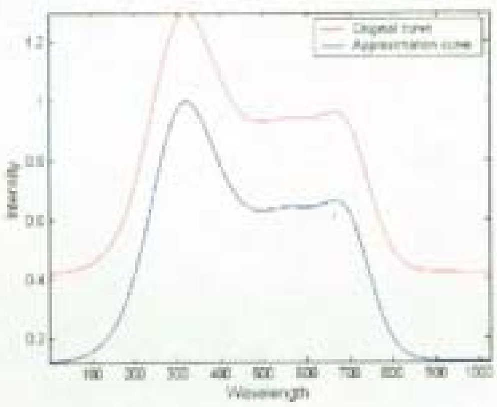
5.1.6 Single initial values using 9 segments



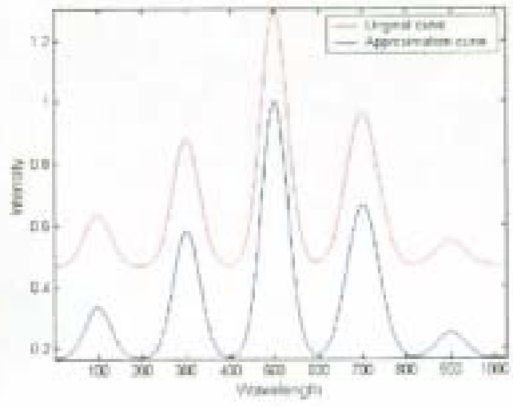
UV in a, 9 segments



UV in b, 9 segments



UV in c, 9 segments

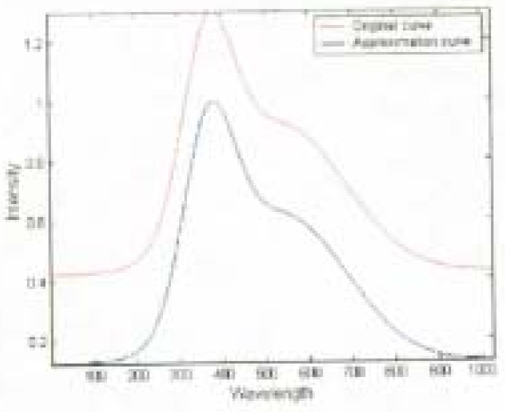


UV in d, 9 segments

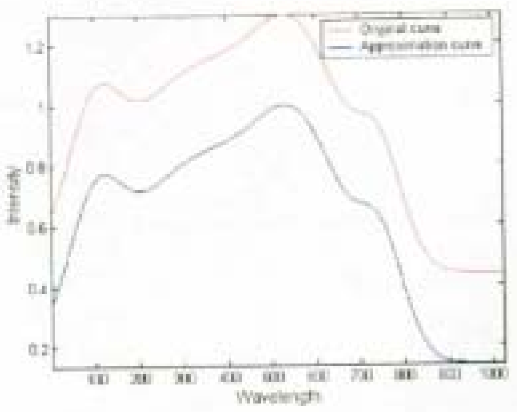
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	7.0922e-006	9.4098e-007	1.9291e-006	0.00081	0.00011	0.00022
UV in b	1.0840e-006	1.9792e-007	3.1093e-007	0.00013	0.00002	0.00004
UV in c	1.1092e-006	1.5636e-007	2.5119e-007	0.00013	0.00002	0.00003
UV in d	1.5855e-004	2.1509e-005	3.5793e-005	0.01903	0.00258	0.00430

Table 5.6: Errors of the UV spectra when 9 segments and learning is used. ($R_{9,1} = \dfrac{1017+7}{9^2+9+1+7} = 10.4490$)

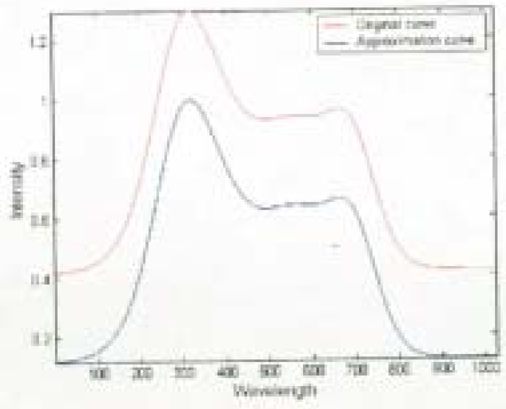
5.1.7 Single initial values using 10 segments



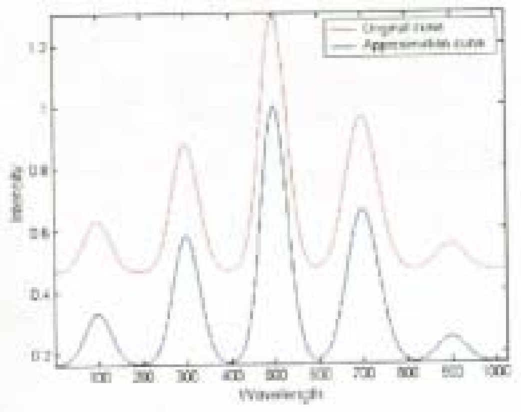
UV in a, 10 segments



UV in b, 10 segments



UV in c, 10 segments



UV in d, 10 segments

Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
UV in a	2.5394e-006	2.1065e-007	4.5805e-007	0.00029	0.00002	0.00005
UV in b	1.6606e-006	3.5702e-007	5.1916e-007	0.00019	0.00004	0.00006
UV in c	7.6635e-007	1.2537e-007	1.9091e-007	0.00009	0.00001	0.00002
UV in d	2.9644e-005	5.1723e-006	7.5185e-006	0.00356	0.00062	0.00090

Table 5.7: Errors of the UV spectra when 10 segments and learning is used. ($R_{10,1} = \dfrac{1020+4}{10^2+10+1+4} = 8.9043$)

5.2 Results - UV Spectra *with some noise* using Learning algorithm

The following diagrams show the result of original data sets and approximated data sets when learning algorithm ("L") with network size of 8 was used. Besides, cyclic transformation, leveling, bounding and normalization were used as the pre-processing techniques. The flow chart of the program for those results of 5.2 is shown in Figure 5.2.

In order to compare the data more easily, the value of 0.3 was added to the original data. In the diagrams, the *upper* curves are the *original curves* which were printed in red, and the *lower* curves are the *recovered curves* which were printed in blue.

The corresponding maximum absolute error(l_∞ error), mean absolute error(l_1 error), root mean squ≀re error ("RMSE") (l_2 error) and their relative error perc≀entages(%) are calculated and tabulated in the tables shown below.

In addition, the compression ratio:

$$R_{i,j} = \frac{\textit{total number of original data}}{\textit{number of data that required for regenerating the approximate data}}$$

where i - number of segments;

j - number of initial point.

have also been calculated.

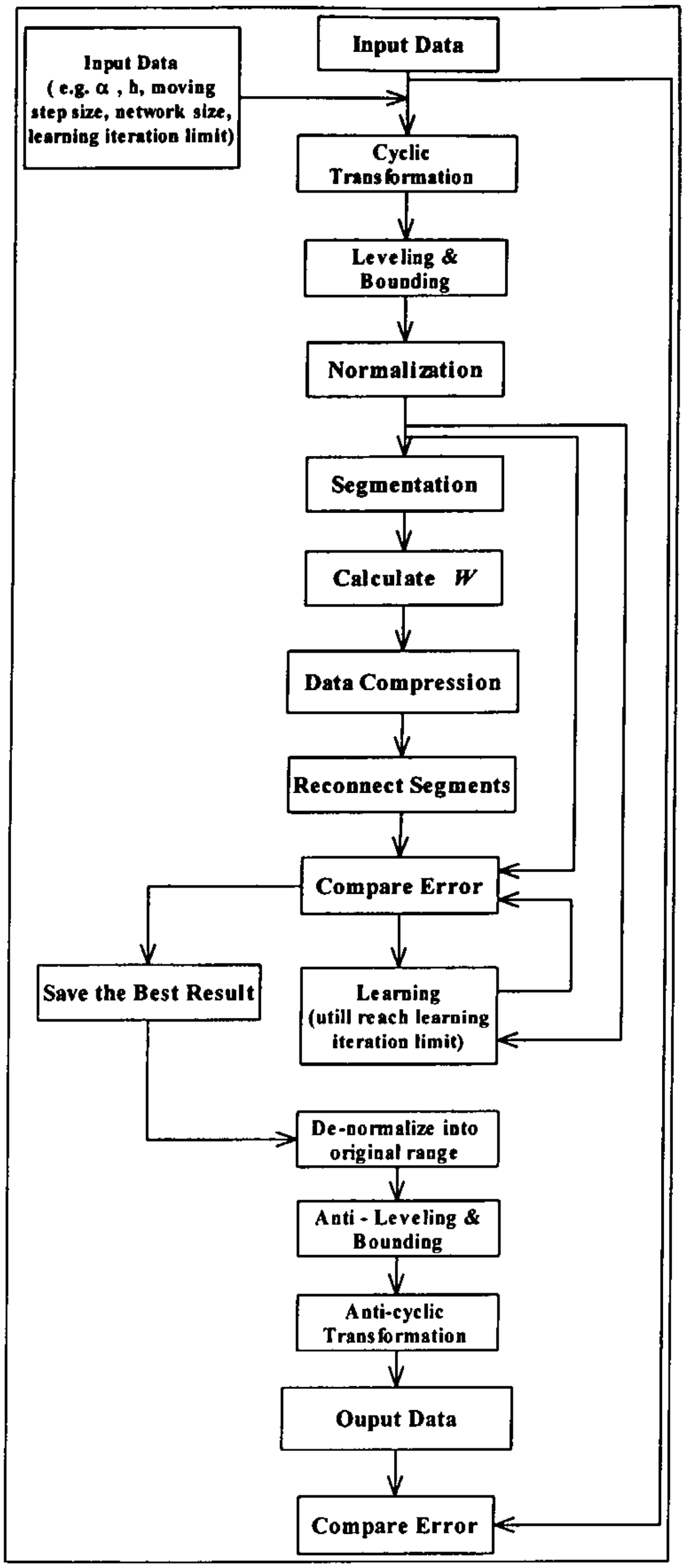
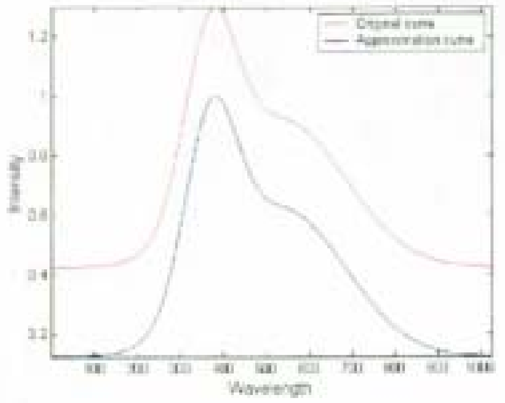
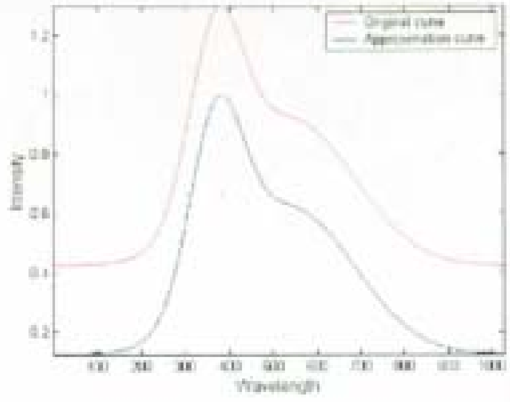

Input Data
Input Data (e.g. α , h, moving step size, network size, learning iteration limit)
Cyclic Transformation
Leveling & Bounding
Normalization
Segmentation
Calculate W
Data Compression
Reconnect Segments
Compare Error
Save the Best Result
Learning (utill reach learning iteration limit)
De-normalize into original range
Anti - Leveling & Bounding
Anti-cyclic Transformation
Ouput Data
Compare Error

Figure 5-2: Flow chart for the results of Section 5.2.

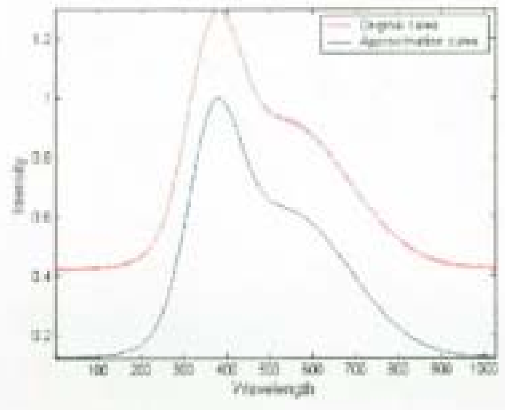
5.2.1 Single initial values using 8 segments for UV in a



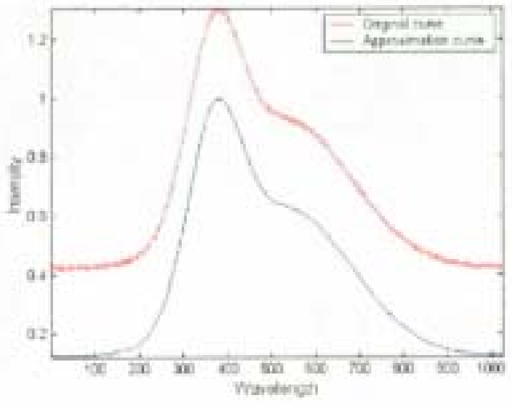
UV in a, 8 segments (Noise free)



UV in a, 8 segments (Noise level $\pm 1 \times 10^{-3}$)



UV in a, 8 segments (Noise Level $\pm 5 \times 10^{-3}$)

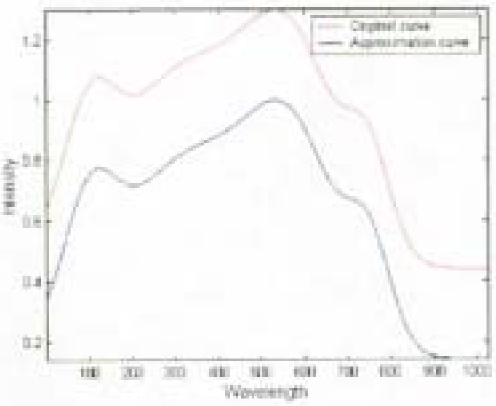


UV in a, 8 segments (Noise Level $\pm 1 \times 10^{-2}$)

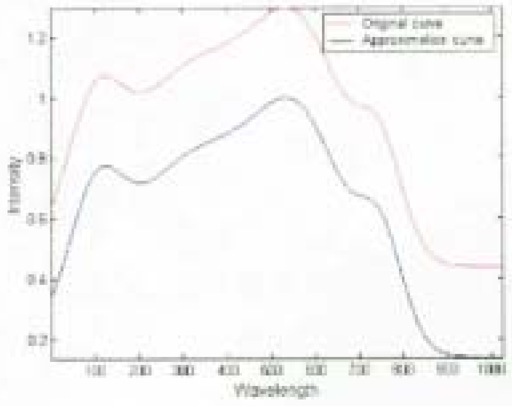
Chemicals UV in a	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
Noise free	1.6181e-004	2.2523e-005	4.0897e-005	0.0185	0.0026	0.0047
$\pm 1 \times 10^{-3}$	7.4880e-004	2.4874e-004	2.9295e-004	0.0855	0.0284	0.0334
$\pm 5 \times 10^{-3}$	4.7061e-003	1.2879e-003	1.5111e-003	0.5371	0.1470	0.1725
$\pm 1 \times 10^{-2}$	9.1116e-003	2.5556e-003	2.9923e-003	1.0399	0.2917	0.3415

Table 5.8: Errors of the UVina with different noise when 8 segments and learning is used. ($R_{8,1} = \dfrac{1024}{8^2+8+1} = 14.027$)

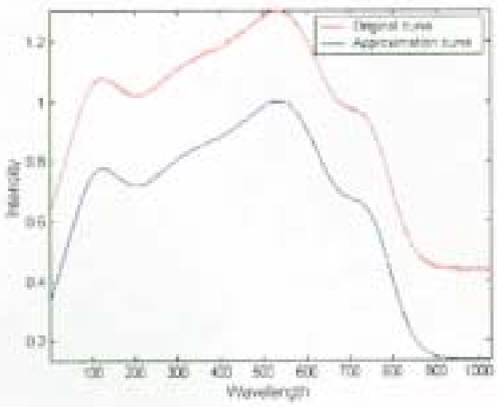
5.2.2 Single initial values using 8 segments for UV in b



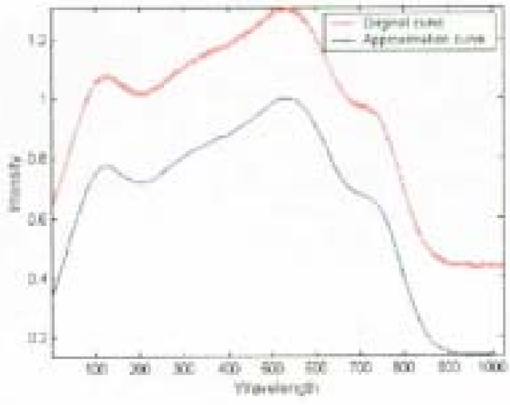
UV in b, 8 segments (Noise free)



UV in b, 8 segments (Noise Level $\pm 1 \times 10^{-3}$)



UV in b, 8 segments (Noise Level $\pm 5 \times 10^{-3}$)

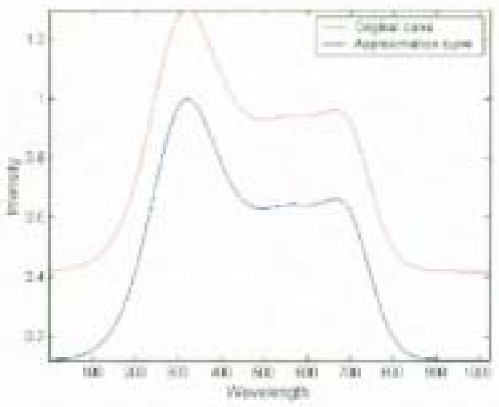


UV in b, 8 segments (Noise Level $\pm 1 \times 10^{-2}$)

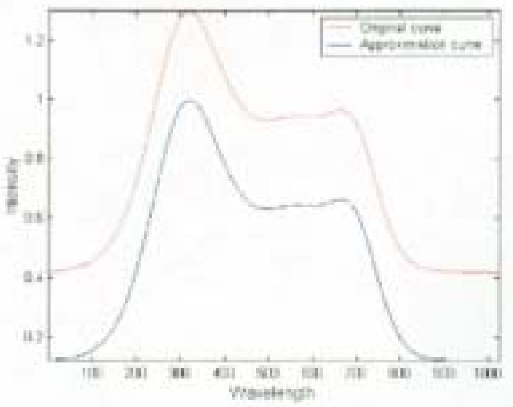
Chemicals UV in a	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
Noise free	1.1254e-005	2.5167e-006	3.7524e-006	0.0013	0.0003	0.0004
$\pm 1 \times 10^{-3}$	7.4292e-004	2.5223e-004	3.0400e-004	0.0860	0.0292	0.0352
$\pm 5 \times 10^{-3}$	3.4152e-003	1.2349e-003	1.4799e-003	0.3952	0.1429	0.1712
$\pm 1 \times 10^{-2}$	6.9719e-003	2.4788e-003	2.9353e-003	0.8067	0.2868	0.3396

Table 5.9: Errors of the UVinb with different noise when 8 segments and learning is used. ($R_{8,1} = \dfrac{1024}{8^2+8+1} = 14.027$)

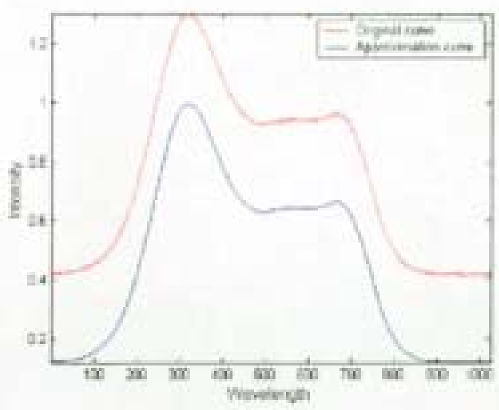
5.2.3 Single initial values using 8 segments for UV in c



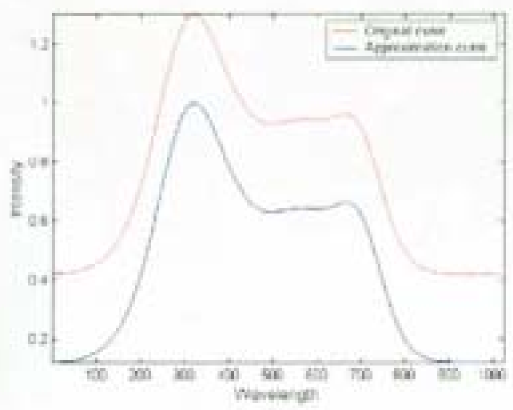
UV in c, 8 segments (Noise free)



UV in c, 8 segments (Noise Level $\pm 1 \times 10^{-3}$)



UV in c, 8 segments (Noise Level $\pm 5 \times 10^{-3}$)

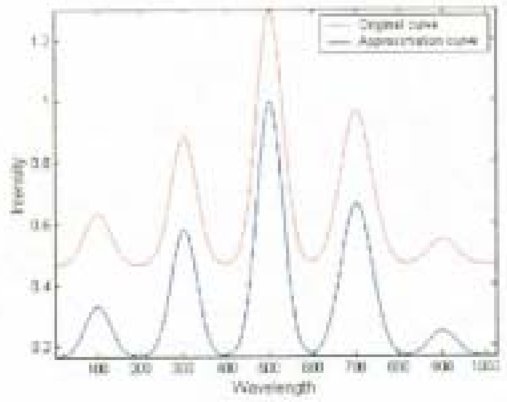


UV in c, 8 segments (Noise Level $\pm 1 \times 10^{-2}$)

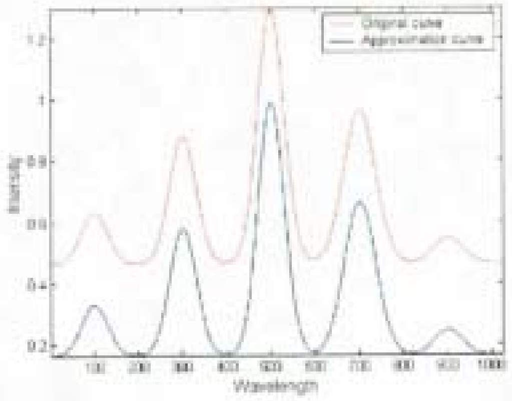
Chemicals UV in a	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
Noise free	5.8094e-005	8.2003e-006	1.4500e-005	0.0066	0.0009	0.0016
$\pm 1 \times 10^{-3}$	7.8578e-004	2.5667e-004	3.0680e-004	0.0892	0.0291	0.0348
$\pm 5 \times 10^{-3}$	4.6245e-003	1.3098e-003	1.5527e-003	0.5249	0.1487	0.1763
$\pm 1 \times 10^{-2}$	9.8315e-003	2.7471e-003	3.3070e-003	1.1160	0.3118	0.3754

Table 5.10: Errors of the UVinc with different noise when 8 segments and learning is used. ($R_{s,1} = \dfrac{1024}{8^2+8+1} = 14.027$)

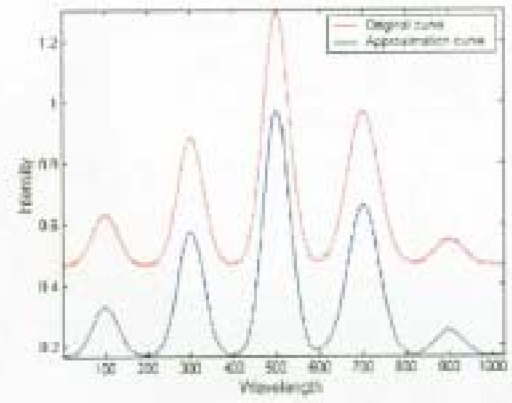
5.2.4 Single initial values using 8 segments for UV in d



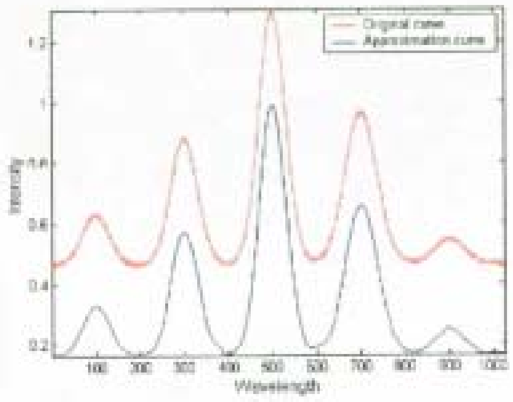
UV in d, 8 segments (Noise free)



UV in d, 8 segments (Noise Level $\pm 1 \times 10^{-3}$)



UV in d, 8 segments (Noise Level $\pm 5 \times 10^{-3}$)



UV in d, 8 segments (Noise Level $\pm 1 \times 10^{-2}$)

Chemicals UV in a	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
Noise free	1.5034e-003	3.7903e-004	5.0780e-004	0.1804	0.0455	0.0609
$\pm 1 \times 10^{-3}$	4.8036e-003	6.5174e-004	9.5790e-004	0.5765	0.0782	0.1150
$\pm 5 \times 10^{-3}$	7.5953e-003	1.8016e-003	2.2631e-003	0.9115	0.2162	0.2716
$\pm 1 \times 10^{-2}$	1.0729e-002	2.9232e-003	3.5660e-003	1.2875	0.3508	0.4279

Table 5.11: Errors of the UVind with different noise when 8 segments and learning is used. ($R_{8,1} = \dfrac{1024}{8^2+8+1} = 14.027$)

5.3 Results - UV Spectra (without/with Learning algorithm)

The following diagrams show the result of original data sets and approximated data sets either with using learning algorithm ("L") or without using learning algorithm, both of which were based on the same network size of 8. Normalization were used as the pre-processing techniques. The flow chart of the program for those results of 5.3 is shown in Figure 5.3 and 5.4.

In order to compare the data more easily, the extra value was added to the data. Moreover the curves were printed in different colors.

The value of 0.3 was added to the *recovered curves* in relation to the results of *without using learning algorithm.* In the lef: hand side of the diagrams, the *upper* curves are the *covered curves* which were printed in blue, and the *lower* curves are the *original curves* which were printed in red.

As regards the results of *using learning algorithm*, the value of 0.3 was added to the original data. In the right hand side of the diagrams, the *upper* curves are the *original curves* which were printed in red, and the *lower* curves are the *recovered curves* which were printed in blue.

The corresponding maximum absolute error(l_∞ error), mean absolute error(l_1 error), root mean square error ("RMSE") (l_2 error) and their relative error percentages(%) are calculated and tabulated in the tables shown below.

In addition, the compression ratio:

$$R_{i,j} = \frac{\textit{total number of original data}}{\textit{number of data that required for regenerating the approximate data}}$$

where i - number of segments;

j - number of initial point.

have also been calculated.

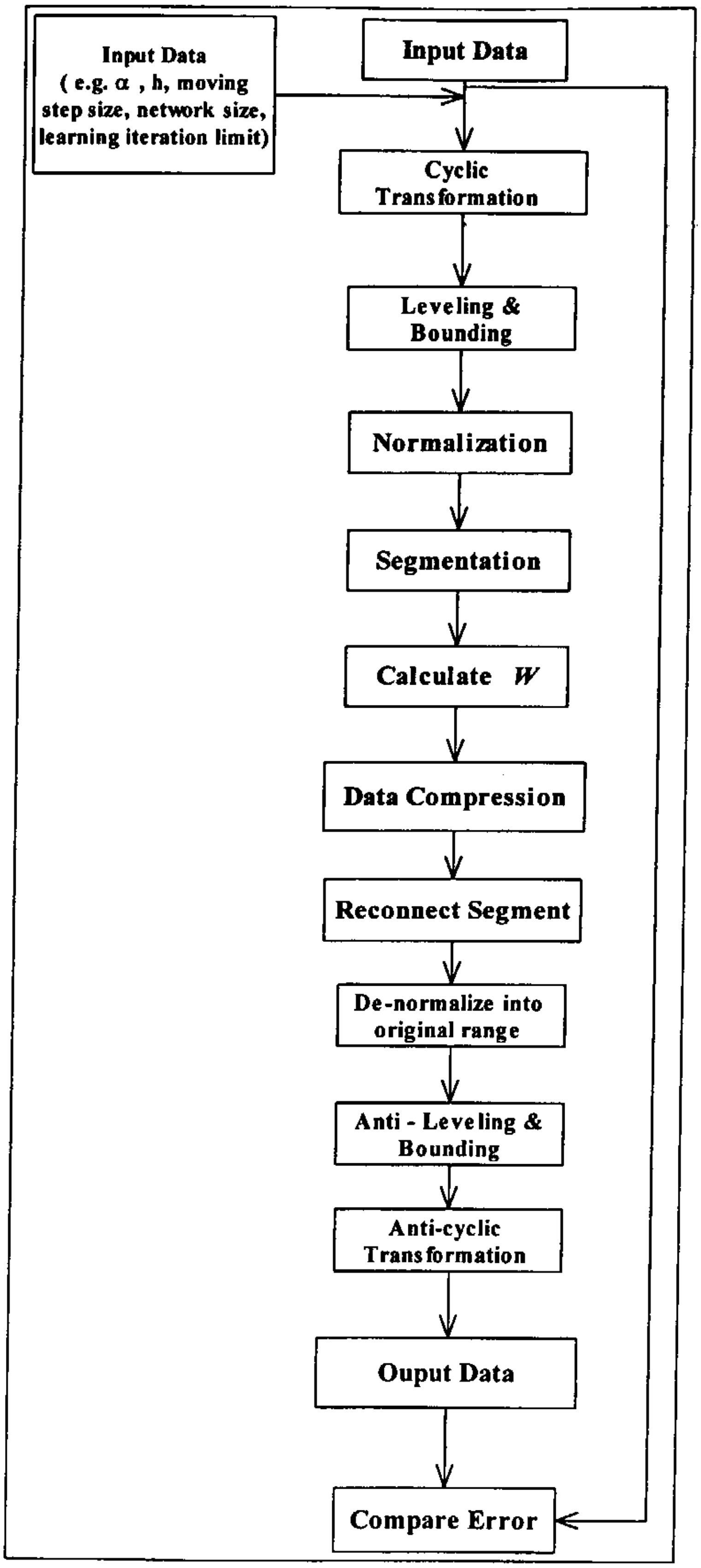

Input Data (e.g. α , h, moving step size, network size, learning iteration limit)
Input Data
Cyclic Transformation
Leveling & Bounding
Normalization
Segmentation
Calculate W
Data Compression
Reconnect Segment
De-normalize into original range
Anti - Leveling & Bounding
Anti-cyclic Transformation
Ouput Data
Compare Error

Figure 5-3: Flow chart for without Learning.

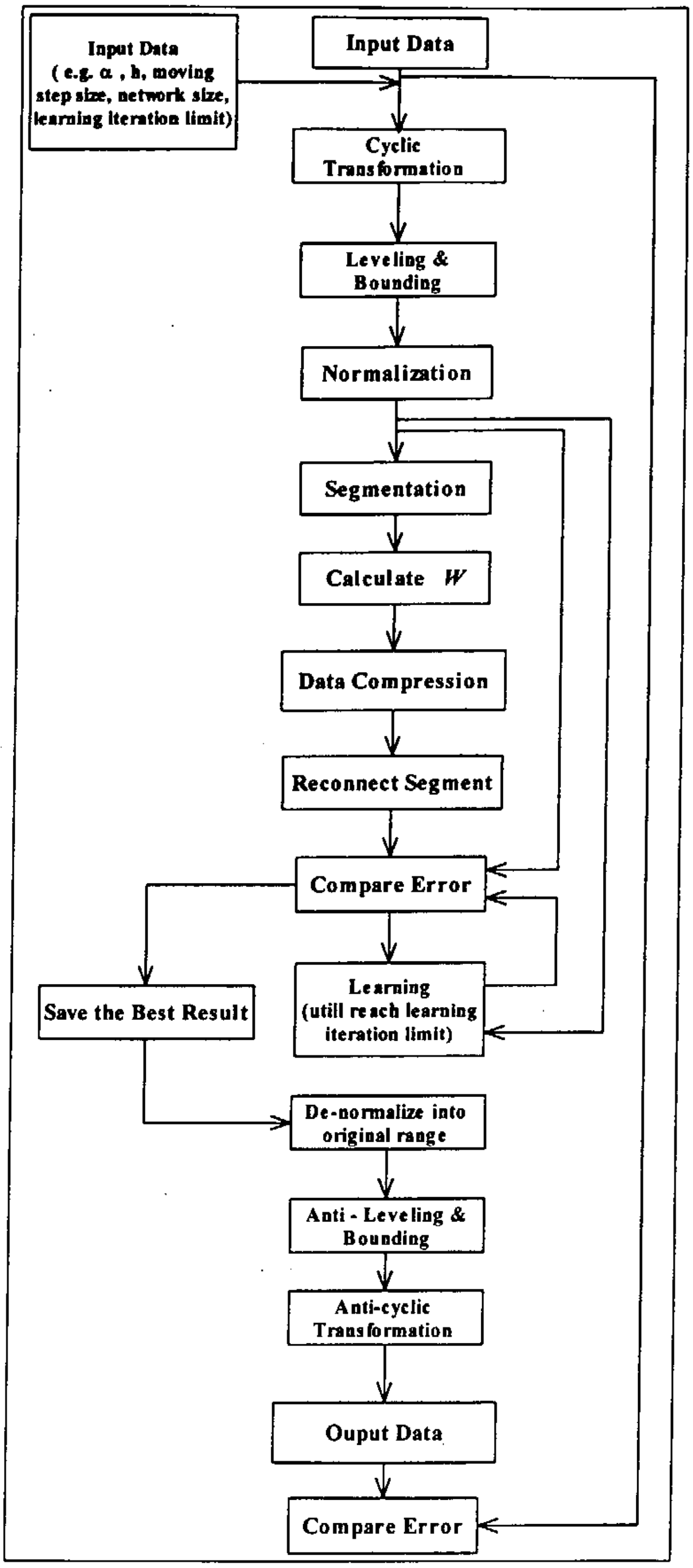
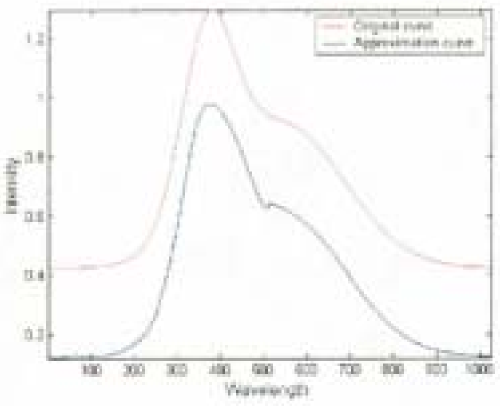
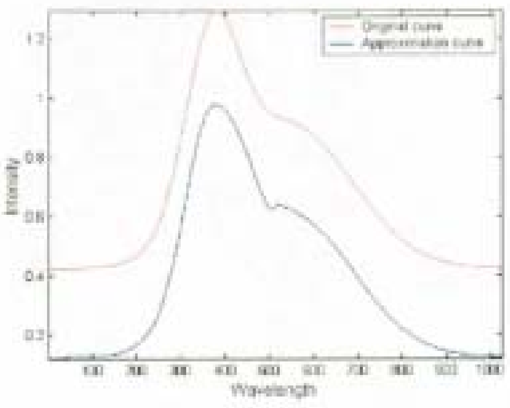

Input Data
(e.g. α , h, moving step size, network size, learning iteration limit)
Input Data
Cyclic Transformation
Leveling & Bounding
Normalization
Segmentation
Calculate W
Data Compression
Reconnect Segment
Compare Error
Save the Best Result
Learning (utill reach learning iteration limit)
De-normalize into original range
Anti - Leveling & Bounding
Anti-cyclic Transformation
Ouput Data
Compare Error

Figure 5-4: Flow chart for using Learning.

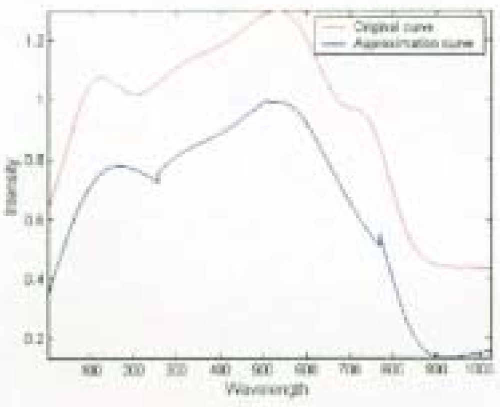
5.3.1 Single initial values using 4 segments for UV Spectra



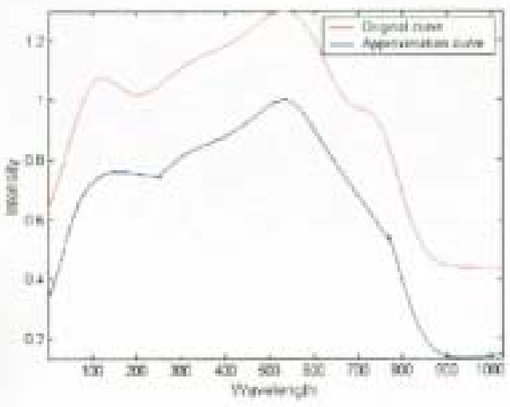
UV in a, without learning



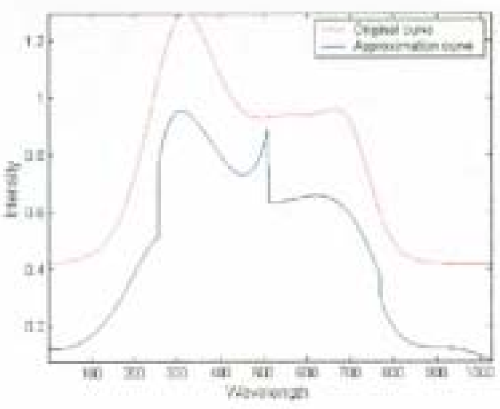
UV in a, with learning



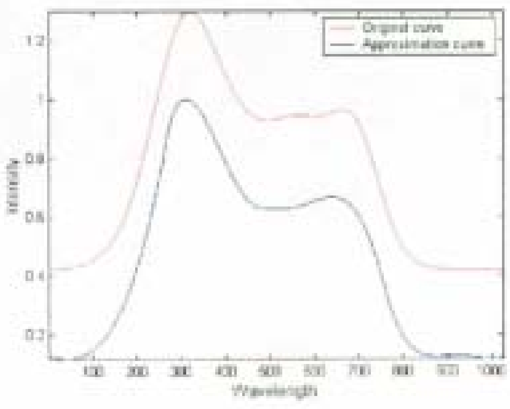
UV in b, without learning



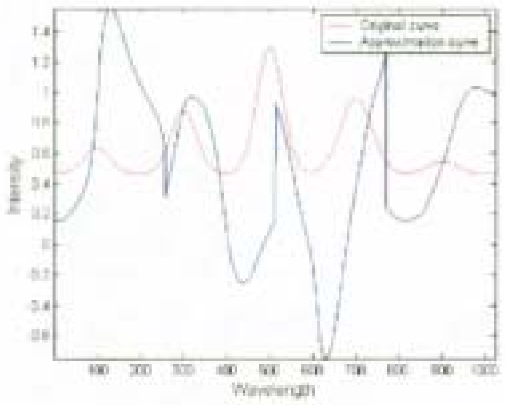
UV in b, with learning



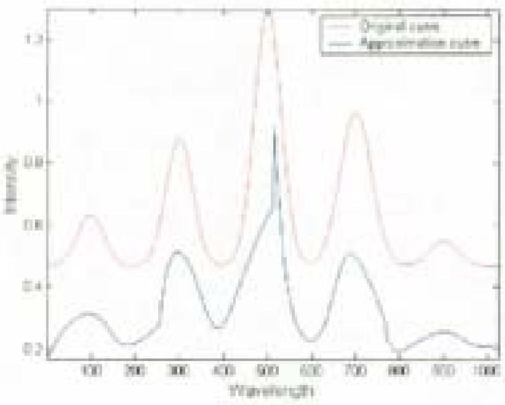
UV in c, without learning



UV in c, with learning



UV in d, without learning

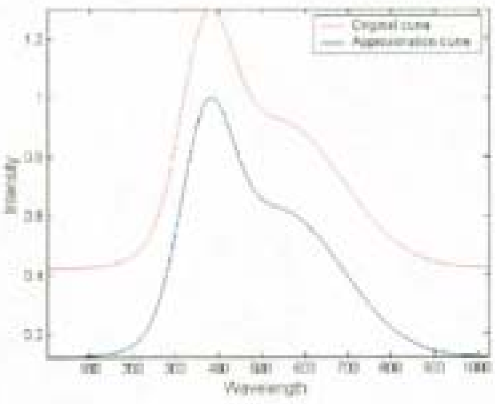


UV in d, with learning

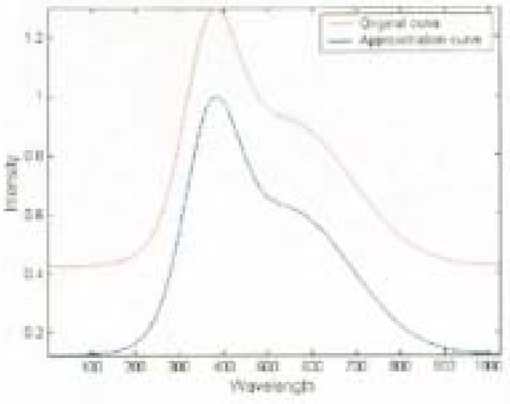
Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
No learning						
UV in a	1.66193e-02	2.31737e-03	4.27153e-03	1.89674	0.26448	0.48750
UV in b	3.93483e-02	7.87609e-03	1.19457e-02	4.49076	0.89889	1.36334
UV in c	1.30599e-01	1.47317e-02	2.75303e-02	14.90502	1.68131	3.14199
UV in d	6.46434e-01	2.38930e-01	3.03869e-01	73.77657	27.26867	34.68013
Learning						
UV in a	1.44072e-02	2.08674e-03	3.88448e-03	1.64427	0.23816	0.44333
UV in b	2.43794e-02	5.00210e-03	8.04164e-03	2.78238	0.57088	0.91778
UV in c	1.99596e-02	3.66692e-03	5.04808e-03	2.27796	0.41850	0.57613

Table 5.12: Errors of the UV spectra when 4 segments, with or without learning. $(R_{4,1} = \dfrac{1024}{4^2+4+1} = 48.7619)$

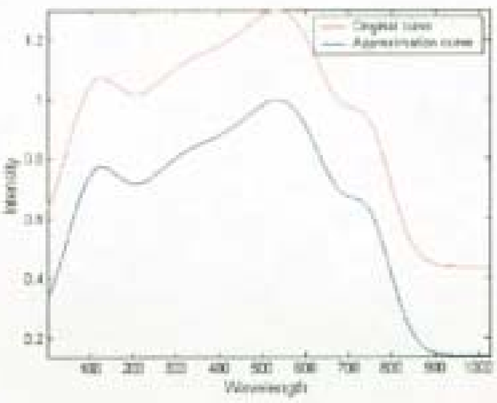
5.3.2 Single initial values using 8 segments for UV Spectra



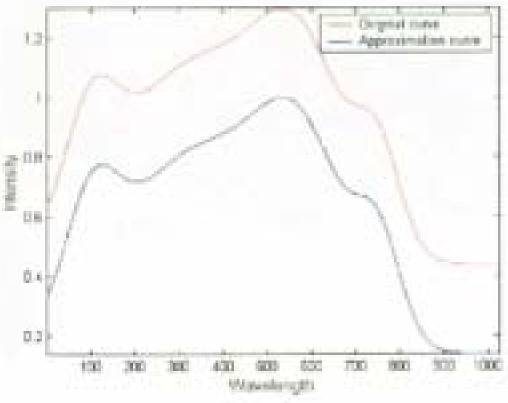
UV in a, without learning



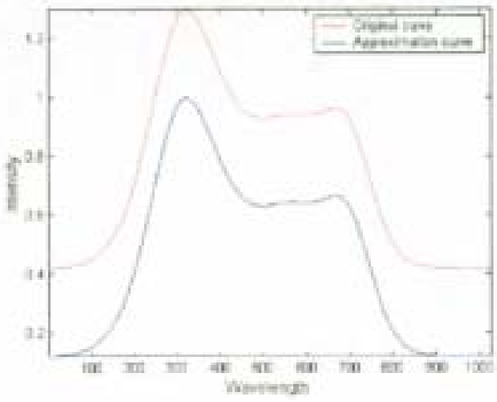
UV in a, with learning



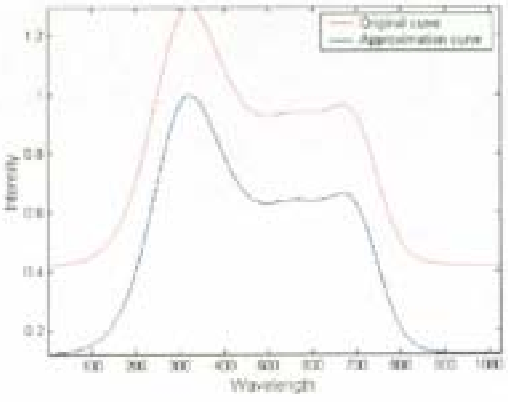
UV in b, without learning



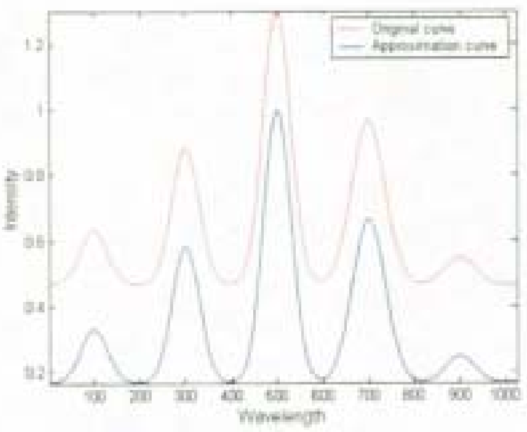
UV in b, with learning



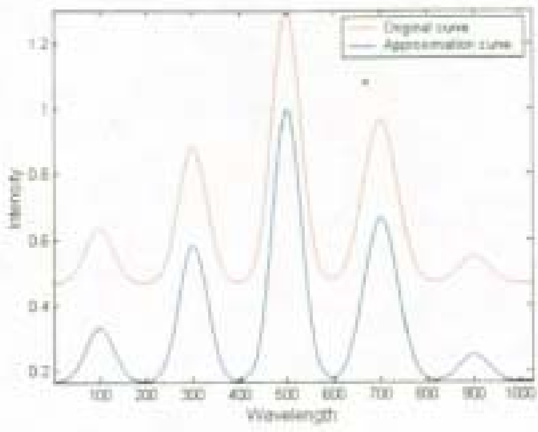
UV in c, without learning



UV in c, with learning



UV in d, without learning



UV in d, with learning

Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
No learning						
UV in a	2.98631e-05	3.53718e-06	7.29423e-06	0.00341	0.00040	0.00083
UV in b	9.44537e-05	1.05136e-05	1.89217e-05	0.01078	0.00120	0.00216
UV in c	2.36235e-03	8.89291e-05	2.53877e-04	0.26961	0.01015	0.02897
UV in d	1.53209e-03	4.35330e-04	5.82788e-04	0.17486	0.04968	0.06651
Learning						
UV in a	2.90151e-05	3.46345e-06	7.21052e-06	0.00331	0.00040	0.00082
UV in b	1.79827e-05	3.99296e-06	5.58392e-06	0.00205	0.00046	0.00064
UV in c	4.89939e-04	4.88247e-05	1.07821e-04	0.05592	0.00557	0.01231
UV in d	1.50344e-03	3.79030e-04	5.07795e-04	0.17158	0.04326	0.05795

Table 5.13: Errors of the UV spectra when 8 segments, with or without learning. ($R_{8,1} = \dfrac{1024}{8^2+8+1} = 14.027$)

5.4 Comments on the results:

The following comments are obtained from the results mentioned in Section 5.1 to 5.3:-

1. By comparing the errors obtained in Section 5.1 with different segments, all UV in a, b, c or d showed improvement while the number of segments were increased. Besides, UV in d showed obviously worse result in all segments.

2. By comparing the errors obtained in Section 5.2 with some noise in UV in a, b, c or d, it seems that although the shape is not change severely, all the approximation errors are obviously drop down if different levels of noise were added.

3. By comparing the errors obtained in Section 5.3.1 and Section 5.3.2 with using and not using learning algorithm, it showed improvement obviously in almost all chemicals as our learning algorithm was added.

Chapter 6

Results from Infra Red Spectra

In chapter [5], the results from UV spectra have been shown. The UV spectra are relatively simple and short in length and so compression of those signals is quite successful.

In this chapter, a second set of data from Infra Red specti a will be presented. The signal lengths of IR spectra are longer than those of UV spectra and the waveforms are much more complicated than those of UV. So compressing IR spectra is not so easy and simple as those compressing UV spectra. Besides, the results from two different segmentation will also be presented. In the first half of this chapter, the results from 8 segments are presented. In the second half of the chapter, the results from 9 segments are presented. All the results are presented both in graphical and numerical formats, so that comparison can be made more easier.

6.1 IR Spectra with 8 segments and learning algorithm

The following diagrams shows the results which are based on the network size of 8. The adopted pre-processing techniques, cyclic transformation, leveling, bounding, normalization and learning algorithm ("L"), were being used. The flow chart of the program for those results of 6.1 is shown in Figure 6.1.

In order to compare the results more easily, different colors were chosen to represent different data sets. In the diagrams, the *original curves* are in *red* and the *recovered curves* of the RNN are in *blue.*

The corresponding maximum absolute error(l_∞ error), mean absolute error(l_1 error), root mean square error ("RMSE") (l_2 error) and their relative error percentages(%) are calculated and tabulated in the tables shown below.

In addition, the compression ratio:

$$R_{i,j} = \frac{\textit{total number of original data}}{\textit{number of data that required for regenerating the approximate data}}$$

where i - number of segments;

j - number of initial point.

have also been calculated.

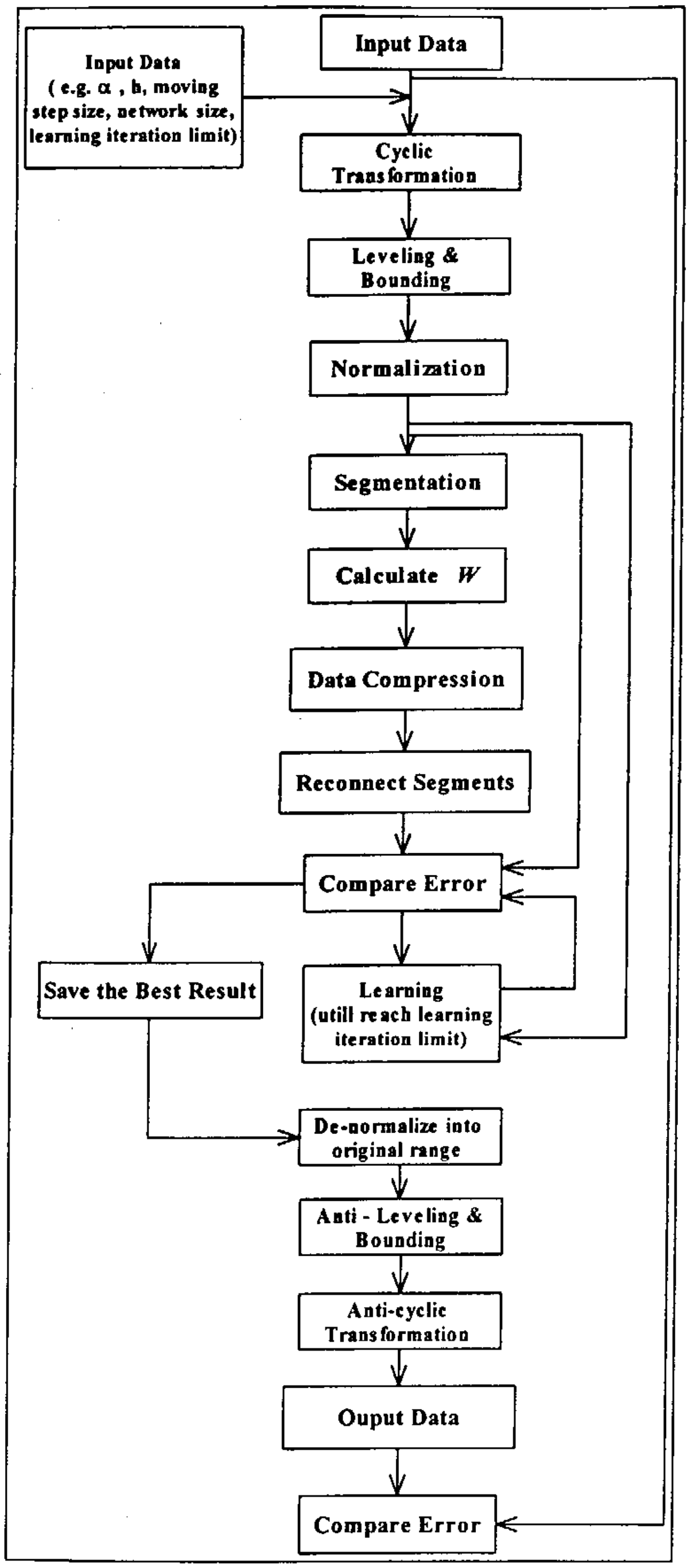
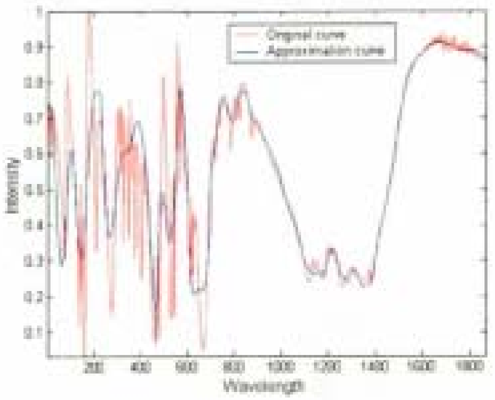
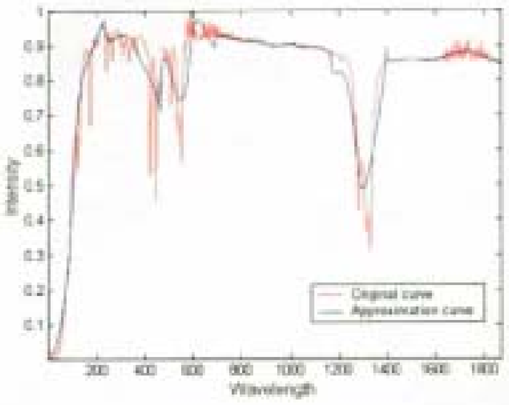

Input Data
Input Data (e.g. α , h, moving step size, network size, learning iteration limit)
Cyclic Transformation
Leveling & Bounding
Normalization
Segmentation
Calculate W
Data Compression
Reconnect Segments
Compare Error
Save the Best Result
Learning (utill reach learning iteration limit)
De-normalize into original range
Anti - Leveling & Bounding
Anti-cyclic Transformation
Ouput Data
Compare Error

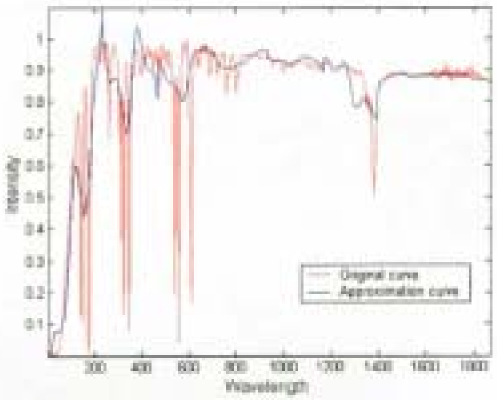
Figure 6-1: Flow chart for the results of Section 6.1.



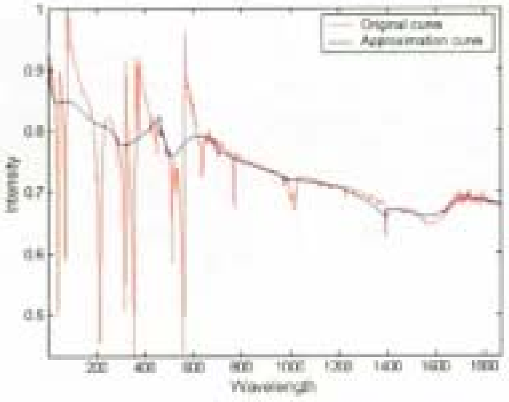
Benzoic Acid, 8 segments (L)



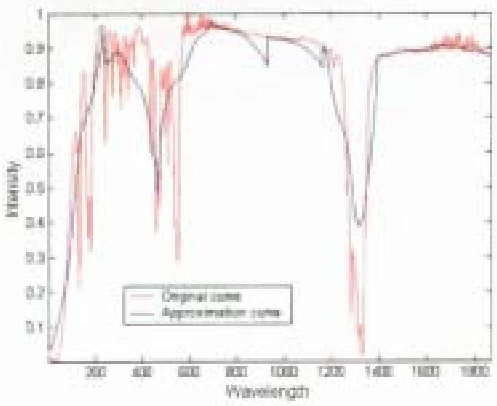
1-bromobutane, 8 segment (L)



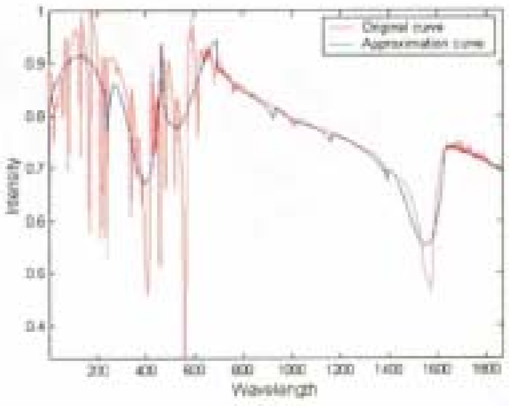
Bromobenzene, 8 segments (L)



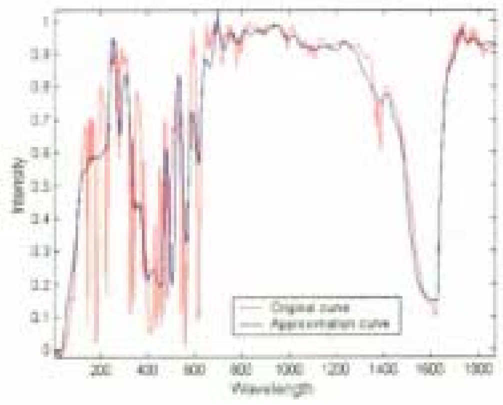
1,4-dichlorobenzene, 8 segments (L)



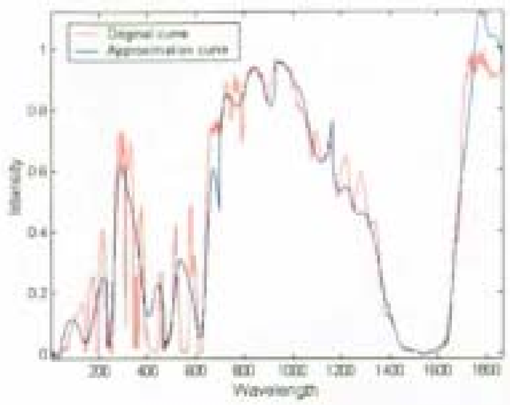
1-chlorobutane, 8 segments (L)



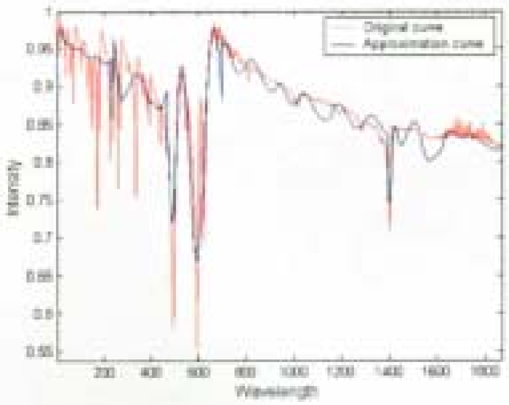
2,4,-dichlorophenol, 8 segments (L)



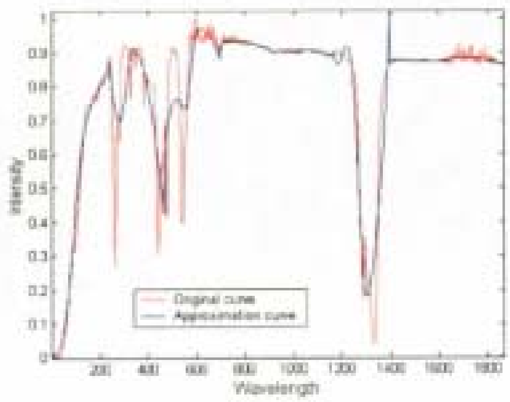
2-chlorophenol, 8 segments (L)



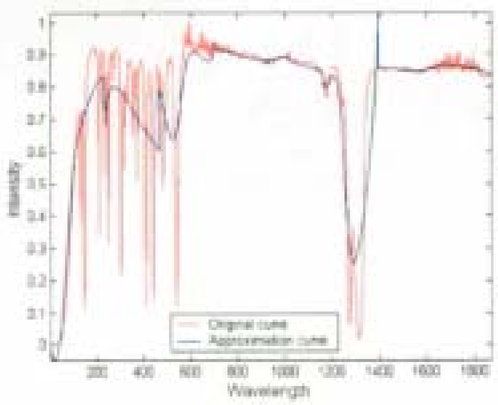
3-chlorophenol, 8 segments (L)



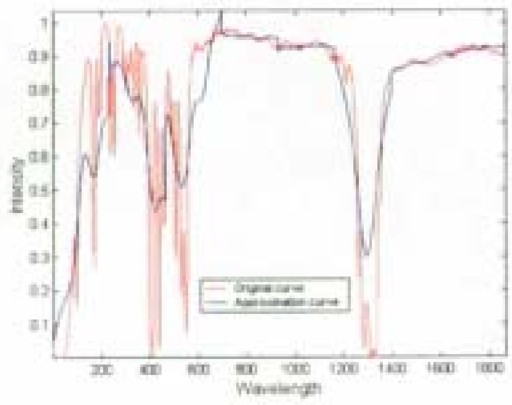
1-chloro-2,4-dinitrobenzene, 8 segments (L)



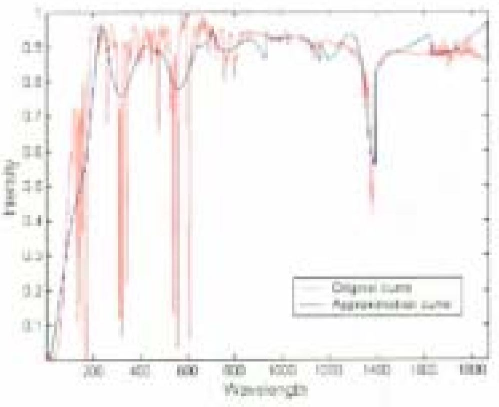
Cyclopentyl Chloride, 8 segments (L)



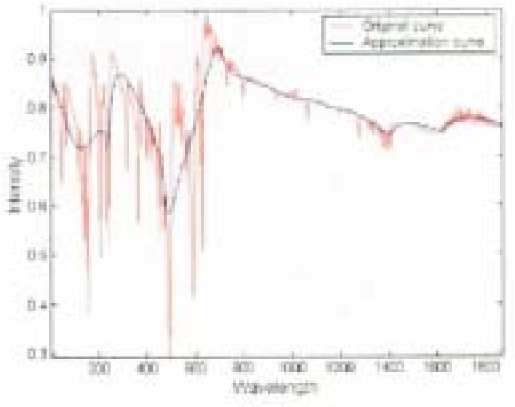
Cyclohexyl Bromide, 8 segments (L)



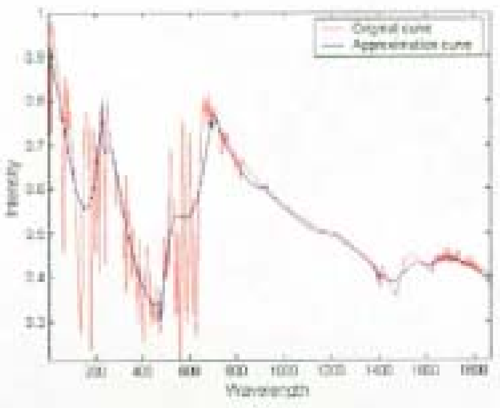
1-iodobutane, 8 segments (L)



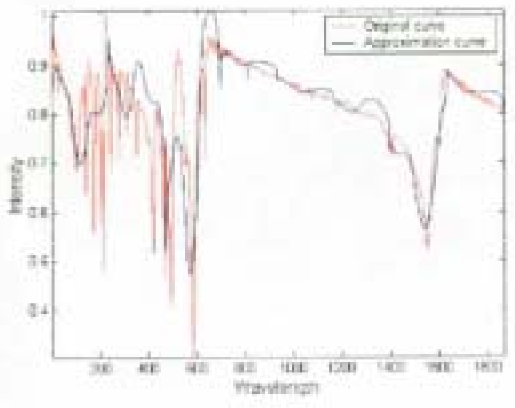
Iodobenzene, 8 segments (L)



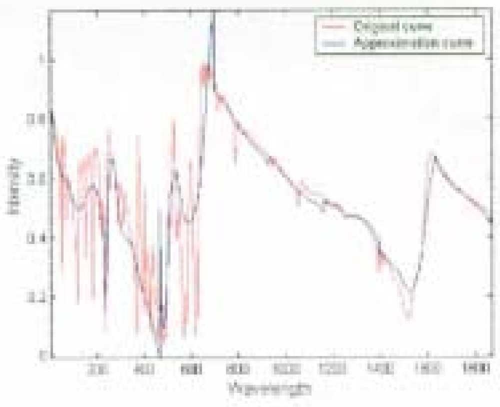
4-nitrobenzyl Chloride, 8 segments (L)



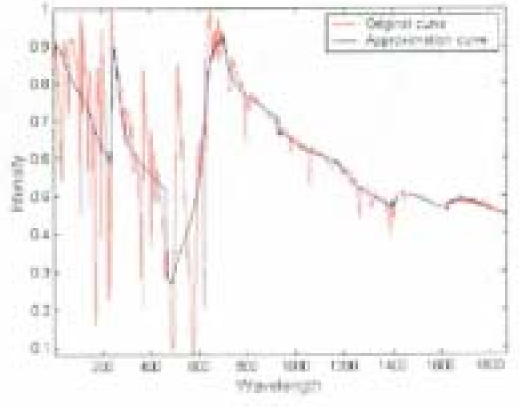
2-nitrophenol, 8 segments (L)



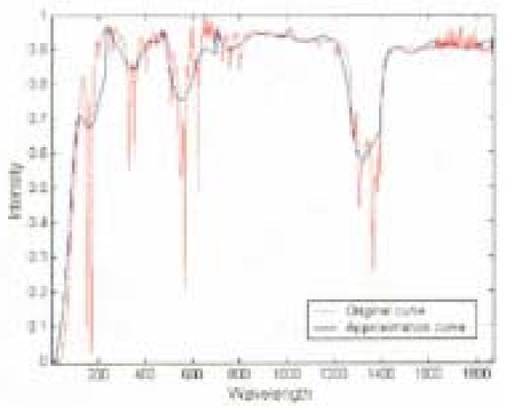
3-nitrophenol, 8 segments (L)



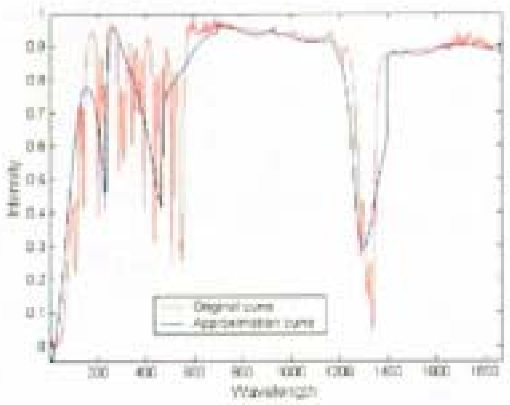
4-nitrophenol, 8 segments (L)



4-nitrotulene, 8 segments (L)



Toluene, 8 segments (L)



2-chlorobutane, 8 segments (L)

Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
benzacid	2.1714e-001	2.8649e-002	5.1316e-002	22.4859	2.9667	5.3140
br1butan	1.8578e-001	1.4487e-002	2.6332e-002	18.6320	1.4529	2.6409
brbenzen	3.8492e-001	2.3706e-002	5.1554e-002	38.4985	2.3710	5.1563
cl14benz	1.7939e-001	1.2480e-002	2.7374e-002	31.6357	2.2009	4.8274
cl1butan	1.9700e-001	2.3467e-002	4.0538e-002	19.7626	2.3542	4.0667
cl24phen	2.0712e-001	1.5840e-002	3.0175e-002	31.2474	2.3897	4.5524
cl2pheno	2.7874e-001	3.2858e-002	5.8966e-002	27.5462	3.2472	5.8273
cl3pheno	2.8566e-001	3.3528e-002	5.1636e-002	28.5033	3.3454	5.1523
cln24ben	1.3530e-001	9.9334e-003	1.8583e-002	29.2642	2.1485	4.0193
cyclo5cl	2.1426e-001	1.8017e-002	3.5943e-002	21.4174	1.8010	3.5929
cyclo6br	2.9897e-001	2.8553e-002	5.2712e-002	29.8671	2.8524	5.2659
i1butane	2.7675e-001	3.5667e-002	5.9202e-002	26.7500	3.5667	5.9202
ibenzene	4.1222e-001	2.7406e-002	5.2417e-002	41.2831	2.7447	5.2495
n4benzcl	1.7760e-001	1.5543e-002	3.1413e-002	25.1110	2.1976	4.4415
no2pheno	1.9050e-001	1.9387e-002	3.6315e-002	24.2185	2.4647	4.6168
no3pheno	1.5472e-001	1.8218e-002	2.9748e-002	22.2491	2.6198	4.2778
no4pheno	2.4980e-001	2.6423e-002	4.4162e-002	26.0588	2.7564	4.6069
no4tulen	2.5541e-001	2.4440e-002	4.8617e-002	27.7460	2.6550	5.2814
toluene	3.4815e-001	2.0081e-002	3.8320e-002	34.6970	2.0013	3.8190
cl2butan	2.2409e-001	2.9350e-002	4.9051e-002	22.4368	2.9386	4.9112

Table 6.1: Errors of the IR spectra when 8 segments and learning is used. ($R_{8,1} = \dfrac{1868}{8^2+8+1} = 25.5890$)

6.2 IR Spectra with 9 segments and learning algorithm

The following diagrams show the results which are based on the network size of 9. The adopted pre-processing techniques, cyclic transformation, leveling, bounding, normalization and learning algorithm ("L"), were being used. The flow chart of the program for those results of 6.2 is shown in Figure 6.2.

In order to compare the results more easily, different colors were chosen to represent different data sets. In the diagrams, the *original curves* are in *red* and the *recovered curves* of the RNN are in *blue.*

The corresponding maximum absolute error(l_∞ error), mean absolute error(l_1 error), root mean square error ("RMSE") (l_2 error) and their relative error percentages(%) are calculated and tabulated in the tables shown below.

In addition, the compression ratio:

$$R_{i,j} = \frac{\textit{total number of original data}}{\textit{number of data that required for regenerating the original data}}$$

where i - number of segments;

j - number of initial point.

have also been calculated.

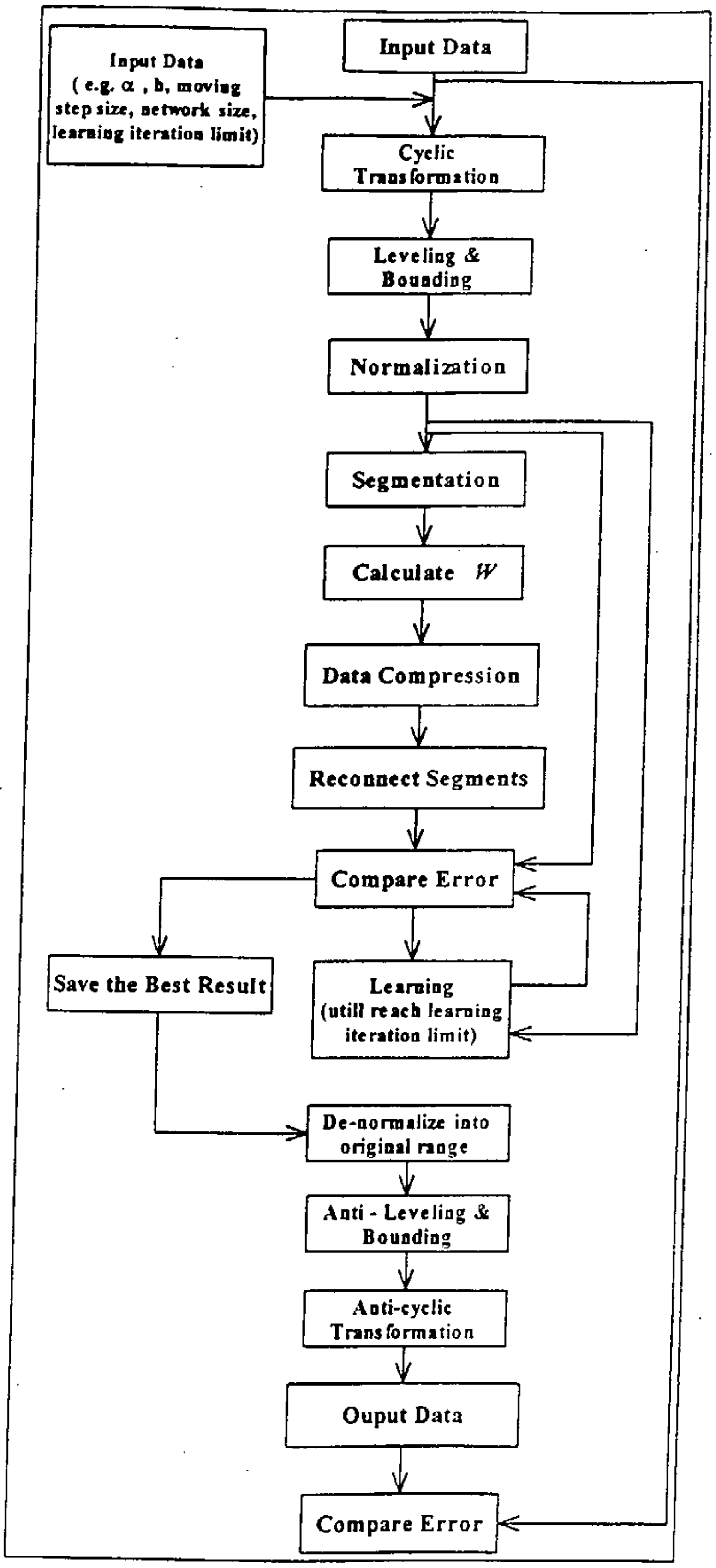
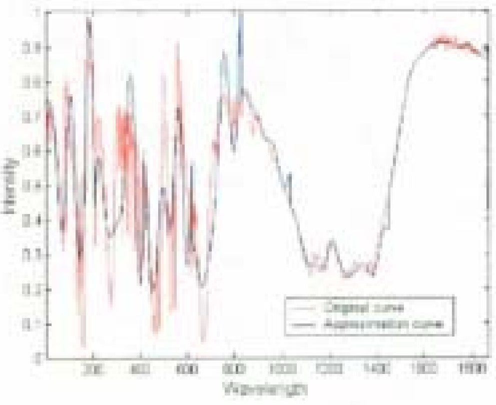
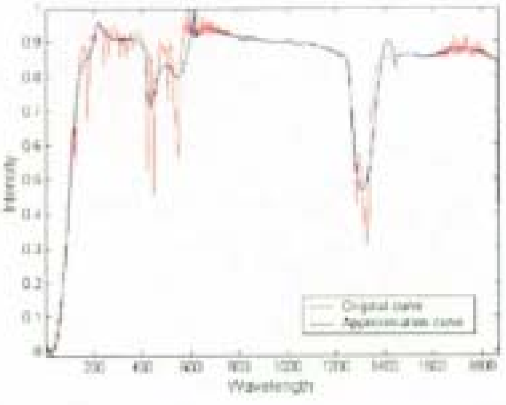

Input Data
(e.g. α , b, moving step size, network size, learning iteration limit)
Input Data
Cyclic Transformation
Leveling & Bounding
Normalization
Segmentation
Calculate W
Data Compression
Reconnect Segments
Compare Error
Save the Best Result
Learning (utill reach learning iteration limit)
De-normalize into original range
Anti - Leveling & Bounding
Anti-cyclic Transformation
Ouput Data
Compare Error

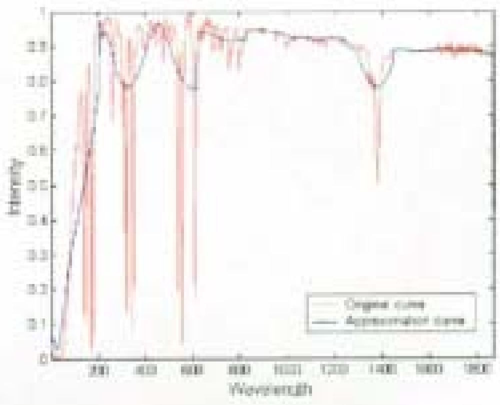
Figure 6-2: Flow chart for the results of Section 6.2.



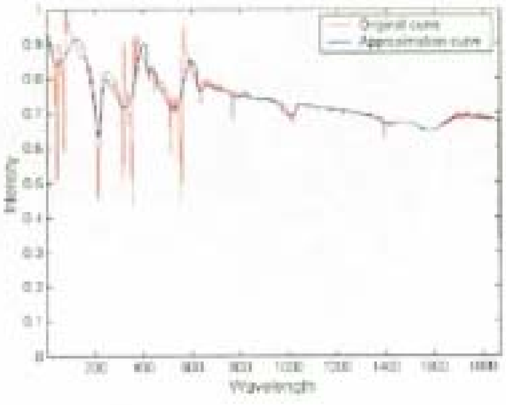
Benzoic Acid, 9 segments (L)



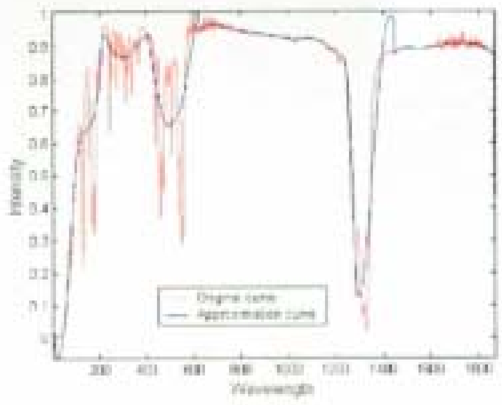
1-bromobutane, 9 segments (L)



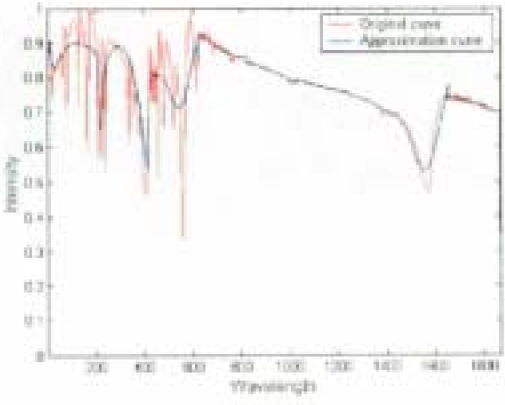
Bromobenzene, 9 segments (L)



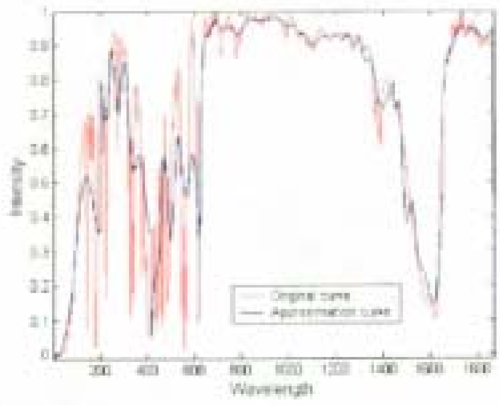
1,4-dichlorobenzene, 9 segments (L)



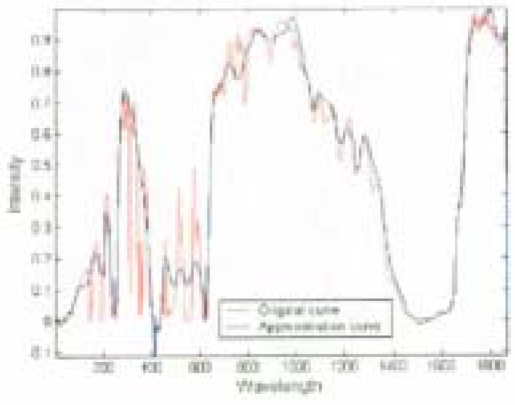
1-chlorobutane, 9 segments (L)



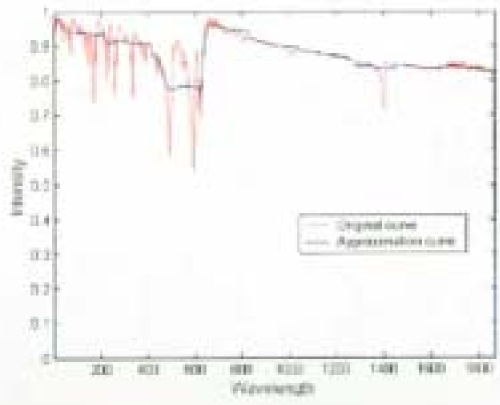
2,4,-dichlorophenol, 9 segments (L)



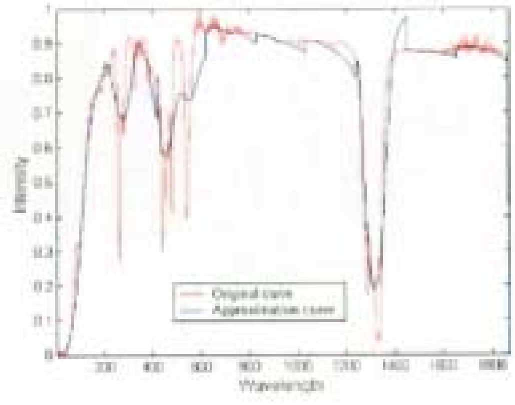
2-chlorophenol, 9 segments (L)



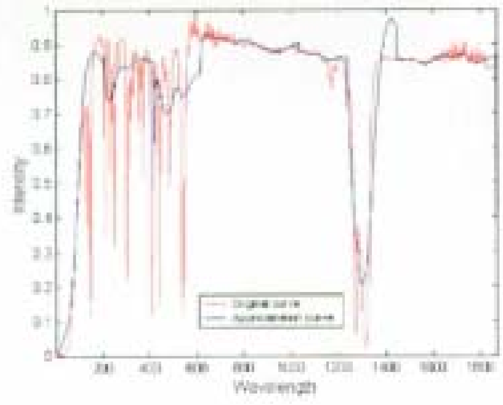
3-chlorophenol, 9 segments (L)



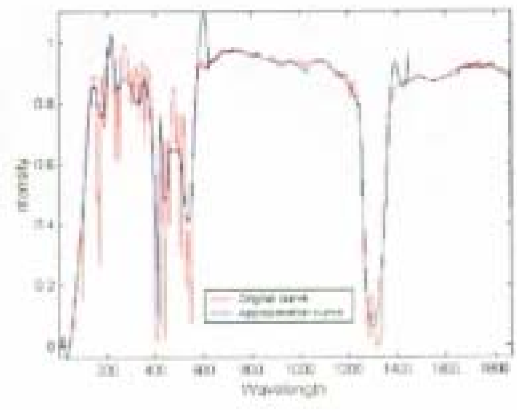
1-chloro-2,4-dinitrobenzene, 9 segments (L)



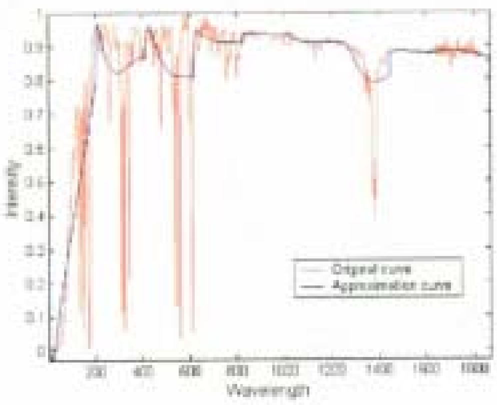
Cyclopentyl Chloride, 9 segments (L)



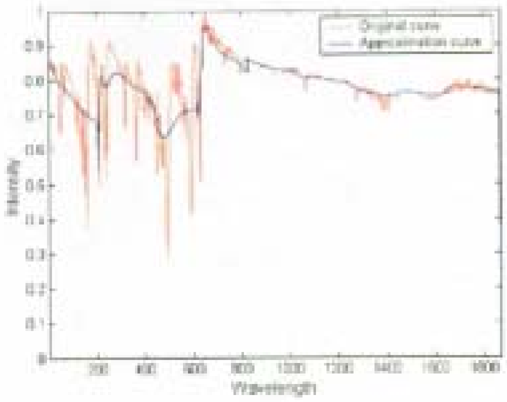
Cyclohexyl Bromide, 9 segments (L)



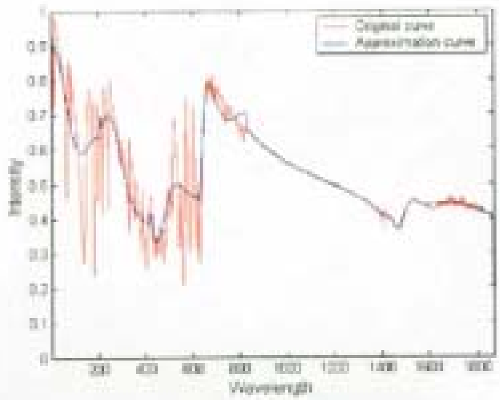
1-iodobutane, 9 segments (L)



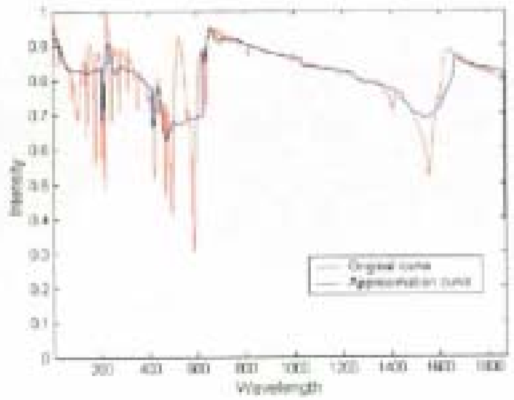
Iodobenzene, 9 segments (L)



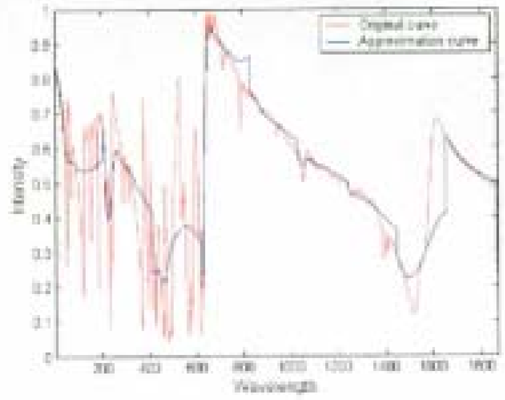
4-nitrobenzyl Chloride, 9 segments (L)



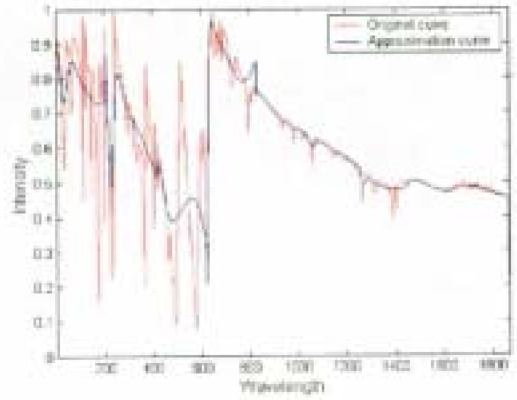
2-nitrophenol, 9 segments (L)



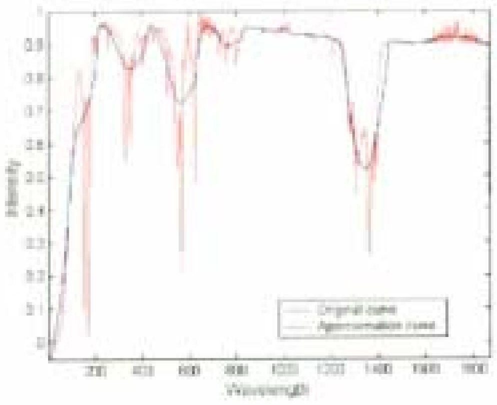
3-nitrophenol, 9 segments (L)



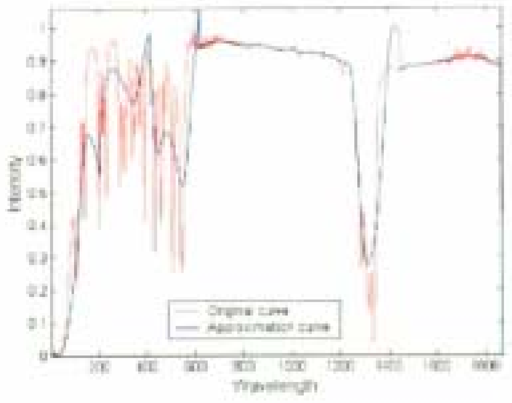
4-nitrophenol, 9 segments (L)



4-nitrotulene, 9 segments (L)



Toluene, 9 segments (L)



2-chlorobutane, 9 segments (L)

Chemicals	l_∞ error	l_1 error	l_2 error	Relative l_∞ error %	Relative l_1 error %	Relative l_2 error %
benzacid	2.0656e-001	2.6628e-002	4.4583e-002	21.3903	2.7575	4.6167
br1butan	1.4607e-001	1.1597e-002	2.2042e-002	14.6495	1.16307	2.2106
brbenzen	3.8029e-001	2.6177e-002	5.4578e-002	38.0355	2.6181	5.4587
cl14benz	1.5778e-001	9.3488e-003	2.1343e-002	27.8247	1.6487	3.7639
cl1butan	2.2018e-001	2.1022e-002	4.0498e-002	22.0880	2.1089	4.0627
cl24phen	1.2721e-001	1.2007e-002	2.4912e-002	19.1917	1.8114	3.7584
cl2pheno	2.6179e-001	2.9236e-002	5.1026e-002	25.8711	2.8892	5.0426
cl3pheno	4.0996e-001	4.0347e-002	6.4926e-002	40.9060	4.0258	6.4783
cln24ben	1.2224e-001	8.4266e-003	1.7313e-002	26.4394	1.8226	3.7446
cyclo5cl	1.9821e-001	1.9330e-002	3.4227e-002	19.8131	1.9322	3.4213
cyclo6br	3.4398e-001	2.6418e-002	5.1739e-002	34.3636	2.6392	5.1687
i1butane	2.2157e-001	2.0930e-002	3.9206e-002	22.1570	2.0930	3.9206
ibenzene	3.8803e-001	2.6235e-002	5.6083e-002	38.8605	26274	5.6166
n4benzcl	1.7426e-001	1.5322e-002	3.0577e-002	24.6387	2.1664	4.3233
no2pheno	1.9855e-001	1.6550e-002	3.2371e-002	25.2419	2.1040	4.1154
no3pheno	1.9185e-001	1.7133e-002	3.2765e-002	27.5884	2.4638	4.7117
no4pheno	2.6424e-001	2.6521e-002	4.4163e-002	27.5652	2.7666	4.6070
no4tulen	2.8489e-001	2.2438e-002	4.4543e-002	30.9485	2.4375	4.8388
toluene	3.4826e-001	1.9358e-002	3.8803e-002	34.7080	1.9292	3.8672
cl2butan	2.6507e-001	3.5978e-002	6.1977e-002	26.5399	3.6023	6.2054

Table 6.2: Errors of the IR spectra when 9 segments and learning is used. $(R_{9,1} = \dfrac{1863+5}{9^2+9+1+5} = 19.4583)$

6.3 Comments on the results:

The followings are comments obtained in Section 6.1 and 6.2:-

1. By comparing the relative l_2 errors percentage obtained in Section 6.1 and 6.2 in 8 and 9 segments, it found that only 3 chemicals do not show any improvement, which are *cl3pheno*, *no3pheno* and *cl2butan*. The others are showed either improved or similar results. Nevertheless, all the l_2 errors are bounded by 6.5×10^{-2} while the relative l_2 errors percentages are bounded by 6.5%.
2. The compression ratios for 8-segments and 9-segments methods are 25.5890 and 19.4583 respectively.

Chapter 7

Conclusions

Data compression is still a very hot research area. Mature compression algorithms and their implementation softwares such as Fourier transforms and Wavelet transforms dominate the market. Neural network is widely used for pattern recognition but seldomly considered for data compression.

Since the title of this project is "A Study on Data Compression by Dynamical System Approach", the most important task of this project is to study the capability of using the dynamical system approach (recurrent neural network) in achieving the goal of data compression. From the results presented in Chapter [5] and Chapter [6], this fundamental goal has been achieved.

The second task of this project is to concentrate on how good the neural network techniques in carrying out the task of data compression. In the following sections, the results from Ultra Violet spectra and that from Infra Red spectra will be discussed briefly.

7.1 Ultra Violet Spectra

For Ultra Violet spectra, almost all the techniques mentioned in Chapter [3] and Chapter [4] were applied. For simplicity, single system method was used while double system method and triple system method were skipped.

Higher system methods have not been used because they will make the programming more complicated and will not keep the method to its simplest format. Another important point is that hyperbolic tangent was used as the activation function.

7.1.1 4, 5, 6, 7, 8, 9 and 10-segments Single RNN System

From the results obtained in Table 5.1 to Table 5.7 for Ultra Violet spectra which showed the compression ratios ranging from 48.7619 to 8.9043 times, the relative l_2 error percenta es are ranging from 0.4433% to 0.00005% for UV in a, 0.9305% to 0.00006% for UV in b, 0.5730% to 0.00002% for UV in c and 5.2470% to 0.00090% for UV in d.

The compression ratios are quite good. Moreover, it is obvious that increasing the size of network (segment number) will improve the results further, while the compression ratios will decrease at the same time. Nevertheless, it seems to be valuable still to increase the network size to achieve the appropriate relative l_2 error percentage.

7.1.2 8-segment Single RNN System for UV spectra with some noise

In the 8-segment tests, the errors in Table 5.8 to Table 5.11 for all UV in a, b, c and d with 3 different levels of noise ($\pm 1 \times 10^{-3}$, $\pm 5 \times 10^{-3}$ and $\pm 1 \times 10^{-2}$) are increased as the noise level applied is increased. However, the errors for the cases are still acceptable as the relative l_2 error percentages were still not too high.

7.1.3 With or without Learning Algorithm

In order to detect the usage of learning algorithm, normalization were the only preprecessing techniques to be used. From the results in Table 5.12 and Table 5.13, the chemicals UV in a, b, c and d show the improvement while learning algorithm are applied. The improvement is more significant when 4 neurons is used instead of 8 neurons is used. It can be due to chemical fluctuation properties.

7.2 Infra Red Spectra

From the results in Table 6.1 and 6.2 for Infra Red Spectra which show the compression ratios ranging from 25.5890 to 19.4583, the best result comes from element 1-bromobutane (*br1butan*) which has an relative l_2 error percentage bounded by 2.6409% for 8-segment method and 2.2106% for 9-segment method. The worst cases come from 1-iodobenzene (*i1butane*) for 8-segment method which has an relative l_2 error percentage bounded by 5.9202%, and 2-chlorobutane (*cl2butan*) for 9-segment method which has an relative l_2 error percentage bounded by 6.2054%. Roughly speaking, an increase of the number of segments can improve the relative l_2 error percentage.

7.3 Future Improvements

Although it is proved in this project that Recurrent Neural Network is capable of carrying out data compression tasks, there are a lot of rooms for improvement.

The examples of improvement are:

- A *systematic* and *simple* way should be developed in classifying the data for compression. A suggestion is to count the turning points within the data set in order to decide the number of segments used.
- The next step is to decide the segmentation method.
- More pre-processing methods should be tried to do the mix and match for all combinations.
- Development of further analysis such as convergence and error bound of the learning algorithm.

Although multiple RNN systems are proved to be much more superior than the single RNN system method, it involves much more complicated steps in deciding the method of segmentation and the sizes of network to be used. As a result, the improvement may not be significant to compensate for the extra effort. Therefore, it is suggested to keep the compression work to be carried out by the single system method and not to extend to higher system unless necessary.

Chapter 8

References

1. Adachi M. and Aihara K., "*Associative Dynamics in a Chaotic Neural Networks*", Vol. 10, No. 1, pp. 83-98, 1997.

2. Brigham E.O., 1940- "*The fast Fourier transform*", Publisher: Englewood Cliffs, NJ, Prentice-Hall [1974].

3. C.S. Burrus, Dept. of Electrical and Computer Engineering, Rice University, Houston, September 1997, "*Notes on the FFT*".

4. Cappellini V., Benelli G. & Del Re E., Lotti F., "*Data Compression Techniques*", Data Compression and Error Control Techniques with Applications, Publisher: London: Orlandor: Academic Press, 1985, pp. 33-82.

5. Chau F.T., Shih T.M., Gao J.B. and Chan C.K., "*Application of the Fast Wavelet Transform Method to Compress Ultraviolet-Visible Spectra*", Applied Spectroscopy, Vol. 50, No. 3, pp. 339-348, 1996.

6. Chau F.T., Tam K.Y., "*Application of the Fast Fourier Transform Method for*

compression of Spectral Data Obtained from a Photodiode Array Spectrophotometer", Computers & Chemistry, 18, pp. 13-20, 1994.

7. Chui C.K. "*Wavelets: a mathematical tool for signal processing*", Publisher: Philadelphia: Society for Industrial and Applied Mathematics, c1997, pp. 171-192.

8. Clark D.W., "*An Introduction to Neural Networks*" *http://members.home.net/neuralnet/introtonn/index.htm*

9. Cohen B., Saad D. and Marom E., "*Efficient Training of Recurrent Neural Network with Time Delays*", Neural Networks, Vol. 10, No. 1, pp. 51-59, 1997.

10. Graps A.L.,"*An Introduction to Wavelets*", IEEE Computational Sciences and Engineering, Volume 2, Number 2, Summer 1995, pp. 50-61. *http://www.amara.com/IEEEwave/IEEEwavelet.html*

11. Kröse B. and Smagt P.v.d., the University of Amsterdam, the Netherlands, Eighth edition 1996, "*An introduction to Neural Networks*".

12. Lelewer D. A. and Hirschberg D.S., "*Data Compression*", *http://www.ics.uci.edu/~dan/pubs/DataCompression.html*

13. Li L.K., Chau F.T., Leung K.M., "*Compression of ultraviolet-visible spectrum with recurrent neural network*", Chemometrics and Intelligent Laboratory Systems 52 (2000), pp. 135-143.

14. Li L.K., "*Data Compression By Recurrent Neural Network Dynamics*", Proceedings of 1996 IEEE Region Ten Conference, Vol. 1, pp. 96-101.

15. Li L. K., "*Approximation Theory and Recurrent Network*", Proceedings of IJ CNN-Baltimore-92, Vol. 2, 1992, pp. 266-271.

16. Madisetti V.K., "*Signal Processing for the NII: Future*"

http://www.ee.gatech.edu/users/215/nii/node49.html

17. Mendelsohn L., "*Preprocessing Data For Neural Networks*", Technical Analysis of Stocks & Commodities, October, 1993

http://www.profittaker.com/preprocessing_data.htm

18. Morse B.S., "*Compression*"

http://iul.cs.byu.edu/450/F96/node29.html

19. Niebur D., New York, June 1995, "*Introduction to Concepts in Artificial Neural Networks*"

http://techreports.jpl.nasa.gov/1995/95-0887.pdf

20. Pedersen M.W. and Larsen J., CONNECT, Department of Mathematical Modelling, Technical University of Denmark, "*Interpretation of Recurrent Neural Networks*"

ftp://eivind.imm.dtu.dk/dist/1997/with.nnsp97.inter.ps.Z

21. Rabiner L.R. and Rader C.M. (editors), "*Digital signal processing*", Selected Reprints. New York, NY:IEEE Press 1972.

22. Rao K.R., and Yip P.C., "*The transform and data compression handbook*", CRC Press, 2001.

23. Schuster M. and Paliwal K.K., "*Bidirectional Recurrent Neural Networks*", IEEE Transactions on Signal Processing, Vol. 45, No. 11, Nov. 1997.

24. Stergiou C. and Siganos D., "*Neural Networks*" *http://www-dse.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html*

25. Stergiou C., "*Neural Networks, the Human Brain and Learning*" *http://www-dse.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/cs11/article2.html*

26. Svarer C., CONNECT, Electronic Institute, Technical University of Denmark, Ph.D. Thesis:"*Neural Network for Signal Processing*" *ftp://eivind.imm.dtu.dk/dist/PhD_thesis/csvarer.thesis.ps.Z*

27. Weisstein E.W.,"*Fast Fourier Transform*" *http://mathworld.wolfram.com/FastFourierTransform.html*

28. Williams R.J., Zipser D., Neural Comput. 1 (1989) pp. 270-280.

29. Wylie C.R., 1911- "*Advanced engineering mathematics*", Publisher: New York: McGraw-Hill, c1995 6th ed., pp. 923-934.

30. Ziv J., & Lempel A., ,"*Compression of individual sequences via variable rate coding*", IEEE Trans on Information Theory, Vol. IT-24, pp. 530-536, Sept 1978.

31. "*A Basic Introduction To Neural Networks*" *http://blizzard.gis.uiuc.edu/htmldocs/Neural/neural.html*

Chapter 9

Appendices

9.1 Programs used in the Project

The follow are program listings used in the project

9.1.1 Program for Ultra Violet Spectra with Learning in Single RNN System (network size = 4, 5, 6, 7, 8, 9, 10) (Section 5.1, 5.2 and 5.3)

% Initialization

```
    n=input('net size is = ');

    am=2;

    kq=1;mr=1;pe=1;rmse=zeros(pe,1);

    alp=.1005;

    h=.0545;

    it=iteration number;
```

```
hk=1;alpk=1;itk=1;

% load file to convert

load filename.dat

x0=filename;

le=length(x0);

x=x0;kw=zeros(n,n);

% Cyclic Transformation

m=floor(le/n)*n;mn=m/n;

x0=zeros(1,m);kx=zeros(1,le-m);

for i=1:le-m, kx(i)=x(m+i); end

for i=1:m, x0(i)=x(i); end

origin=x0;

xn=[x0(m)-x0(1)]/[m-1];

for i=1:m-1, x0(i+1)=x0(i+1)-i*xn; end

% Leveling & Bounding

mmx0=mean(x0);

x0=x0-(mmx0);

mx0=max(x0);x0=x0/mx0;

% Normalization

z=x0(1:m);zn=x0(1:m);

ma=max(z);mi=min(z);rz=ma-mi;

for i=1:m, z(i)=((z(i)-mi)/rz-.5)*am; end
```

```
zn=zeros(n,mn);zp=zeros(1,m);

err=zeros(it,1);

% divide into segments

for i=1:n,

     for j=1:mn, zn(i,j)=z(mn*(i-1)+j); end

end

% Learning (Starting)------------------------------------------

y=zn;

for k=1:it,

y=(1-alp)*y+alp*zn;

% Calculate W for segmentation

u=zeros(n,mn-1); v=zeros(n,mn-1);

u=y(:,1:mn-1); v=y(:,2:mn); tu=tanh(u);

uv=[(v-u)/h+u]*tu';

w=uv*inv(tu*tu');

% Processing Data Compression with Dynamic RNN System

for i=1:mn-1,     y(:,i+1)=y(:,i)+h*(-y(:,i)+w*tanh(y(:,i)));     end

% Reconnect Segments together

for i=1:n,

     for j=1:mn,     zp(1,mn*(i-1)+j)=y(i,j);     end

end

% Comparing the error
```

```
 rmse=sqrt([zp-z]*[zp-z]'/m)/am;kp=1;
 err(k)=rmse;
 if err(k) < mr,
      mr=err(k);hk=h;alpk=alp;itk=k; kq=kp; kw=w;
      [err(k) alp h k n]
 end
 end
```

% Learning (Ending)——————————————————————

% Re-produce the best output

```
for i=1:mn-1,      zn(:,i+1)=zn(:,i)+hk*(-zn(:,i)+kw*tanh(zn(:,i)));      end
```

% Reconnect Segments together

```
for i=1:n,
      for j=1:mn,      zp(1,mn*(i-1)+j)=zn(i,j);      end
end
```

% Re-normalize into original range

```
for i=1:m, zp(i)=((zp(i)/am)+.5)*rz+mi; end
```

% Undo Leveling and Bounding

```
zp=zp*mx0;
zp=zp+mmx0;
```

% Re-Cyclic Transformation

```
for i=1:m-1; zp(i+1)=zp(i+1)+i*xn; end
```

```
rmse=sqrt(([zp-origin]*[zp-origin]')/m)/am
mae=sum(abs(zp-origin))/m/am
max_a_e=max(abs(zp-origin))/am

% Combine all together
zzp=zeros(1,le);
for i=1:m,      zzp(1,i)=zp(1,i);      end
for i=1:le-m,      zzp(1,m+i)=kx(1,i);      end

te=1:le;
figure(sn);
plot(te,x+.3,'r',te,zzp,'b');axis('tight');
xlabel('Wavelength','FontSize',12),ylabel('Intensity','FontSize',12);
legend('Original curve','Approximation curve');
print ('-dbitmap',s);;
```

9.1.2 Program for Ultra Violet Spectra without Learning in Single RNN System (network size = 4 & 8) (Section 5.3.1 & 5.3.2)

```
% Initialization
  n=input('net size is = ');
  am=2;
  r1=1;r2=200;re=zeros(r1,r2)+1;
  kq=1;%p=7;
```

```
mr=1;pe=1;rmse=zeros(pe,1);

it=1;

l1=.100; ll1=.0005; step=.0005;

hk=1;alpk=1;itk=1;

% load file to convert

load filename.dat

x0=filename;

le=length(x0);

x=x0;origin=x0;kw=zeros(n,n);

% Cyclic Transformation

m=floor(le/n)*n;mn=m/n;

x0=zeros(1,m);kx=zeros(1,le-m);

for i=1:le-m, kx(i)=x(m+i); end

for i=1:m, x0(i)=x(i); end

origin=x0;

xn=[x0(m)-x0(1)]/[m-1];

for i=1:m-1, x0(i+1)=x0(i+1)-i*xn; end

% Leveling & Bounding

mmx0=mean(x0);

x0=x0-(mmx0);

mx0=max(x0);x0=x0/mx0;

% Normalization
```

```
z=x0(1:m);zn=x0(1:m);
ma=max(z);mi=min(z);rz=ma-mi;
for i=1:m, z(i)=((z(i)-mi)/rz-.5)*am; end
zn=zeros(n,mn);zp=zeros(1,m);

% divide into segments
for i=1:n,
for j=1:mn, zn(i,j)=z(mn*(i-1)+j); end
end

% Learning (Starting)------------------------------------
y=zn;
for k=1:it,
y=(1-alp)*y+alp*zn;

% Calculate W for segmentation
u=zeros(n,mn-1); v=zeros(n,mn-1);
u=y(:,1:mn-1); v=y(:,2:mn); tu=tanh(u);
uv=[(v-u)/h+u]*tu';
for p=1:pe,
pp=10^(-p-7)*eye(n);
w=uv*inv(tu*tu'+pp);

% Processing Data Compression with Dynamic RNN System
for i=1:mn-1, y(:,i+1)=y(:,i)+h*(-y(:,i)+w*tanh(y(:,i))); end

% Reconnect Segments together
```

```
for i=1:n,

for j=1:mn, zp(1,mn*(i-1)+j)=y(i,j); end

end

% Comparing the error

rmse=sqrt([zp-z]*[zp-z]'/m)/am;kp=1;

end

err(k)=rmse;%min(rmse);

if err(k) < mr,

mr=err(k);hk=h;alpk=alp;itk=k; kq=kp; kw=w;

[err(k) alp h k n]

end

end

% Learning (Ending)————————————————————

% Re-produce the best output

for i=1:mn-1, zn(:,i+1)=zn(:,i)+hk*(-zn(:,i)+kw*tanh(zn(:,i))); end

% Reconnect Segments together

for i=1:n,

for j=1:mn, zp(1,mn*(i-1)+j)=zn(i,j); end

end

% Re-normalize into original range

for i=1:m, zp(i)=((zp(i)/am)+.5)*rz+mi; end

% Undo Leveling and Bounding
```

```
zp=zp*mx0;

zp=zp+mmx0;

% Re-Cyclic Transformation

for i=1:m-1; zp(i+1)=zp(i+1)+i*xn; end

rmse=sqrt(([zp-origin]*[zp-origin]')/m)/am

mae=sum(abs(zp-origin))/m/am

max_a_e=max(abs(zp-origin))/am

save (s,'rmse','mae','max_a_e','kw','alpk','hk','itk','-ascii','-double','-tabs');

% Combine all together

zzp=zeros(1,le);

for i=1:m, zzp(1,i)=zp(1,i); end

for i=1:le-m, zzp(1,m+i)=kx(1,i); end

te=1:le;

figure(sn);

plot(te,x+.3,'r',te,zzp,'b');axis('tight');

xlabel('Wavelength','FontSize',12),ylabel('Intensity','FontSize',12);

legend('Original curve','Approximation curve');

print ('-dbitmap',s);
```

9.1.3 Program for Infra Red Spectra with Learning in Single RNN System (network size = 8 & 9) (Section 6.1 and 6.2)

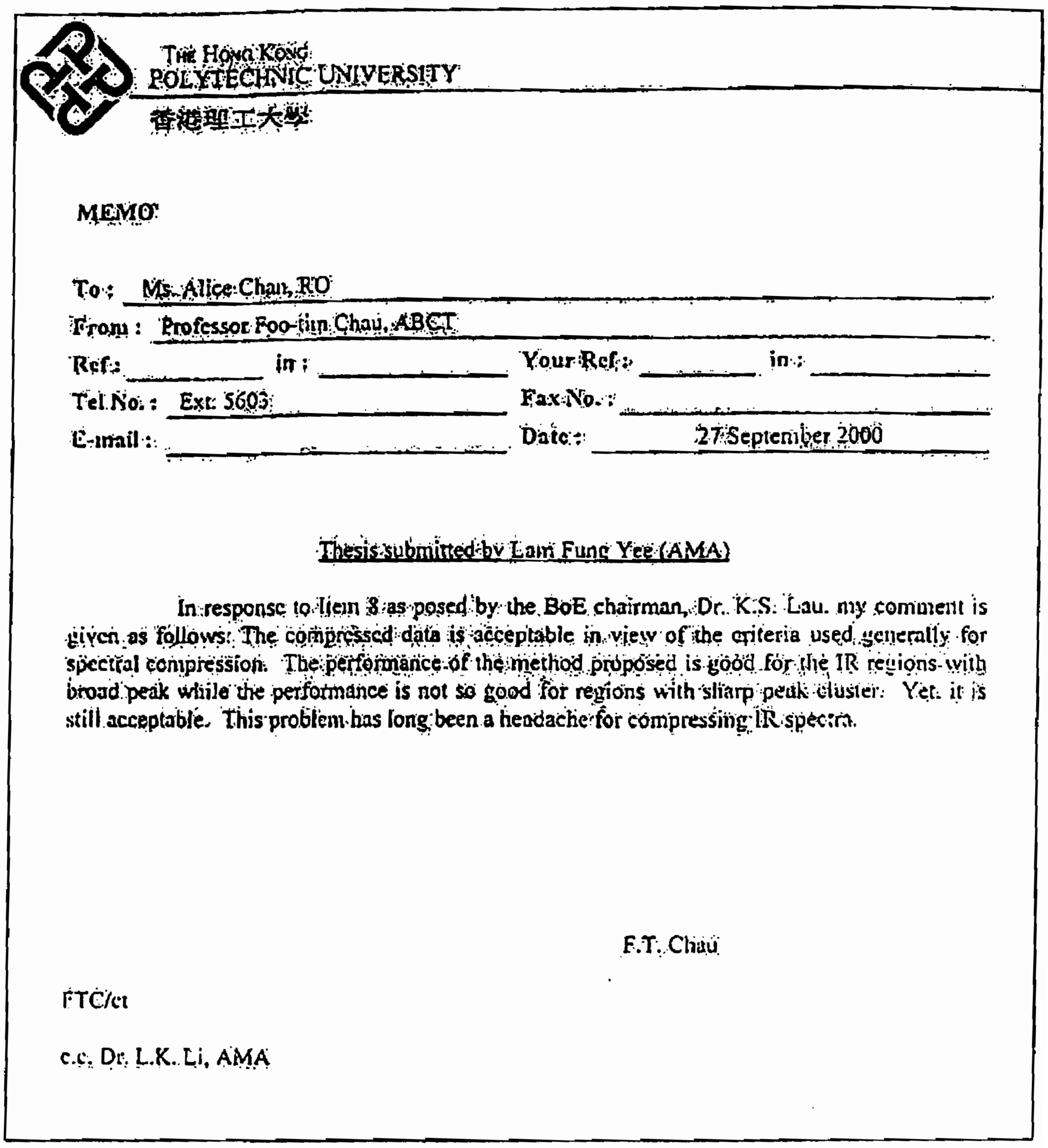
```
% Initialization

 n=input('Network size is = ');

 alp=0.1090;

 h=0.1495;

 it=iteration number;

 % load file to convert

 load filename.dat

 x0=filename;

 le=length(x0); am=2;

 x=x0(1:le);kw=zeros(n,n);

 % Cyclic Transformation

 m=floor(le/n)*n;

 x0=zeros(1,m);kx=zeros(1,le-m);

 for i=1:le-m,     kx(i)=x(m+i);     end

 for i=1:m,     x0(i)=x(i);     end

 origin=x0;

 xn=[x0(m)-x0(1)]/[m-1];

 for i=1:m-1,     x0(i+1)=x0(i+1)-i*xn;     end

 % Leveling & Bounding

 mmx0=mean(x0);x0=x0-(mmx0);

 mx0=max(x0);x0=x0/mx0;

 mn=m/n;
```

```
kq=1;mr=1;pe=1;rmse=zeros(pe,1);
hk=1;alpk=1;itk=1;

% Normalization
z=x0(1:m);zn=x0(1:m);
ma=max(z);mi=min(z);rz=ma-mi;
for i=1:m,     z(i)=((z(i)-mi)/rz-.5)*am;     end
zn=zeros(n,mn); zp=zeros(m,1);
err=zeros(it,1);
for i=1:n,
     for j=1:mn,     zn(i,j)=z((i-1)*mn+j);     end
end
y=zn;

% Learning ====================
for k=1:it,
y=(1-alp)*y+alp*zn;

% Calculate W for segmentation
u=y(:,1:mn-1);v=y(:,2:mn);tu=tanh(u);
uv=[(v-u)/h+u]*tu';
w=uv*inv(tu*tu');

% Processing Data Compression with Dynamic RNN System
for i=1:mn-1,     y(:,i+1)=y(:,i)+h*[-y(:,i)+w*tanh(y(:,i))];     end

% Reconnect Segments together
```

```
for i=1:n,
     for j=1:mn,      zp(j+(i-1)*mn)=y(i,j);      end
end

% Comparing the error

rmse=sqrt([zp-z']'*[zp-z']/m)/am;kp=1;

err(k)=rmse;

if err(k) < mr,

     mr=err(k);hk=h;alpk=alp;itk=k;kq=kp;kw=w;

     [err(k) alp h k n]

end

end

% End of Learning =====================

% Processing by Dynamic RNN System

for i=1:mn-1, zn(:,i+1)=zn(:,i)+hk*[-zn(:,i)+kw*tanh(zn(:,i))]; end

% Connect Segments together

zp=zeros(m,1);

for i=1:n,

     for j=1:mn,      zp(j+(i-1)*mn)=zn(i,j);      end

end

% Re-normalization

for i=1:m;      zp(i)=((zp(i)/am)+.5)*rz+mi;      end

% Undo Leveling and Bounding
```

```
zp=zp*mx0;      zp=zp+mmx0;

% Re-Cyclic Transformation

for i=1:m-1;      zp(i+1)=zp(i+1)+i*xn;      end

rmse=sqrt(([zp-origin']'*[zp-origin'])/m)/am

mae=sum(abs(zp-origin'))/m/am

max_a_e=max(abs(zp-origin'))/am

% Combine all together

zzp=zeros(le,1);

for i=1:m, zzp(i,1)=zp(i,1); end

for i=1:le-m, zzp(m+i,1)=kx(1,i); end

te=1:le;

igure(sn);

plot(te,x,'r',te,zzp','b');axis('tight');

xlabel('Wavelength','FontSize',12),ylabel('Intensity','FontSize',12);

legend('Original curve','Approximation curve');

print ('-dbitmap',s);
```

9.2 Feedback from Professor Chau



THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

MEMO

To :	Ms. Alice Chau, RO		
From :	Professor Foo-tim Chau, ABCT		
Ref.:	in :	Your Ref.:	in :
Tel. No. :	Ext. 5605	Fax No. :	
E-mail :		Date :	27 September 2000

Thesis submitted by Lam Fung Yee (AMA)

In response to Item 8 as posed by the BoE chairman, Dr. K.S. Lau, my comment is given as follows: The compressed data is acceptable in view of the criteria used generally for spectral compression. The performance of the method proposed is good for the IR regions with broad peak while the performance is not so good for regions with sharp peak cluster. Yet, it is still acceptable. This problem has long been a headache for compressing IR spectra.

F.T. Chau

FTC/ct

c.c. Dr. L.K. Li, AMA