Design and Development of a Neural Network based Speech Recognition System

Sean C. Lennon B.E. (Electronic)

Submitted to the National University of Ireland in fulfilment of the requirements for the Degree of Master of Engineering Science.

Department of Electronic Engineering
University College
Galway

Head of Department
Prof D. Wilcox

Supervisor
Dr. E. Ambikairajah

March 1993

Volume 1
(of two volumes)
Abstract

The object of this research was to develop a neural network based speech recognition system and then implement it on transputers. To this end a two-layer model of Kohonen's self-organising network was developed. Two pre-processors were tried as front-ends for the neural network. One was a physiologically based cochlear model while the other was a parallel filter bank. Sammon's non-linear mapping and K-means clustering algorithm were both used to analyse N-dimensional data produced by the cochlear model.

Two isolated word recognition systems were implemented on five transputers. System 1 uses a 19 channel parallel filter bank based on the mel frequency scale as a front-end to the two-layer Kohonen network. System 2 uses an 8 kHz cochlear model outputting vectors of dimensionality 90 as its front-end.
### Contents

**Volume 1**

**Chapter 1**  
Speech Recognition Systems and Neural Networks  
1.1 Introduction  
1.2 Conventional Speech Recognition Systems  
1.3 Neural Networks  
1.4 Kohonen's Self-Organising Neural Network  
1.5 Transputer Technology  
1.6 Transputer Implementation  

**Chapter 2**  
Kohonen's Self-Organising Neural Network  
2.1 Introduction  
2.2 Description of the Neural Array  
2.3 Kohonen's algorithm  
2.4 Training parameters  
2.5 Tuning the Kohonen network  
2.5.1 Tuning of a 2-d network to 2-d input vectors  
2.5.2 Tuning of a 2-d network to 60-d input vectors  
2.6 Characteristic trajectories  
2.7 The Second Kohonen Neural Layer  

**Chapter 3**  
Pre-processors: Parallel Filter Bank, Triangular Filter Bank, Linear Predictive Coding  
3.1 Introduction  
3.2 Pre-processing of Speech Signals  
3.3 Development of the Parallel Filter Bank  
3.3.1 Band-Pass Filter Section  
3.3.2 Frequency Response of Band-Pass Filters
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.3 Nonlinearity and Low-Pass Filter Section</td>
<td>31</td>
</tr>
<tr>
<td>3.3.4 Logarithmic Encoding and Postprocessing</td>
<td>33</td>
</tr>
<tr>
<td>3.3.5 Mel Frequency Cepstral Coefficients</td>
<td>33</td>
</tr>
<tr>
<td>3.3.6 Description of the Parallel Filter Bank Program</td>
<td>34</td>
</tr>
<tr>
<td>3.4 Triangular Filter Bank</td>
<td>36</td>
</tr>
<tr>
<td>3.4.1 Fast Fourier Transform</td>
<td>37</td>
</tr>
<tr>
<td>3.4.2 The Triangular Filters</td>
<td>38</td>
</tr>
<tr>
<td>3.4.3 Overall Frequency Response</td>
<td>40</td>
</tr>
<tr>
<td>3.4.4 Example Inputs and Outputs</td>
<td>40</td>
</tr>
<tr>
<td>3.5 Linear Predictive Coding</td>
<td>41</td>
</tr>
<tr>
<td>3.5.1 Theory of Linear Predictive Coding</td>
<td>42</td>
</tr>
<tr>
<td>3.5.2 Example LPC outputs from Hypersignal</td>
<td>43</td>
</tr>
<tr>
<td>3.6 Collection of Speech Samples</td>
<td>43</td>
</tr>
<tr>
<td>Chapter 4 Preprocessors: The Cochlear Model</td>
<td>45</td>
</tr>
<tr>
<td>4.1 An Introduction to the Anatomy of the Cochlea</td>
<td>45</td>
</tr>
<tr>
<td>4.2 The Basilar Membrane</td>
<td>46</td>
</tr>
<tr>
<td>4.3 The Physiologically-based Cochlear Model</td>
<td>47</td>
</tr>
<tr>
<td>4.4 Use of the Cochlear Model in this research</td>
<td>49</td>
</tr>
<tr>
<td>4.5 The 48 kHz Cochlear Model</td>
<td>50</td>
</tr>
<tr>
<td>4.5.1 Pre-emphasis</td>
<td>51</td>
</tr>
<tr>
<td>4.5.2 Interpolation</td>
<td>52</td>
</tr>
<tr>
<td>4.5.3 Inner Hair Cells</td>
<td>54</td>
</tr>
<tr>
<td>4.5.4 A Smoothing Algorithm</td>
<td>57</td>
</tr>
<tr>
<td>4.6 The 8 kHz Cochlear Model</td>
<td>58</td>
</tr>
<tr>
<td>4.7 Simulation Results with the 48 kHz model</td>
<td>58</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Cluster Analysis of the Cochlear Model Outputs</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>Sammon's Non-linear Mapping</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Implementation of Sammon's Algorithm</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Linear Time Warping</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Dynamic Time Warping</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Results obtained using Sammon's algorithm</td>
</tr>
<tr>
<td>5.3</td>
<td>K-means clustering algorithm</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Results obtained using K-means algorithm</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Speech Recognition Results</td>
</tr>
<tr>
<td>6.1</td>
<td>Recognition with the two-layer Kohonen Neural Net</td>
</tr>
<tr>
<td>6.2</td>
<td>Recognition results using the Cochlear Model</td>
</tr>
<tr>
<td>6.3</td>
<td>Recognition results using the Parallel Filter Bank</td>
</tr>
<tr>
<td>6.4</td>
<td>Recognition results using the Triangular Filter Bank</td>
</tr>
<tr>
<td>6.5</td>
<td>Recognition results using Linear Predictive Coding Outputs</td>
</tr>
<tr>
<td>6.6</td>
<td>Techniques for monitoring topological ordering</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Transputer Implementation of Speech Recognition System1 using the Parallel Filter Bank</td>
</tr>
<tr>
<td>7.1</td>
<td>The Inmos Transputer Family</td>
</tr>
<tr>
<td>7.2</td>
<td>Transputer hardware used in this application</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Topology of Transputers used in System Implementation</td>
</tr>
<tr>
<td>Contents</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>7.3 Implementation of System 1 using the Parallel</td>
<td>92</td>
</tr>
<tr>
<td>7.3.1 Configuration of Transputers used in System 1</td>
<td>93</td>
</tr>
<tr>
<td>7.3.2 A Description of the Processes used in System 1</td>
<td>94</td>
</tr>
<tr>
<td>7.3.3 Stage 1- Implementation of the Front-end Processor</td>
<td>95</td>
</tr>
<tr>
<td>7.3.4 Stage 2- Implementation of the Two-layer Neural Network</td>
<td>95</td>
</tr>
<tr>
<td>7.3.5 Labelling the Second Layer</td>
<td>98</td>
</tr>
<tr>
<td>7.4 Optimisation factors involved in the Implementation</td>
<td>99</td>
</tr>
<tr>
<td>Chapter 8 Transputer Implementation of System 2 using the Cochlear Model</td>
<td></td>
</tr>
<tr>
<td>8.1 Introduction</td>
<td>101</td>
</tr>
<tr>
<td>8.2 The Cochlear Model- a cascade structure</td>
<td>101</td>
</tr>
<tr>
<td>8.3 Topological structure of the Implementation of System 2</td>
<td>105</td>
</tr>
<tr>
<td>8.4 A description of the Processes used in System 2</td>
<td>106</td>
</tr>
<tr>
<td>8.4.1 Stage 1- Implementation of the Cochlear Model</td>
<td>106</td>
</tr>
<tr>
<td>8.4.2 Stage 2- Implementation of the Two-layer Neural Network</td>
<td>108</td>
</tr>
<tr>
<td>8.5 Optimisation issues in System 2</td>
<td>109</td>
</tr>
<tr>
<td>8.6 A Comparision of both Systems</td>
<td>111</td>
</tr>
<tr>
<td>Chapter 9 Conclusion</td>
<td>112</td>
</tr>
<tr>
<td>References</td>
<td>Page</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>References</td>
<td>116</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Publications</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Paper presented at 2nd International Conference on Spoken Language Processing (ICSLP-92), Oct 92, Banff, Alberta, Canada</td>
<td>119</td>
</tr>
<tr>
<td>(2) Paper presented at 2nd IEEE Workshop on Neural Networks for Signal Processing, Aug 92, Helsingor, Denmark</td>
<td></td>
</tr>
<tr>
<td>(3) Paper presented at AI and Cognitive Science (AICS-90), Aug 90, Belfast, Northern Ireland</td>
<td></td>
</tr>
</tbody>
</table>
Contents

Volume 2

Appendix A

Neural Network programs

Appendix A1 "k15": a pascal program simulating the first-layer of a Kohonen network
Appendix A2 "klayer2": a pascal program simulating the second layer of the two-layer Kohonen network
Appendix A3 "labelnet": a pascal program which labels the second layer of the Kohonen network

Appendix B

Pre-processor programs and filter coefficients

Appendix B1 "parallel_filter_bank": a pascal program implementing the parallel filter bank
Appendix B2 "triangular_filter_bank": a pascal program implementing the triangular filter bank
Appendix B3 "band_pass_filter": a pascal program implementing a band pass filter which converts inputted speech into telephone bandwidth speech
Appendix B4 "48kHz_cochlear_model": a pascal program simulating the cochlear model operating at 48 kHz
Contents

Volume 2

Appendix B5  "8kHz_cochlear_model": a pascal program
             simulating the cochlear model operating at 8 kHz
Appendix B6  IIR band-pass filter coefficients for the parallel
             filter bank
Appendix B7  FIR filter coefficients for the band-pass filter
             program
Appendix B8  FIR filter coefficients for the interpolation filter
Appendix B9  IIR filter coefficients for the 48 kHz cochlear
             model
Appendix B10 IIR filter coefficients for the 8 kHz cochlear model

Appendix C

Clustering Analysis and Time Warping programs

Appendix C1  "Sammon's_nonlinear_mapping": a pascal program
             implementing Sammon's nonlinear mapping process.
             The program uses some procedures stored in unit
             "domap" and unit "mygraphs"
Appendix C2  "unit domap": a pascal unit containing mapping
             procedures
Appendix C3  "unit mygraphs": a pascal unit containing graphic
             and other procedures
Contents

Volume 2

Appendix C4 "time_warp_of_speech": a pascal program implementing linear time warping of speech inputs

Appendix C5 "dynamic_time_warping": a pascal program implementing dynamic time warping of linearly warped speech inputs

Appendix C6 "k_means_clustering": a pascal program implementing the k-means clustering algorithm

Appendix D

Occam programs implementing Speech Recognition System 1

Appendix D1 "amalg": an occam configuration file

Appendix D2 "filt": an occam process implementing several parallel filter bank channels

Appendix D3 "filtneu": an occam process implementing parallel filter bank channels and the neural network

Appendix D4 "gather": a distributive process which accumulates results and distributes data to the slave processes

Appendix D5 "pfb": the master process which has access to the host
Contents

Volume 2

Appendix E

Occam programs implementing Speech Recognition System 2

Appendix E1 "cochpgm": an occam configuration file
Appendix E2 "coch" : the master occam process which has access to the host computer
Appendix E3 "filtneug": an occam process implementing a cascade of n filters and also the neural network
Appendix E4 "filtont1" an occam process executing on transputer 1 (t1) implementing n filters starting at filter number 1
Appendix E5 "filtont5" an occam process executing on transputer 5 (t5) implementing the last n filters of the cascade
Appendix E6 "holder" : a distributive process which collects results and sends data to slave processes
Chapter 1

Speech Recognition Systems and Neural Networks

1.1 Introduction

The objective of this research was to develop a neural network based speech recognition system and to implement it on Inmos transputers. In this chapter, a general rational is given for the approach adopted. A neural network recognition system is compared with a conventional recognition system and their differences are highlighted. Some background information is provided on neural networks in general and the motivation behind using Kohonen’s self-organising network in particular, is explained. Also in this chapter, the advantages of using transputers as an implementation platform for this application are discussed and a brief account of the two speech recognition systems developed is given.

1.2 Conventional Recognition Systems

Conventional speech recognition systems normally use a front-end processor to extract certain features from the incoming speech waveform (see Figure 1.1). A transformation from a time domain
representation to its equivalent frequency domain is a common property of these pre-processors.

![Diagram of Conventional Speech Recognition Systems]

Figure 1.1 Conventional Speech Recognition Systems

The output emerging from the front-end processor is usually an n-dimensional vector. A comparison is made between reference templates stored in memory and the current pattern vector produced by the pre-processor. A decision is then made based on how similar the pattern vector is to its most closely matched reference pattern. If the two vectors are deemed to be sufficiently similar, the system responds by declaring the sound associated with the reference template as being the current input.

The speech recognition systems described in this research use a different approach. Rather than trying to match reference templates to the outputs of the front-end processor, the outputs are directed
instead into a neural network model (see Figure 1.2). A number of different varieties of neural networks may be used here including the Multi-Layer Perceptron, but the network that was chosen is a self-organising network based on Kohonen's algorithm.

![Diagram](image)

Many different neural networks suitable here including Multi Layer Perceptron and Kohonen's Self Organising Feature Maps.

Figure 1.2 A Neural Network based Speech Recognition System

A two-layer neural network model based on this algorithm was designed, simulated and developed. Two different front-end processors were also simulated.

1.3 Neural Networks

The design of artificial neural networks is based on our understanding of how the brain uses networks of neurons. This understanding is still limited. In crude terms, the human brain is a natural computer composed of 10 billion to 100 billion neurons, each of which connects
to about 10000 others, and all of which function in parallel. It is believed that neurons perform simple computations and operate very slowly by comparison to electronic computers. Yet the brain can solve difficult problems of vision and language in about half a second.

A neuron consists of a cell body, branching extensions called dendrites that receive input, and an axon that carries the neurons’s output to the dendrites of other neurons. The junction between an axon and a dendrite is called a synapse. A neuron carries out a simple calculation: it collects signals at its synapse and adds them; if the combined strength of the signal exceeds a certain threshold the neuron sends out a signal. This natural phenomena may be modelled mathematically as in Figure 1.3 where the neuron output will fire only if the weighted sum of its n inputs exceeds the set threshold. The weights (w_i) are adaptive and are modified during the learning process.

Figure 1.3 Mathematical model of biological neuron
Artificial neural networks consist of many simple interconnected processing elements, operating in parallel as in biological nervous systems. In recent years, they have attracted a resurgence of interest, principally due to the failure of conventional von Neumann techniques in solving such diverse problems as speech and image processing, robotic control and pattern recognition. Such networks are particularly suited to carrying out speech and image recognition tasks where high computation rates are necessary.

The field of neural networks first came into existence in 1943 when McCulloch and Pitts [18] published a paper entitled "A logical calculus of the ideas imminent in nervous activity". They showed that networks of neuron-like elements are general computing devices. The elements in their model were threshold devices. The concept of learning was added to this in 1949 when Hebb [8] suggested that synapses are the site of biological learning.

Hebb's theory states that synapses that are frequently active should have an increased chance of becoming active again. This rule is the basis of modelling neural networks and also leads to understanding of how animals learn.

Researchers became interested in the possibility of building artificial neural networks in 1957 when Rosenblatt [23] proposed a model using elements called "perceptrons". A single layer of perceptrons, based on neurons with Hebbian learning, could solve simple problems involving
recognition of patterns. Active research continued throughout the late 1950's and the 1960's.

Then Minsky and Papert [19] published a paper which demonstrated that a neural network in a single layer would require an absurdly large number of elements to solve any interesting problems. They also suggested that extending the model to allow multiple layers would not dramatically increase the processing power. The paper dampened interest in the field and many researchers turned their attention to other projects in the main stream of artificial intelligence.

Interest in artificial neural networks was renewed in the early 1980's when Hopfield [9] developed a new learning model with multiple layers that provided the basis for a number of other models. These developments and others demonstrated that extensions of the perceptron model with multiple layers overcame the limitations that were predicated by Minsky and Papert. Today, the more well-known neural networks in use include the Multi-Layer Perceptron, the Hopfield net, the Boltzmann and Cauchy machines and Kohonen's self-organising network.

1.4 Kohonen's Self-Organising Neural Network

Recent research suggests that the brain contains various topologically ordered maps where different neural cells respond optimally to different signal quantities. The self-organising maps discovered by Kohonen are based upon the behaviour of biological neurons. In its
most basic form the network consists of a rectangular array of elements which form a topologically ordered map of the data to which it is exposed. This means that the relative physical positions of neurons fired by different stimuli mirrors some significant relationship between the stimuli. This is useful when the perceptual relationship between stimuli is reflected in the pattern space from which they are drawn as in the case of speech. For example, when the network is trained on spectral vectors which represent different vowel sounds, different neural elements in the array tune themselves to different vowel sounds. This property allows the Kohonen net to be used for speech recognition purposes.

The Kohonen network maps signals from a high-order dimensional space to a topologically-ordered low-order dimensional space. This low-order dimensional space has a dimensionality usually equal to two. The front-end processors used in today's speech recognition produce high-order dimensional vectors in the frequency domain. A sequence of high dimensional vectors representing a word, outputted from these front-ends, is converted by the Kohonen network, into a sequence of two-dimensional co-ordinates. This dimensionality reduction simplifies later analysis. In using Kohonen's self-organising network, a method by which the brain accommodates high-dimensional incoming signals can be modelled.
1.5 Transputer Technology

The Inmos transputer family, designed with parallel applications in mind, is particularly suitable for use, when working with neural networks. This is because neural networks attempt to mimic biological nervous systems by utilising a large number of interconnected processing elements which interact on a massively parallel scale. Transputers too, may be used on a very large parallel scale. Some supercomputers are on sale using in excess of 60,000 transputers working in parallel.

The flexibility of transputers being used in unison to achieve an impressive numerical performance has allowed many applications including speech recognition, with its high computational requirement, to become feasible. The ability to use a variety of different topologies, when connecting transputers together, to implement multiple configurations is an added advantage, especially when working with neural networks.

1.6 Transputer Implementation

The implementation of two speech recognition systems on a series of transputers is described in Chapter 7 and 8. Both systems may be thought of as comprising of two stages:- a front-end processor section and a two-layer neural network section (Figure 1.4). Each system uses a different front-end processor while both systems use a
similar neural network model, based on Kohonen's algorithm, as the second stage.

Speech Recognition System 1

Speech Recognition System 2

Speech recognition system 1 uses a parallel filter bank, which is constructed using 19 fourth-order elliptical filters, and spans the bandwidth 250Hz to 4kHz, as its front-end processor section. The second system's pre-processor is a cochlear model and consists of a cascade of 128 digital filters. The outputs from 90 of these 128 filters are tapped off producing a vector spanning the same speech bandwidth of 250Hz to 4kHz. After the two-layer neural network has been trained, it accepts outputs from the front-end processors and decides which word most closely represents the unknown input.
Chapter 2

Kohonen's Self Organising Neural Network

2.1 Introduction

Kohonen's neural network is an unsupervised self-organising network. It consists of a rectangular array, which, after sufficient training, forms a topologically-ordered map of the input data. This means that input data vectors which are close together in their N-dimensional space will be mapped to neurons which are physically close together in two dimensional space. This property may be exploited as follows: if a sequence of slowly varying input vectors is presented to the trained network, neurons with adjacent co-ordinates in the two-dimensional array become successively excited. A trajectory is generated connecting the sequence of excited or "fired" neurons. This trajectory is then characteristic of the sequence of incoming high-dimensional pattern vectors. In this way, data from a multi-dimensional space may be mapped to a sequence of two dimensional co-ordinates, which when joined, produces a characteristic trajectory.
The front-end processor used for these simulations was a cochlear model and can be considered well suited as a pre-processor for a model of the brain. The cochlear model carries out a transformation from the time domain to the frequency domain. When stimulated, the output of the model is a sequence of high-dimensional vectors, each vector element indicating the relative magnitudes of the stimulus frequency components at a particular time instant.

It is difficult to organise large quantities of high-dimensional data using conventional techniques. This is where Kohonen's self-organising net is so useful. The Kohonen net quantises the input vectors into classes according to their relative frequency of occurrence. Thus, for a commonly occurring input, a class representing that input will be found in the net. Due to the fact that the map is non-linear, much of the original high-dimensional information is preserved, while a lower dimensional representation is obtained.

In this chapter a description of Kohonen's algorithm is presented. Also included are plots of trajectories produced by the neural network when excited by different speech sounds. An account of how a second neural layer may accept inputs from the first is described. The manner in which the second layer may be used for word recognition purposes is explained. The simulation results presented here used 60-dimensional spectral outputs from the cochlear model operating at 48kHz. In the implementation phase
(Chapter 8) 90-dimensional vectors from a cochlear model operating at 8 kHz were used as inputs to the neural net.

2.2 Description of the Neural Array

The network is arranged as a two-dimensional array, the position of each neuron being defined by i and j co-ordinates. It contains M*M neurons, each neuron having N weights associated with it. The neurons which fire to each of the L successive inputs are labelled as (U1,V1) to (UL,VL) (Figure 2.1).

Kohonen’s Neural Array

\[
\begin{array}{c}
X \\
X_1 \\
X_2 \\
\vdots \\
X_n \\
\end{array}
\quad
\begin{array}{c}
W_{ij} \\
W_1 \\
W_2 \\
\vdots \\
W_n \\
\end{array}
\]

Input Vector $X$    Weight Vector $W$

Trajectory consists of sequence of ‘winning’ neurons

Figure 2.1 A Layer of Kohonen’s Neural Array
These weights, which represent the synaptic connections of the biological neurons, correspond directly to the N dimensions of the input vector. It is generally accepted that the process of learning is associated with the modification of these weights. There are no physical lateral connections between the neurons.

The training of the array takes place in the following manner: an N-dimensional input vector, X, chosen at random from a large representative set of input pattern vectors, is compared with the weights of each neuron in the array (the weights are initially set to small random values). The neuron whose weights most closely match the input vector is chosen as the 'winner'. This is the neuron with the minimum Euclidean distance between its weights and the input vector. A set region, encompassing a number of neurons, is defined around the winning neuron, and is called its neighbourhood. The process of topological ordering requires that this neighbourhood initially include all neurons in the array and subsequently decrease in size. As more and more input vectors are applied to the network, different parts of the array become selectively sensitised to different classes of inputs.

2.3 Kohonen's Algorithm

The steps involved in implementing Kohonen's self-organising network algorithm [15], [16] are as follows:
Step 1. Initialise the weights
The weights of each of the \( M^*M \) neurons are initialised to random values and the initial neighbourhood is set to cover all neurons.

Step 2. Present the input vector
The current input vector \((x_1, x_2, \ldots, x_N)\) is randomly chosen from a large representative set of input pattern vectors, and applied without supervision to the array.

Step 3. Compute the distance between the input and all neurons
The distance, \( d_{ij} \), between the input vector and the weights of each neuron is computed as follows:

\[
    d_{ij} = \sum_{k=1}^{N} \left( x_k(t) - W_{ijk}(t) \right)^2
\]  \hspace{1cm} (2.1)

where \( x_k(t) \) is the current input at time \( t \); \( W_{ijk}(t) \) is the weight at time \( t \);
\( i = 1, 2, \ldots, M \); \( j = 1, 2, \ldots, M \); \( k = 1, 2, \ldots, N \).

Step 4. Select the best matching neuron
The neuron with co-ordinates \( i,j \) such that \( d_{ij} \) is a minimum is selected.
Step 5. Update weights to neuron i,j and neighbours
The weights of the winning neuron i,j and of all neurons within the current neighbourhood are updated.
The new weights are calculated from:

\[ W_{ij}^{(t+1)} = W_{ij}^{(t)} + a(t) \{ X_k^{(t)} - W_{ij}^{(t)} \} \]

i and j range through all values within the current neighbourhood. a(t) is a gain term which decreases in time (0 < a(t) < 1); the size of the neighbourhood also decreases in time (section 2.4). One can see from the above equation that the weight vector will tend to take on values which match the values of commonly occurring input vectors.

Step 6. Repeat Steps 2 to 5
until sufficient iterations have been carried out to tune the network.

2.4 Training parameters

During this investigation, the following values were found to provide good topological ordering of the network. For improved ordering, lower values of gain and lower shrinkage rates of the neighbourhood may be chosen, but only at the expense of more iterations.
Training is carried out in two phases with varying gain parameter \( a(t) \).

The gain parameter is defined as:

\[
    a(t) = \begin{cases} 
        c_1 \left(1 - \frac{t}{T_1}\right) & \text{during the first training phase } (0 < t < T_1); \\
        c_2 \left(1 - \frac{t}{T_2}\right) & \text{during the second training phase } (T_1 < t < T_2);
    \end{cases}
\]

(2.3)

\( c_1 = 0.1, \ c_2 = 0.008, \ T_1 = 10000, \ T_2 = 90000. \)

where \( T_1 \) and \( T_2 \) are the number of iterations of the first and second training phases respectively.

The size of the neighbourhood surrounding the winner, within which the weights of the neurons are updated, varies as follows:

\[
    R(t) = \begin{cases} 
        R + (1-R)\frac{t}{T_1} & \text{during first training phase; } \\
        1 & \text{during second training phase; }
    \end{cases}
\]

(2.4)

where \( R(t) \) is the current size of the neighbourhood (for example a 6*6 array will have \( R = 6 \)).

A final value of \( R(t) = 1 \) means only the winner and the eight neurons which are closest to it are updated in phase 2 (see Figure 2.2). There must be a sufficient number of statistically distributed inputs in order to obtain the topologically ordered map, especially when increasing the dimensionality of the input vector. A ratio of ten inputs to one neuron is recommended, and ordering improves if
more inputs are added. According to Kohonen, between 500 and 5000 iterations are required per neuron, to completely tune a network.

Neighbourhood decreasing in time

Figure 2.2 Neighbourhood around the winning neuron at three different time instants. The same winning neuron is assumed for each size of neighbourhood.
2.5 Tuning the Kohonen network

The Kohonen network may be trained on input vectors of any dimensionality. However, as the dimensionality of the vector increases, topological ordering becomes more difficult to attain and training the network takes longer.

2.5.1 Tuning of a 2-D Network to 2-D Input Vectors

An 8*8 neural array was trained with a two-dimensional random elliptical input vector. This data is represented by the dots in Figure 2.3. Also shown in Figures 2.3(a) to 2.3(d) are the synaptic weight values of the array for four different time instants during the training period. Lines are drawn to connect the synaptic weight values of elements which are adjacent in the array. Figure 2.3(a) shows the array after 20 iterations. Folding is evident in this diagram. The array is shown after 100 iterations in Figure 2.3(b). The net is now beginning to spread out and cover the input data. Figure 2.3(c) shows the situation after 500 iterations. After 1000 iterations (Figure 2.3(d)), the synaptic weights have taken on positions which match the input data space in a topologically-ordered manner, in other words, the grid is not folded. Note that the input data was presented to the network in a completely random order, without supervision.
Figure 2.3 Kohonen network training on 2-d vectors
2.5.2 Tuning of a 2-D Array to 60-D Input Vectors

For accurate mapping from a higher-dimensional to a lower-dimensional space the inherent dimensionality of the data must be of the same order as the dimension to which it is mapped [25]. For example the inherent dimensionality of vowel sounds in speech is only two. Thus, these sounds may be mapped from a higher dimensional representation to a two dimensional array, without folding occurring between dimensions. If data is mapped without folding between dimensions, it can be stated that the data has the same inherent dimensionality as the lower space to which it is mapped.

The phenomenon of folding may be observed graphically during the training phase, by connecting neurons to their four nearest neighbours. For neurons located at the edges of the array, three connections will suffice. By plotting the weights of dimension p as an x co-ordinate and the weights of dimension q as a y co-ordinate, the ordering between the two dimensions may be observed. For a 60-dimensional map, p and q are any integers between 1 and 60. If the same dimensions p and q of the input data are also plotted, the tuning of the network to the input data may also be observed.

A plot of ordering between dimensions for a 10*10 array is shown in Figure 2.4., after 40,000 iterations. For example, in the top left corner of Fig. 2.4, the weights of dimension 20 of the neural array are plotted against the weights of dimension 30 of the array. The
Figure 2.4 Kohonen network training on 60-d speech vectors
same dimensions of the input data vectors are plotted as dots. The absence of folding is evident, and is due to the small values of gain (0.004 and 0.001) being used to train the network. In this case, the input data vectors consisted of one hundred utterances outputted from the cochlear model.

2.6 Characteristic Trajectories

A 9*9 neural array was trained using ten digit sounds, "zero" to "nine". For each digit, the ten utterances collected were divided into eight training utterances and two test utterances. Hence, the network was trained using eighty utterances, while twenty utterances were available for testing purposes. A test utterance consists of a variable number of frames (each of length 16 ms), with each frame represented by a 60-dimensional vector. When this sequence of frames is presented to the trained network, neurons will fire in a certain sequence. It is possible that one neuron will fire for more than one frame. As a result, the number of distinct neurons which fire will be less than the number of frames in the utterance. By connecting the positions of the neurons which fire (i.e. the neurons most similar to the inputted sequence of frames) a characteristic trajectory may be plotted.

Examples of trajectories for the digit sounds "two" and "three" are presented in Figures 2.5 and 2.6. The test utterance for digit "two" consisted of 22 frame and for digit "three", 29 frames. The arrows indicate the direction in which the trajectory moves. One may
Figure 2.5 Trajectory of neuron firing for the digit "two" sound.
Figure 2.6 Trajectory of neuron firing for the digit "three" sound
observe from the diagrams how these digits can be distinguished by the shapes of their trajectories. For more details on results recorded during simulations refer to chapter 6.

When the same word is spoken more than once, even by the same speaker, the speaking rate varies. This causes problems in speech recognition systems, and is normally eliminated by using linear/dynamic time warping techniques.

The feature of time warp invariance may be observed by comparing the trajectory of Figure 2.7(a) with that of Figure 2.7(b). The trajectories were produced by two different utterances of the digit "one". The first utterance was 22 frames long, while the second was 14 frames long. However, the network produces similar trajectories for the two utterances, by eliminating the differences in speaking rate (note that one utterance is almost twice as long as the other). This property was used when preparing inputs to the second layer of the neural network.

2.7 The Second Kohonen Neural Layer

A second neural array may be used to accept inputs from the first array (See Figure 2.8). The inputs to the second layer are created by concatenating the sequence of "fired" neurons on the first layer. This then produces a P-dimensional column vector. In order that these inputs each have the same dimensionality it is necessary to pad the unused remaining entries of the vector with zero's. This
Figure 2.7 Illustration of time warp invariance using the digit "one"
Figure 2.8 The Kohonen two layer neural network using a cochlear model as a preprocessor.