Speech Recognition
and
Blackboard Expert Systems

Guy Marchand Loureiro

Submitted in partial fulfilment of the requirements for the degree of Masters of Science in the Department of Computer Science, University of Natal.
1992

Durban
1992

Preface

The work described in this thesis was carried out in the Department of Computer Science, University of Natal, Durban, between January 1990 and November 1992, under the supervision of Professor Alan G Sartori-Angus.

These studies represent original work by the author and have not been submitted in any form to another University. Where use has been made of the work of others it has been duly acknowledged in the text.

Acknowledgements

I am particularly grateful to my supervisor, Prof Alan G Sartori-Angus who allowed me the freedom to explore this vast field of research and for his helpful guidance.

Many thanks to Jane Meyerowitz who, towards the end of the project monitored my last strides, made numerous suggestions on the presentation and gave the sort of encouraging criticism which spurs one on.

To Stuart Melville for the great discussions and the loan of your library of books on artificial intelligence; and proofreadings of the AI related sections of this thesis - I am most indebted to you.

Within the department, my thanks go to Hilton Goldstein who assisted me with the building of the hardware preprocessing and to Alan Powell for the discussions and material on neural networks as well as fellow Masters students Andrew Deighton, Dave Carson, Carl Rautenbach and John du Preez, and Gary Nicholson and Mike Hayley of the 1991 Honours class, all of whom have been sources of reference and encouragement.

To Murray Small, thanks for introducing me to Vygotsky, cognitive psychology and a string of cognitive scientists involved in similar work but from a refreshingly different perspective.

To Roland Patterson-Jones for his contributions on Prolog and problems related to natural language processing (including the French lessons, *merci beaucoup*).

To my family and friends, especially those who gave up precious relaxing time to help with voice samples - thank you for your constant support and patience, I am ever grateful.

This thesis would not have been possible without the work done by Kari. I am sure she did not realise that two becoming one (as we did in the middle of this project) would mean sharing the weight of a Masters degree in a field she knew (knows) nothing about. For all the other things I should have done which you did, thank you!

Special thanks to the UND Electrical Engineering Department for access to their journals especially the IEEE on Acoustics, Speech, and Signal Processing which contains a wealth of new ideas on speech and signal processing monthly. In a similar vein, I am very grateful to the UND library staff for their assistance in retrieving information from near and wide, and especially to Kirsty Hoile for her assistance in finding material this end.

Finally, I wish to thank the Foundation for Research Development for the much needed financial assistance.

Abstract

Spoken language is used by people to communicate naturally with one another. A simplistic view of the communication process is as follows. Person A wishes to communicate an idea to person B. The idea, initiated in the mind/brain of person A is encoded into speech signals by means of the person A's speech production mechanism, the vocal apparata in the vocal tract. Various kinds of noise may interfere with the speech signals as they travel to person B. The resulting signal is captured by person B's speech receiving mechanism, the ear. It is then analysed and decoded into a meaningful message by the brain of person B.

This thesis concerns itself with the investigation of and attempt to automate the receiving and decoding of English sentences using a machine - that is to perform the task of person B in the above scenario using a computer. The aim is not only to produce a sequence of phonetic sounds, but to look at the problems of building in the 'mind of the machine', a picture of the meanings, intentions, absurdities and realities of the spoken message.

The various models, algorithms and techniques of speech recognition and speech understanding systems are examined. Speech signals are captured and digitised by hardware. The digital samples are analysed and the important distinguishing features of all speech sounds are identified. These are then used to classify speech sounds in subsequent spoken words. The way speech sounds are joined together to form syllables and words introduces difficult problems to the automatic recognition process. Speech sounds are blurred, overlapped or left out due to the effects of coarticulation. Finally, natural language processing issues, such as the importance of syntax (the structure) and semantics (the meaning) of sentences, are studied.

A system to control and unite all the above processing is considered. The blackboard expert system model of the widely reported HEARSAY-II speech recognition system is reviewed as the system with the best potential for the above tasks.

Contents

Chapter 1 - Introduction

> "It's April 1993 (sic), and you've just purchased a top-of-the-line personal computer from Big Apple. You open the box and begin to assemble it, but to your surprise, you don't see a keyboard, only a special pad and pen that resemble those in an executive portfolio. The box also contains clothing: a body suit, a pair of gloves, and a headband. You see no monitor, only a helmet and goggles. What's going on here? What you are about to experience is your first taste of natural computing: making computers interact with users in a human-like manner." [Cau92]

The above quote from **BYTE** magazine predicts the day (in the not too distant future) when computers will be transformed into intelligent human-like systems. The onus will be on the computer to understand the user's speech, gestures, mood and train of thought. At the same time, the computer will respond and react to the user not only via graphic information on the screen but also verbally in his (or her[1]) own language.

Speech is the natural means of communication between people and it therefore is desirable that machines should be able to understand speech and talk to the user. *(Automatic) speech recognition* is the term used to describe the assimilation of speech by a machine. *Speech synthesis*, although not within the scope of this thesis, is the process of getting the machine to generate speech sounds.

The dream of a natural interface with a machine is perhaps only the surface of a deeper goal - that of machine intelligence. Important theories in cognitive psychology maintain that man's capacity for speech separates him from all other living creatures in terms of intelligence. Vygotsky has shown that children outstrip the performance of apes in solving practical tasks because of their ability to "reconstruct their perceptions" with "the help of their speech" [Vyg78]. According to these theories, speech is the very mechanism which allows humans to build abstract interpretations and solutions to problems. Thus, speech might be the key to solving the problem of machine intelligence. These ideas are not the central focus of this thesis and yet they support the need for a holistic approach to speech recognition. That is, speech recognition should not simply be a case of identifying the sounds of word utterances but should rather involve developing a sophisticated system that

[1] The masculine is adopted without prejudice henceforth.

can predict, associate, validate, think and understand. Chapters 8 and 9 on the higher levels of speech processing highlight these necessary components in speech recognition.

Speech recognition has not had a history of instant successes. The problem has been around since the early '50s but researchers were quickly disillusioned by the complexity of their subject. As a result, the goal of a speaking, understanding machine was reduced to make the goals more attainable. Several less sophisticated classes of speech recognition systems have emerged, each extending the scope of the problem or shifting the perspective or approach of existing classes.

There are two aspects which determine the way speech recognition systems are referred to: firstly, the *unit of recognition* used and secondly, the *scope of the recognition problem* tackled.

The unit of recognition refers to the size of the sound used to identify speech. People perceive several units of sound namely phonemes[1], syllables and words. Speech recognition systems using these and other 'in between' units of recognition, have been developed. The word is the largest and the phoneme the smallest. The unit of recognition is often included in the name describing the type of speech recognition system. For example, *word recognition systems* have the *word* as their unit of recognition.

The second way speech recognition systems are identified is by the scope of speech they attempt to recognise.

Isolated speech recognition (ISR) systems identify single word utterances from a dictionary of possible words.

Continuous speech recognition[2] *(CSR) systems* involve recognising phrases or sentences of words spoken in continuous speech. Word spotting systems are closely related to CSR. They attempt to detect a single word in a string of continuous speech.

[1] Phonemes are defined and discussed in chapter 2.

[2] CSR systems with a *word* unit of recognition are called *connected word recognition systems.*

Speech understanding is the ultimate goal of speech recognition research in that it attempts to integrate higher levels of speech processing and language issues into CSR systems. These issues include:

- the prosodics (stress, rhythm and pitch) of speech sounds
- the syntax or structure of a language
- the semantics or meanings of words and the logic and truth value of sentences
- the pragmatics or context of ideas in the sentence(s) and the intentions (vs. literal meaning) of the speaker's words.

Speech understanding draws on ideas from artificial intelligence and expert systems are used to attempt to embody the multiple knowledge sources.

Two topics related to speech recognition but not covered in this thesis, are *speaker (or voice) recognition* [Ata76] [Sam76] and *speaker verification (or authentication)* [Ros76]. Both deal with the problem of identifying speakers from the sound of their voice. (For the differences between speaker recognition and verification see [Ros76]. A summary of papers including [Ata76] [Sam76] and [Ros76] on these topics is included in [Dix79]). A person's voice like his thumb-print, can be used to identify him because each person's vocal apparatus is different from anyone else's. People have the ability to perform voice recognition which is apparent when they answer telephones. Voice recognition can be considered an important component of speech understanding from the point of anticipating the subject-matter and context of a speaker and following the conversation of multiple speakers.

1.1 General Speech Recognition Process

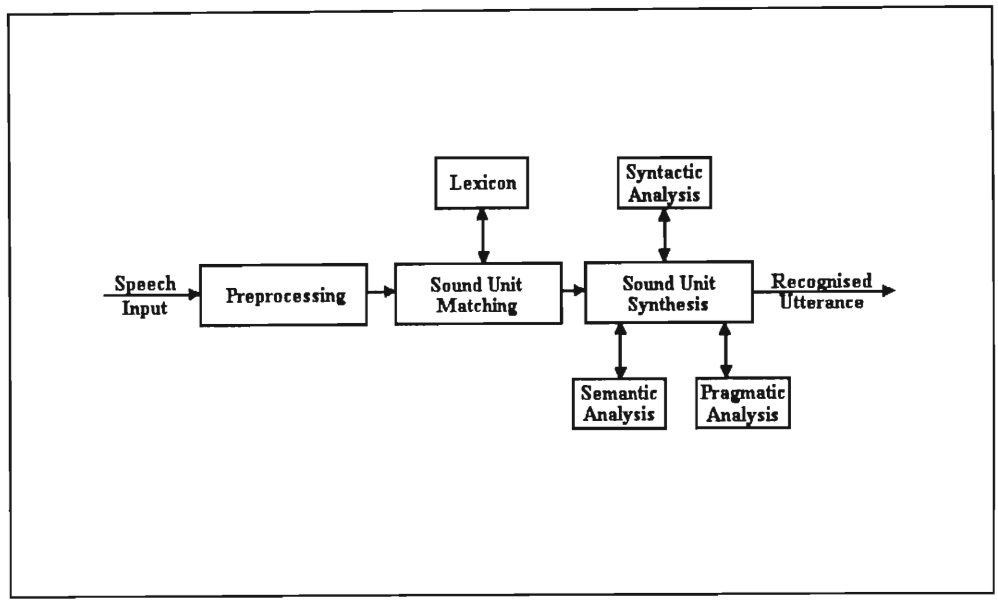

Figure 1.1 General Speech Processing System

Figure 1.1 shows the process adopted by most digital speech recognition systems. The speech input is captured by a microphone in the form of an analogue electronic signal. This signal is digitised and then preprocessed to extract a set of speech features characterising it as it varies in time (chapter 3). The speech features are then classified and labelled by matching them against a set of reference patterns in the lexicon (or dictionary). The unit of recognition must be specified at this stage. *Word* recognition systems are discussed in chapters 5 and 6 while *subword* (eg. phoneme) recognition systems are the focus of chapter 4. The lexicon of any speech recognition system is made up of reference patterns (chapter 4 and 5) or statistical models (chapter 6) representing the words in the system's vocabulary. All speech recognition systems will require a training stage to build their lexicons. Training is very tedious and thus ways of reducing this time consuming task are investigated (chapter 4). Testing the system involves matching input signal patterns which are statistically different from the set of training signals, against the reference patterns or models in the lexicon. The best matching reference pattern is chosen as the 'winning word'. The recognised units of speech can then be merged with neighbouring recognised speech units until words, phrases and eventually sentences are formed. The higher levels of speech processing (chapter 8) can be integrated with this merging process to restrict and guide it.

1.2 Broad Aims

This thesis examines the main approaches to speech recognition. It is holistic in approach, attempting to integrate the main ideas in speech recognition. Due to the size of the topic, certain areas are omitted or covered briefly. Where possible, references to further research are given. Several speech recognition systems have been implemented, the results of which are compared with those from similar systems found in the literature.

1.3 Chapter Outline of the Thesis

Chapter 1 - Introduction

The introduction defines the field of speech recognition and outlines the aims and objectives of this thesis.

Chapter 2 - Anatomy of the Speech Production and Perception Mechanisms

The anatomy of the speech production and perception mechanisms are investigated in order to parallel machine recognition and synthesis with the human model. In addition, a study of how sounds are produced provides useful insights into the recognition problem. Finally, the source-filter model [Fan60] is described as a powerful tool for modelling speech production. Coupled with statistical predictive analysis of speech, this model can be used to develop the all-important linear predictive coding (LPC) technique which has been the cornerstone of speech processing applications since the late '60s.

Chapter 3 - The Preprocessing Stage

This chapter deals with the problems in the first stage of the general speech recogniser (see figure 1.1) including:

- Filtering and amplification of the signal
- Sampling or converting the analogue speech signal to a digital form
- 'Short-time' characterisation of time varying speech signals
- Various preprocessing functions for extracting characteristic features from the speech signal in both the time- and frequency-domains.
- Linear predictive coding of the speech signals

Chapter 4 - Segmentation and Classification

Segmentation involves splitting the signal into regions of similar acoustic content. Classification is the process of determining to which particular sound classes the segments of the speech signal belong. Several techniques for both problems are presented in this chapter.

Chapter 5 - Template Matching Word Recognition

Template matching is the technique whereby an unknown input signal is matched against a set of reference words. Several algorithms for time aligning the input signal with the reference signal are presented. Both isolated and connected word recognition systems are investigated using this technique.

Chapter 6 - Hidden Markov Modelling

The notion of parametric modelling of speech using stochastic *hidden Markov models* is developed. Isolated and connected word recognition systems using this approach are studied. The chapter concludes with a comparison between template matching and hidden Markov modelling approaches.

Chapter 7 - Implementation

A description of the practical aspects of implementing several speech recognition systems are detailed in this chapter including:

- A template matching isolated word recognition system (chapter 5)
- An isolated word recognition system using hidden Markov modelling (chapter 6)
- A template matching connected word recognition system (chapter 5)
- A continuous speech recognition system based on the classification ideas presented in chapter 4.

A program for graphing the speech signal and various features of the signal is also described along with a segmentation algorithm based on the ZAPDASH segmenter discussed in chapter 4.

The results of the systems implemented in this project are presented and compared with those of similar systems found in the literature.

Chapter 8 - Higher Levels of Speech Processing

This chapter examines the higher levels of speech and language processing with a view to incorporating them into the speech recognition systems.

Chapter 9 - Expert Systems

Expert systems are studied as the means for incorporating the concepts of chapter 8 into a speech understanding system. In particular, the blackboard expert system model is seen to have the greatest potential for expressing and integrating the higher levels of speech and language knowledge into a speech understanding system. The HEARSAY-II blackboard speech understanding system is discussed in detail.

Chapter 10 - Conclusions

General conclusions are drawn about the current state and future of speech recognition.

Chapter 10 - Conclusions

In this thesis, several aspects of speech recognition have been examined:

- the way people talk and listen, the way linguists classify sounds and the theories related to sound production, quality of voice and problems associated with speech processing (chapter 2)

- the 'short-time' analysis of the speech signal in order to compress the large number of speech digital samples into a condensed form (several forms were mentioned especially using spectral information and linear predictive coding) which carries all the important information of the signal (chapter 3)

- presegmentation of the signal into speech events eg. sounds or words, was found to be difficult while several techniques for classifying (labelling) the speech sounds in the signal were studied as a front-end to a *continuous speech recognition* system (chapters 4 and 7)

- *word recognition* techniques using the template-matching and hidden Markov modelling approaches (chapters 5, 6 and 7)

- an integrated system for embodying natural language processing into the speech recognition process using the blackboard model (chapters 8 and 9)

The aim of the thesis was to expose the most important aspects of speech recognition and in particular, to include discussion on the natural language processing aspect of speech and how it might be integrated in speech recognition using the blackboard expert system. It seems likely that the current trend of speech recognition research will bend towards NLP as the low-level pattern recognition problems are exhausted. In this regard, there are areas in speech recognition research which need considerable development (also see [Moo92]):

- prosodics or the contribution of rhythm, stress and timing to the speech message and the speaker's mood and intentions
- natural language acquisition theories eg. [Ree78] can possibly give way to more powerful NLP systems with the ability to learn a language rather than starting with the data structures which researchers think model human cognitive processes

- new technology for handling large speech data, high processing speeds but more importantly, concurrent processing

Small-medium vocabulary (<200 words) word recognisers have long been achieving excellent recognition results. It seems that this technology has finally become available, with user-friendly response times. Earlier on in the year, Apple MacIntosh announced that their workstations would come with this technology and more recently (November '92) IBM announced that they would be introducing speech recognition technology with their mid-range systems.

As far as speech understanding is concerned, the problem is so vast that it will probably be many years before a reasonably 'intelligent' system is produced which performs as a human should even if in a restricted problem domain.

References

[Abe74] Abercrombie, D, Elements of General Phonetics, Edinburgh University Press Edinburgh, (1974).

[Aho73] Aho, A V & Ullman, J D, The Theory of Parsing, Translation, and Compiling, Vol II: Compiling, Prentice-Hall Inc, Englewood Cliffs, New Jersey, (1973).

[Ata71] Atal, B S & Hanauer, S L, Speech Analysis and Synthesis by Linear Prediction of the Speech Wave, *Journal of the Acoustic Society of America*, Vol 50, No 2 (Part 2), August (1971), p 637-655.

[Ata76] Atal, B S, Automatic Recognition of Speakers from their Voices, *Proceedings of the IEEE*, Vol 64, April (1976), p 460-475.

[Ata85] Atal, B S, Computer Speech Processing, Fallside, F & Woods, W A (eds), Prentice-Hall International, London, (1985).

[Bla85] Bladon, R A W, Acoustic Phonetics, Auditory Phonetics, Speaker Sex and Speech Recognition : A Thread, Computer Speech Processing, Fallside, F & Woods, W A (eds), Prentice-Hall International, London, (1985).

[Bor87] Born, R (ed), Artificial Intelligence: The Case Against, Croom & Helm, London, (1987).

[Bra90] Bratko, I, Prolog Programming Language for Artificial Intelligence (2nd ed), Addison-Wesley, Reading, Massachusetts, (1990).

[Bro82] Brown, M K & Rabiner, L R, An Adaptive, Ordered, Graph Search Technique for Dynamic Time Warping for Isolated Word Regognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-30, No 4, August (1982), p 535-544.

[Bro91] Brown, M K, McGee, M A, Rabiner, L R & Wilpon, J G, Training Set Design for Connected Speech Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol 39, No 6, June (1991), p 1268-1281.

[Cau92] Caudill, M, Kinder, Gentler Computing, *Byte (UK)*, McGraw-Hill Inc. Peterborough, April (1992).

[Cha85] Charniak, E & McDermott, D V, Introduction to Artificial Intelligence, Addison-Wesley, Reading, Massachusetts, (1985).

[Cho68] Chomsky, N & Halle, M, The Sound Pattern of English, Harper & Row, New York, (1968).

[Chu47] Churchill, W S, My Early Life, Odhams Press Ltd, London, (1947).

[Coh82] Cohen, P R & Feigenbaum, E A (eds), Handbook of AI Vol 3, Addison-Wesley, Reading, Massachusetts, (1982).

[Coh86] Cohen, D I A, Introduction to Computer Theory, John Wiley & Sons, Toronto, (1986).

[DeM90] De Mori, R, Palakal, M J & Cosi P, Perceptual Models for Automatic Speech Recognition, Yovits, M C (ed) *Advances in Computers* Vol 31, Academic Press Inc, Boston, (1990), p 100-173.

[Dev82] Devijver, P A & Kittler, J, Pattern Recognition: A Statistical Approach, Prentice-Hall International, London, (1982).

[Dix79] Dixon, N R & Martin, T B, (eds), Automatic Speech and Speaker Recognition, Proceedings of the IEEE, John Wiley & Sons, New York, (1979).

[DuP91] Du Preez, J A, Modelling Durations in Hidden Markov Models with Application to Word Spotting, *ComSig Proceedings*, IEEE, August (1991).

[Eng88] Engelmore, R & Morgan, T (eds), Blackboard Systems, Addison-Wesley, Reading, Massachusetts, (1988).

[Erm76] Erman, L D, Fennell, R D, Lesser, V R & Reddy, D R, System Organizations for Speech Understanding: Implications of Network and Multiprocessor Computer Architectures for AI, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol-C 25, No 4, April (1976), p 414-421.

[Fal85] Fallside, F & Woods, W A (eds), Computer Speech Processing, Prentice-Hall International (UK), London, (1985).

[Fan60] Fant, G, Acoustic Theory of Speech Production, Mouton, The Hague, (1960).

[For84] Forsyth, R (ed), Expert Systems - Principles and Case Studies, Chapman and Hall, London, (1984).

[For89] Forsyth, R (ed), Machine Learning: Principles and Techniques, Chapman and Hall Computing, London, (1989).

[Fur80] Furui, S, A Training Procedure for Isolated Word Recognition Systems, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-28, No 2, April (1980), p 129-135.

[Gao90a] Gao, Y Q, Chen, Y B & Huang, T Y, A New Method for Estimation of Hidden Markov Model Parameters, 10th International Conference on Pattern Recognition, Vol II, IEEE, Computer Society Press, Washington, June (1990).

[Gao90b] Gao, Y Q, Chen Y B & Huang T Y, The Speech Recognition System for All the Chinese Syllables Using Hidden Markov Model, 10th International Conference on Pattern Recognition, IEEE, Computer Society Press, Washington, (1990).

[Gaz89a] Gazdar, G & Mellish, C, Natural Language Processing in Lisp: An Introduction to Computational Linguistics, Addison-Wesley, Reading, Massachusetts, (1989).

[Gaz89b] Gazdar, G & Mellish, C, Natural Language Processing in Prolog: An Introduction to Computational Linguistics, Addison-Wesley, Reading, Massachusetts, 1989.

[Gre78] Greene, M C L, The Voice and its Disorders, *Pitman Medical*, London, p 1-45, (1978).

[Har89] Hart, A, Machine induction as a form of knowledge acquisition in knowledge engineering, Machine Learning, Forsyth, R (ed), Chapman and Hall, London, (1989).

[Hay83] Hayes-Roth, F, Waterman, D A & Lenat, D B (eds), Building Expert Systems, Addison-Wesley, Reading, Massachusetts, (1983).

[Hur88] Hurford, J R & Heasley, B, Semantics: A Coursebook, Cambridge University Press, Cambridge, (1988).

[Ios80] Iosifescu, M, Finite Markov Processes and their Applications, John Wiley and Sons, Bucuresti, (1980).

[Ita75] Itakura, F, Minimum Prediction Residual Principle Applied to Speech Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-23, February (1975), p 62-72.

[Jac89] Jackson, L B, Digital Filters and Signal Processing (2nd ed), Kluwer Academic Publishers, Boston, (1989).

[Jak52] Jakobson, R, Fant, G & Halle, M, Preliminaries to Speech Analysis, MIT Press, Cambridge, Massachusetts, (1952).

[Jel85] Jelinek, F, The Development of an Experimental Discrete Dictation Recognizer, *Proceedings of the IEEE*, Vol 73, No 11, November (1985).

[Joh76] Johnson, D E, Introduction to Filter Theory, Prentice-Hall Inc, Englewood Cliffs, New Jersey, (1976).

[Joh88] Johnson, L & Keravnou, E T, Expert Systems and Architectures, Kogan Page Ltd, London, (1988).

[Jua85a] Juang, B-H, Maximum-Likelihood Estimation for Mixture Multivariate Stochastic Observations of Markov Chains, *AT&T Technical Journal*, Vol 64, No 6, July-August (1985), p 1235-1249.

[Jua85b] Juang, B-H & Rabiner, L R, Mixture Autoregressive Hidden Markov Models for Speech Signals, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-33 No.6, December (1985), p 1404-1413.

[Jua86] Juang, B-H, Levinson, S E & Sondhi M, Maximum Likelihood Estimation for Multivariate Mixture Observations of Markov Chains, *IEEE Transactions on Information Theory*, Vol IT-32, No 2, March (1986).

[Kla77] Klatt, D H, A review of the ARPA Speech Understanding Project, *Journal of the Accoustical Society of America*, Vol 62, December (1977), p 1345-1366.

[Kit82] Kittler, J, Fu K S & Pau L F (eds), Pattern Recognition Theory and Applications, D Riedel Publishing Co, Dordrecht, Holland, (1982).

[Kni90] Knight, K, Connectionist ideas and algorithms, *Communications of the ACM*, Vol 33, No 11, New York Association for Computing Machinery Inc. New York, November (1990).

[Koh88] Kohonen, T, The "Neural" Phonetic Typewriter, *Computer*, Vol 21, No 3, March (1988), p 11-22.

[Koh89] Kohonen, T, Speech Recognition based on topology-preserving neural maps, Neural Computing Architectures : The Design of Brain-like Machines, Aleksander, I (ed), North Oxford Academic, London, (1989).

[Koh90] Kohonen, T, The Self-Organizing Map, *Proceedings of the IEEE*, Vol 78, No 9, September (1990), p 1464-1477.

[Koh91][1] Kohonen, T, Kangas, J, Laaksonen, J & Torkkola, K, The Learning Vector Quantization Program Package, Version 2.0 (January 1992), Helsinki University of Technology, Rakentajanaukio, Finland.

[Lad85] Ladefoged, P, The Phonetic Basis for Computer Speech Processing, Computer Speech Processing, Fallside, F & Woods, W A (eds), Prentice-Hall International (UK), London, (1985).

[Lan79] Lancaster, D, Active Filter Cookbook, Howard W Sams & Co Inc, Indianapolis, (1979).

[1] Can be FTP'ed from anonymous login on the machine: "cochlea.hut.fi" (or INTERNET address: 130.233.168.48). Use your own e-mail address as password and obtain files from the /pub/lvq_pak directory.

[Lee90a] Lee, K-F, Hon, H-W & Reddy, R, An Overview of the SPHINX Speech Recognition System, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol 38, No 1, January (1990), p 35-45.

[Lee90b] Lee, K-F, Context-Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition, *IEEE Transactions on Acoustics, Speech, Signal Processing*, Vol 38, No 4, April (1990), p 599-609.

[Lee92] Leedham, G, Pattern Recognition and its Application to Speech, Rowden, C (ed), McGraw-Hill International (UK), Maidenhead, (1992).

[Les75] Lesser, V R, Fennell, R D, Erman, L D & Reddy, D R, Organisation of the HEARSAY-II Speech Understanding System, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-23, February (1975), p 11-24.

[Lev85a] Levinson, S E, A Unified Theory of Composite Pattern Analysis for Automatic Speech Recognition, Computer Speech Processing, Fallside, F & Woods, W A (eds), Prentice-Hall International (UK), London, (1985).

[Lev85b] Levinson, S E, Structural Methods in Automatic Speech Recognition *Proceedings of the IEEE*, Vol 73, No 11, November (1985), p 1625-1647.

[Lip82] Liporace, L A, Maximum Likelihood Estimation for Multivariate Observations of Markov Sources, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol IT-28, No 5, September (1982), p 729-734.

[Low77] Lowerre, B T, Dynamic Speaker Adaption in the Harpy Speech Recognition System, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, May (1977), p 788-790.

[Mak75] Makhoul, J, Linear Prediction : A Tutorial Review, *Proceedings of the IEEE*, Vol 63, No 4, April (1975), p 561-580.

[Mak85] Makhoul, J, Roucos, S & Gish, H, Vector Quantization in Speech Coding, *Proceedings of the IEEE*, Vol 73, No 11, November (1985), p 1551-1585.

[McT87] McTear, M, The Articulate Computer, Basil Blackwell, Oxford, (1987).

[Mer89] Merritt, D, Building Expert Systems in Prolog, Springer-Verlag, New York, (1989).

[Moo92] Moore, R, Recognition - the Stochastic Modelling Approach, Speech Processing, Rowden, C (ed), McGraw-Hill International (UK), Maidenhead, (1992).

[Mye80] Myers, C, Rabiner, L R & Rosenberg, A E, Performance Tradeoffs in Dynamic Time Warping for Isolated Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-28, No 6, December (1980), p 623-635.

[Mye81] Myers, C S & Rabiner, L R, A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol Assp-29, No 2, April (1981), p 284-297.

[Ney84] Ney, H, The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-32, No 2, April (1984), p 263-271.

[Ney91] Ney, H, Dynamic Programming Parsing for Context-Free Grammars in Continuous Speech Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol 39, No 2, February (1991), p 336-340.

[Nie81] Niemann, H, Pattern Analysis, Springer-Verlag, Berlin, (1981).

[Opp90] Oppenheimer, A V & Schafer, R W, Discrete-Time Signal Processing, Prentice-Hall Inc, Englewood Cliffs, New Jersey, (1990).

[Pal82] Pal, S K, Fuzzy Set Theoretic Approach : A Tool for Speech and Image Recognition, Pattern Recognition Theory and Applications, Kittler, J, Fu, K S & Pau, L F (eds), D Reidel Publishing Company, Dordrecht: Holland, (1982).

[Pal86] Pal, S K & Majumder, D K, Fuzzy Mathematical Approach to Pattern Recognition, John Wiley & Sons, New York, (1986).

[Pan85] Pan, K-C, Soong, F K & Rabiner, L R, A Vector-Quantization-Based Preprocessor for Speaker-Independent Isolated Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-33, No 3, June (1985), p 547-559.

[Par86] Parsons, T, Voice and Speech Processing, McGraw-Hill Inc, New York, (1986).

[Pas82] Passingham, R, The Human Primate, W H Freeman & Co, San Francisco, (1982).

[Pit90] Pitchers, R C, A Comparative Study of Various Speech Recognition Techniques, Master of Science in Engineering, Department of Electronic Engineering, University of Natal, Durban, (1990).

[Pou83] Poulton, A S, Microcomputer Speech Synthesis and Recognition, Sigma Technical Press, Wilmslow, Cheshire, UK, (1983).

[Pri89] Priemer, R, Introductory Signal Processing, World Scientific, Singapore, (1989).

[Qui87] Quinlan, J R, Compton, P J, Horn, K A & Lazarus, L, Inductive Knowledge Acquisition : A Case Study, Applications of Expert Systems, Quinlan, J R (ed), Turing Institute Press in association with Addison Wesley, Sydney, (1987).

[Rab76] Rabiner, L R & Sambur, M R, Some Preliminary Experiments in the Recognition of Connected Digits, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-24, April (1976), p 170-182.

[Rab79] Rabiner, L R, Speaker-Independent Isolated Word Recognition for a Moderate Size (54 Word) Vocabulary, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-27, No 6, December (1979), p 583-587.

[Rab80] Rabiner, L R & Schmidt, C E, Application of Dynamic Time Warping to Connected Digit Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-28, No 4, August (1980), p 377-388.

[Rab84] Rabiner, L R, Wilpon, J G, Quinn, A M & Terrace, S G, On the Application of Embedded Digit Training to Speaker Independent Connected Digit Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32, No 2, April (1984), p 272-279.

[Rab85a] Rabiner, L R & Levinson, S E, A Speaker-Independent, Syntax-Directed, Connected Word Recognition System Based on Hidden Markov Models and Level Building, *IEEE Transactions on Acoustics, Speech, and Signal Processsing*, ASSP-33, No 3, June (1985), p 561-573.

[Rab85b] Rabiner, L R, Juang, B-H, Levinson, S E & M M, Sondhi, Recognition of Isolated Digits Using Hidden Markov Models With Continuous Mixture Densities, *AT&T Technical Journal*, Vol 64, July-August (1985), p 1211-1233.

[Rab89] Rabiner, L R, A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, *Proceedings of the IEEE*, Vol 77, No 2, February (1989), p 257-286.

[Ree78] Reeker, L H, Computational Study of Language Acquisition, Rubinoff, M & Yovits, M C (eds), *Advances in Computers*, Vol 15, (1978), p 181-237.

[Red76] Reddy, D R, Speech Recognition by Machine: A Review, *Proceedings of the IEEE*, Vol 64, April (1976), p 501-531.

[Rey83] Reynolds, A G & Flagg, P W, Cognitive Psychology, Little, Brown & Co. Boston, (1983).

[Ric91] Rich, E & Knight, K, Artificial Intelligence, McGraw-Hill Inc, New York, (1991).

[Rob87] Roberts, R A & Mullis, C T, Digital Signal Processing, Addison-Wesley, Reading, Massachusetts, (1987).

[Ros76] Rosenberg, A E, Automatic Speaker Verification, *Proceedings of the IEEE*, Vol 64, April (1976), p 475-487.

[Row92a] Rowden, C, Analysis, Speech Processing, Rowden, C (ed), McGraw-Hill International (UK), Maidenhead, (1992).

[Row92b] Rowden, C & Hall, S, Parametric Coding of Speech, Speech Processing, Rowden, C (ed), McGraw-Hill International (UK), Maidenhead, (1992).

[Rum87] Rumelhart, D.E. & McClelland, J.L., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol 1 & 2, The Massachusetts Institute of Technology, Cambridge, Massachusetts, (1987).

[Sak78] Sakoe, H & Chiba, S, Dynamic Programming Algorithm Optimization for Spoken Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-26, February (1978), p 43-49.

[Sak79] Sakoe, H, Two-Level DP-Matching - A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-27, No 26, December (1979), p 588-595.

[Sam76] Sambur, M R, Speaker Recognition using Orthogonal Linear Prediction, *IEEE Transactions on Acoustics, Speech, Signal Processing*, Vol ASSP-24, August (1976), p 283-289.

[Sch75] Schafer, R W & Rabiner, L R, Digital Representation of Speech Signals in *Proceedings of the IEEE*, Vol 63, No 4, April (1975), p 662-677.

[Sch77] Schank, R C & Abelson, R P, Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures, Addison-Wesley, Reading, Massachusetts, (1977).

[Sch77] Schank, R C, Dynamic Memory and Theory of Reminding and Learning in Computers and People, Addison-Wesley, Reading, Massachusetts, (1982).

[Sco1889] Scott, E & Liddle H, A Greek-English Lexicon, Longhams, Oxford, 1889.

[Tap78] Tappert, C C & Subrata, K D, Memory and Time Improvements in a Dynamic Programming Algorithm for Matching Speech Patterns, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-26, No 6, December (1978), p 583-586.

[Toh87] Tohkura, Y, A Weighted Cepstral Distance Measure for Speech Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-35, No 10, October (1987), p 1414-1422.

[Van91] Van Wyk, G, Nel, I H & Coetzer, W, A Real Time, Speaker Independent, Speech Recognition System, *ComSig Proceedings*, IEEE, August (1991).

[Vas85] Vassiere, J, Speech Recognition: A Tutorial, Computer Speech Processing, Fallside, F & Woods, W A (eds), Prentice-Hall International (UK), London, (1985)

[Vee88] Veeneman, D E, Speech Signal Analysis, Signal Processing Handbook, Chen, C H (ed), Marcel Dekker Inc, New York, (1988).

[Vic87] Vich, R, Homomorphic Vector Quantisation, *Electronic Letters*, 23(11), May (1987), p 561-562.

[Vyg78] Vygotsky, L S, Mind in Society, Cole, M, John-Steiner, V, Scribner, S & Souberman, E (eds), Harvard University Press, Massachusetts, (1978).

[Wai88] Waibel, A, Prosody and Speech Recognition, Pitman, London, (1988).

[Wai89] Waibel, A, Hanazawa, T, Hinton, G, Shikano, K & Lang, K J, Phoneme Recognition Using Time-Delay Neural Networks, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol 37, No 3, March (1989).

[Wat92] Waters, G, Speech Production and Perception, Speech Processing, Rowden, C (ed), McGraw-Hill International (UK), Maidenhead, (1992).

[Wei84] Weiss, S.M. & Kulikowski, C A, A Practical Guide to Designing Expert Systems, Chapman and Hall Ltd, London, 1984.

[Whi76a] White, G M & Neely, R B, Speech Recognition Experiments with Linear Prediction, Bandpass Filtering, and Dynamic Programming, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-24, April (1976), p 183-188.

[Whi76b] White, G M, Speech Recognition : A tutorial Overview, *Computer*, Vol 9, No 5, May (1976), p 40-53.

[Wil83] Wilensky, R, Planning and Understanding : A Computational Approach to Human Reasoning, Addison-Wesley, Reading, Massachusetts, (1983).

[Wil85] Wilpon, J G & Rabiner, L R, A Modified *K*-Means Clustering Algorithm for Use in Isolated Work(sic) Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-33, No 3, June (1985), p 587-594.

[Wil90] Wilpon, J G, Rabiner, L R, Chin-Hui, L & Goldman, E R, Automatic Recognition of Keywords in Unconstrained Speech using Hidden Marcov Models, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol 38, No 11, November (1990), p 1870-1878.

[Win84] Winston, P H, Artificial Intelligence (2nd ed), Addison-Wesley, Reading, Massachusetts, (1984).

[Wol79] Wolf, J J & Woods, W A, The HWIM Speech Understanding System, Automatic Speech and Speaker Recognition, Dixon, N R & Martin, T B (eds), IEEE Press, New York, (1979).

[Woo75] Woods, W A, Motivation and Overview of SPEECHLIS : An Experimental Prototype for Speech Understanding Research, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-23, p 2-10, February (1975).

[Woo85] Woods, W A, Language Processing for Speech Understanding, Computer Speech Processing, Fallside, F A & Woods, W A (eds), Prentice-Hall International, Englewood Cliffs, New Jersey, (1985).

[Zad88] Zadeh, L A, Fuzzy Logic, *Computer*, April (1988), p 83-93.

[Zue85] Zue, V W, The Use of Speech Knowledge in Automatic Speech Recognition, *Proceedings of the IEEE*, Vol 73, No 11, November, (1985).

Chapter 2 - Speech Production and Perception

2.1 Introduction

In this chapter, an overview of the anatomy of the human vocal and hearing mechanisms is presented, followed by an examination of the production and perception of the speech sounds.

A study of the speech production mechanism shows not only how people speak but helps ascertain the problems and limitations as well as the potential of speech sounds, based on the anatomical constraints. This study provides useful insights into the acoustic nature of speech sounds.

The speech production mechanism is divided into three main categories:

- the lungs - the power supply to the system
- the larynx - the main source of excitation
- the vocal tract - the modulator and filter of the signal

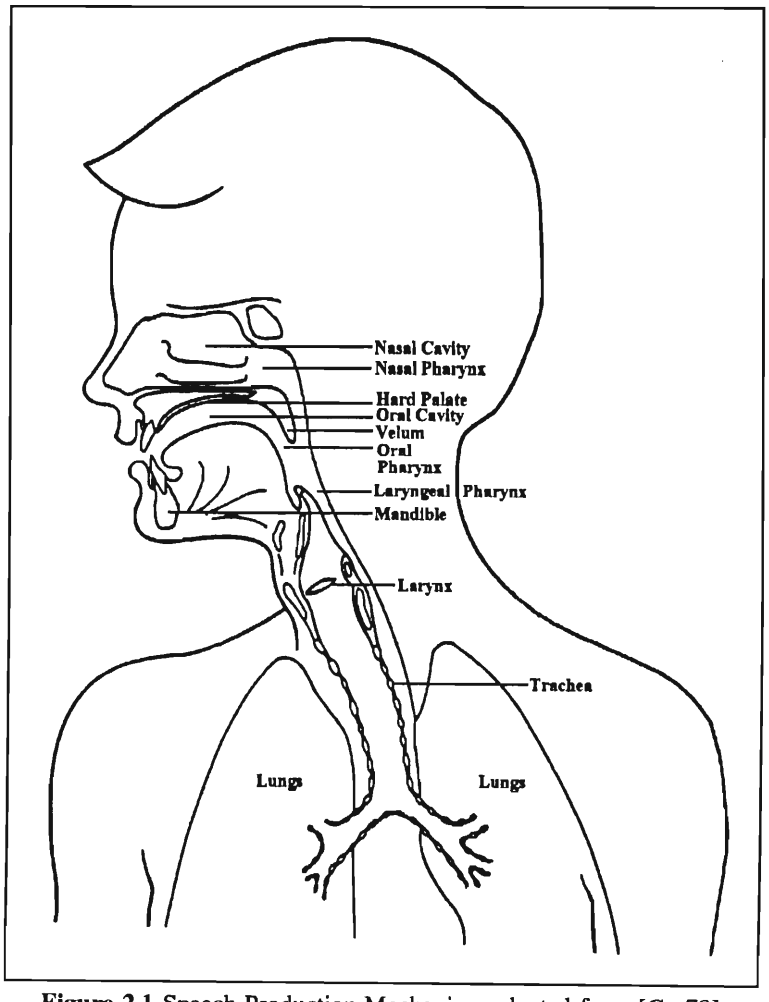

Figure 2.1 Speech Production Mechanism adapted from [Gre78]

2.2 Speech Production Mechanism

2.2.1 Lungs

The speech production mechanism is driven or powered by air pressure generated when the lungs are compressed. Generally the lungs do not contribute to the production of sound except for pulmonic bursts of air in the form of gasps and coughs.

The primary function of the lungs is to provide a surface for oxygen to enter the bloodstream. When people speak, they use air from the lungs thus competing directly with the continuous inward-outward flow of air in breathing. People overcome this conflict by speaking in "breath groups", which entails inhaling in between groups of sounds - as at the end of words, phrases and sentences - and exhaling while speaking. The competition between breathing and speaking is most evident when, for example, one struggles to breathe after strenuous exercise.

The sound level or loudness of speech is predominantly determined by the force of the air emitted from the lungs.

2.2.2 Larynx

The larynx (or glottis) is a complicated organ, comprised of cartilages and muscles which manipulate and control the vocal chords, the primary source of excitation in the speech production mechanism. The vocal chords are set vibrating by airstreams from the lungs like the strings or reed of a musical instrument. This phenomenon is called voicing or phonation. The rate at which the vocal chords vibrate determines pitch - the faster the rate, the higher the pitch. The larynx consists of four major components: the cricoid cartilage, thyroid cartilage, arytenoid cartilages and the vocal chords or folds.

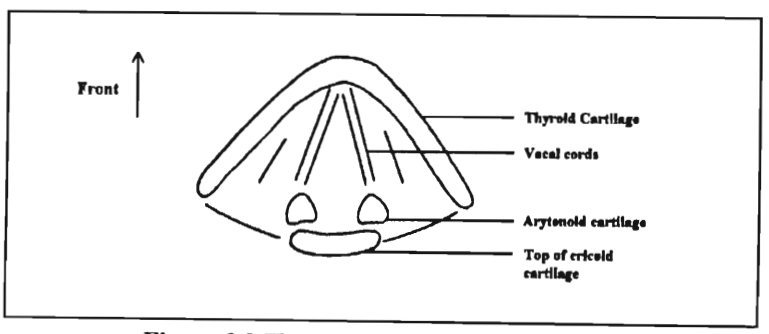

Figure 2.2 The Larynx adapted from [Par86]

The cricoid and thyroid cartilages are used mainly for supporting the vocal chords. The arytenoid cartilages control the positioning of the vocal chords, together or apart. In normal breathing, the vocal chords must be open; when they are closed, they provide an airtight plug between the lungs and the vocal passage. Some sounds are constructed by closing off and building air pressure behind the glottal folds before releasing a burst sound. The glottal stop (explained later), the "k"[1] sounds in *kick* and the "g" sounds in *gag* are made in this way.

2.2.3 Vocal Tract

This term encompasses all the components of the speech production system from the vocal chords to the lips including the nasal cavity. Figure 2.1 shows the location and identity of the main parts which are divided into the following five regions:

- the laryngeal pharynx (beneath the epiglottis)
- the oral pharynx (behind the tongue, between the epiglottis and the velum)
- the nasal pharynx (above the velum, rear end of the nasal cavity)
- the oral cavity (forward of the velum and bounded by lips, tongue and palate)
- the nasal cavity (above the palate and extending from the pharynx to the nostrils)

These regions act as the acoustic resonance chambers for speech sounds. By changing their shape, the sounds are modified and as a result the boundaries of these chambers are known as articulators. They can obstruct or alter the shape of the vocal tract because of their elastic nature. The primary articulator is the tongue which is the most flexible; but the lips, teeth, nasal cavity, epiglottis, lower jaw (mandible), velum (soft palate) and hard palate are also articulators.

The velum is an elastic muscle which controls the percentage of the vibrating air passing into the nasal cavity. Several English sounds are specifically nasalised like "n", "m" and "ng" sounds in *numbering*; but some people have difficulty controlling their velum during normal speech, resulting in 'nasalised speech'.

---

[1] Double quotes "" around an alphabetic character(s) denotes a speech sound(s). An example of the sound (highlighted by underlining) in an English word often accompanies it. The conventional phonetic or phonemic symbols denoting the atomic speech sounds are not used because they are generally not well-known and constant looking up in a table tends to frustrate the reader. Verbalising the word examples accompanying the sounds will help to follow the argument. See [Par86] for two phonetic alphabets.

2.3 Speech Production

Production of speech sounds can be divided into two parts: sound generation (excitation) and sound modification. Excitation occurs at the sound source whereas modification is carried out by the articulators.

2.3.1 Excitation

Excitation is achieved in several ways, namely:

- phonation (voicing)
- whispering
- frication
- compression
- vowel-like vibration in consonants
- nasality

2.3.2 Phonation (Voicing)

Phonation is the vibration or excitation of the vocal folds when compressed air from the lungs is forced through them. The stretched out folds are opened and closed by the arytenoid cartilages to produce bursts or pulses of air.

Due to the low pressure level of the air coming from the lungs at the end of words or sentences, the pulse rate is sometimes reduced to about half, or the pulses occur in pairs. This phenomenon known as 'vocal fry' [Par86], does not audibly effect the speech sounds but can be detected in amplitude-time graphs of the speech signal. An example of 'vocal fry' is shown in the circled region of graph in figure 2.3.

2.3.3 Whispering

When air from the lungs rushes through a triangular opening between the arytenoid cartilages instead of passing through the vocal chords, the turbulent, voiceless sound called whispering is created.

2.3.4 Frication

Frication is the sound created by turbulence when air passes through a constricted opening in the vocal tract. The nasal cavity is blocked and the vocal tract is partially obstructed by one or more of the articulators. An example of fricative sound is the "s" sound in *see-saw*,

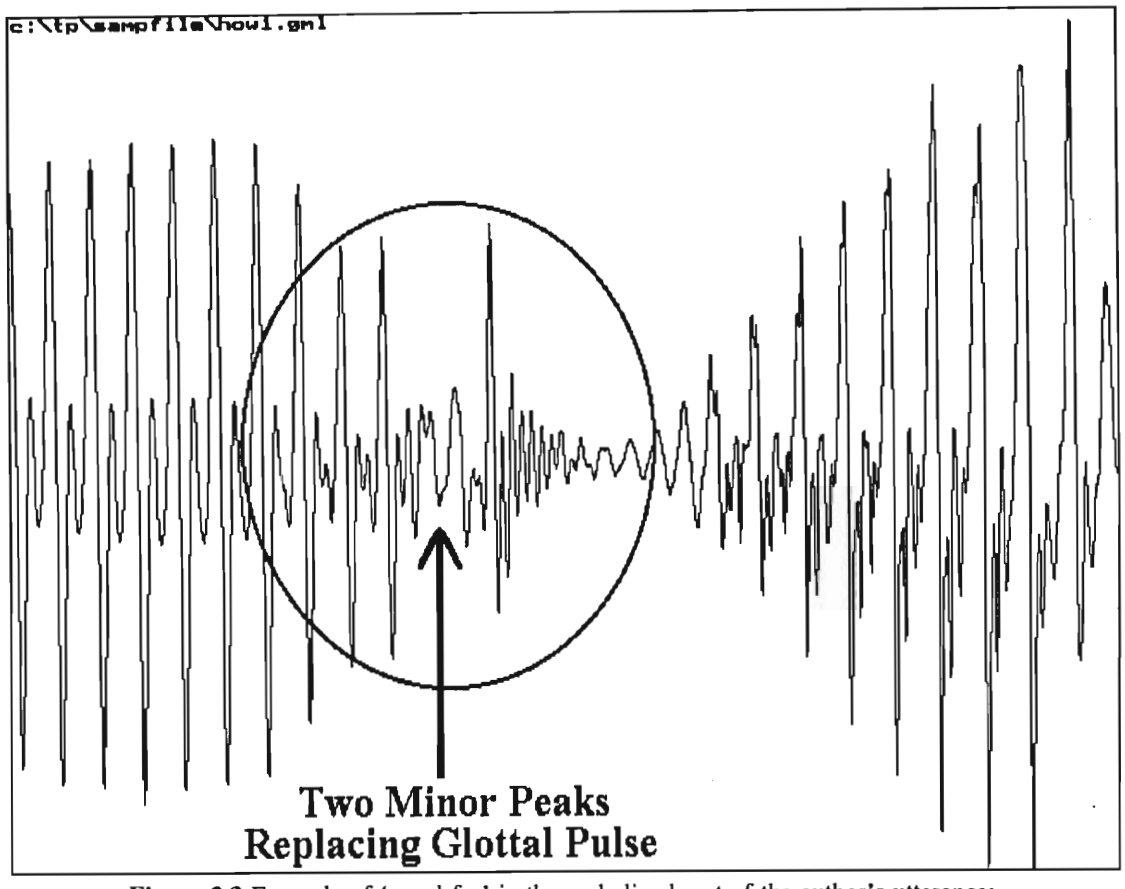

Figure 2.3 Example of 'vocal fry' in the underlined part of the author's utterance:
How many objects are on the table?

in which air is forced through a narrow opening between the hard palate and a slightly grooved tongue and then between the teeth. The "z" sound in *zoo* is produced in a similar manner to that of the "s". The major difference between them is the additional phonation in the "z" sound. Such phonated fricatives are called voiced fricatives. All voiceless fricatives, in English, have a voiced equivalent. A list of these is produced in Table 2.3 later in the chapter.

2.3.5 Compression

These sounds are made up of plosives (also called bursts) and stops. Plosives occur after a build up of air pressure behind some point of complete obstruction in the vocal tract. When the obstruction is removed, the sound that results from the little 'burst' or 'explosion' due to the release of pressure is termed plosive. The stop, on the other hand, occurs when the speaker suddenly closes the vocal tract at some point to form a complete obstruction. Stops are often followed by a plosive sound. The glottal stop is a good example to illustrate this. The British dialectal *be'er* and *bu'er* for *better* and *butter* exhibit the glottal stop (closure of the glottis) about where the apostrophe is positioned.

2.3.6 Nasality

Nasality is the result of partial or complete vibration of a voiced sound resonating in the nasal cavity. Three sounds are deliberately nasalised in English namely "m" in *sum*, "n" in *sun* and "ng" in *sung*.

2.3.7 Oscillations

These sounds are produced by relatively slow vibrations (±5-10 Hz) of the articulators (eg. tongue or lips). The "r" sound in some languages (eg. Afrikaans) and dialects sometimes produces such oscillations of the tongue.

2.4 Hearing and Perception Mechanisms

This section briefly describes the anatomy of the human hearing mechanism and the perception of speech sounds. The ear can be compared with the front-end module of speech processing systems (described in the beginning sections of chapter 3). Perception, on the other hand, is the analysis and interpretation of the sound signals, carried out by the brain. Speech recognition is an attempt to perceive speech sounds by means of a machine.

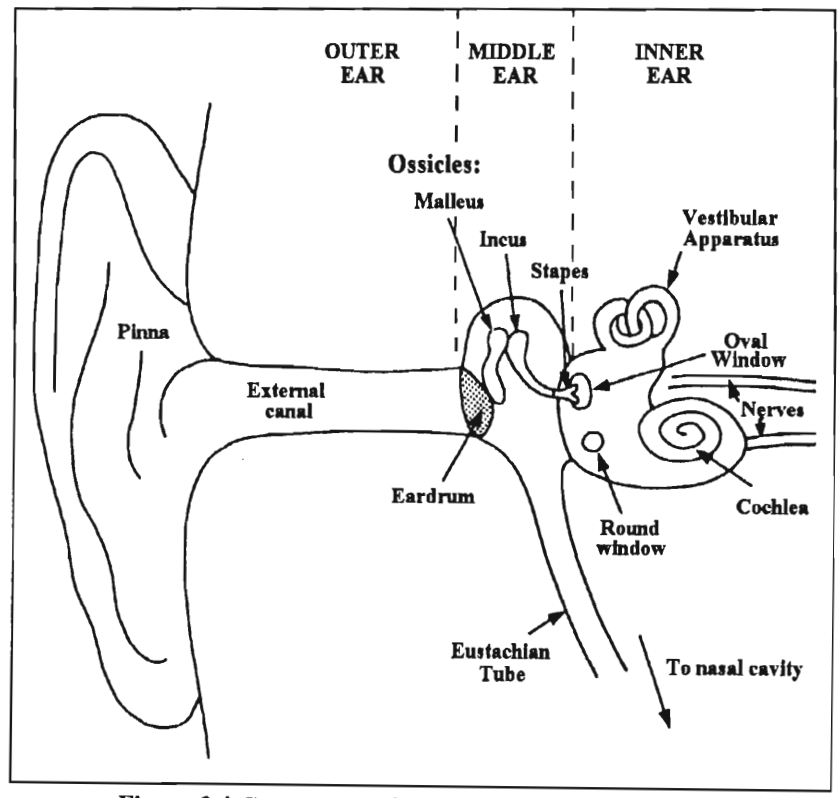

Figure 2.4 Components of the Ear adapted from [Wat92]

2.4.1 Physiology of the Ear

The human ear is divided into three parts (see figure 2.4):

- the outer ear
- the middle ear
- the inner ear

The outer ear is the sound receiver, passing sounds to the tympanic membrane (or eardrum) which separates the outer and middle ears. The tympanic membrane is very like the thin metal plate in a dynamic microphone. In both cases, acoustic pressure causes them to oscillate and this kinetic or mechanical energy is then translated into electrical impulses (microphone) and chemical, liquid impulses (inner ear).

The function of the middle ear is succinctly described by Waters:

> "The middle ear overcomes the air-to-liquid impedance to ensure that a detectable signal reaches the liquid-filled cochlea, to which it interfaces by two membranes called the oval and the round window." [Wat92]

The ossicles which translate the oscillations of the tympanic membrane to the oval window, serve to amplify the signal by about 30dB[1] [Wat92]. Again, one can see the parallel between this amplification and that usually found in the hardware preprocessor of speech systems.

The oval window interfaces the middle and inner ears. The vibrations of the oval window displace the liquid in the adjoining 'snail-like' cochlea to form 'waves'. It is believed that the cochlea performs a Fourier-type analysis of these 'waves' - the high frequencies not travelling as deep into the cochlea as the low frequencies [Wat92]. Minute hairs attached to the nerves of the cortex line the cochlea in order to extract the 'Fourier' information.

The cortex is an exceedingly complex biological structure. It is the interface between the ear and the brain through which nerve impulses carrying sound information pass.

---

[1] The decibel is a measure of the relative power or loudness of a signal. It is determined as ten times the common logarithm of the ratio of the powers of two signals. Thus in this case, a 30dB amplification means the amplified signal has 1000 times the power it had before amplification.

It is evident that the details of speech perception get progressively less clear moving inwards to the cortex and, as a result, speech systems start to look different from the human model, certainly as far as complexity is concerned.

2.4.2 Perception

Perception involves the interpretation of both the content message and the quality of speech signals. It also encompasses the distinctions between non-speech and speech sounds and one's ability to mask off sounds which are distracting or unimportant. Although extensive research is being undertaken, to date the brain remains a 'black-box' concealing the mysteries of human performance.

2.4.3 Loudness Perception

The loudness of sounds is not perceived the same for different frequencies. Figure 2.5 uses contours to illustrate similar loudness across the spectrum as perceived by the human ear. It can be seen that at very low and high frequencies one needs a greater intensity for sounds to be perceived. The human ear is most sensitive to loudness in the 250-5000Hz range and, as a result, many speech systems (including telephone) only preserve the frequencies in this range.

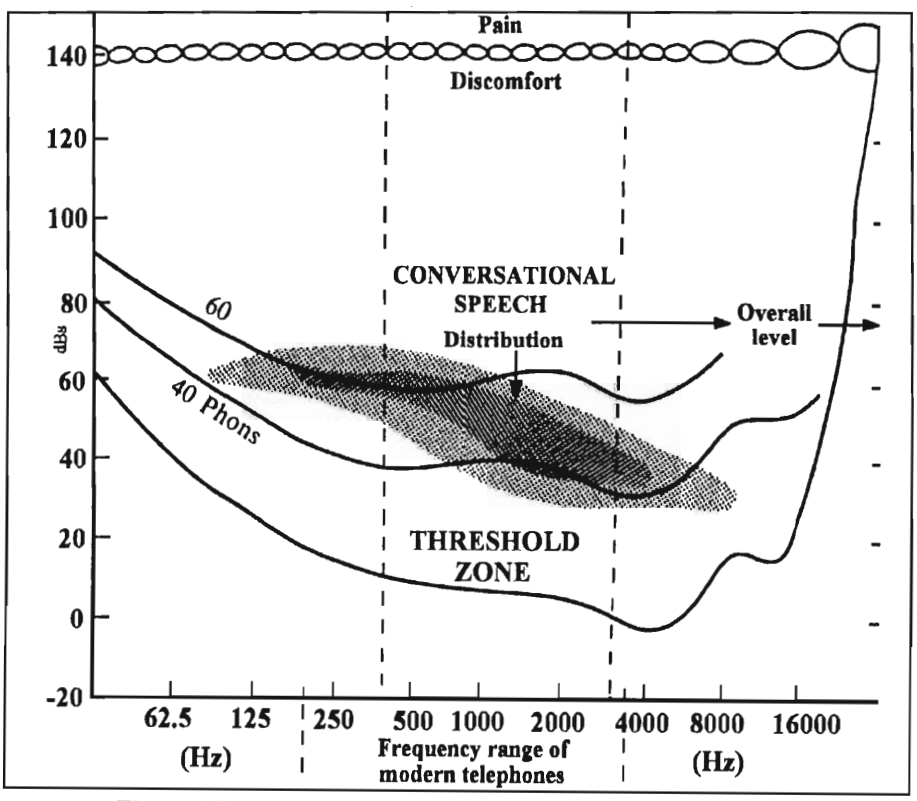

Figure 2.5 Loudness Perception Contours adapted from [Wat92]

2.4.4 Pitch Perception

Pitch is related to the rate at which the vocal chords vibrate (the fundamental frequency). The human ear has the ability to distinguish a 2Hz difference in the fundamental frequency of a sound with 60-70 SPL at about 1000Hz [Wat92]. How is pitch important in speech recognition and understanding? Pitch, along with loudness (intensity) and duration, is a primary ingredient for determining prosodic cues of speech. Combining these ingredients is a particular way makes it possible for humans to produce the effects known as rhythm and stress (or emphasis). The importance of these speech cues has been underplayed in speech recognition research and it seems essential that future systems address these issues. They are not discussed in any detail in this thesis but a comprehensive study has been undertaken by Waibel [Wai88], consolidating and expanding previous studies in this area.

The following statement is evidence of the role of pitch in determining the rhythm in speech.

> "A remarkable property of the human auditory system is that for some complex tones, even if the fundamental pitch is not present in the source signal, it may still be heard." [Wat92]

Pitch is thus even perceived during unvoiced sounds; the continuity gives rise to the rhythm of the speech utterances.

2.4.5 Auditory Feedback

The ear not only receives sounds from speakers but also plays a vital role in providing feedback to the speaker. The importance of auditory feedback is highlighted by the fact that infants who are born totally deaf inevitably remain dumb as a result [Abe74]. Intensity or loudness of speech is also judged by auditory feedback. A good example of this is people tend to talk louder when they are under a hairdrier because they cannot hear themselves talking against the background noise of the hairdrier.

2.5 Phonetics

Phonetics is the study of the production and perception of speech sounds in general, without attention to a specific language or dialect. How speech sounds should be identified, labelled and recorded is therefore one of the central concerns. In phonetics, the smallest distinguishable speech sounds are called *phons* usually denoted by standard symbols in

slanted lines. The phon is language independent. Therefore /k/ is the "k" phon in the English word *king*, the Afrikaans word *koning* (≡ king) and the French words *qu'est-ce que* (≡ what is). Phonetic symbols and notations are continually being revised to label all the sounds of all languages. The International Phonetic Alphabet (IPA) and the Arpabet are two recognised standards of phonetic symbols and are listed in [Par86]. *Phonemes* are the smallest distinguishable sounds *of a particular language*. They are represented by the same symbols as phons are but have square brackets around them. Therefore, the "th" phoneme in the English word *think* is denoted by [θ] whereas if one were broadly referring to "th" sounds across all languages, the "th" sound would be denoted by /θ/. The notation for labelling sounds in this thesis is the non-standard but more user-friendly "<sound>" followed by an example in which the <sound> is underlined.

In the following sections a phonetic theory is developed to understand how humans perceive speech sounds.

2.5.1 Syllable

The syllable is a sub-word unit of speech sounds which people generally have little difficulty recognising. Abercrombie points out that:

> "writers on linguistics have nevertheless *not* found it at all easy to say what a syllable is and there have been many arguments about how it should be defined" [Abe74].

A typical school grammar lesson would reduce the syllable to a set of rules. One such rule might state that every syllable must contain a vowel. This would not however cater for speech sounds like "shh" (a call for silence) and "hmm" (a sound uttered during contemplation). The rules do not explain why or how people can ordinarily distinguish the syllable and as a result, researchers are searching for a biological explanation.

One theory proposes that during speech, the airstream from the lungs is not emitted in one continuous flow, but as a series of short bursts of air called *chest-pulses* [Abe74]. The syllable is thus determined as the unit of sound produced by each chest-pulse. Although there are contradictions to this theory, its physiological nature makes it more plausible than a set of grammatical 'tricks'.

2.5.2 Segments

During the formation of a syllable, the articulators in the vocal tract move to produce several audibly distinct sounds. The syllable is thus comprised of smaller, distinguishable units of speech referred to as segments. Four main problems arise when trying to determine segments.

1. The segments are produced by the simultaneous action of several vocal organs.
2. The articulators can move fairly rapidly. For instance, the tongue can move to 12 different positions in the vocal tract in one second of normal speech [Abe74].
3. Even very slight displacements by the articulators are discernible by the human ear as different speech sounds.
4. Speech sounds are not separated by silent intervals. They are constructed by continuously moving articulators and sound sources which do not switch off and on between each segment.

Speech recognition would be a simple matter if these problems did not exist. But they do. So the describing and naming of all the possible speech sounds are the most daunting of tasks for the phonetician.

In order to simplify matters, syllable production is divided into three stages. The first stage involves the release of air as in a chest-pulse. Prior to such a release, there is usually some constriction in the vocal tract which causes the pressure build up needed to power the production of the syllable. Such conditions favour the formation of consonants.

The second stage is filled by a vowel because there is little or no obstruction in the vocal tract allowing air to be emitted from the chest-pulse with relative ease.

The final stage, the termination of the syllable before the beginning of the next, is, usually taken up by a consonant to mark the end of the vowel.

These three stages of syllable production are often denoted in terms of V, C and O - where 'V' stands for vowel, 'C' for consonant and 'O' for omitted sound. A point '.' is used to denote a syllable boundary. For example, the word *because* consists of two syllables and can be denoted as: CVO.CVC.

The labels, vowel and consonant, have been used without precise definition. In terms of how sounds are produced, vowels result from phonation in a relatively unconstrained vocal tract. Consonants, on the other hand, are formed when there is some constriction in the

vocal tract. This is not quite true if one considers the consonants "r" (in *right*), "l" (in *long*), "w" (in *wet*) and "y" (in *yet*). These sounds exhibit the production features of vowels but perform the role of consonants in the sentence. As a result, they are referred to as semi-vowels or approximants (the term *approximant* is used because the articulators of these sounds only approximate a constriction in the vocal tract unlike other fricatives which form a complete constriction). Vowels and consonants are defined in terms of their roles in syllable production. Vowels occupy the middle stage while consonants occupy the peripheral positions. Sounds like "shh" and "hmm" are exceptions.

2.5.3 Description of Vowels and Consonants

2.5.3.1 Consonants

Consonants can be categorised physiologically according to three characteristics, namely:

- Point of articulation
- Manner of articulation
- Voicing (Phonation)

Point of Articulation

This category distinguishes which articulators are involved, and where they are positioned. For example, the "b" sound in *bib* involves both top and bottom lip. The "b" sound is thus called bi-labial. Table 2.1 below lists most of the possible points of articulation.

Name	Place Description
labial	the lips
dental	the teeth
apico-	tip of tongue
gingival	the gums (usually behind upper teeth)
alveolar	the alveolar ridge
domal	the hard palate
lamino-	blade of the tongue
centro-	middle of tongue
dorso-	back of tongue
velar	the velum
pharyngeal	root of tongue and pharynx
glottal	between vocal chords

Table 2.1 Naming the Points of Articulation

Manner of Articulation

This describes the type of constriction at the point of articulation and how the sound is produced. For example, the "b" sounds in *bib* are produced by compression and are therefore called plosives. A list of most of the ways in which consonants are articulated is presented in table 2.2 below.

Name	Description
Plosive	Closed vocal tract followed by a burst
Aspirate	Closed vocal tract followed by puff of air
Affricate	Closed vocal tract followed by gradual release producing turbulence
Fricative	Turbulence created at point of articulation
Lateral	Closed vocal tract but open at the sides
Semivowel or Approximant	Open at point of articulation and thus without turbulence
Nasal	Velum open and the vocal tract closed
Trill	Oscillatory movement of the tongue

Table 2.2 Naming the Manner of Articulation

Voicing

Voicing refers to sound created by the vibration of the vocal chords. Examples of voiced fricative phonemes are listed below, together with their unvoiced equivalents.

Voiced Sound	Unvoiced Equivalent
"b" in *bat*	"p" in *pat*
"d" in *dip*	"t" in *tip*
"g" in *goat*	"k" in *coat*
"v" in *van*	"f" in *fan*
"j" in *jew*	"ch" in *chew*
"z" in *zoo*	"s" in *sue*
"dz" in *leisure*	"sh" in *she*
"th" in *that*	"th" in *think*

Table 2.3 Voiced and Unvoiced Fricative Pairs

Summary of Consonant Classification

Consonants are named using the following convention. Firstly, say whether the sound is voiced or voiceless; then, express the place of articulation as an adjective; and finally, describe the manner of articulation as a noun. Several examples of describing consonants are displayed in the table below.

Example	Description
"p" in *pet*	voiceless bilabial plosive
"s" in *sit*	voiceless (grooved) alveolar fricative
"v" in *vat*	voiced labiodental fricative
"m" in *mat*	voiced labial nasal

Table 2.4 Complete Naming of Consonants

The apparent ease with which consonants can be defined is contrasted with that of vowels in the next section.

2.5.3.2 Vowels

Vowels are not as easy to classify physiologically as consonants are because, in most cases, the vowel sound is different as a result of only the slightest variation in the position of the tongue (the primary articulator). To compound this problem, the tip or blade of the tongue often does not quite touch the sides of the vocal cavity and this makes a description of the precise location of the articulators very difficult. Consider, for example, the initial part of the "I" (capital i) sound in *high*. The tongue is positioned low in the mouth and the voiced sound resonates in the cavity of the open mouth. Notice how vague "low in the mouth" sounds when compared to apicoalveolar or "the tip of the tongue touching the alveolar ridge". As a result, the following categories have been determined for vowels:

- tongue, high or low
- tongue, back or front
- lips, rounded or open
- nasalised or not (eg. the "ar" sound in *garçon* = boy in French)

Example	Description
"u" in *cut*	low back tongue, lips open, not rounded
"ar" in *garçon*	low back tongue, lips open, nasalised
"oo" in *sue*	high middle tongue, lips rounded
"e" in *head*	middle tongue, lips open

Table 2.5 Examples of Descriptions of Vowels

Vowel diagrams have been developed to illustrate the relative position of the tongue at the time of vowel articulation.

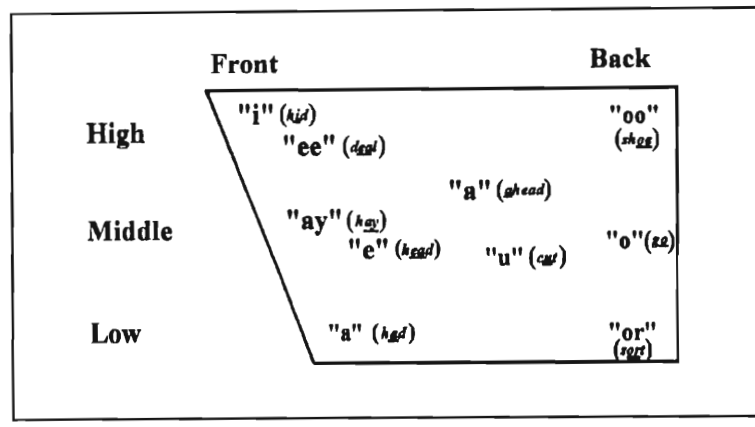

Figure 2.6 Vowel Diagrams adapted from [Par86]

Voiceless Vowels

These are vowels produced during whispered speech. However, several examples of voiceless vowels do occur in normal speech. The first vowel in the word *potato* is voiceless because of the aspiration after the "p" and the rapid movement of the articulators to the "t" sound.

Diphthongs

Diphthongs are formed by vowel segments which are constantly changing. For example, the "ow" vowel sound in *how* changes during its production as the lips are gradually rounded. The "ay" vowel sound in *day* is formed by gradually raising the blade of the tongue. Diphthongs can be thought of as two 'joined' vowels produced in one syllable. The vowel sounds in *ruin* do not form a diphthong since they are produced in two separate syllables not one.

Secondary Articulations

Although vowels have been described in terms of lip posture and positioning of the tongue, there are obviously several other influencing "points of articulation" in the vocal tract. This is in fact true of vowel and consonantal segments alike. A good example of secondary articulation is found in the "m" sound in words like *me*, *make* and *mine*. The tongue position of the vowel sound following the "m" sound, alters the "m" sound slightly in each of these examples. The primary point of articulation of "m" is the lips. Thus, "m" is usually referred to as a voiced labial nasal. However, sound variations in the "m"s of the above example are due to the position of the tongue which produces a secondary articulation.

2.6 Phonology

From the study of the techniques used for naming different sounds, it is evident that people have the ability to produce a great variety of speech sounds. However, a language typically contains *only a small number* of all the possible segments from which words are formed. In addition, certain combinations of segments are never formed while other combinations of segments are found in some position in the word more frequently than in others. Phonology is the study of segments and their characteristics in a particular language. Divided into two main areas, the first deals with the selection of a set of "distinguishable segments" for a language and the second examines the structural composition of the segments that make up the syllables and words of a language.

2.6.1 Selection of Distinguishable Segments

Allophones are the members of a set of phonetically similar sounds accepted by speakers of a language as being the same sound [Par86]. The sound that allophones are "accepted as being" is called the *phoneme* denoted by the phonetic symbol in square brackets. For example, consider the "k" sound in *kin* and *cup*. These "k" sounds exhibit a secondary articulation determined by the vowel sounds which follow them and are therefore technically different sounds. They are however perceived by English speakers to be the same sound and are thus allophones of the [k] phoneme. Two more allophones of [k] are found in *cope* (where the "k" sound is aspirated) and *scope* (where it is not aspirated).

Every language contains hundreds of segments and yet relatively few phonemes. Thus most segment sounds are allophones of the phonemes of a language. Humans, fortunately, have the ability to identify phonemes.

2.6.2 Structural Composition of Segments in Syllables

The CVC combination of segments is typical of English syllables. For example: *pick, ball, shape, house.*

It is also common for the peripheral C segments to be omitted. *tie*, *the* and *do* all exhibit the form CVO (where O denotes the omitted segment); examples of the form OVC are: *up*, *on* and *at.* Single segments like *a*, *I* are of the form OVO.

The forms discussed above are also found in multisyllabic words. Eg. the word *internally* has the form OVC.CVC.CVO.CVO.

In English, it is also possible to have clusters of consonant segments in syllables. A consonant cluster is a group of several consonant segments replacing a single C segment in the first or last stage of syllable production. The monosyllabic word *streets*, denoted by CCCVCC, illustrates this very well.

Several languages restrict the possible forms of syllable construction. A speech recognition system of such languages can exploit these constraints to make the identification process much easier.

One last point must be made concerning the composition of segments in syllables. It is evident that various combinations of phonemes within syllables occur more frequently than others in a particular language. In fact, some combinations may not be found at all. Information of this nature is very useful to the speech recognition system as is shown by the the following examples.

The "ps" sound combination is hardly ever found in the first syllable of an English word (eg. *psst*). It is found in syllables other than the first eg. *synapse*, and *napsack.* In fact, any noun ending in the "p" sound (eg. *turnip*) will result in a "ps" sound combination for its plural form (ie. *turnips*). Similarly, verbs which end in the "p" sound (eg. *help*) have a "ps" sound combination for their first person singular present tense form (ie. *he, she or it helps*).

2.7 Broad Speech Classification using Distinctive Features

Jakobson proposed that speech sounds be categorised by so-called *distinctive features*. The distinctive features of a phoneme are a set of broad speech production categories. By definition, either a phoneme has an attribute or it does not. The distinctive features of a

Distinctive Feature	Description
Vocalic/Nonvocalic	Refers to the presence or absence of a well-defined formant structure. (Formants are the natural resonances occurring in the vocal tract).
Consonantal/Nonconsonantal	Consonantal implies a relatively small amount of total energy.
Compact/Diffuse	Refers to distribution of spectral energy.
Tense/Lax	Tense implies larger total energy with wider bandwidth and longer duration.
Voiced/Unvoiced	Voicing indicates the presence of low-frequency components due to the vibration of the vocal chords.
Nasal/Oral	Nasal shows a wider distribution of spectral energy resulting from additional nasal resonances.
Discontinuous/Continuous	Discontinuous phonemes show abrupt changes in spectral energy spread.
Strident/Mellow	Strident phonemes have stronger and more random noise components.
Checked/Unchecked	Energy in checked phonemes appears as a burst, as in plosives.
Grave/Acute	Grave sounds are dominated by low-frequency resonances, acute ones by high-frequency resonances.
Flat/Plain	Difference is one of relative energy of high-frequency resonances: flat or weaker vs. plain or stronger.
Sharp/Plain	Sharp phonemes show a raising in the relative frequency of higher-frequency resonances.

Table 2.6 The Original 12 Distinctive Features

speech sound are typically represented by an array of twelve binary values (usually "+" if the feature is exhibited, else "-", ie. "1" or "0", or "TRUE" or "FALSE"). The attributes are listed in table 2.6 adapted from Parsons [Par86]. Revisions have been made to Jakobson's original list of distinctive features by Jakobson [Jak52] and later Chomsky and Halle [Cho68]. The definite 'yes-or-no' character of these features has been criticised. One way to avoid this is to assign a probability value (or a fuzzy confidence qualifier) rating the degree of certainty of that feature being found in the speech sound.

2.8 Coarticulation

Coarticulation refers to the 'overlapping' of phonemes. This occurs when the brain anticipates the next phoneme and the articulators, due to lethargy, begin moving into position to articulate the next phoneme before the current phoneme has been fully

articulated. Commonly used words like *and* and *the* are often greatly effected by coarticulation to such an extent that they can be left out altogether (omission is called *ellipsis*) because higher-level speech processing can determine their presence in context without actually having to 'hear' them. *Up and down*, for example, is reduced to *up 'n down*.

2.9 Voice Characteristics and Voice Dynamics

For the purpose of speech recognition it has been of primary importance to examine speech production and perception in terms of segmental features as this determines the actual phonemes in a speech utterance. There are however other aspects of speech sounds which also impart valuable information to the listener. They are voice characteristics (or quality) and voice dynamics.

Voice characteristics are those features of speech which are present more or less all the time that a person is talking [Abe74]. They are the speech cues which inform the listener about the identity of the speaker. They involve all the following components: language, dialect, gender, physical injuries to the vocal apparatus, ailments (eg. asthma) and the shape and size of the speech production mechanisms of the speaker. A good example of this is the similarity of voice characteristics in members of the same family. Genetically, the shape and size of their vocal tracts are similar. In addition, a 'domestic dialect' results of their constant interaction.

How are voice characteristics important to speech recognition? Firstly, the human speech recognition system 'adapts' to recognise speech from different people. This is particularly noticeable for example, when listening to a foreigner speaking English with a strong accent. The brain quickly 'adjusts' to decipher and understand the new 'dialect'. Moreover, knowing that the speaker is foreign will prepare the listener for semantic and syntactic errors where the speech might be idiomatic in sense and the sentences unstructured. A detailed discussion of the differences between male and female voice features is undertaken by Bladon[1] [Bla85].

More important to the speech recognition problem are the feaures of voice dynamics. These are speech feaures that are under the speaker's control and can be acquired. They include: loudness, tempo, continuity, rhythm, tessitura, register and pitch fluctuation [Abe74]. The most important of these are investigated in the following sections.

---

[1] In chapter two of Computer Speech Processing [Fal85]

2.9.1 Loudness

Loudness depends on the amount of compressed air supplied by the lungs and the extent to which the vocal chords are opened. The range of loudness is quite substantial and can be and is controlled by the speaker through the feedback from the ears. It is interesting to note that if there is interference with the feedback mechanism (eg. an elderly person whose hearing is impaired) the speaker tends to speak too loudly.

2.9.2 Pitch

Pitch is the rate at which the "pulses of air" generated from voicing repeat themselves. It is controlled primarily by the tension of the vocal folds, and is also regulated by feedback from the speaker's hearing perception mechanism. The pitch fluctuates continually in normal speech. These fluctuations are however by no means random and usually follow well-defined "tunes". The pitch disappears during unvoiced sounds but humans do not actually notice these momentary gaps. Pitch fluctuations are perceived to be continuous across the utterance. These fluctuations generate meaning of their own to the listener. Consider the following utterances spoken with different pitch intonations:

We have arrived? and *We have arrived!*

The phonemic content is identical. The difference in meaning is translated by the sound of the utterances, that is, their pitch fluctations.

2.10 Higher-levels of Speech and Language Processing

As speech can be categorised and analysed in a number of ways using units of speech greater than the phoneme, it is important to look briefly at these higher-levels of speech and language processing. One such way classifies sentences of speech into grammatical units (parts of speech) which together form the structure or syntax of the sentence. Another approach analyses the semantics or meaning of words in the sentence. Pragmatics looks at the speaker's intentions and the context of the sentence in the conversation. Yet another perspective on speech is found in the rhythm, pitch and stress or emphasis of words in the sentence. These features are called prosodics. Pitch fluctuations have already been mentioned in this chapter. The various perspectives of speech analysis mentioned here are referred to (in this thesis) as the higher-levels of language and speech analysis. These are essential in the speech understanding process and are covered in more detail in chapter 8.

2.11 Acoustics of Speech Production

The previous analysis of speech production was from a physiological perspective. In this section, the acoustic nature of speech is investigated and a model for speech production developed.

Vowel Sound	Example	f_1 (Hz)	f_2 (Hz)	f_3 (Hz)
"ee"	*beat*	250	2300	2750
"i"	*bit*	330	2500	2900
"e"	*bet*	500	2090	2830
"er"	*bird*	490	1660	2540
"ar"	*bar*	700	1210	2460
"u"	*but*	700	1370	2340
"oo" (short)	*book*	470	850	2420
"oo" (long)	*boot*	390	780	2000
"o"	*lot*	660	1050	2500

Table 2.7 Formant Frequencies of Common Vowels

The fundamental frequency, denoted by f_0, of voiced speech is the rate at which the vocal folds vibrate, and therefore only applicable to voiced speech. According to Fourier theory, f_0 determines the natural harmonics of the resulting speech sound. However, certain frequencies (called the *formants*) are more susceptible to resonance than others due to the shape and size of the resonance cavities in the vocal tract. A spectrogram can be used to detect frequencies of greater intensity due to the formants. There can be as many as six formants but usually the first three are the most predominant. Table 2.7 contains the first three formants of various vowel sounds spoken by the author. As a comparison, the first three formants of the "er" in *bird* by a female voice were measured and found to be: 625Hz, 1680Hz and 2850Hz. The higher frequencies are characteristic of female voices, largely due to their smaller vocal apparata.

The formants of vowels can be used to identify them. Formant-tracking is a technique used to trace the first three formant frequencies in order to determine the identity of the vowel sounds. In this method, the peak amplitudes of the smoothed amplitude or power spectra can be used to determine the formants.

The vocal tract is often modelled by a simple 'lossless' uniformly cylindrical tube open at the one end and with a sound source at the other. Sound is propagated along the tube and out at the open end. Standard physics textbooks will show that resonances occur in such a tube when the tube length is at odd multiples of a quarter of the wavelength of the sound.

The length, ℓ, of the vocal tract of the average man is about 17cm and the speed of sound c is roughly $340ms^{-1}$. Therefore the first resonant frequency can be determined as:

$$f_1 = \frac{c}{\ell} = \frac{340ms^{-1}}{4\times0.17m} = 500Hz$$
$$f_2 = \frac{c}{3\ell} = \frac{340ms^{-1}}{4/3\times0.17m} = 1500Hz \quad (2.1)$$
$$f_3 = \frac{c}{5\ell} = \frac{340ms^{-1}}{4/5\times0.17m} = 2500Hz$$

The lossless tube model best matches a sound produced by an unimpeded vocal tract ie. the neutral "er" sound. Notice the similarities in the three formant frequencies of the "er" sound in *bird* in table 2.7 and those of the tube model. Though an interesting analogy, the above model is however quite different from the soft, elastic, non-uniform vocal tract.

2.12 Filter-Model of the Speech Production Mechanism

The filter-model was first derived by Fant [Fan60] as a system for producing speech sounds. It is based on several assumptions namely that speech sounds vary slowly in time and that the shape of the vocal tract and the source of excitation determine the sound at any moment. Using these assumptions, speech sounds can be produced by adapting the shape of the vocal tract and the type of excitation for short time intervals (±10-45ms long). Moreover, the shape of the vocal tract can be modelled by the parameters of a linear filter which acts on input sound generated either by a quasi-periodic train of pulses (emulating the glottal source in vocalic sounds), or by random noise (emulating the turbulence in frication), or both to produce voiced fricative sounds. The model is illustrated by figure 2.7:

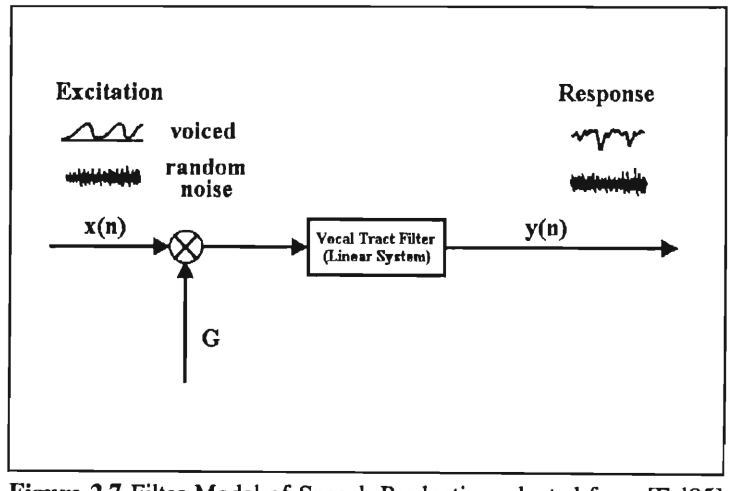

Figure 2.7 Filter-Model of Speech Production adapted from [Fal85]

The filter-model is used in speech recognition by a speech processing technique called linear predictive coding (LPC), discussed in the following chapter. Because LPC is so widely used in speech recognition and in order to understand the filter-model, several points are made about discrete-time signals and filters.

In the time-domain, a filter is defined as a discrete-time system which operates on an input signal x(n) to produce an output signal y(n). A special input signal δ(n) called the *impulse* (or *unit-pulse*) sequence is defined as:

$$\delta(n) = \left.\begin{matrix} 1 \\ 0 \end{matrix}\right\} \quad \begin{matrix} n=0 \\ otherwise \end{matrix} \tag{2.2}$$

The filter output of an impulse sequence is called the *impulse response* and is usually denoted by h(n). Any discrete-time signal x(n) can be written as the sum of an infinite number of weighted time-delayed impulse sequences ie.

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\,\delta(n-k) \tag{2.3}$$

The filter output y(n) of a signal x(n) can thus be written in terms of the impulse response h(n) as:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\,h(n-k) \tag{2.4}$$

Provided the filter is a Linear Time-Invariant filter, the output y(n) of a signal x(n) can be written in terms of the *convolution* operator * as y(n) = x(n) * h(n). Determining the response of a filter by convolution is computationally expensive, but when transformed to the frequency-domain, the result is a single multiplication.

To transform any signal x(n) to the frequency-domain, the complex z transform is used:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \tag{2.5}$$

Using equations 4 and 5 the filter can be expressed in the frequency domain in terms of the Frequency Response as

$$Y(z)=H(z)X(z) \tag{2.6}$$

Note:

- the above equation is a single multiplication in the frequency-domain.

- H(z) is referred to as the *transfer function* of the filter and sometimes A(z)=1/H(z), the inverse of the transfer function is used.

If it is further assumed that the filter is Casual and that it satisfies a linear constant-coefficient difference equation of the form

$$\sum_{k=0}^{p} a_k y(n-k) = \sum_{k=0}^{q} b_k x(n-k) \tag{2.7}$$

then H(z) will be a rational function and can be written as a ratio of two polynomials as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{q} b_k z^{-k}}{\sum_{k=0}^{p} a_k z^{-k}} \tag{2.8}$$

Note:

- The *zeroes* of above equation are those values of z which result in the numerator equating to 0 whereas the *poles* are the values of z which result in the denominator equating to 0.
- The resonances of the filter have been shown to be adequately described by the poles of the filter. In fact, by including extra poles, the effects of nasality and frication can be modelled [Mak75].

The above observations are used in the next chapter to show that it is possible to ignore the contribution of the zeroes and use the all-pole filter to model the effects of the vocal tract.

Chapter 3 - Preprocessing Speech Signals

3.1 Hardware Preprocessing

3.1.1 Sampling

Speech is transmitted through the air by sound waves. These waveforms can be represented mathematically by functions of the form x(t), continuous in time and amplitude, called *analogue signals.*

The sudden, ubiquitous use of powerful digital computers necessitated a means of capturing and storing a *digital* (or discrete-time) form of these analogue signals in computer memory. This is achieved by *sampling* the analogue signal at uniform time intervals and converting them to a digital form using a process known as *analogue-to-digital conversion* (ADC). The sequence of digital values (or *samples*) as a result of ADC are represented by the discrete function x(n), where n is related to time by n=kT, for k = 0,1,2,... Also, the *sampling period* T, is the inverse of the *sampling frequency* f_s ie. $f_s=1/T$.

3.1.2 Sampling Frequency, Nyquist Rule and Aliasing Error

Sampling results in a loss of information about an analogue signal due to the time intervals between samples. Therefore, if the sampling period is "too large" (or equivalently, f_s too small), much information about the analogue signal will be lost. This phenomenon is called *undersampling or aliasing.* On the other hand, if the sampling period is very small (ie. f_s is very large), the number of samples can become excessive, retarding the speed of analysis and processing without significant improvement in the results of the system. This effect is known as *oversampling.*

What therefore is the lowest sampling frequency which preserves the important characteristics (especially the frequency information) of the original signal? The solution lies in the well-known *Nyquist rule* which states that

> *the sampling frequency must be at least twice the highest significant frequency.*

The "highest significant frequency" is called the *Nyquist frequency.* A sampling frequency less than twice the Nyquist frequency is considered undersampled and therefore exhibits

aliasing. (A more detailed investigation of the above ideas can be found in most digital signal processing textbooks eg. [Opp90]).

What value should be used for the Nyquist frequency in speech recognition systems? Although the human ear can perceive sound in the frequency range 30Hz-20kHz, it was shown in the previous chapter that the frequencies less than 5kHz are perceived the best. In addition, frequencies less than 5kHz are adequate to recognise all the speech sounds (even the high frequency fricative sounds) and most voice qualities by virtue of the fact that the current standard of digital telephone is a sampling frequency of 8kHz and therefore a Nyquist frequency of 4kHz [Vee88]. Consequently, most speech recognition systems have a sampling frequency of 8-10kHz and thus a Nyquist frequency of 4-5kHz.

3.1.3 Filtering, Low Frequency Noise and High Frequency Feed-back

A filter is a system for eliminating unwanted frequencies from the signal. There are two[1] important classes of filters in speech processing applications namely, the *high-pass* (HP) and *low-pass* (LP) filters.

The ideal HP filter eliminates all the signal frequencies below some frequency (eg. 125Hz) in order to erradicate any interfering, non-speech, low frequency noise - particularly the 50Hz mains' hum.

The ideal LP filter eliminates all the signal frequencies above the Nyquist frequency to prevent *frequency foldback* ie. to prevent the interpretation of signal frequencies greater than the Nyquist frequency as frequencies in the significant frequency range.

The *passband* of a filter is the range of signal frequencies untouched by the filter whereas the *stopband* is the range of signal frequencies eliminated by the filter.

The above definitions describe the properties of an 'ideal filter', that is one with a perfect cut-off at a particular frequency between the passband and stopband. In practice however, filters gradually attenuate the signal outside the cut-off frequency. Consequently, the *cut-off frequency* of a non-ideal filter is defined to be the frequency at which the power of the signal is attenuated by -3dBs ie. is half the power of the signal in the passband.

The effect of the non-ideal LP filter poses the problem of how to prevent foldback of the gradually attenuated, high signal frequencies in its stopband. There are two ways of

[1] A third type of filter, the band-pass, is a combination of both HP and LP filters.

reducing this problem. Firstly, a filter with a steep cut-off slope can be chosen eg. a high-order Chebychev filter[1]. Secondly, one can ensure that the signal at the Nyquist frequency be attenuated by -30 or -40dB. The -3dB low-pass cut-off frequency would thus need to be set slightly below the Nyquist frequency, low enough so that the attenuation at the Nyquist frequency is less than -40dB.

3.1.4 Storage Considerations

The ADC process stores the digital samples as *n*-bit words. The size of *n* is called the resolution of the analogue-to-digital (AD) converter. The minimum resolution of an AD converter for speech recognition is 12-bits, requiring two byte data words to store it in a computer [Par86]. Using a sampling frequency of 10kHz and a two byte sample word, 20 000 bytes of sample data is generated per second of speech. In normal speech, it takes about 3 seconds to utter a sentence, thus generating 60 000 bytes of sample data. This amount of data is too great (and 'raw') even for expert phoneticians to decipher. What is needed, is some way of extracting the important speech characteristics from these samples. Most digital processing systems have a front-end preprocessing stage to execute this. It is either referred to as *feature extraction* or *preprocessing*. The preprocessing stage can also be thought of as a speech compression process since many raw data samples are compressed into relatively few characteristic *speech features*, extracted in the time- and/or frequency-domain.

3.2 'Short-Time' Analysis of the Speech Signal

The 'short-time' analysis of speech is founded on the assumption that:

> *speech sounds vary relatively slowly at a rate of less than 100Hz.*

As a result, it is possible to determine the characteristic information about the signal by processing abutted or slightly overlapping short-time intervals of the speech signal called *speech frames*. The characteristic information called *speech features*, is extracted for each speech frame and usually comprises representative spectral (frequency) and/or temporal (amplitude-time) metrics. The functions used to extract this information are called *preprocessing functions*. The speech features of successive speech frames are often represented as vectors, called *speech feature vectors*.

---

[1] The different types of filters are described in most filter theory textbooks eg. [Joh76] while Lancaster's Filter Cookbook [Lan79] is an excellent practical guide to building active filters.

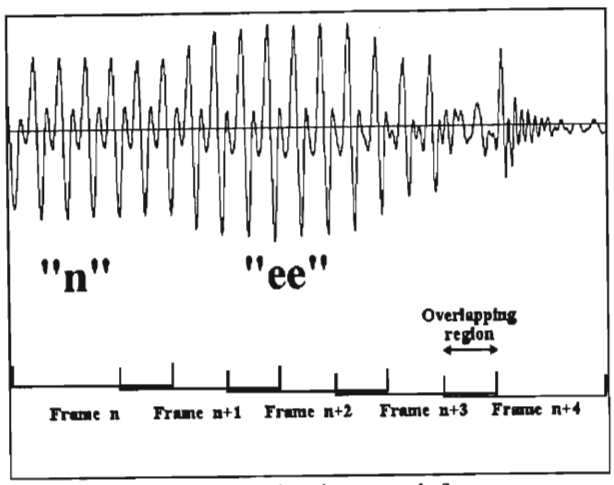

Figure 3.1 Overlapping speech frames.

3.2.1 Windowing

'Short-time' speech frames can be isolated from the rest of the speech signal by multiplying the signal with a *window function* w(n). A window function sets all the values of the signal outside the 'window' to zero and may, in addition, apply some appropriate function to the sample values inside the window.

The *length of the window* (typically 5-45ms) is very important. If it is too large, the speech features become 'blurred' by different acoustic events in the window; if it is too small, low frequency spectral information may be omitted and temporal features can be under-averaged.

An important consideration in determining the length of the window is pitch. Recall from the previous chapter that the pitch period is the inverse of the fundamental frequency ie. the rate of the vibrations of the vocal chords in voiced speech. Pitch is so important because it determines the lowest speech frequency that plays an essential role in the spectral make-up of speech sounds and in prosodics. Everyone can achieve a range of pitches by manipulating their vocal chords. However, due to physiological constraints, males generally have lower pitches than females and children. The highest pitch found in baby cries and high-pitched females reaches a period of about 5ms while the deepest male voice can approach a pitch period of 25ms.

When extracting the spectral features from part of the signal, it is necessary to include all the frequencies making up an acoustic event, especially the fundamental frequency. In fact, the window should contain exactly one pitch period of speech for most spectral extraction techniques to be accurate. This is very difficult to regulate due to the large number of variables in normal speech production. To circumvent the problem, one should choose a window length which is just greater than the pitch period 'most of the time' and taper the

samples within the window at the window boundaries to zero. This practice will reduce the possibility of erroneous frequencies in the spectrum. Various window functions have been developed to achieve this effect; the one referred to and used in this project being the Hamming window.

3.2.2 Window Functions

The most common window functions are the *rectangular* $w_R(n)$ and *Hamming* $w_H(n)$ window functions. Other window functions, like the Hanning, cosine taper, the raised cosine and Bartlett functions, are not discussed here. Detailed investigations of these window functions are undertaken in most digital signal processing texts[1].

Rectangular Window Function

The rectangular window function is used predominantly in temporal preprocessing functions. It isolates a particular segment of the signal without affecting the signal values inside the window in any way. This is mathematically expressed as [Fal85]:

$$\begin{aligned} w_R(n) &= 1 && 0 \le n \le N\text{-}1, \text{ where N is the length of the window} \\ &= 0 && \text{otherwise} \end{aligned}$$

The rectangular window is unsuitable for spectral analysis because it does not cater for the probability of discontinuities at the window boundaries as do the tapered window functions. An example of a rectangular window is shown in figure 3.2.

Hamming Window Function

The Hamming window function is used primarily by spectral preprocessing functions. It tapers the signal to zero at the window boundaries (see figure 3.3). In this way, the discontinuities at the window edges are eliminated by forcing the first sample at the left window boundary to match the last sample at the right window boundary (both equal zero). The Hamming window is given as [Fal85]:

$$\begin{aligned} w_H(n) &= 0.54 - 0.46\cos(2\pi n/(N\text{-}1)) && 0 \le n \le N\text{-}1, \text{ N is the window length} \\ &= 0 && \text{otherwise} \end{aligned}$$

[1] Oppenheimer and Schafer [Opp90] and Priemer [Pri89] include good analyses of these functions in the frequency domain and also examine the errors they introduce.

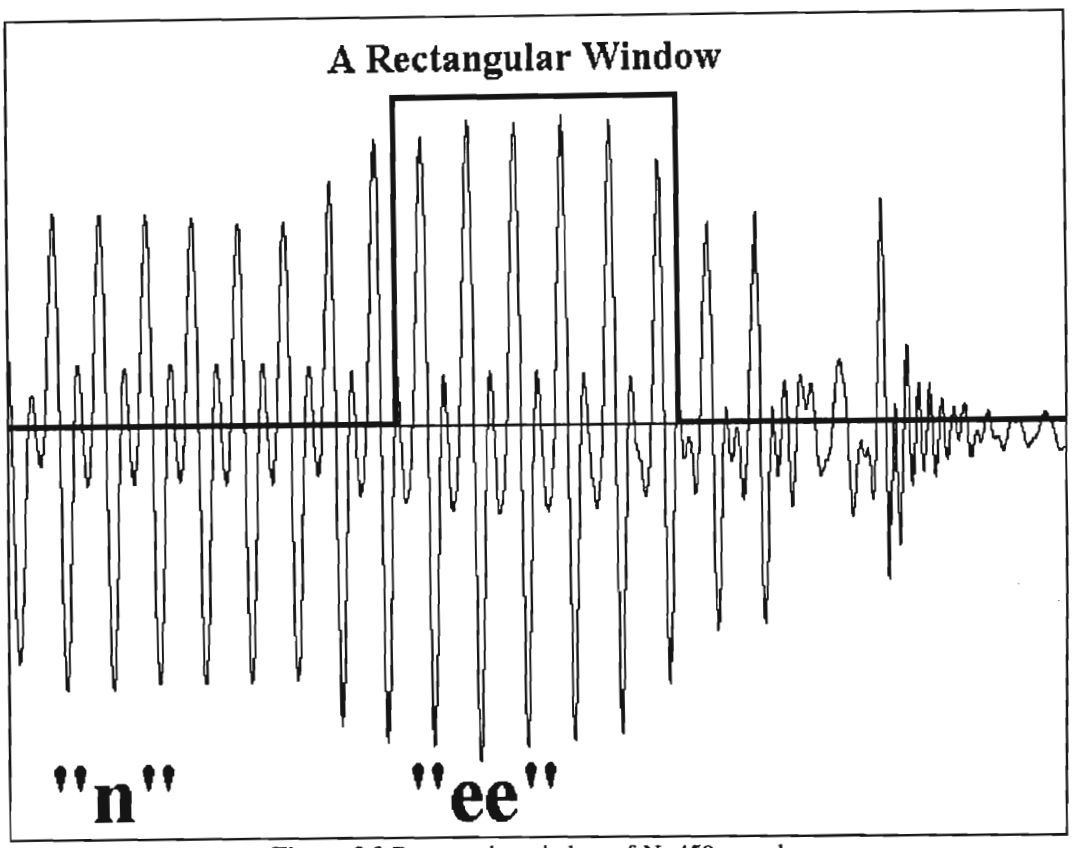

Figure 3.2 Rectangular window of N=450 samples

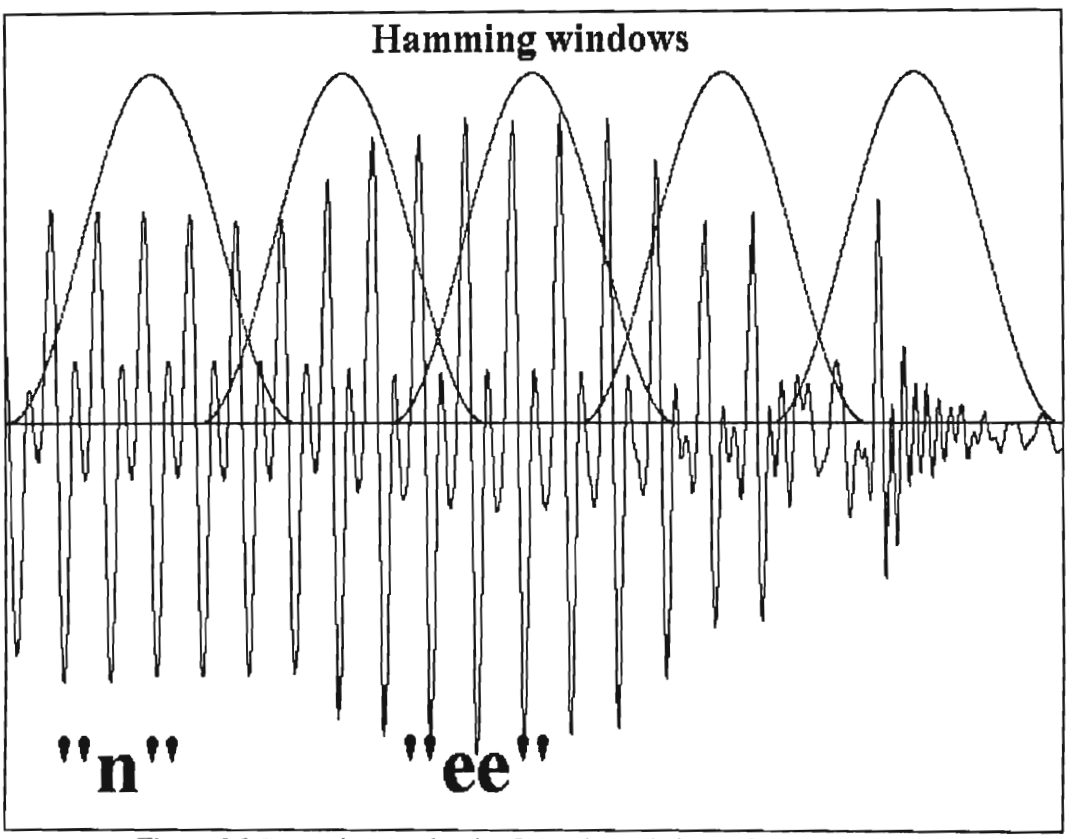

Figure 3.3 Successive Overlapping Hamming Windows of N=450 Samples

Both figure 3.2 and figure 3.3 are from the underlined part of the utterance:

How many objects are on the table?

3.3 Preprocessing Functions

Preprocessing condenses 'raw' sample data into speech features which are easier to interpret and analyse.

3.3.1 Temporal Preprocessing Functions

Temporal preprocessing functions extract speech features directly from the speech samples in the amplitude-time domain. They typically find the average of some amplitude-based metric over each speech frame. The output of a temporal preprocessing function can be graphed with respect to time to determine *trends*. Analysis of these trends is aided by the smoothness of the resulting curve, which is related to the length of the speech frames. The shorter the speech frames, the more perturbated the resulting curve tends to be and the poorer the speech compression. On the other hand, if they are too large, although the speech compression is greater, the speech frame can encompass several different acoustic events resulting in blurring or over-averaging. Dips, curve shape and gradient can be used to indicate acoustic changes and acoustic class information (eg. vocalic vs. fricative).

The temporal functions and their theory has been assimilated from various sources, the most important of which are Veeneman [Vee88], [Row92a], Fallside [Fal85] and Oppenheimer and Schafer [Opp90].

'Short-time' Energy Function

The total energy E of a real analogue signal, x(t), is given by:

$$E = \int_{-\infty}^{\infty} x(t)^2 \, dt \tag{3.1}$$

The 'short-time' energy (also called *intensity*) function is defined for a portion of the discrete signal x(n) approximating x(t) as:

$$E_n = \sum_{m=-\infty}^{\infty} [x(m) \; w(n-m)]^2 \tag{3.2}$$

In equation (3.2), the window function is typically a rectangular window $w_R(n)$. The above equation also shows that the window function can be thought of as a linear filter[1] with impulse response h(n).

A short-time absolute amplitude function is computationally less expensive than the intensity function and is therefore sometimes used in its place:

$$\hat{E}_n = \sum_{m=-\infty}^{\infty} | x(m) | w(n-m) \tag{3.3}$$

The intensity function is useful for distinguishing between segments of silence and sound in the signal since speech frames with sample amplitudes close to zero (silence) result in a small energy. As a result, a simple energy threshold value below which the speech frame is said to exhibit silence, can be determined by experimentation.

The 'short-time' energy function viewed graphically in time tends to produce a smoothed curve similar to the amplitude-time signal. From this graph one can identify regions of acoustic similarity or changes in acoustic events (dips in curve).

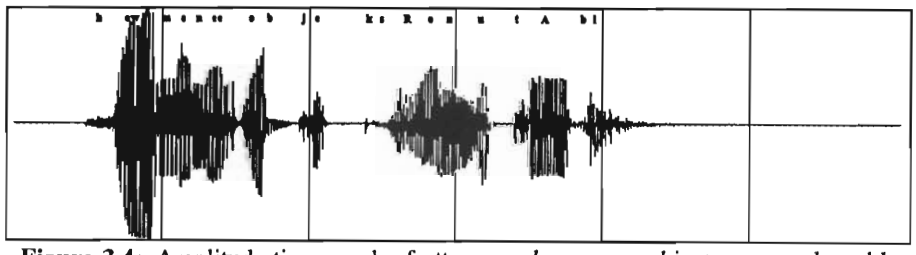

Figure 3.4a Amplitude-time graph of utterance: *how many objects are on the table*

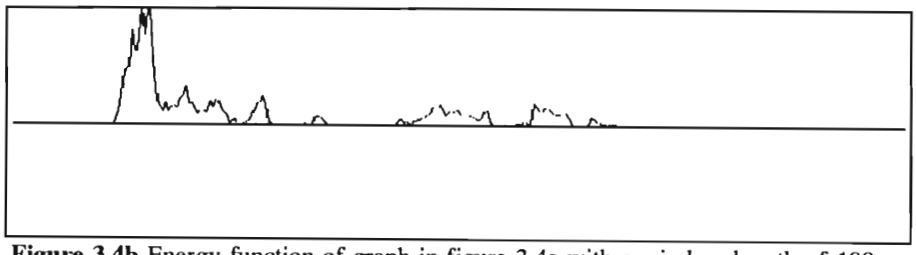

Figure 3.4b Energy function of graph in figure 3.4a with a window length of 100ms

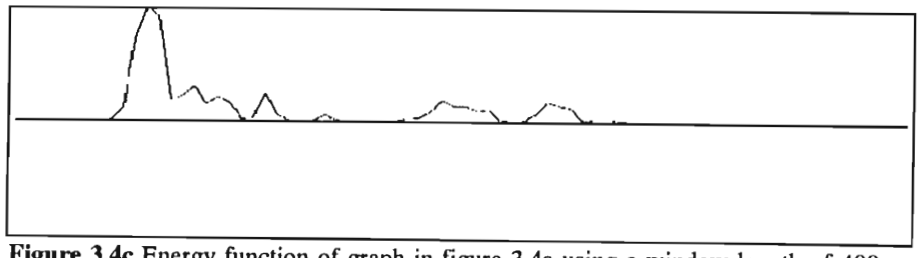

Figure 3.4c Energy function of graph in figure 3.4a using a window length of 400ms

[1] See last sub-section of chapter 2 for introduction to linear filter. Also see [Opp90].

The signal in figure 3.4a was sampled at 10kHz. In figure 3.4b, a window length of 100ms was used whereas in figure 3.4c, a window length of 400ms was used. In both energy graphs, no overlap between speech frames was used. The above examples demonstrate the effects of averaging with respect to the size of the window length that were mentioned previously. That is, the shorter the window length, the more 'ragged' the curve (fig. 3-4b) while the larger window length is prone to averaging over different acoustic events though its resulting curve is 'smooth' (fig 3-4c).

As a summary to the energy function, Waibel points out its importance in speech processing:

> "Intensity is mostly viewed as a contributing factor to stress or emphasis, but is so far considered to contribute little to recognition. This view is supported by perceptual studies that indicate that distortions in the amplitude or energy contours of speech have little or no detrimental effect on intelligibility... Indeed, DeMori et al [DeMori 76] have shown using a syntactic pattern recognition approach to speech recognition that intensity profiles could be used explicitly as an integral part of a recognition strategy." [Wai88].

Short-time Zero-crossing Function

The short time zero-crossing function counts the number of times the speech signal crosses the zero amplitude-axis (i.e. 0 volts) in a short time interval. The function is given by:

$$ZC_n = \sum_{m=-\infty}^{\infty} \begin{matrix} 1 \\ 0 \end{matrix} \Bigg\} \begin{matrix} sign[x(m)\ w_R(n-m)] \neq sign[x(m+1)\ w_R(n-m-1)] \\ otherwise \end{matrix} \qquad (3.4)$$

where sign(k) is positive or negative

The average zero-crossing rate is therefore ZC_n/N. This can be used as a rough indication of the frequency of the signal. If the signal has a sampling frequency f_s then the approximate signal frequency in the window can be calculated by:

$$f_n = f_s \times \frac{ZC_n}{N} \qquad (3.5)$$

The narrower the bandwidth of the signal, the more accurate the frequency estimation of the above equation. However, even as a rough estimation of the frequency in the signal, the zero-crossing function has proved to be useful since high frequencies are an indication of frication. Pitchers [Pit90] reports a recognition system that achieved 92.6% recognition

accuracy on a small 10-digit vocabulary using only zero-crossing and energy metrics. For modest applications, this technique is very attractive owing to its computational efficiency and real-time response.

An important precaution of the zero-crossing function is to ensure that the effective zero amplitude of the sample data is the same as the nominal zero amplitude axis. A slight zero-offset can destroy the performance of this function.

Another source of error for the zero-crossing function is noise ie. non-speech sounds. The human ear has the ability to attune itself to various sounds. Speech systems have not yet achieved this level of sophistication. Filtering reduces some sources of noise (eg. 50Hz mains' hum). One can set up the system in a noise-free environment but this is not practical since most systems must operate in the noisy world. One possible solution for eliminating noise is to subtract the input from a microphone set up to monitor only the background noise from a microphone focussed on the speaker.

Short-time "Turning Points" Function

This function determines the number of local amplitude maxima and minima in a short-time interval. Viewed in another way, it determines the number of zero-crossings of the first derivative of the signal. This function, like the zero-crossing function, gives an indication of the frequency of the signal.

Mathematically, the function is described as:

$$TP_n = \sum_{m=-\infty}^{\infty} \begin{cases} 1 & \textit{TurningPoint}[(m)\ w(n-m)]=\textit{true} \\ 0 & \textit{otherwise} \end{cases} \qquad (3.6)$$

where TurningPoint(m) determines whether x[m] is a turning point

Autocorrelation function

Correlation functions determine the similarity between signals. The autocorrelation of a function x(t), as the name suggests, measures the periodicities and duration of the function.

The autocorrelation function of an analogue signal x(t) is given by:

$$R(k) = \int_{-\infty}^{\infty} x(t)\ x(t-k)\,dt \tag{3.7}$$

The autocorrelation of a discrete signal x(n) can be written as:

$$R(k) = \sum_{m=-\infty}^{\infty} x(m)\ x(m+k) \tag{3.8}$$

And, the short time autocorrelation of x(n) can be approximated by:

$$R_n(k) = \sum_{m=-\infty}^{\infty} x(m)\ w(n-m)\ x(m+k)\ w(n-m+k) \tag{3.9}$$

Several points can be made about the autocorrelation function:

- In the above equations, k is called the lag.
- $R_n(0)$ in equation is equivalent to the short time energy function previously determined.
- R(k) = R(-k), in the above equations.
- If x(n) is periodic, then so is R(k) ie. R(k) = R(k+mT), where m =1,2... and T is the same period of x(n). The same is true for an analogue signal x(t). However, if x(t) is periodic, it does not necessarily mean that x(n) is. Nevertheless, it can be shown by example that if x(t) is periodic (or even quasi-periodic) though x(n) the sampled version of x(t) is not, R(k) is periodic (quasi-periodic). This property is called the periodicity of the autocorrelation function. It is useful for determining the pitch period of voiced speech signals.
- The short-time autocorrelation function is often used in the all-important linear predictive coding discussed at the end of this chapter.

As already mentioned, the autocorrelation function uses the property of periodicity for pitch detection. Speech sounds are never quite periodic. Unvoiced fricative sounds are obviously non-periodic. Voiced sounds are not perfectly periodic and therefore termed quasi-periodic. The periodicity of the autocorrelation function isolates the pitch period from these signals, exhibiting a distinct peak at the pitch period. This is shown in figures 3.5a and 3.5b. The first graph is an amplitude-time graph of the word *bird* spoken by the author. The second is the output of the 'short-time' autocorrelation function (amplitude vs. lag k) of the highlighted section of figure 3.5a. There are several local maxima and minima in the

autocorrelation graph but the peaks at the pitch period are clearly distinguishable (aided by the markings of the successive pitch periods).

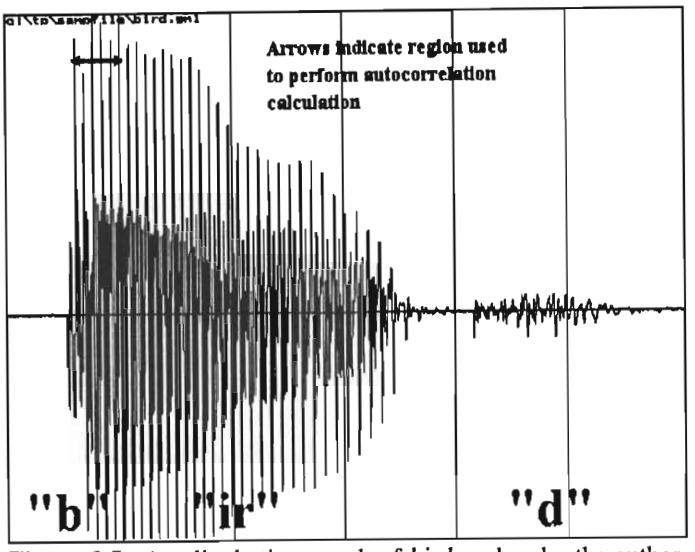

Figure 3.5a Amplitude-time graph of *bird* spoken by the author.

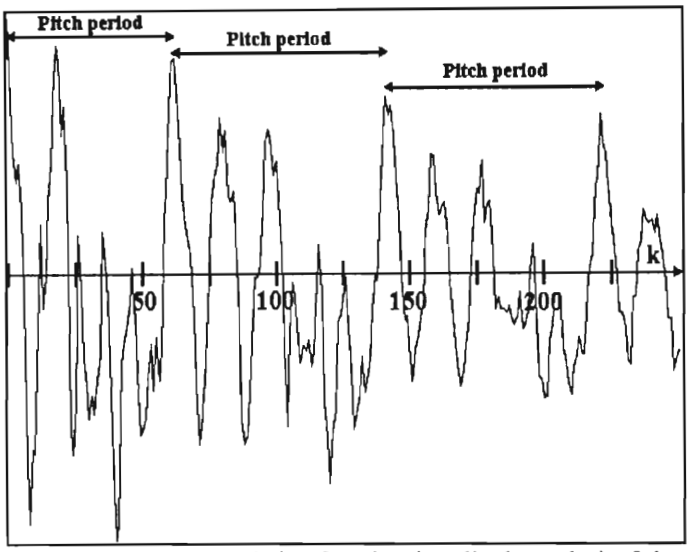

Figure 3.5b Autocorrelation function (amplitude vs. lag) of the highlighted section of figure 3.5a

Spectral Flattening to Determine Pitch Period

The pitch period can also be determined by a technique known as *spectral flattening*. The signal is first low-pass filtered with a cut-off frequency of ± 900Hz. Then a threshold is used to remove most of the signal excepting the major peaks. The distance between successive peaks (if they are a certain distance away from each other) is the pitch period. Several pitch detection techniques have been compared and investigated, see for example [Rab76] [Ata85] [Mak75].

3.3.2 Frequency-Domain Preprocessing Functions

Pre-emphasis

In voiced speech there is an attenuation of -6dB per octave across the spectrum. That is, the signal power is reduced by a factor of four for each doubling of the frequency across the spectrum [Mak75]. See Veeneman for a detailed analysis of these effects [Vee88]. Voiced speech should therefore be enhanced by +6dB/octave prior to speech processing. This procedure called *pre-emphasis* is important for reducing the variance of *distance metrics* [Rab76] (discussed in chapter 4) which determine the similarities and differences between speech sounds.

A pre-emphasis transfer function $H(z) = 1 - az^{-1}$ performs the necessary enhancement across the spectrum. In the time domain, this function can be written as: $y(n) = x(n) - ax(n-1)$, where y(n) is the enhanced signal and $0.9 \leq a \leq 1$.

Pre-emphasis is not necessary for fricative sounds since their constituent frequencies tend to be spread randomly across the higher end of the spectrum due to their turbulent nature. A useful way of negating pre-emphasis of these sounds is to define the scaling factor as $a = R_n(1)/R_n(0)$, where $R_n(k)$ is the output of the 'short-time' autocorrelation function with lag *k*. In vocalic sounds $R_n(1)$ is very close to $R_n(0)$ and therefore $R_n(1)/R_n(0)$ is close to 1. In fricative sounds $R_n(1) << R_n(0)$ and $R_n(1)/R_n(0)$ is close to 0 thus negating the effect of the enhancing scaling factor.

Fourier Analysis

Spectral information is essential for identifying the acoustic sounds in the signal. This is supported by phoneticians being able to identify phonemes with a high percentage accuracy using spectral information encoded in spectrograms.

How to extract spectral information from signals is the main emphasis of Fourier theory. In digital speech processing, the most important process is the discrete Fourier transform (DFT). Most digital signal processing textbooks contain the details of how the DFT is derived and its derivation is therefore omitted from this thesis. In particular, Poulton [Pou83] derives the Fourier transform (FT) for analogue periodic and aperiodic signals. A thorough investigation of FT and DFT can be found in most texts (eg. [Opp90], [Pri89], [Jac89] and [Rob87]). Fallside [Fal85] summarises the important results as does Rowden [Row92a].

Discrete Fourier Transform

The discrete Fourier transform (DFT) approximates the Fourier transform of discrete-time signals over a short-time interval. A window function is used to isolate the 'short-time' interval of N sample points from which the DFT is determined as:

$$X_n(k) = \sum_{m=-\infty}^{\infty} x(m)\, w(n-m) e^{-j\frac{2\pi}{N}km}; \qquad k = 0, 1, 2, \ldots, N-1 \tag{3.10}$$

where w() is a tapered (eg. Hamming) window function

The signal x(n) can be recovered from its DFT by the inverse DFT (IDFT) as follows:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn}; \qquad n = 0, 1, \ldots, N-1 \tag{3.11}$$

The spectrum from the DFT is discrete in frequency variable k. It is therefore possible to think of this discrete spectrum as a sampled version of the z- or Fourier transform of a finite length sequence [Fal85].

Fast Discrete Fourier Transform

There are a number of ways to optimise the DFT algorithm by exploiting Fourier transform properties given in the textbooks. A very efficient implementation of the DFT developed by Cooley and Tukey in 1965, has given rise to a class of DFTs known as the fast Fourier transform (FFT). Many variations of the FFT exist, the most well-known being the common-factor algorithms (eg. radix-2, radix-4). The details of this algorithm are investigated in most DSP textbooks eg. [Jac89].

Points about the DFT and FFT:

- The DFT requires the signal be preprocessed by a tapering window function like the Hamming window.
- Two sources of error leakage and ripple error, are introduced by windowing. A Hamming window covering at least one pitch period seems adequate to reduce the effects of these errors.
- The resolution of the spectrum is determined by the size of N; the greater the size of N, the finer the frequency resolution. However if N is too large, the 'short-time' sequence can include spectral components from different sounds (ie. blurring), destroying its potential for identifying individual speech sounds. To achieve fine

resolution without blurring it is possible to pad the N samples of a 'short-time' segment of the signal with zeroes.

- By reapplying the DFT to successive speech frames, the spectral information of the (slowly changing) speech sounds in the signal can be monitored.
- The radix-2 FFT algorithms requires that the length of the window N be a power of 2; the radix-4 algorithm requires N be a power of 4. N is typically in the range 32-1024.
- The FFT extracts the spectral composition of the signal into N/2 frequency 'bins'. The remaining N/2 'bins' mirror the results of the first N/2 'bins'. That is, the $((N/2) - k)^{th}$ 'bin' = $((N/2) + k)^{th}$ 'bin', for $1 \leq k \leq N/2$. Each 'bin' covers an equal part of the spectrum. The frequency resolution of the FFT is thus equal to: 1/(N/2) * sampling frequency.
- The DFT (and therefore the FFT) is a complex transform and thus has a real and imaginary part. When transformed by the DFT, real signals (eg. speech signals) have an even real part and an odd imaginary part.
- The spectrum is often represented by combining the real and imaginary components of the DFT to get the magnitude and phase in the following way:

$$|X(k)| = \sqrt{[Re\{X(k)\}]^2+[Im\{X(k)\}]^2}$$
$$\textit{phase angle } \phi = \arctan\frac{Im\{X(k)\}}{Re\{X(k)\}} \qquad (3.12)$$

  The magnitude is often expressed in dB as the *log magnitude*.

- Another way of expressing the spectrum is by the power spectrum (Parseval's Theorem [Opp90]), which is the square magnitude of the DFT ie. $|X(k)|^2$.
- The power spectrum can also be determined by padding the N samples of a segment of the signal with N zeroes and then taking the DFT of the 'short-time' autocorrelation sequence of the combined 2N samples [Opp90].

Formant Tracking

The magnitude or power spectrum can be used to single out the formant frequencies which exhibit greater intensity in certain regions of the spectra due to resonance in the vocal chambers. The formants are identified by peaks in the smoothed spectra and can be 'tracked' for the duration of the speech signal by linking from the specific formant peaks (ie. the 1st, 2nd etc.) of successive speech frames using 'short-time' analysis. Occasionally the formants merge (eg. the 2nd and 3th may become very close together in the spectrum), making it difficult to track them. Heuristics based on the context in which the merged formants are found, can be used to resolve them when they merge.

Cepstrum

The cepstrum is the spectrum of the logarithm of the frequency spectrum of a signal [Opp90]. The cepstrum c(n) of a segment of speech signal x(n) can thus be evaluated in the following way:

i. Calculate the short time DFT of a segment of the signal using a Hamming window function.
ii. Log the DFT spectral components.
iii. Take the inverse DFT (IDFT) of ii. The result is the cepstrum of the speech frame.

The original segment of the speech signal x(n) cannot be recovered from c(n) since it contains no separate information about the phase of x(n). A smoothed power spectrum and the pitch period can however be evaluated from c(n).

If the segment of x(n) from which the cepstrum is calculated is quasi-periodic, the cepstrum exhibits a distinguishable peak at the quasi-period of the signal.

The power spectrum is calculated by multiplying c(n) by a raised-cosine window (eg. Hamming) which isolates the points in the cepstrum below the pitch peak in the cepstrum. A DFT is then employed to these samples producing a smoothed power spectrum.

3.4 Linear Predictive Coding

3.4.1 Introduction

Linear predictive coding (LPC) is widely used in speech processing systems. Its ubiquity can be attributed to its ability to compress sample data into a compact form (see Makhoul [Mak85]) and yet still characterise the acoustic content of the speech signal with excellent results in recognition and synthesis applications.

The underlying principle in LPC analysis is embodied in the following:

> *the output (response) of a system can be written as a linear combination of the past outputs and the current and previous inputs (excitation).*

This idea has its roots in statistical prediction theory[1].

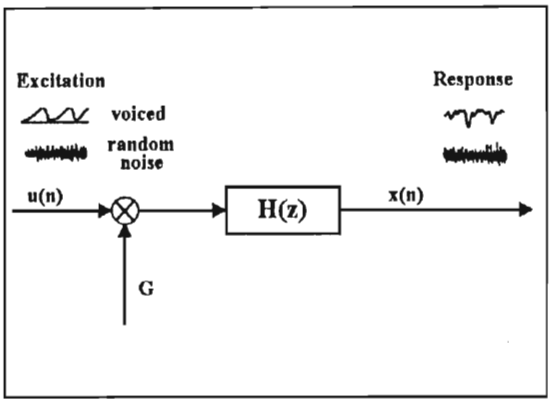

Figure 3.6 Linear-filter Model of the Speech Production Mechanism

Figure 3.6 depicts the filter model of the speech production system introduced in the last chapter. A source of excitation generates quasi-periodic impulses for voiced speech and/or random white noise for unvoiced speech. The vocal tract can be modelled by a single linear, time-varying filter. The speech produced as a result of the filtered sound source is called the response of the system.

The input of the system is denoted by u(n). For voiced speech, u(n) is a series of impulses ie. $u(n) = \delta(kT)$, where T a constant time period and k = 0,1,2,... For unvoiced speech, u(n) is white (or random) noise. The response of the filter to u(n) is the output signal denoted by x(n).

Using the filter model and the statistical prediction principle, the speech signal x(n) can be written as:

$$x(n) = \sum_{k=1}^{p} a_k\, x(n-k) + G \sum_{\ell=0}^{q} b_\ell\, u(n-\ell) \qquad (3.13)$$

that is, the present output is equal to the sum of the weighted past outputs and the weighted present and past inputs, where the a_k and b_ℓ are the linear coefficients of the equation and G is the gain factor.

The z-transform[2] of equation (3.13) characterises the frequency response of the system. Using the linearity and time-shift properties of the z-transform, equation (3.13) becomes:

---

[1] Makhoul relates the history of LPC in his tutorial on LPC [Mak75].

[2] The z-transform of a discrete signal is shown in equation (2.5) of chapter 2.

$$X(z) = \sum_{k=1}^{p} a_k z^{-k} X(z) + G \sum_{\ell=0}^{q} b_\ell z^{-\ell} U(z) \tag{3.14}$$

which can be rewritten as

$$X(z)\left[1 - \sum_{k=1}^{p} a_k z^{-k}\right] = U(z)\left[G \sum_{\ell=0}^{q} b_\ell z^{-\ell}\right] \tag{3.15}$$

The transfer function H(z) of a linear time invariant[1] system is given by the ratio of the z-transform of the system response to the z-transform of the system excitation. That is,

$$H(z) = \frac{X(z)}{U(z)} = \frac{G\, b_0 + G \sum_{\ell=1}^{q} b_\ell z^{-\ell}}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{3.16}$$

A special case of the above is the *all-pole filter*[2], achieved by restricting $\ell = 0$ which results in the transfer function being rewritten as

$$H(z) = \frac{G\, b_0}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{3.17}$$

where b_0 for convenience, is often incorporated in the gain factor, G.

In the time-domain, the all-pole model represents the output x(n) as a linear combination of the weighted past *p* outputs and the present input. That is,

$$x(n) = a_1\, x(n-1) + a_2\, x(n-2) + \ldots + a_p\, x(n-p) + G\, u(n) \tag{3.18}$$

The only *parameters* of the all-pole system are therefore the *p* linear coefficients a_k and the gain factor G (with the present input (b_0) included in G).

What is the effect of considering only the contribution of the poles to the system? Atal and Hanauer [Ata71] and Atal [Ata85] have reported on the contributions of both the poles and zeroes of the original filter model (see equation (3.17)) to the spectrum of the signal. The zeroes represent both local dips in the spectrum and changes in the spectral balance. The poles can effectively approximate the contribution of the zeroes to the changes in the spectral balance, though poorly represent the local dips. In speech analysis however, the

---

1 The system can be thought of as time invariant if 'short time analysis' of the speech signal is used.

2 The all-pole model is called the autoregressive model by statisticians.

dips in the spectrum are not as important as are the peaks (which indicate the formants) and the spectral balance, both of which can be represented by poles. Thus, the all-pole filter is ideal for speech recognition systems.

Due to the time variant nature of speech, the parameters of the model are recalculated at intervals (perhaps overlapping) of about 10-45ms. The gain factor is necessary to determine the pitch period, but for simply identifying the speech sounds can be ignored.

3.4.2 Calculating the Linear Predictor Coefficients

In speech recognition (unlike speech production) the excitation source u(n) and gain factor G are unknown. By discarding Gu(n) term from equation (3.18) it is possible to approximate x(n) by $\tilde{x}(n)$ in the following manner:

$$x(n) \approx \tilde{x}(n) = \sum_{k=1}^{p} a_k \, x(n-k) \tag{3.19}$$

The *prediction error* ε (also called the *prediction residual*) is the difference between the actual sample value x(n) and the predicted or approximated value $\tilde{x}(n)$, that is

$$\begin{aligned} \varepsilon(n) &= x(n) - \tilde{x}(n) \\ &= x(n) - \sum_{k=1}^{p} a_k \, x(n-k) \end{aligned} \tag{3.20}$$

It is evident that ε(n) in the equation (3.20) is equal to the Gu(n) term in equation (3.18).

The total squared prediction error denoted by E, is the sum of the squared prediction residual over the speech frame interval $[n_0,N]$ ie.

$$E = \sum_{n=n_0}^{N} \varepsilon_n^2 = \sum_{n=n_0}^{N} [x(n) - \tilde{x}(n)]^2 \tag{3.21}$$

To find the p optimal linear predictive coefficients a_k which minimise the total squared error E, the partial derivative of E with respect to *each* of the predictive coefficients a_k is equated to zero and solved, yielding

$$\frac{\partial E}{\partial a_i} = \frac{\partial \sum_{n=n_0}^{N} \left[x(n) - \sum_{k=1}^{p} a_k \, x(n-k) \right]^2}{\partial a_i} = 0, \qquad 1 \le i \le p \tag{3.22}$$

Simplifying the above gives

$$\sum_{n=n_0}^{N} x(n)\ x(n-i) = \sum_{k=1}^{p} a_k \sum_{n=n_0}^{N} x(n-k)\ x(n-i), \qquad 1 \le i \le p \tag{3.23}$$

The minimum total squared error E_p is determined by expanding equation (3.21) and substituting (3.23) into it, resulting in the following:

$$E_p = \sum_{n=n_0}^{N} x(n)^2 + \sum_{k=1}^{p} a_k \sum_{n=n_0}^{N} x(n)\ x(n-k) \tag{3.24}$$

There are p linear equations with p unknowns (ie. the a_k) expressed in equation (3.23). The predictive coefficients a_k can be evaluated by the well-known elimination or Crout reduction method. These methods require $p^3/3+O(p^2)$ operations (multiplications and divisions) and p^2 storage space [Mak75]. Two other methods, named after their inventors *Levinson* and *Durbin*, exploit special properties of the linear equations in (3.23) which reduce the computational effort of the traditional solutions for finding the unknowns in linear equations.

Autocorrelation Approach

This approach derives its name from the fact that the linear equations (3.23) can be rewritten in terms of the short time autocorrelation function described in (3.9). That is,

$$\sum_{k=1}^{p} a_k\ R(i-k) = R(i), \qquad 1 \le i \le p \tag{3.25}$$

E_p in equation (3.24) can be written in terms of the short time autocorrelation function and becomes

$$E_p = R(0) + \sum_{k=1}^{p} a_k\ R(k) \tag{3.26}$$

Equation (3.25) can be written in expanded matrix form as

$$\begin{bmatrix} R_0 & R_1 & R_2 & \cdots & R_{p-1} \\ R_1 & R_0 & R_1 & \cdots & R_{p-2} \\ R_2 & R_1 & R_0 & \cdots & R_{p-3} \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ R_{p-1} & R_{p-2} & R_{p-3} & \cdots & R_0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \cdot \\ \cdot \\ \cdot \\ a_p \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \cdot \\ \cdot \\ \cdot \\ R_p \end{bmatrix} \tag{3.27}$$

This matrix has both the Toeplitz (ie. the elements along any diagonal are equal) and symmetric properties. This system of equations can be efficiently solved by either Levinson's or Durbin's recursive methods.

Levinson's Method

The following are Levinson's recursive procedures adapted from [Ata85]:

$$a_k^{(i)} = a_k^{(i-1)} - a_i^{(i)} a_{i-k}^{(i-1)}, \quad 1 \le k \le i-1, \quad 2 \le i \le p, \tag{3.28a}$$

where

$$a_i^{(i)} = \frac{[R(i) - \sum_{k=1}^{i-1} R(|i-k|) a_k^{(i-1)}]}{[R(0) - \sum_{k=1}^{i-1} R(k) a_k^{(i-1)}]} \tag{3.28b}$$

Durbin's Method

This method makes use of the observation that the column vector on the right hand side of (3.27) contains the same elements that are found in the autocorrelation matrix on the left hand side. It is twice as fast as Levinson's method requiring $2p$ storage spaces and $p2+O(p)$ operations [Mak75].

The following equations describe Durbin recursion (after [Mak75]):

$$
\begin{aligned}
E_0 &= R(0) \\
k_i &= \frac{\left[R(i) + \sum_{j=1}^{i-1} a_j^{(i-1)} R(i-j)\right]}{E_{i-1}} \\
a_i^{(i)} &= k_i \\
a_j^{(i)} &= a_j^{(i-1)} + k_i a_{i-j}^{(i-1)}, \qquad 1 \le j \le i-1 \\
E_i &= (1-k_i^2) E_{i-1}
\end{aligned}
\tag{3.29a}
$$

The above equations are solved recursively for i = 1,2,...,p. The a_k are finally determined by

$$a_j = a_j^{(p)}, \quad 1 \le j \le p \tag{3.29b}$$

Covariance Approach

The covariance approach is similar to the autocorrelation except that equation (3.23) is written in terms of the autocovariance function φ rather than the autocorrelation function $R_n(\ )$, giving:

$$\sum_{k=1}^{p} a_k \varphi_{ki} = \varphi_{0i}, \quad 1 \le i \le p \tag{3.30}$$

The minimum total squared error E_p (see equation (3.24)) can also be written in terms of the autocovariance function φ:

$$E_p = \varphi_{00} + \sum_{k=1}^{p} a_k \varphi_{0k} \tag{3.31}$$

φ is symmetric, ie. $\varphi_{ik} = \varphi_{ki}$ but not Toeplitz. It is not difficult to see that as the interval extends to infinity, the covariance approach reduces to the autocorrelation approach. In 'short-time' analysis, the covariance matrix for each speech frame uses sample values in the range $n_0\text{-}p \le n \le N$ to avoid problems at the left frame boundary. The autocorrelation matrix on the other hand, sets all values outside the speech frame to zero. The system of equations resulting from the covariance approach can be efficiently solved using Cholesky decomposition [Mak75].

To summarise the advantages and disadvantages of both approaches, Markhoul notes that:

> "The covariance method is quite general and can be used with no restrictions. The only problem is that of the stability of the resulting filter, which is not a severe problem generally. In the autocorrelation method, on the other hand, the filter is guaranteed to be stable, but problems of parameter accuracy can arise because of the necessity of windowing (truncating) the time signal. This is usually a problem if the signal is a portion of an impulse response." [Mak75]

Lattice or Partial Correlation (PARCOR) Method

For completeness, the lattice or PARCOR method is given. Veeneman describes this method in the following way (also see [Row92b]):

> "In both the autocorrelation and covariance methods of LPC, the processing has two stages: the calculation of the correlation matrix, and the solution of the resulting set of linear equations. However, in lattice methods, the two stages have been effectively combined into a recursive procedure for determining the LPC parameters, where not just one but p forward and p backward linear predictors are used and the parameters are calculated one stage at a time." [Vee88]

A full description of how they are obtained is detailed by Makhoul [Mak75]. The PARCOR coefficients are found to be the intermediate terms $a_i^{(i)}$ in equations (3.28a) and (3.28b). Similarly they can be derived from the intermediate terms of Durbin's recursive method in equations (3.29a) & (3.29b). The predictor coefficients a_k are therefore directly obtained by the PARCOR coefficients.

Linear Prediction in Spectral Estimation

LPC parameters can be employed to approximate the signal power spectrum P(w) by the all-pole model approximation of the power spectrum, $\tilde{P}(w)$.

$$\begin{aligned} \tilde{P}(\omega) &= |H(e^{j\omega})|^2 \\ &= \frac{G^2}{|A(e^{j\omega})|^2} \end{aligned} \tag{3.32}$$

The gain G is shown by Makhoul [Mak75] to be related to E_p in the following way:

$$G^2 = E_p = R(0) + \sum_{k=1}^{p} a_k R(k) \tag{3.33}$$

and can thus be easily calculated while the $|A(e^{jw})|^2$ term in equation (3.32) can be determined by the magnitude squared of the Fourier transform of the sequence of predictor coefficients $\{1, a_1, a_2, ..., a_p\}$. By appending zeroes to the end of this sequence, the frequency resolution can be improved.

Number of Poles

It has been shown [Ata85] that the greater the number of predictor coefficients the better the model parameters and thus the smaller the recognition error rate. However, it was also shown that the increase in the number of parameters p resulted in only very small improvements of the model parameters greater than some value p_0. For greater values of p, greater processing effort is required. Atal found that at least 2 poles are required to represent each 1kHz region in the spectrum of the signal [Ata85]. An additional 4-5 poles are necessary for the spectral balance [Opp90]. Thus, for a signal bandwidth of 10kHz, between 14 and 15 poles are required for the model.

Cepstral Coefficients using Linear Prediction Analysis

The cepstral coefficients can be easily determined from the linear predictor coefficients, a_k. The cepstral coefficients c_n are defined by the following iterative definition [Mak75]:

$$c_n = a_n - \sum_{m=1}^{n-1} \frac{m}{n} c_m a_{n-m}, \quad 1 \le n \le p \tag{3.34}$$

The cepstral coefficients are a useful representation of the linear predictor coefficients since the "distance" between two speech sounds represented by their cepstral coefficients is computationally less expensive (because can use the Euclidean distance measure - see chapter 4) to express and determine than their equivalent linear predictor coefficients, a_k (which require the Itakura-Saito distance measure). These ideas are elaborated on in chapter 4.

Formant Tracking using Linear Prediction

Formant tracking has already been mentioned with respect to Fourier analysis. On a spectrogram, the formants are easily identifiable as those regions on the frequency axis with a darker, denser shade. The peaks in the approximation of the power spectrum of the signal, given by equation (3.32), can be used to approximate the formant frequencies. The peaks of the resulting power spectrum are the formants. If the formants merge, heuristics can be used to resolve them from their context. See [Ata85] for further details.

Linear Prediction in Pitch Analysis

It has already been mentioned that the pitch period can be determined from the linear predictive coefficients. The techniques to achieve this are not investigated here but Atal [Ata85] reports on several different techniques using the LPC analysis.

Chapter 4 - Segmentation and Classification

4.1 Introduction

The lower levels of speech recognition involve the following aspects:

- sampling and preprocessing (see chapter 3)
- segmentation
- classification

In this chapter the problems of segmentation and classification of speech sounds are investigated.

Segmentation is the process of dividing the speech signal into distinct, *quasi-homogeneous acoustic regions*. Looking at figure 4.1, the human eye can segment the (graphical form of the) signal into regions of similarity with a high degree of accuracy. Classification, on the other hand, is the process of *identifying the different speech sounds* in the signal.

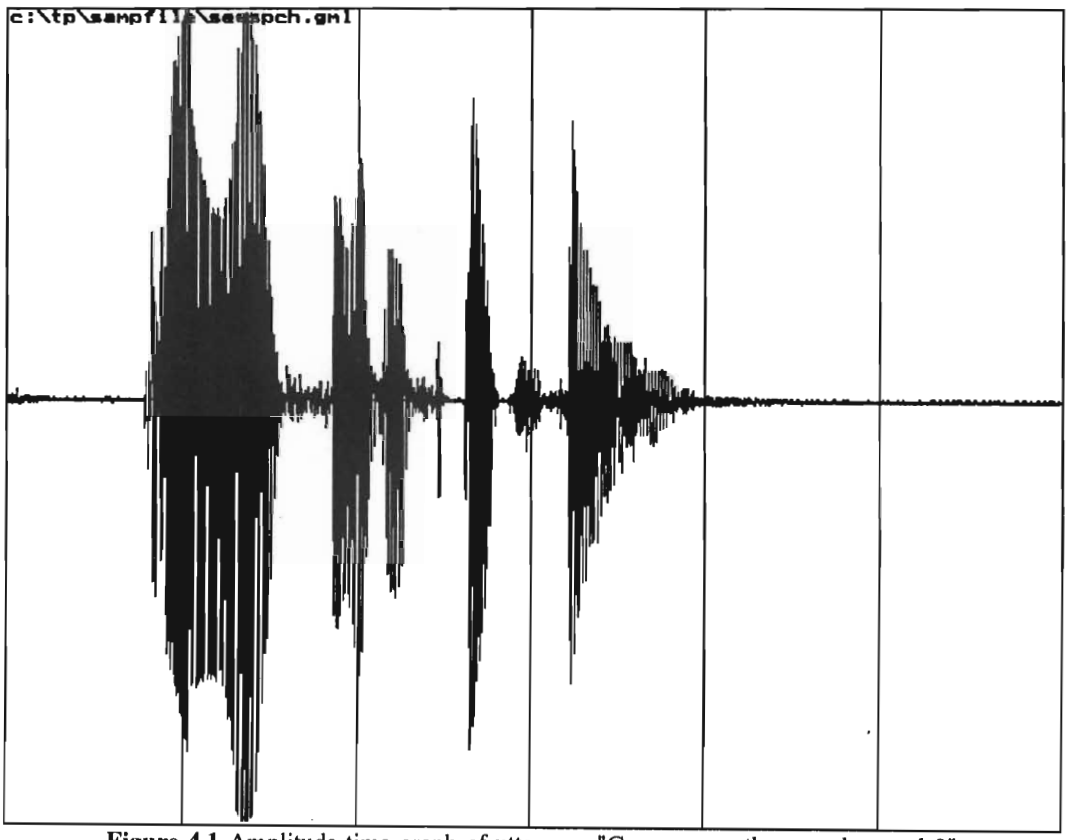

Figure 4.1 Amplitude-time graph of utterance "Can you see the speech sounds?"

How is segmentation related to classification? There are two possibilities: one, the signal is first segmented, and then these segments are classified into speech sounds; and two, successive speech frames of the signal are classified as speech sounds and segmentation occurs naturally as a result of this classification.

In this chapter, the emphasis is on the *subword speech sound unit* - what it is, how to detect and label it and how it interacts with its neighbouring sounds. These issues form the basis for a continuous phoneme (or some other subword sound unit) recogniser. An example of such a phoneme recogniser is presented in chapter 7. In chapters 5 and 6, two different approaches to speech recognition are studied: template matching and stochastic hidden Markov model recognition techniques. These techniques in essence deal with determining larger speech units like words or groups of words eg. BIGRAMS [Bro91]. For simplification, one can think of chapter 4 as concentrating on speech sound recognition while chapters 5 and 6 on word recognition.

4.1.1 Preprocessing Notation

The preprocessing stage condenses the large number of digital samples into representative speech features, extracted by the temporal and/or spectral preprocessing functions discussed in the previous chapter. Speech features are easy to analyse and manipulate and characterise speech information like speech sounds and speaker information (gender, age, dialect, tone etc). In order to represent the time variant nature of speech, speech features are evaluated for successive, abutted or overlapping speech frames ('short-time' analysis technique).

The speech features of each speech frame are represented by a feature vector α_{ti} where t denotes the speech frame index in time and i references one of the p speech feature measurements called *speech parameters* eg. the LPC coefficients. By representing the speech signal in a speech frame as a vector, the speech sound associated with that partial signal can be thought of as being situated in the feature vector space of dimensionality p, the number of speech parameters. The speech parameters should be chosen such that they group speech sounds of the same speech class (eg. phoneme) in the same region of the feature vector space. On the other hand, speech sounds from different classes should be separated in the feature vector space. Devijver and Kittler express this idea succinctly:

> "[One has to] extract from the raw data the information which is most relevant for classification purposes, in the sense of minimizing the within-class pattern[1] variability while enhancing the between-class pattern variability." [Dev82].

The authors of the above quotation also point out that many of the problems with classification can be attributed to the failure to establish robust (ie. characteristic) speech

[1] A *pattern* can refer to a single feature vector (as it does here) or a set of feature vectors grouped together in a word or sentence (see chapters 5 and 6).

parameters. The performance of a set of speech parameters can be determined either by empirical recognition results or by measuring the variances of the within- and between-class distances of various feature vectors. If the system results in large between-class and relatively small within-class variances, then it is considered to have robust parameters.

The number of speech parameters depends on three factors, namely: time taken to compute them, the space required to store them and the ease with which they can be implemented [Nie81].

4.2 Segmentation

One can segment the speech signal in two ways, either by grouping the signal into regions of acoustic similarity or by detecting the boundaries which separate one speech sound from another. These approaches are comparable with region and boundary extraction in image-processing [Nie81]. The first merges neighbouring frames which preserve the acoustic character of the segment. The second places segment boundaries between speech frames with significantly different speech parameters.

One can segment before or after (as a result of) classification. By first segmenting, one need only classify one of the possibly many speech frames in each segment, thus reducing the time-costly task of classifying each speech frame. The efficiency of the segmentation procedure should be as inexpensive as possible in order to make this option viable. In other words, it is pointless performing the same costly calculations used in classification on the speech frames when segmenting since one might as well classify each segment and perform segmentation as a postprocess. Consequently, the computationally less expensive temporal preprocessing functions are usually used with simple thresholding techniques to determine the segment boundaries. There are several problems with this approach.

Firstly, segmentation is seldom 100% accurate [Wai88]. Even a human expert has difficulty pin-pointing the segment boundaries. Two types of segmentation errors can occur: a segment boundary could be omitted (undersegmenting) or erroneously added (over-segmenting). Undersegmenting has catastrophic results - segments are lost forever and speech sounds in the lost segments are subsumed into other speech sounds. The consequences are less heinous in oversegmenting. Oversegmented parts of the signal can be merged with their same neighbouring segments during post-processing.

Secondly, classifying a pre-segmented signal introduces the problem of which part (ie. speech frame) of the segment to use for classification. The problem is augmented by

coarticulation and the continual varying of the articulators in speech. As a result, a speech sound is not constant during a segment. The middle of the segment is probably the most representative of the speech sound, where the sound is most stable and well-formed; with the exception of diphthongs and plosives. Diphthongs are recognised by the variation in the sound production over the segment. Plosives, on the other hand, include a silent period (stop) when air pressure is built up behind a point of constriction before releasing the pressure with a burst or small 'explosion'.

Waibel describes and compares the performances of several segmenters [Wai88]. One of these, the ZAPDASH segmenter is discussed in detail below.

4.2.1 The ZAPDASH[1] Segmenter

The ZAPDASH segmenter was part of the front-end preprocessor of the Harpy[2] speech understanding system. The aim of this segmenter, according to Gill (one of the author's of the ZAPDASH system), was not to determine phonemic boundaries but to break the utterance into nearly uniform acoustic segments [Wai88]. The speech signal is low-pass (or smoothed) and high-pass filtered and then preprocessed by the zero-crossing and peak-to-peak amplitude functions. Frequency information is extracted using the zero-crossing function whereas the peak-to-peak amplitude function gives an indication of the energy or intensity in the speech frames of the signal. Both these temporal preprocessing functions are computationally inexpensive, making them suitable for a segmentation procedure.

The ZAPDASH algorithm executes eight left-to-right passes of the preprocessed speech signal in a hierarchical manner such that each pass performs a more specific segmentation of segments determined by the previous passes in a 'divide-and-conquer' manner. For each pass, a set of heuristic decision rules are applied to determine the segment boundaries. The eight passes are listed below [Wai88]:

1. Separate silence from sound using thresholds on the differenced zero-crossing and smoothed peak-to-peak amplitude functions (after this step, segments of silence are ignored)
2. Separate non-silence segments into fricative and vocalic regions using thresholds on speech parameters

---

[1] ZAPDASH = Zero-crossing And Peak-to-peak amplitude of Difference And SmootHed data [Wai88].

[2] The HARPY system is a well-known speech recognition system, developed at Carnegie-Mellon University as part of ARPA's speech recognition project in the 1970s. See chapter 9 and [Low77].

3. Correction rules and merging of similar regions created by oversegmenting in steps 1 and 2
4. Detect and separate low amplitude regions (especially for determining weak fricative sounds "f", "th" in *think* not *that* and "h")
5. Correction rules for oversegmenting in 4
6. Detect and separate regions of aspiration
7. Detect nasal and liquid sounds from significant dips in the amplitude function in vocalic regions
8. Split large segments with significant changes in speech parameters

A segmentation algorithm based on the ideas of the ZAPDASH segmenter was attempted in this project. The performance of the segmentation algorithm is graphically demonstrated in chapter 7. The graphical results expose the rather poor performance of the segmenter (when compared against a human expert). This result is consistent with those of similar systems reported in the literature. For example, Waibel [Wai88] reports that the ZAPDASH segmenter achieves between 5-15% error rates when segmenting into broad phonetic classes. All of the other types of segmenters reported by Waibel are also prone to poor results although several (eg. the perceptron classifier) achieved lower average error rates than the ZAPDASH segmenter.

4.3 Classification

This section focuses on the classification of frame-sized, preprocessed segments of speech into speech sound units eg. phonemes. Once classified, a post-processor would merge the labelled segments into syllables, words and finally sentences. These stages group together to form a continuous speech recognition system.

In order to classify a speech feature vector, one must have prior knowledge about the speech sound classes, particularly information about where the sounds of each class are situated in the feature vector space. A *training or learning phase* is obligatory in any speech recognition system to 'learn' what constitutes a sound from a particular class. The details of the training phase follow the section on classification. A good analogy of the learning phase is found in the acquisition of language by an infant. The infant does not automatically speak but listens, observes and practices the various sounds in a lengthy and gradual learning period.

Classification can therefore be defined as the process of matching an unknown speech pattern against a set of known reference speech patterns (or models - see chapter 6).

4.3.1 Direct Matching

A naive pattern matching technique is to represent the speech pattern as an array of sample values and match it against a set of reference speech patterns (one pattern for each sound class) represented in the same way. The problem with this approach is that speech patterns are highly variable. The same word, spoken by the same speaker more than once, will most probably never be exactly the same. Even if two utterances were identical, slight time shifts would result in a mismatch. Hence, pattern matching techniques require the preprocessing stage where the robust parameters are used to describe the 'short-time' characteristics of the signal and are encoded as speech feature vectors.

4.3.2 The Classification Problem

Classification of speech patterns is not straight forward for several reasons. Apart from the noise in the signal, speaker variations and coarticulation; the speech sounds themselves are often distinguished from one another by small adjustments in the vocal apparata, resulting in subtle (almost negligible) differences in the acoustic make-up of the signal. In fact the human sound classification system has been shown to be less than 100% accurate when sounds are spoken in isolation (ie. without an acoustic context) [War83]. Humans tend to rely heavily on the context in which the sounds are made and on information from the higher levels of speech processing in order to determine the identity of acoustically similar sounds. It seems as if a perfect classifier will never be achieved. Instead the classifier must have the ability to determine the best options and moreover, must be postprocessed by a sophisticated system which can resolve the identity of acoustically similar segments from their context. Because of the uncertainty involved in classifying a speech sound, a statistical or 'fuzzy' approach for determining the most likely options is used in most speech systems.

The above argument is not only true of speech classification but also to many other pattern recognition problems in the world [Nie81] eg. image processing, meteor-burst classification etc.

4.3.3 Distance Measures in the Pattern Space

It has already been said that speech patterns from the same class should be clustered close together in the same region of the pattern space whereas patterns from different classes should be separated in the pattern space. Distance between patterns in the feature space is used to determine how 'closely' they are related to each other and to which sound class they belong. The distance between patterns of the same class should therefore be small,

while the distance between patterns of different classes much larger. The distance measures most frequently used in speech classification are presented below[1].

In this thesis, the notation used for any non-specific distance measure between two speech patterns x and y is $\delta(x,y)$.

A distance measure is defined to have the following properties (after [Row92]):

i	Symmetry	$\delta(x,y) = \delta(y,x)$
ii	Triangle inequality	$\delta(x,z) \geq \delta(x,y) + \delta(y,z)$
iii	Positive definiteness	$\delta(x,y) > 0$ for $x \neq y$
		$= 0$ for $x = y$

and the following properties are desirable, although not essential

iv should be computationally efficient

v should be physically meaningful ie. for speech, the distance measure should relate to the perceptual 'distance' between sounds

The *Euclidean distance* measure between two vectors x and y is given as:

$$\delta_E(x,y) = \left[\sum_{j=1}^{p} (x_j - y_j)^2 \right]^{1/2} = [\, (x-y)^t \, (x-y) \,]^{1/2} \qquad (4.1)$$

where t denotes the transpose of the column vector using matrix representation. Note that if the speech parameters are not orthogonal, the Euclidean distance can be misleading. For example if 15 speech parameters are used (14 LPC coefficients + the average amplitude of the speech frame), the average amplitude value is typically greater than 1 whereas the LPC coefficients are less than or equal to 1. A slight difference (say 5) between the average amplitude features of the two speech feature vectors being compared will result in a large distance, regardless of the LPC coefficients, since their contribution to the distance is a value less than 1. One would hardly use an amplitude feature (which monitors loudness) with spectral features though the above example does illustrate the necessity to normalise the feature values.

Although the Euclidean distance is easily and efficiently computable (it is not necessary to perform the costly square root of the sum of the squared differences since comparatively they are the same), it treats all the speech feature parameters as if each carried the same perceptual significance. Certain of the speech parameters tend to be more significant than others and therefore a weighting factor needs to be introduced to cater for this.

---

[1] A comprehensive list of the best known distance measures is given in [Dev82].

The *Mahalanobis distance* measure overcomes the problem of features having different perceptual importance by weighting each of the features differently using a scaling matrix W^{-1}:

$$\delta_M(x,y) = (x-y)^T W^{-1} (x-y) \tag{4.2}$$

The matrix W^{-1} is the inverse of the auto-covariance matrix of the feature vector y. The auto-covariance matrix is only meaningful if y is a reference feature vector since W^{-1} must be constructed from a set of feature vectors representing that sound class. This matrix can be thought of as providing an indication of the importance of the speech features with respect to each other by means of a weighting factor. A separate matrix W^{-1} is required for each reference feature vector y, but in practice often a general auto-covariance matrix is computed of all the reference vectors. The Mahalanobis distance reduces to the Euclidean distance if W^{-1} is the identity matrix.

If the speech features are LPC coefficients, either the Euclidean or Mahalanobis distance measures can be used although another distance measure, the *Itakura-Saito* (a log likelihood measure), was specifically developed for linear predictive analysis. It has proved to be significantly better than Euclidean measures for LPC coefficient features [Ita75]. Since LPC coefficients are used as features by many speech systems, this distance measure is widely used eg. by [Pit90] [Toh87] [Rab85a] and [Rab85b].

The Itakura-Saito distance measure is given as

$$\delta(x_i,y_j) = \log \frac{(a^j)^T R^i (a^j)}{(a^i)^T R^i (a^i)} \tag{4.3}$$

where:

- (a^i) is a column vector of the set of linear predictive coefficients of the i^{th} frame of the test input pattern x,
- (a^j) is a column vector of the set of linear predictive coefficients of the j^{th} frame of the reference pattern y,
- R^i is the autocorrelation matrix of the i^{th} frame of the input pattern x (depicted in equation (3.27) in chapter 3)
- T denotes the transpose of the column vectors

Notes on the implementation of this distance measure can be found in [Ita75] and [Pit90]. A major disadvantage of this measure is that it is computationally expensive. It has been shown that minimising the Euclidean distance in the cepstral domain is equivalent to minimising the Itakura-Saito distance in the frequency domain [Vic87]. In addition, the

cepstral coefficients are derived with little extra overhead from the LPC coefficients (see chapter 3). Empirical results eg. [Rab89] [Toh87] show that cepstral analysis using the Euclidean distance measure is computationally less expensive and produces only marginally poorer results than LPC analysis using the Itakura-Saito measure. As a result, the cepstral coefficients, or variations of them (see [Rab89] and [Toh87]), are also a popular choice of speech features [Ney92] [Ken90] [Shi86].

4.3.4 Classification using Nearest Reference Pattern

Consider a set of sound classes C_i $1 \le i \le L$, where L is the total number of sound classes in the feature space. During training, one can calculate a reference vector r_i associated with each class C_i situated in the feature vector space.

Classification of an unknown speech vector x is undertaken by choosing the class C_i with the minimum distance between x and its reference vector r_i for $1 \le i \le L$. This is written mathematically as

$$\hat{w}(i) = \underset{i}{\mathit{argmin}}\ \delta(x, r_i), \qquad 1 \le i \le L \tag{4.4}$$

where the argmin function returns the value of the argument which results in a minimum distance between x and one of the r_is.

Each reference pattern r_i can be determined by finding the mean vector of an independent set of training vectors categorising a particular class C_i ie.

$$r_i = \frac{1}{n_i} \sum_{k=1}^{n_i} \alpha^{(k)} \tag{4.5}$$

where n_i is the number of training vectors for class C_i.

The problem with this method is that the training vectors can be quite spread out in a particular region of the feature vector space especially if the set of training vectors is made up from a group of speakers. The mean vector may not necessarily be a good reference point for a particular input vector. One solution is to establish several smaller clusters of vectors within a class and use the mean vectors of these clusters as reference points for classifying an input vector. Another solution, requiring greater computational effort, is to use all the training vectors as reference vectors and then to apply a nearest neighbour classification rule described in the next section.

4.3.5 Nearest Neighbour Classification Rules

The nearest neighbour rule (abbreviated 1-NNR) states that an unknown pattern x can be classified according to the class of the training pattern y closest to it in the feature space, populated with correctly classified training patterns. "Closeness" is determined by a distance measure $\delta(x,y)$. The 1-NNR is prone to incorrect classification if the closest training vector happens to be an outlier found in the region of another class. To combat this problem, the *k*-nearest neighbour rule (*k*-NNR) determines the class of an unknown pattern x by choosing the class of the majority of its *k* nearest neighbours $y_1, ..., y_k$. A threshold value can be set to determine the percentage majority of the *k* nearest neighbours that must belong to the same class in order to result in a class matching. This results in a more robust classification system but will require heuristic decisions to resolve the case where a majority is not obtained.

An algorithm to classify patterns using the k-NNR rule is as follows:

1. Gather a training set of T correctly classified patterns $Y = \{y_i | \ 1 \le i \le T\}$.
2. Generate another (statistically independent) set of patterns from which to select the test pattern x.
3. Determine the *k* patterns (from Y) closest to the pattern x (ie. the *k* $y_i \in Y$ which result in the smallest distances $\delta(x,y_i)$ between themselves and x.
4. The class membership of x is determined by choosing the class with the most "votes"[1] from the *k* nearest patterns selected in step 3. A threshold percentage majority of "votes" ℓ below which x will not be classified by this method and will require further heuristics for its classification, can be incorporated to provide a more robust classifier.

4.3.6 Classification using a Statistical Approach

The classification problem can be stated in the following way:

> Given a set of L mutually exclusive classes C_i $1 \le i \le L$, an unknown input pattern x belongs to the class C_k if the probability of x being in C_k is greater than the probability of x being in any other class C_i $i \ne k$.

[1] Each of the *k* patterns "votes" for the class to which it belongs. Consequently the class with the most "votes" (the "winner") determines the class of the test pattern $\underline{x}$.

4.3.6.1 Bayes Decision Rule

Bayes decision theory can be used to determine the probability that x is in some class C_i by the following:

$$P(C_i|x) = \frac{P(x|C_i)\, p_i}{\sum_{j=1}^{L} P(x|C_j)\, p_j} \qquad (4.6)$$

where p_i is the *a priori* probability that pattern x belongs to class C_i. The denominator term in equation (4.6) is called the *unconditional probability density function* which determines the distribution of pattern x in the feature space. The major drawback of the Bayes rule is that it relies on prior knowledge, both the *a priori* probabilities and prior knowledge of the probability density function of the test pattern x. As a result, other techniques are introduced in this chapter to approximate the Bayesian decision surface[1] and to introduce contextual information into the decision making process.

4.3.6.2 Markovian Stochastic Principles

When people listen, they use contextual information to predict and later to verify speech events (eg. sounds, words and even ideas) from the current and previous speech events. Contextual information can be interpreted statistically by Markov models. Markovian-based word recognisers are investigated in chapter 6. In this section, the general Markovian principles are presented with respect to the classification problem.

A Markov model is a network of *states* linked to other states by arcs. There are two ways of interpreting such a model. The first and traditional method is to consider the states as physical (speech) events while the arcs represent the probability of traversing from one event to another, joined by the arc. In the second method, the arcs themselves represent the events. The latter method is not considered further, yet it is worth noting that the IBM continuous speech recogniser using a maximum likelihood approach used this model [Jel83].

An *ergodic* model is one in which every state can be reached from every other state. A simple example of an ergodic model is one which is *fully connected*; that is, where every state is connected to every other state [Ios80].

---

[1] A decision surface is a plane in the p dimensional feature vector space, separating one sound class from another.

The *order* of the Markov model determines how many of the previous events influence the current event. A first order Markov model is said to exhibit the *Markov property* which simply stated, means that the current event can be determined by the previous event. First order Markov models are often used because they are less complex to manipulate [Ios80].

The context of speech sounds is dependent on time. For example, the "n" sound (in *nun*) is seldom if ever followed by "z" (in *zoo*) at the beginning of a word. At the end of the word however, "n" followed by "z" is common due to plurality of nouns ending in "n" (eg. *balloons*) and the third person singular "z" suffix of verbs ending in "n" (eg. *he runs*). This complicates matters considerable with the result that Markov models are generally only used if the application is independent of time. If the model is time independent, only one set of *transition probabilities* (ie. the set of probabilities of traversing between any two states in the network) need be determined.

By knowing the state transition probabilities, the Markov model can be used to predict or verify a sequence of events and thus be used as a classifier.

The above description of Markov models should be seen as an overview and introduction to a more detailed analysis developed in chapter 6. Because speech sounds do vary with time, the model presented above is not sufficient to describe the contextual dependencies between speech sounds. Therefore in chapter 6, the hidden Markov model is described to overcome this problem. The Markovian principles presented above can be used for syntax modelling where the parts of speech are related to one another without being dependent on time.

4.3.7 Problems with Statistical Decision Theory

The main problem with the Bayes rule and the Markovian statistical classifiers is that the *a priori* probabilities are assumed to be known beforehand (as the Latin indicates), as are the underlying probability distributions. Unfortunately in real applications like speech classification, they are not generally known. As a result, statistical sampling ie. approximating these unknowns from a large number of known and correctly classified patterns, is used to determine these probability values. Most speech classifiers use information from statistical sampling gathered in a training phase. It is essential that the training phase should use a set of data which is *statistically independent of the testing data.* The training phase is discussed later.

4.3.8 Fuzzy Approach to Pattern Recognition

There is strong motivation for a fuzzy approach to speech recognition. Its strength is highlighted by the following argument. Speech patterns are not so difficult to classify because of random noise in the speech signal, but rather as a result of the intrinsic vagueness or fuzziness of speech due to a kaleidoscope of factors like the speaker's identity (age, sex, genetics, environment), the speaker's physiological and psychological status (health, natural or temporary impediments and emotions) and the mood, context and intentions of the speaker.

Pal and Dutta Majumder differentiate between the fuzzy membership function and the probability density function in the following way:

> "Probability is about how frequently a sample occurs in a population while fuzzy membership value means how closely or how accurately a sample resembles an ideal element (including its contextual informational sense) of a population." [Pal86]

From the above, the fuzzy approach seems well suited to the speech recognition domain. Pal used a fuzzy-approach to classify Telegu (an Indian language) vowels into ten vowel classes [Pal82]. The system used only the first three formants as speech features. Classification accuracies of about 72-75% were reported. The comparatively poor results can probably be attributed to the small number of non-robust features used.

The fuzzy approach to pattern recognition is beyond the scope of this thesis. The interested reader should see Zadeh [Zad88] for an introduction to the concepts of fuzzy logic, and [Pal82] and [Pal86] with respect to speech recognition.

4.3.9 Connectionist Classifiers

The human brain has for many years been the model and inspiration for building 'intelligent' systems. Expert systems for example draw on the organisational and rational models that the human mind exhibits. In a more direct way the human brain has inspired research in the field of neural networks or more correctly, connectionist architectures. Connectionist architectures aspire to constructing an artificial brain using a computer by simulating (what is thought to be) the neurons of the human brain.

The motivation for using a "neurological" model is expressed perfectly by Knight:

> "Another thing people seem to do better than computers is handle fuzzy situations. We have very large memories of visual, auditory, and problem-solving episodes, and one key operation in solving new problems is finding closest matches to old solutions. Inexact matching is something brain-style models seem to be good at, because of the diffuse and fluid way in which knowledge is represented." [Kni90]

This matches quite closely the nature of the speech classification problem. The neural classification approach is not examined in this thesis though these techniques have great potential in speech sound classification applications. Rumelhart and McClelland [Rum87] provide a useful introduction to the field of neural networks. The results of several neural-based speech recognition systems are summarised as an indication of their importance in the general study of speech recognition.

The neural-inspired "phonetic typewriter" uses self-organising feature maps discussed later in the chapter, to segment (Finnish and Japanese) speech into a string of phonemes. The system is the first commercially available complete speech recogniser that employs neural networks. Kohonen reports impressive classification accuracies of between 96% and 98% [Koh88]. Its outstanding feature is its ability to reduce the system's dependency on a human phonetic expert using "self-organising maps" [Koh90].

Waibel et al [Wai89] used a neural network to tackle the problem of distinguishing between the notoriously confusable [b], [d] and [g] phonemes. They reported recognition accuracies greater than 80%.

In another project, DeMori et al [DeM90] have developed a neural network using the backpropagation learning algorithm to distinguish between ten vowel classes. Excellent results of 99.4% accuracy inspired them to extend the scope of the classifier to determine the broad distinctive features (see chapter 2) of vowels. The reported error rates of this application were between 2.5% and 10.5%.

4.4 Training

The training stage is often the bottle-neck in the development of the classifier because the patterns have to be manually and correctly labelled - a tedious task. A number of training techniques have been developed, the choice of which depends on the nature of the classification problem.

The goal of the training procedure is to determine the set of reference patterns which represent the classes in the feature space. In the following sections, two different training approaches are presented, the "supervised" and "unsupervised".

The difference between supervised and unsupervised training is in the content of the set of training patterns. If both the training patterns together with their correctly identified class information are given in the training set, then the training routine (which uses the known class information) is termed *supervised.* If the class information is not used or known during (the initial part of) classification, the technique is referred to as *unsupervised training* or *clustering.* Unsupervised techniques are used as far as possible in order to reduce the human effort in the training operation. In the next section, two unsupervised training algorithms are presented. Supervised training techniques, eg. linear discriminant functions and the least mean-squared-error procedure, can be found in [Dev82].

4.4.1 Introduction to Unsupervised Training

The training phase causes a bottle-neck in the development of the classifier because it typically requires many (in the order of thousands) patterns to be manually classified. It would be ideal if this task could be automated. But if this were the case, the classification problem would already be solved. Unsupervised training is a technique which eliminates as much of the tedious manual labelling component of training stage as possible. It will be seen however, that some (though substantially less) manual classification of training patterns is necessary. Two unsupervised training algorithms are presented here: the *k-means segmental algorithm* and the *Self-Organising Map* (SOM).

4.4.2 k-Means Segmental Algorithm

This algorithm is used in several of the systems developed in this thesis. The algorithm partitions the training patterns into L clusters C_j where $1 \le j \le L$. The distortion or distance between any two training patterns x and y in any cluster C_k is determined by one of the distance measure $\delta(x,y)$ described previously. The average distortion D_j between patterns within each cluster C_j is given by:

$$D_j = \frac{1}{L_j} \sum_{x \in C_j} \delta(x, M_j) \qquad (4.7)$$

where L_j is the size of the subset of training patterns $X_j = \{x: x \in C_j\}$ in cluster C_j, and m_j is the centroid of cluster C_j. The centroid of a cluster C_j is the feature vector m_j which minimises the average distortion of C_j. It can be shown [Mak85] that for the Euclidean

distance measure, the centroid m_j is the mean vector of the training feature vectors in cluster C_j given by

$$m_j = \frac{1}{L_j} \sum_{\forall x \in C_j} x \tag{4.8}$$

Clusters defined in this way are known as nearest neighbour clusters, Voronoi cells or Dirichlet regions [Rab89]. The k-means segmental algorithm as presented below is adapted from [Dev82], [Mak85] and [Rab89].

k-Means Segmental Algorithm:

1. Time index t is initialised to 0. The time index determines which training cycle (steps 2-4 represent 1 training cycle) is currently being executed. The set of T training patterns $X = \{x_i | \ 1 \le i \le T\}$ is partitioned into L clusters, either arbitrarily or based on some previous partitioning of training patterns. The centroid vectors $m_j(t)$ of each cluster C_j are then calculated.
2. Use the distance measure $\delta(x_i, m_j(t))$ to reclassify the training patterns x_i using the centroids $m_j(t)$ as reference vectors and applying a nearest neighbour rule (1-NNR) for reclassification ie. each x_i is a member of the cluster C_j whose centroid m_j is closest to it. Stated formally

$$x \in C_j \quad iff \quad \delta(x, m_j) \le \delta(x, m_i), \qquad \forall \ i \neq j \tag{4.9}$$

3. Increment the time index ie. $t = t + 1$ and redetermine the centroids $m_j(t)$ of all the newly-grouped clusters C_j.
4. If no change was made to the centroids in step 3 then terminate, else goto step 2 and redo steps 2-4 using the updated centroids.

This algorithm cannot be guaranteed to terminate, although in all the tested cases, termination did occur.

4.4.2.1 Vector Quantisation

The k-means segmental algorithm is the cornerstone of a speech compression or coding technique called *vector quantisation* (VQ) [Mak85]. In VQ, frame-sized segments of the speech signal (preprocessed using eg. LPC coefficients) are coded as one of L *code vectors* stored in a *codebook*. The code vectors do not correspond to specific sound categories (eg. phonemes) but are arbitrarily determined by the k-means segmental algorithm using a suitable set of training speech signals. Typically L is greater than 32 and can be as large as 256, compared with the approximately 40 phoneme classes in standard English. Each codevector thus represents a class of sounds from the set of training speech signals. If the training set is representative of all the sounds in the language being modelled, then these

codevectors can act as 'phonemes' for the recognition system. Therefore, VQ is often used as a preprocess to speech recognition systems (eg. [Rab89]), mapping the speech feature vectors of successive speech frames onto the codewords of the codevectors which are closest to them in the speech feature space. The procedure used in VQ is sketched below in order to highlight the unsupervised character of the k-means segmental algorithm.

VQ Procedure:

1. Construct the codebook of code vectors using the k-means segmental algorithm.
   - A set of training speech feature vectors (independent of those used in the actual execution of the speech coding application) is gathered from which the code vectors will be constructed.
   - Decide on the size of the codebook L. The size of the codebook is typically a power of 2 eg. 32, 64, 128 or 256 because speech coding requires maximum usage of the code word which is comprised of a prespecified number of bits.
   - Use the k-means segmental algorithm to determine the L code vectors based on the set of training speech feature vectors.
2. Encoding
   - A new speech feature vector is encoded as an integer between 1 and L by determining to which of the L code vectors it is closest (ie. using a distance measure $\delta()$).
3. Decoding
   - In order to decode the coded sequence of speech, one must have a copy of the codebook. The codebook contains the code vector number (between 1 and L) and the speech feature vector (eg. the LPC coefficients) associated with it.

The reason for including the VQ process was primarily to introduce the idea that one might be able to circumvent the need for manually classifying each speech feature vector in the training set when constructing a set of reference feature vectors.

Note also that if the training set of speech feature vectors represents all the sounds of a language (as it should), the code vectors should roughly align themselves with the phonemic classes of the language. Obviously there will be some duplication in that several code vectors might map onto one phoneme class but there should not be any omissions.

4.4.3 The Self-Organising Map

The Self-Organising Map (SOM) is a clustering unsupervised learning algorithm developed by Kohonen [Koh90].

The fundamental idea of the SOM is to construct a two-dimensional spatial ordering of the *p* dimensional feature vector space. The two-dimensional spatial ordering is called the *map* which is comprised of G *cells or units* representing speech sounds in a similar way that the code vectors represented speech sounds in the k-means segmental algorithm. The SOM algorithm organises the speech sounds associated with the cells such that similar sounds are clustered together while dissimilar ones are separated from each other in the two-dimensional map.

The SOM can be interpreted as a competitive learning, unsupervised neural network. In competitive learning, "neurons" compete against each other by means of

> "mutual lateral interactions and develop adaptively into specific detectors of different signal patterns" [Koh90].

The similarities of the SOM and competitive learning neural networks are beyond the scope of this thesis but can be found in [Koh90]. The "spatial" component of the SOM has not been a feature of "neural" algorithms excluding the SOM but Kohonen demonstrates that the brain may exhibit a "spatial" character analogous to that found in the SOM [Koh90].

The map is a two-dimensional, single layer of cells arranged in some topological configuration, typically a rectangle, square or hexagon. Each cell represents a 'sound' (at some time t) by a *weight speech feature vector*, $m_i(t)$. The cells are not connected to each other as they are in most neural networks. Instead their connectivity is intrinsically modelled by their proximity to other cells in their neighbourhood of the map. A large amount of speech input (eg. LPC coefficients of frame-sized speech signals) is used to configure the weight feature vectors associated with the cells. During each step[1] in the training stage, an input speech feature vector *stimulates or excites* the cell closest to it ('closeness' is determined by the shortest distance between the input vector and one of the weight vectors). The 'closest' cell to the input vector in a training step is called the *centre cell* with weight vector $m_k(t)$. All the cells within a specified radius N_k of the centre cell (this region is called the 'bubble' [Koh88]) are stimulated in a similar way that the centre cell was stimulated. 'Stimulation' is formally defined in the algorithm below, but it can be thought of as scaling or influencing the weight vectors of those cells in the 'bubble' closer to the input feature vector. The radius of the 'bubble' decreases monotonically as the time index (ie. the training step number) increases. To begin with the radius of the 'bubble' is large to ensure that the map is globally well-ordered. Towards the end of training, the 'bubble' should have a radius covering only the centre cell and its immediate neighbours.

[1] A step refers to an iteration of the outer loop of the SOM algorithm.

This has the effect of finely attuning the weight vectors to represent the various sound classes of the language.

At the end of the training phase, if the input feature vectors were representative of all the speech sounds of the language, the weight vectors of the cells should roughly characterise the sound classes of that language. One needs only gather a single, manually labelled feature vector for each sound class (eg. phoneme) to determine the position of the sound class in the map.

An input speech feature vector x is thought to be connected in parallel to every cell C_i in the network so that the distance between the input and each weight vector can be determined. A weight vector $m_i(t)$ is associated with each cell C_i $1 \leq i \leq G$ at time *t*. The weight vectors can be thought of as representing various regions in the pattern feature space. A distance metric (as previously discussed) is used to match an input feature vector x with the m_i.

The SOM algorithm consists of an ordering and a labelling stage. The ordering stage creates the spatial ordering of the map as described above. The labelling stage determines the speech class labels corresponding to the various cells in the map.

The SOM algorithm described below is adapted from [Koh88] and [Koh90].

4.4.3.1 Phase 1: Spatial Ordering of the SOM

List of variables used in the algorithm:

- t is the time index
- x is the input feature vector
- m_i is the weight feature vector associated with each cell C_i
- N_k is the radius of the excitatory "bubble" with centre cell C_k
- $\alpha(t)$ is the "adaptation gain" at time t which scales the amount by which the weight feature vectors will change if they are in the radius of influence

1. Let t = 0. Initialise the weight feature vectors, $m_i(0)$, by choosing arbitrary values as their speech parameters. It is also necessary that $m_i \neq m_j$ for $1 \leq i \neq j \leq G$ where G is the number of cells in the map.
2. Get the next training input feature vector x from a set of training input feature vectors.
3. Determine the centre cell C_k which results in the best (ie. 'closest') matching weight vector $m_i(t)$ with x using some distance measure $\delta(x,m_i(t))$. This is mathematically expressed by the following:

$$k = \underset{i}{argmin} \{\delta(x, m_i(t))\} \tag{4.10}$$

   where argmin{δ()} returns the value of argument i which results in a minimum distance δ().
4. Update all the weight vectors of the cells contained in N_k according to the following rule:

$$m_i(t+1) = \begin{cases} m_i(t) + \alpha(t)\,[x(t) - m_i(t)] & \text{if } i \in N_k(t) \\ m_i(t) & \text{if } i \notin N_k(t) \end{cases} \tag{4.11}$$

   where $0 \leq \alpha(t) \leq 1$ is the "adaptation gain".
5. Let t = t + 1. If not finished then goto step 2.

The radius of the 'bubble' N_k around the centre cell C_k is initially set very large, typically greater than half the diameter of the network. Starting with a large radius ensures a global ordering of the map. For the initial 1000 training steps the radius is allowed to decrease monotonically (eg. linearly) with time. Thereafter it maintains a radius of one cell ie. the centre cell C_k and its nearest neighbours.

The "gain adaptation" factor $\alpha(t) = 0.9(1 - t/1000)$ for the initial 1000 steps and then for the remaining steps was set to a small value in the order of 0.01. These values were determined by empirical results of several test experiments conducted by Kohonen [Koh90].

The number of training steps should be relatively large. A good "rule of thumb" is 500 times the number of cells in the map [Koh90]. The map can over-learn if too many steps are used. Over-learning occurs when the weight vectors learn to recognise the training patterns themselves and thus to lose its ability to generalise as a classifier. On the other hand if too few steps are used, the map can be under-trained which will result in poor classification results.

Finally it is emphasised that the map is trained in an unsupervised manner ie. the feature vectors need not be labelled.

4.4.3.2 Phase 2: Determining the Classes in Map

The second phase, determining the classes corresponding to the cells in the map, requires a set of manually labelled training feature vectors. A single, labelled feature vector per speech sound (of the language) will determine which cells (or set of cells) correspond with which speech classes.

4.4.3.3 Phase 3: Classifier

The results from phase 2 are sufficient to build the classifier which consists of the list of the weight feature vectors and their respective sound class labels. A new input speech feature vector can be classified simply by returning the sound class label of the closest weight feature vector to it.

4.4.3.4 General Comments

The main advantage of unsupervised training is that the phonetic expert does not have the tedious task of manually labelling thousands of training feature vectors.

Unfortunately, the SOM classifier does not generate good results since it provides a very rough approximation of the Bayesian classifier [Koh90]. Consequently, Kohonen and a team of researchers have developed a set of techniques called the Learning Vector Quantisation (LVQ) algorithms[1] to improve these results.

4.4.3.5 Learning Vector Quantisation (LVQ) Techniques

The need for the LVQ techniques are described by Kohonen in the following way:

> "If the Self-Organizing Map is to be used as a pattern classifier in which the cells or their responses are grouped into subsets, each of which corresponds to a discrete class of patterns, then the problem becomes a decision process and must be handled differently. The original Map, like any classical Vector Quantization (VQ) method is mainly intended to approximate input signal

[1] Kohonen's team at the Helsinki University of Technology have developed LVQ_PAK, the LVQ program PAcKage, to perform the LVQ algorithms. LVQ_PAK can be obtained by anonymous ftp (see [Koh91]).

values, or their probability density functions, by quantized "codebook" vectors that are localized in the input space to minimize a quantization error functional. On the other hand, if the signal sets are to be classified into a finite number of categories, then several codebook vectors are usually made to represent each class, and their identity within the classes is no longer important. In fact, only decisions made at class borders count. It is then possible, as shown below [by the LVQ algorithms] to define effective values for the codebook vectors such that they directly define near-optimal decision borders between the classes, even in the sense of classical Bayesian decision theory." [Koh90]

The LVQ algorithms are briefly described below because of their importance in the classification process. The LVQ1/2/3 algorithms is based on the notation used in the vector quantisation (VQ) and SOM algorithms.

4.4.3.6 LVQ1 Algorithm

The LVQ1 algorithm is a technique which finely tunes the rough approximation of the class clusters determined by either a vector quantisation (eg. k-means segmental) or SOM algorithm. The tuning process aims to move the decision surfaces between two classes towards the Bayesian limit; in this way approximating the optimal solution.

The LVQ1 training algorithm begins where the codebook or weight vectors $m_i(T)$ (T is the final training step number) have already been determined by either a k-means segmental or SOM algorithm. A set of training feature vectors X = {(x,c)}, where each training feature vector x has a speech class *c* associated with it, is used by all the LVQ algorithms. The LVQ1 algorithm aims at moving the codebook vectors from the class decision boundaries towards the 'centre' of the class [Koh90]. The following equations achieve this end by updating the codebook vectors $m_i(t)$ for each new training feature vector from X.

$$\begin{aligned} m_c(t+1) &= m_c(t) + \alpha(t)\,[x(t) - m_c(t)], && \textit{if } x \textit{ is classified correctly} \\ m_c(t+1) &= m_c(t) - \alpha(t)\,[x(t) - m_c(t)], && \textit{if } x \textit{ is misclassified} \\ m_c(t+1) &= m_c(t), && \forall\, i \neq c \end{aligned} \tag{4.12}$$

The value of $\alpha(t)$ is small (±0.02) to begin with because it is a fine tuning technique. It reduces monotonically to zero in about 100 000 steps. This does not mean that a large number of manually labelled training feature vectors are necessary. Three feature vectors per sound class (ie. ±150 labelled training feature vectors) is sufficient though they must be repeatedly (and randomly) applied to make up the ±100 000 training steps.

How does one interpret these equations? If training vector x is classified correctly using a 1-NNR (ie. the 1-NNR classification is the same as the sound class label of x), the equations become identical to those of the SOM algorithm. On the other hand if x is misclassified (ie. the 1-NNR classification is different from the sound class according to manual labelling[1]), the minus sign in the second equation can be thought of as removing the contributions of the interfering (overlapping) neighbouring class probability density function from that of the m_c.

It can be shown that the LVQ1 method approximates in a piecewise linear manner the Bayes decision surface [Koh90].

4.4.3.7 LVQ2 Algorithm

This algorithm is a modification of the LVQ1. It can result in a better approximation of the Bayes decision surface.

Assume that m_i and m_j are two closest neighbour codebook vectors which belong to different classes C_i and C_j. Suppose also that m_i and m_j are not optimally positioned in the feature space (ie. they are near the Bayesian decision boundary). Define a symmetric window of width W about the midplane, where the width W is typically 10-20% of the shortest distance between m_i and m_j. Under these conditions the LVQ2 algorithm is given by the following:

$$\begin{aligned} m_i(t+1) &= m_i(t) - \alpha(t)[x - m_i(t)] \\ m_j(t+1) &= m_j(t) + \alpha(t)[x - m_j(t)] \end{aligned} \quad \left\{ \begin{array}{l} \textit{if } C_i \textit{ is the nearest class to } x, \\ \textit{but } x \in C_j \neq C_i \textit{ where } C_j \textit{ is} \\ \textit{the next nearest class to } x \textit{ and,} \\ x \textit{ is in the window } W \end{array} \right. \tag{4.13}$$

$$m_k(t+1) = m_k(t) \qquad \textit{in all other cases}$$

Relatively few steps (± 10 000) should be used with $\alpha(t)$ starting at 0.02 and decreasing monotonically to zero. The smaller number of steps is due to that observation that after many steps the m_i "drift away" from the Bayesian limit [Koh90].

In a high dimensional feature space x is determined to be in the window if

$$min\ \{\delta(x,m_i)/\delta(x,m_j),\ \delta(x,m_j),\ \delta(x,m_i)\} > (1\text{-}V)/(1+V)$$

where the *min*{} function returns the smallest result of the functions in its list and V is the shortest distance between m_i and m_j.

[1] The labels determined manually are assumed to be correct (else the results are ruined).

4.4.3.8 LVQ3 Algorithm

The LVQ3 algorithm tackles two problems inherent in LVQ2. The first problem is that corrections to m_i and m_j in LVQ2 result in monotonically decreasing shortest distances between m_i and m_j. How is this so? The correction of m_j in the correct class is greater than the correction of m_i in the incorrect class since m_j is further away from x than m_i. To rectify this problem, the condition of equation (4.13) is changed quite subtly in equation (4.14) to allow either m_i or m_j be closest to input vector x in the window W.

The second change in LVQ3 compensates for the possibility that if LVQ2 is continued for "too long", m_i may "drift away" from its optimal position in the feature space. This is remedied by introducing a further equation which ensures that the m_i continue approximating the class distributions [Koh90].
The equations are given as,

$$\left.\begin{aligned} m_i(t+1) &= m_i(t) - \alpha(t)\,[x - m_i(t)] \\ m_j(t+1) &= m_j(t) + \alpha(t)\,[x - m_j(t)] \end{aligned}\right\} \begin{array}{l} \textit{if } m_i \textit{ and } m_j \textit{ are the two closest vectors to} \\ \textit{and } x \textit{ and } m_j \in C_p, \textit{ while } m_i \notin C_p \\ \textit{and } x \textit{ is in the window } W \end{array}$$

$$m_k(t+1) = m_k(t) + \varepsilon\,\alpha(t)\,[x - m_k(t)] \qquad \textit{for } k \in \{i,j\}, \textit{ if } x, m_i \textit{ and } m_j \in C_r \tag{4.14}$$

The value of ε is dependent on the size of the window though experiments have shown values between 0.1 and 0.5 to be best [Koh90]. The number of steps is not restricted as it was in LVQ2 since the drifting away problems are catered for by this algorithm.

Both the k-means segmental and SOM (along with the LVQ) algorithms have been implemented in this project.

4.5 Problems of Classification

The failure of speech recognition systems to a large extent, can be attributed to the failure of their segmentation and classification processes [Nie81][Dev82]. Vaissière emphasises this point:

> "It becomes more and more evident that progress in continuous speech recognition essentially will depend on a better bottom-up analysis of the speech signal" [Vas85].

What is the reason for these poor results and how can they be improved? Firstly, better feature selection and extraction techniques (ie. preprocessing functions) are needed. Work has been conducted with cognitive psychologists and linguists to determine what preprocessing is performed by the brain. DeMori et al [DeM90] have shown that their classifier modelled on the human ear model, achieves significantly better results than spectral information used directly from the DFT of the signal.

Even if one only has spectral information of the speech signal, Vaissière points out:

> "The score obtained in sentence spectrogram reading (15% of errors in phoneme identification for expert readers) demonstrates that enough information is present to decrease the present error rate [±30-40%] by at least half" [Vas85]

Ask the phonetic expert to explain how he decodes the spectrogram and he will usually struggle to explain this. The same is true of a human listener when asked to explain how he heard and interpreted a speech sound or word. This type of problem lends itself to solution by artificial neural networks (ANNs). ANNs also find it difficult to 'explain' how they achieve a result since the information is encoded in the configuration of the network. (There is work being done on ANNs that can explain themselves but the researchers acknowledge this underlying difficulty). The best results in speech sound classification have been achieved by ANNs. The SOM with tailoring from the LVQ algorithms appears to be the most successful "phoneme-typewriter" to date.

One final point that underlies the ideas in this chapter, is that the ultimate success of a continuous recognition system will be determined by the contribution and interaction of the higher level speech and language processing ie. the syntax, semantics and pragmatics of a language, examined in chapter 8.

Chapter 5 - Template Matching Word Recognition

5.1 Introduction

In this and the next chapter two important approaches to *word* recognition are investigated, namely:

- *template matching* (the so-called *non-parametric approach*) and
- *hidden Markov modelling* (the *parametric approach*).

The two approaches are covered in separate chapters because of their unique approaches to the recognition problem. A comparison is undertaken at the end of chapter 6 which summarises their differences.

This chapter is divided into two main parts. The first deals with template matching in isolated word recognition while the second extends the template matching theory to handle connected word recognition.

5.1.1 Role of Word Recognition Systems

Word and *speech* recognition systems were defined in the introductory chapter. At this point it seems appropriate to recall and highlight their differences and roles in the overall speech recognition problem.

The general speech recognition process involves the following stages (see figure 1.1 in chapter 1):

- digitising the analogue speech signal
- feature extraction or preprocessing
- classification
- post-processing with integrated higher-level language processing

The difference between word and speech recognition is the *unit of recognition*, a *word* in the case of word recognition and a *subword* type (eg. a phoneme) in the case of speech recognition. The process of extracting the words and therefore the meaning from the speech signal is therefore handled differently in these different types of systems. In word recognition, sequences of speech feature vectors (or statistical models, see chapter 6) represent words and are used to (pattern) match against other sequences of speech feature

vectors. In subword recognition system, a single speech feature vector is classified as one of a set of atomic speech sounds.

In other words, in speech recognition systems, classification using one of the techniques in chapter 4, results in each input frame being labelled as a subword speech class (eg. phoneme). Thereafter, adjacent frame with identical classes are merged together by a post-processing routine to form larger sound segments (like syllables), words, phrases and finally sentences.

In word recognition systems, the speech input is matched against a set of reference words (or models in chapter 6) representing each word in the system's vocabulary. Therefore, 'classification' or matching occurs directly in terms of words.

Why do word recognisers exist when one can build words from recognised strings of subword sounds? Firstly, the word unit is easily identified by the human speech recognition system. Secondly, they developed as a result of research in the restricted area of *isolated* speech recognition, producing good recognition results with reasonable response times for small (<200 word) vocabularies [Red76].

5.1.2 Difference between Isolated and Connected Word Recognition

Isolated word recognition (IWR) is a simpler task than connected word recognition (CWR) because the start- and end-point of an isolated word is easily segmented from the surrounding silence, whereas the boundaries of connected words are very difficult to determine without actually recognising the entire sequence of words. In addition to this, connected words suffer from the effects of coarticulation due to 'sloppiness' in colloquial speech.

The term *end-point detection* is used in IWR to describe the task of detecting the start- and end-points of a word utterance. Usually a simple threshold of the intensity function performed on the speech signal is sufficient to determine where silence becomes sound (start-point) and sound becomes silence (end-point) at the end of the word utterance. Care must be taken of the possibility of a period of silence in the word due to a stop followed by a plosive sound eg. the *t* in the word *eight*. It is also advisable to monitor the 'frequency' of the signal using the zero-crossing or turning points function to detect low energy (amplitude) fricative or aspirated sounds at the word boundaries. If the system is operating in real-time and the start of a word has been detected, the end of word can be decided by a silence period of greater than some duration, typically 150ms [Lee92]. Detailed studies of end-point detection are undertaken by [Pit90], [Red76], [Rab76] and [Lee92].

In CWR it is more difficult (often impossible) to determine the word boundaries since adjacent words often "run into one another" due to coarticulation. Rabiner and Sambur tried with some success, to implement a preprocessing module which determined the word boundaries of connected digits[1] [Rab76]. This technique was however tailored to the connected digit recognition problem with little prospect of generalisation to other problems. As a result, connected word recognition systems generally do not presegment the speech signal but rather attempt to 'spot' words in the speech signal without knowing their start- and end-points beforehand.

5.1.3 Template Matching

In template matching word recognition systems, a word is represented by a sequence of n frame-sized, speech feature vectors x_1, ..., x_n and is sometimes referred to as a *(word) pattern*[2]. Template matching in IWR is the process of matching a single unknown word pattern against a set of reference word patterns which constitute the system's dictionary called a *lexicon*.

Several ways of using template matching in CWR are discussed later in this chapter. Temporarily one can think of this problem as matching several connected word patterns (possibly with repetition of any word) against a concatenated sequence of reference word patterns from the lexicon.

Template matching word recognition systems are often speaker dependent with a relatively small-sized lexicons and achieve an impressive rate of correct matches, typically between 88.6-99% for IWR (see summary of results in [Red76]). The small size of the vocabulary is influenced by the need for a real-time response and an expedient training stage to adapt the speaker dependent system for a different speaker. If the system is speaker dependent, a new set of reference patterns is needed for each new user of the system. This requires saying each word at least once during the training stage - the duration of which is dependent on the number of words in the lexicon and the system's processing power since it has to generate the reference pattern feature vectors for each spoken word. Speaker-independent template matching systems have also been built. See, for example, [Rab79], [Rab84] and [Pan85]. Generally, these systems first preprocess the speech signal by labelling frame-sized speech segments in terms of codewords from a codebook constructed using the vector quantisation (VQ) technique described in chapter 4. Template matching

---

[1] The spoken digits 0-9 are a widely used vocabulary set in word recognition systems.

[2] **N.B.** A *pattern* which referred to a single feature vector in the previous chapter, now refers to the sequence of feature vectors making up a word.

then takes place using the codevectors associated with the codewords rather than the speech feature vectors for the particular spoken utterance itself. The results of independent-speaker template matching systems tend to be slightly poorer than for otherwise equivalent speaker-dependent systems because of the additional VQ classification step which is not very robust.

5.1.4 The Template Matching Problem

Template matching would be trivial if the duration of a speech sound in various utterances of the same word were constant. This is never the case because of the many variables which influence speech including the speaker's dialect, intentions and mood; and the context of the word in the sentence. As a result, several approaches to *normalise time variations* in the utterance of a word have been developed.

5.2 Isolated Word Recognition

5.2.1 Linear Time Alignment

Absolute Time Alignment

Absolute pattern matching is the simplest method of matching two patterns. This method directly compares an unknown word pattern[1] $X = x_1, ..., x_N$ with each of the reference word patterns, $R_s = r_1^{(s)}, ..., r_{J_s}^{(s)}$, $1 \le s \le W$ in a lexicon of W words. This is done by determining the minimum *total distance* $D(X,R_s)$ between the unknown pattern X and one of the reference pattern R_s. The total distance $D(X,R_k)$ between X and the k^{th} reference pattern is determined as the sum of the local distances between their (time aligned) feature vectors beginning with the first feature vector of each ie. x_1 and $r_1^{(k)}$ (and ignoring the feature vectors at the end of the longer pattern). That is,

$$D(X,R_j) = \sum_{i=1}^{\min\{N,J_j\}} \delta(x_i, r_i^{(j)}) \qquad (5.1)$$

The "winning match" is the k^{th} word in the lexicon whose reference pattern results in a minimum total distance with the input pattern X. Or, put mathematically,

---

[1] The terms *unknown pattern*, *input pattern* and *test pattern* are synonymous.

$$k = \underset{j}{argmin}\ D(X,R_j) = \underset{j}{argmin} \sum_{i=1}^{\min\{N,J_j\}} \delta(x_i, r_i^{(j)}), \quad 1 \le j \le W \tag{5.2}$$

where $\underset{j}{argmin}$ *is the index of* R_j *(ie.*k*) which results in a minimum distance between* X *and any* R_j

The problem with this method is that it does not take into consideration that utterances of the same word can have varying durations. It aligns the first feature vector of the reference patterns with that of the unknown pattern and compares only the overlapping regions of the patterns. It is probable that some region at the end of either reference or input pattern is not part of the matching procedure.

The *best absolute time alignment* algorithm [DeM90] is an improvement of the above absolute time alignment approach. It involves computing the absolute time alignment repeatedly, where each repetition aligns the first feature vector of the shorter of either the input and reference patterns with successive feature vectors (starting at the first) of the larger, before performing the total distance measure. This process terminates when the ends of the two patterns (ie. x_N and $r_{J_s}^{(s)}$) are time aligned. The best absolute time alignment of any input-reference pattern pair is the minimum distance obtained by the best alignment of that pattern pair. The "winning match" is determined as the word in the lexicon which results in the minimum best absolute time alignment with the input pattern. Although this algorithm results in an improved performance over the previous, it requires far greater computational effort. In addition, it does not solve the problem of varying durations in utterances of the same word. It simply finds the best absolute match between patterns.

Linear Time-Normalisation

Linear time-normalisation treats the time variations in utterances as linear with respect to time. That is, the time variations are thought to be uniform over all parts of the utterance.

Linear time-normalisation can be achieved in two ways. Either all patterns (input and reference) are uniformly stretched or compressed to a fixed pattern length, or the reference patterns are uniformly stretched or compressed to the length of the unknown pattern. Slight time shifting has also been experimented with to improve the results of the linear time-normalisation procedure [Whi76].

The results of a comparison between the linear and non-linear (discussed shortly) time-normalisation by White and Neely [Whi76] expose the weakness of the linear approach. As can be seen from Table 5.1, the linear time-normalised results were

comparable for the Alpha-Digit vocabulary set (the spoken alphabetic capitals "A-Z" and digits "0-9") while significantly poorer for the vocabulary of North American State names. This is attributed to non-linear time variations found in multisyllabic words eg. North American State names. The utterances of the Alpha-Digit vocabulary on the other hand, are almost all monosyllabic which are generally too short to exhibit non-linear time variations.

Time Normalisation Method	Alpha-Digit	North American States
Linear	98	90
Non-Linear	98	99.6

Table 5.1 Comparison of Linear and Non-linear Time Alignment Techniques [Whi76]

A commercial linear time-normalisation system **VRM** with a small 10-30 word vocabulary has been built [DeM90]. Its performance depends heavily on the choice of words in the vocabulary, the number of the speakers and their cooperation.

5.2.2 Non-linear Time Alignment

Non-linear time normalisation is the ability to align utterances of the same word taking into consideration local, non-uniform variations. The non-linear variations can be thought of as insertions, deletions or substitutions of feature vectors of the input pattern with respect to the matching reference pattern.

The *dynamic time warping (DTW) algorithm* is a well-known method for solving the problem of non-linear time normalisation in isolated and connected word recognition systems.

5.2.2.1 Notation

An unknown pattern X is comprised of a sequence of N feature vectors $X = x_1, ..., x_N$. The lexicon is made up of W reference patterns R_s $1 \le s \le W$. The k^{th} reference pattern is stored as a sequence of feature vectors $R_k = r_1^{(k)}, ..., r_{J_k}^{(k)}$, where $r_i^{(k)}$ is the i^{th} feature vector and J_k is the number of feature vectors in the sequence.

5.2.2.2 Basic DTW Procedure

The set of reference patterns is constructed in a *training stage*, where the speaker is prompted to utter each word in the vocabulary once. Several refinements in the training

phase have been suggested, eg. [Rab84]. Pitchers [Pit90] suggests a simple refinement in adding reference patterns of those words which the system recognises badly. This improves the recognition accuracy by about 5%.

When *testing* or using the system, the speaker utters a test pattern X which is matched against all the reference patterns in the lexicon to find the best matching word. The input pattern X and each reference pattern R_s are first non-linearly time aligned (warped) before determining $D(X,R_s)$, the distance between them. The best matching reference pattern is the one that results in a minimum distance with X.

5.2.2.3 Time Warping

The optimal, non-linear time alignment of the input pattern X and reference pattern R_s can be determined by a *warping function* w(k)=(i(k),j(k)) where k=1,2,...K. The i(k)s and j(k)s determine which input pattern feature vector is aligned with which reference pattern feature

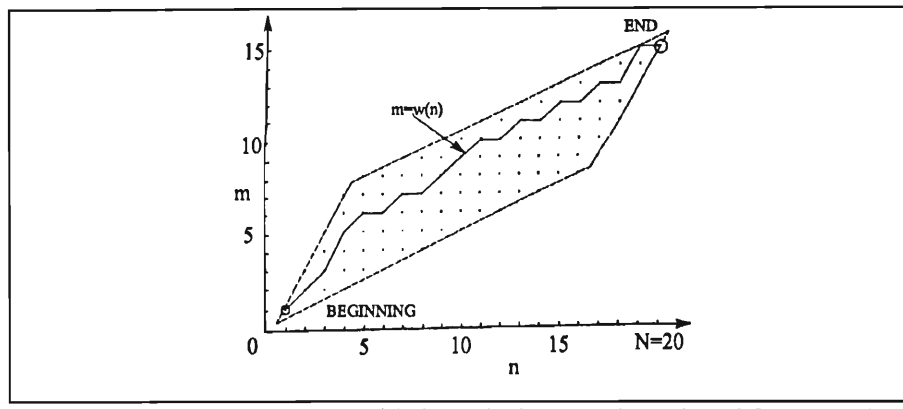

Figure 5.1 Warping function w(k) time aligning X and R_s adapted from [Ita75]

vector and therefore, $1 \le i(k) \le N$ and $1 \le j(k) \le J_s$. The warping function illustrated in figure 5.1, is the sequence of points w(1), w(2), ..., w(K) which depict the optimal alignment between the $x_{i(k)}$ vectors of the input pattern X with the $r_{j(k)}^{(s)}$ vectors of the reference pattern R_s. Once the optimal warping path is determined, the distance between X and R_s, $D(X,R_s)$, can be calculated as the sum of the local distances between the feature vectors of X and R_s at each point along the warping path. The optimal warping path must comply with the following conditions[1] adapted from [Sak78]:

I. *Boundary conditions:*

$w(1) = (1,1)$ and

$w(K) = (I,J_s)$

That is, the optimal path aligns the first and last feature vectors of X and R_s.

[1] See [Ita75] for a slightly different set of conditions.

II. *Monotonicity:*

$i(k+1) \geq i(k)$ and

$j(k+1) \geq j(k)$, $\quad 1 \leq k \leq K-1$

III. *Continuity:*

$i(k+1) - i(k) \leq 1$ and

$j(k+1) - j(k) \leq 1$, $\quad 1 \leq k \leq K-1$

and as a result, the point w(k+1) on the warping path can be reached from one of the following three points only:

$$w(k+1) = \begin{cases} (i(k)+1, j(k)) \\ (i(k)+1, j(k)+1) \\ (i(k), j(k)+1) \end{cases} \qquad \textbf{(5.3)}$$

IV. *Adjustment window of length h:*

$|\, i(k) - j(k) \,| \leq h$, $\quad$ where $h \geq 0$ is a suitable window length.

The adjustment window condition ensures that the warping path does not err too far from the "diagonal"[1].

V. *Slope condition:*

The "*effective intensity of the slope*" of the warping function is the ratio of the number of consecutive diagonal moves n to the number of consecutive vertical or horizontal (but not both) moves m where the $m+n$ moves are consecutive. It is conveniently called the *slope constraint* and is denoted by $P = n/m$. It is evident that the greater the slope constraint, the more the warping function is restricted to the diagonal. In the extreme case where $P \rightarrow \infty$ (ie. $n \rightarrow \infty$ and $m \rightarrow 0$), the path lies exactly along the "diagonal". This is just the degenerate case of the linear time alignment of the input pattern with the reference pattern. When $P \rightarrow 0$, there are no restrictions on the warping function. The optimum slope constraint (in terms of generating the best results) has been shown to be $P = 1$, while $P = 0$ and $P = \infty$ require the least computational effort [Sak78]. In this project, a slope constraint of 0 was used since the difference in performance between $P = 0$ and $P = 1$ (which is less than 1%[2] [Sak78]) does not warrant the extra computational effort.

---

[1] The "diagonal" is approximated since the N x J_s matrix need not be square.

[2] Performances were comparable for Japanese digits but ±1% different for 50 Japanese geographical names.

5.2.2.4 Implementation of the DTW Algorithm

The implementation of the DTW algorithm can be divided into two parts. The first stage involves constructing an NxJ_s distance matrix called Dist(i,j), which stores the distances between the feature vectors x_i and $r_j^{(s)}$ for $1 \le i \le N$ and $1 \le j \le J_s$ ie. $Dist(i,j) = \delta(x_i, r_j^{(s)})$. Figure 5.2 adapted from Moore's illustrations [DeM90], depicts a small example of this matrix where the input pattern X is of the same word though different utterance as the reference pattern R_k and, as a result, the distances (ie. the values not in brackets) stored in the matrix close to the "diagonal" tend to be relatively small while those further away from the "diagonal" are relatively large.

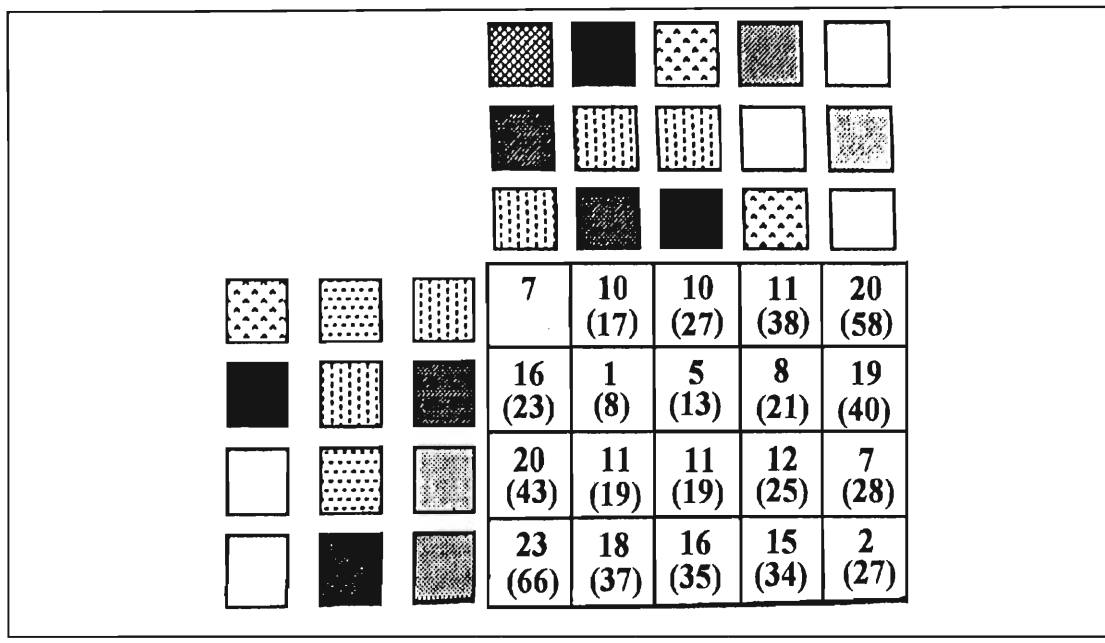

Figure 5.2 The Distance Matrix Dist(i,j) of X and R_k with the values for the Accumulated Distance Matrix AccDist(i,j) stored in brackets in the equivalent cell [Dem90].

The second stage involves finding the optimal warping path through the distance matrix[1] by determining the lowest accumulated distance along any path through Dist(i,j) adhering to all the warping path conditions I-V. This is achieved by constructing an accumulated distance matrix AccDist(i,j), also depicted in figure 5.2, which stores the accumulated distance along the best path to every point (i,j) in the warping grid. This is often written in the literature as a dynamic programming (DP) equation:

For slope constraint P = 1, the DP equation is given as:

$$g(i,j) = \min \begin{bmatrix} \delta(i,j) + g(i,j-1), \\ 2\delta(i,j) + g(i-1,j-1), \\ \delta(i,j) + g(i-1,j) \end{bmatrix} \qquad (5.4)$$

[1] Starting from the top left element and ending at the bottom right element in order to satisfy the boundary conditions (see constraint I above) of warping functions.

For slope constraint P = 0, the DP equation is:

$$g(i,j) = \min \begin{bmatrix} \delta(i,j) + g(i-1,j), \\ \delta(i,j) + g(i-1,j-1), \\ \delta(i,j) + g(i,j-1) \end{bmatrix} \quad \textbf{(5.5)}$$

The recursive function g() in the DP equation (5.5)[1] can be modified and used to evaluate the accumulated distance matrix iteratively in the following way:

Algorithm:

```
For i := 1 to N do
   For j := 1 to J_s do
      AccDist(i,j) := Dist(x_i,r_j^(s)) + Min{AccDist(i-1,j), AccDist(i-1,j-1), AccDist(i,j-1)}
   EndFor {j}
EndFor {i}
```

where
- x_i and $r_j^{(s)}$ are the feature vectors of the input and reference patterns respectively and,
- $Dist(x_i,r_j^{(s)})$ is the value of element (i,j) of the Dist matrix defined above ie. the distance measure $\delta(x_i,r_j^{(s)})$ and,
- Min{a,b,c} is a function returning the argument a, b or c with the smallest value and,
- AccDist(0,u) and AccDist(v,0) which will be accessed by the above algorithm when i=1 and/or j=1, are therefore initialised to ∞ for $1 \le u \le J_s$ and $1 \le v \le N$ and AccDist(0,0) = 0.

If the adjustment window condition (IV) is applied, the above algorithm must set to ∞ those elements in AccDist which do not satisfy the condition $j-h \le i \le j+h$ (where h is a suitable adjustment window).

It is evident that the actual warping path through the warping grid is not stored or returned from the above algorithm. In fact, the optimal warping path does not need to be preserved in isolated word recognition because the minimum accumulated distance of all the input-reference pattern matchings determines the best matching reference pattern (and therefore the "winning" word in the lexicon).

The optimal warping path is important in connected word recognition where the between-word links must be monitored to determine the sequence of connected words. The optimal path through the grid can be determined by tracing back through the accumulated distance matrix, starting from the bottom right cell (N,J_s) and recursively, until cell (0,0) is reached, determining the cell from which the current cell was reached in order to achieve a minimum accumulated distance at that position in the grid.

[1] As previously mentioned, a slope constraint of P=0 was used for computational ease.

Determining the Warping Path

A more efficient manner (in terms of speed but not space) of determining the warping path can be achieved by storing a matrix of "from cells", ie. the previous position from which each cell in the grid would be reached if it were on the optimal warping path. The "from cell" for each cell in the grid can be determined in the Min{} function of the above algorithm by saving the i and j indices of the argument (of Min{}) with the smallest distance value. The new matrix FromCell(i,j) is established to store the best local path decisions at every point (i,j) in the NxJ_s grid in the following way:

$$\begin{aligned} &let\ a = AccDist(i-1,j) \\ &let\ b = AccDist(i-1,j-1) \\ &let\ c = AccDist(i,j-1) \end{aligned} \qquad (5.6)$$

$$FromCell(i,j) = \begin{cases} (i-1,j) & if\ a = Min(a,b,c) \\ (i-1,j-1) & if\ b = Min(a,b,c) \\ (i,j-1) & if\ c = Min(a,b,c) \end{cases}$$

The warping path can thus be determined by backtracking along the best local path decisions starting at FromCell[N,J_s] for reference pattern R_s and stopping at (0,0). The warping path is (in reverse order):

{(N,J_s), FromCell(N,J_s), FromCell(FromCell(N,J_s)), ..., (1,1)}

A simple postfix recursive procedure will produce the warping path in the correct order.

5.2.3 Review of the Literature on Isolated Word DTW Algorithms

Sakoe and Chiba have compared the recognition results of different slope constraints and compared the results of different hybrid DTW algorithms developed in the '70s [Sak78]. Rabiner et al investigate the performance of an isolated DTW word recognition system with respect to the various parameters of the system [Rab78] as do Myers et al [Mye80] and [Tap78] with respect to memory and time savings in the DTW algorithm. Itakura developed a DTW isolated word recognition system using the maximum log likelihood ratio (described in the previous chapter). This system (200 word lexicon) matched words spoken over a telephone system with a reported accuracy of 97.3% and had a response time about 22 times greater than real time [Ita75]. A good overview and clearly presented DTW algorithm is contained in DeMori et al [DeM90] and Leedham [Lee92] while Levinson compares the template matching and hidden Markov model approaches in a holistic study of word recognition systems [Lev85a]. An algorithm called the order graph search developed by Brown and Rabiner [Bro82] uses ordered tree and graph searching techniques to reduce the distance computation threefold with no loss in recognition accuracy. Pitchers has described and implemented this algorithm [Pit90].

5.3 Template Matching in Connected Word Recognition

5.3.1 Introduction to the Problem

Recognising words in normal speech[1] is difficult because coarticulation is drastically increased in normal speech and as a result, there are very few definite cues in the speech signal to mark the between-word boundaries. However, it turns out that the non-linear time normalisation mechanism of the DTW algorithm is a suitable tool for overcoming these difficulties.

The connected word recognition problem using template matching can be formulated as follows:

> The input consists of a sequence of connected word patterns of normal speech chosen from a predefined vocabulary. The reference patterns are constructed and stored as they are for isolated word recognition. The recognition problem is to determine the sequence of concatenated reference patterns which best match the input pattern. The concatenation of reference patterns is often referred to as a *super reference pattern.*

The input pattern differs from the super reference pattern of the same sequence of words in that sounds appear to be omitted, blurred and/or compressed. The reason for this is that the reference word patterns are produced from isolated utterances of the words and are therefore generally "well-formed" and less succeptible to coarticulation than the words generated in normal speech. (Coarticulation is more pronounced in normal speech because the rapidity or tempo of normal speech is far greater than it is in isolated word utterances. As a result, the articulators often fail to reach the exact point of articulation in their haste to produce the subsequent sounds in the sentence). These differences of the input and super reference patterns of the same sequence of words (ie. the omissions, blurrings and deletions) can be overcome by the non-linear time alignment mechanism of the DTW algorithm. In fact this mechanism is the very reason for extending the use of the template matching approach to connected word recognition.

5.3.2 A Naive Solution

A naive solution of the template matching CWR problem is to generate all possible super reference patterns, comparing each with the input pattern using the DTW algorithm. This

---

[1] *Normal speech* is the term used to describe continuous, colloquial speech.

solution is non-tractable since the number of words in the input pattern is unknown and an infinite number of super reference patterns could thus be generated for matching with the input pattern. Even if the exact number of words in the input pattern were known (say *k*), the total number of executions of the DTW algorithm[1] for a vocabulary of V words is V^k which grows exponentially in *k*. In comparison, the number of executions of the DTW algorithm in isolated word recognition is V (ie. one for each reference pattern in the lexicon).

5.3.3 Optimised Solutions

Several algorithms using the DTW technique have been developed to improve the naive solution offered above, namely:

- the two-level algorithm [Sak79]
- the "sampled" algorithm [Rab80]
- the level-building algorithm [Mye81]
- the one-stage algorithm [Ney84]

The last algorithm which Ney revised from the earlier but "overlooked" works of Vintsyuk 1971 and Bridle and Brown in 1979, has been shown to be the best of the four algorithms in terms of simplicity in approach, computational efficiency and storage costs [Ney84]. The "sampled" algorithm by Rabiner and Schmidt [Rab80] looks at ways of optimising the word recognition algorithms and provides some useful insights to the problem of connected word recognition (CWR) as a whole. The other two algorithms are outlined to highlight their different approaches to the problem.

A standard notation based on that used by the DTW algorithm in isolated word recognition, is presented at the outset to make the task of comparing the different algorithms easier.

5.3.4 Notation

The reference patterns stored in the lexicon are represented as they were previously. That is, the s^{th} reference pattern in the lexicon is written as $R_s = r_1^{(s)} r_2^{(s)} \ldots r_{Js}^{(s)}$ where $r_j^{(s)}$ is the j^{th} feature vector of the s^{th} reference pattern.

---

[1] That is, the total number of orderings of the reference word patterns (with repetition) in a *k*-length super reference pattern.

The notation for the input pattern is more complex than the one used previously in order to depict that it is composed of several joined word utterances. An input pattern X with *k* word utterances is represented as:

$$X = x_1^{(s1)}\ x_2^{(s1)} \ldots x_{Ns1}^{(s1)}\ x_1^{(s2)} \ldots x_{Ns2}^{(s2)} \ldots x_1^{(k)}\ x_2^{(k)} \ldots x_{Nk}^{(sk)}$$

where $x_i^{(c)}$ is the i^{th} speech feature of the c^{th} word utterance. $x_{Nc}^{(c)}$ is the last feature vector of the c^{th} word utterance which is followed (if c<k) by $x_1^{(c+1)}$ or the first feature vector of the $(c+1)^{th}$ word pattern. Although the word boundaries are indicated in the above notation, they are not known beforehand and are very difficult to determine. Therefore the input pattern is also represented as a single pattern: $X = x_1, x_2 \ldots x_N$.

5.3.5 The Two-Level DP Algorithm [Sak79]

The two-level DP algorithm was developed as a result of the efforts of Sakoe and Chiba and Nakatsu and Kohda [Sak79]. The main ideas of the algorithm are presented in this section while a detailed account can be found in [Sak79]. The algorithm uses two stages to recognise the words in a continuous speech signal (as the name suggests).

Stage 1: Word Level Matching

The first stage is called word level matching. The DTW algorithm is used to determine the best matching reference pattern R_k $1 \le k \le W$ of every possible partial sequence of the input pattern. That is, the minimum accumulated distance and the reference word (ie. its index) associated with the minimum accumulated distance is evaluated and stored for every partial sequence of input pattern $[x_\ell;x_m]$, $\ell \le m$. The minimum accumulated distance of the partial sequence $[x_\ell;x_m]$ is stored in a matrix AccDist(ℓ,m) and the matching reference word index *k* which resulted in the minimum accumulated distance is stored in a matrix WdHyp(ℓ,m). The matrix WdHyp(ℓ,m) embodies a word hypothesis and can be expressed in words as: the partial sequence of the input pattern starting at the ℓ^{th} and ending at the m^{th} input feature vector is thought (hypothesised) to be the k^{th} reference word in the lexicon.

For a lexicon size of V words, an input pattern of length M frames and ignoring the DTW constraints, the number of times the DTW algorithm is executed, is given as:

$$\frac{M}{2}\left[M+1\right] V \qquad (5.7)$$

The word level matching stage involves the bulk of the computation and therefore any opportunity of limiting the number of DTW matches should be taken. For example, the adjustment window *h* (DTW condition IV) can limit the possible matching sequences.

Stage 2: Phrase Level Matching

The second stage attempts to concatenate adjacent word hypotheses (evaluated above) in the best way in order to determine the optimal, hypothesised connected sequence of words in the input pattern. All potential concatenations of word hypotheses must extend over the entire input pattern. That is, the starting frame of the first word hypothesis is 1 and the ending frame of the last word hypothesis in the concatenated sequence of word hypotheses is N. The word hypotheses in the concatenated sequence are abutted in that the starting frame of the $(c+1)^{th}$ word hypothesis is one frame greater than the ending frame of the c^{th} word hypothesis. The optimal concatenation of word hypotheses (obeying the above conditions) is the one which results in the smallest sum of the minimum accumulated distances of the word hypotheses in a concatenated sequence. That is, the "winning" concatenation of word hypotheses is the one whose sum of the accumulated distances of its intermediate *p-1* word hypotheses, $AccDist(1,w_1)+AccDist(w_1+1,w_2)+...+AccDist(w_p,N)$, is less than that of any other.

To solve the phrase level matching problem, two new matrices T(x,m) and L(x,m) are introduced:

- T(x,m) stores the smallest sum of the accumulated distances of *x* word hypotheses ending at input frame *m* - ie.
  $T(x,m) = AccDist(1,w_1) + AccDist(w_1+1,w_2) + ... + AccDist(w_{x-1}+1,w_x)$
  where w_i is the last frame of the i^{th} word hypothesis and $w_x = m$.

- L(x,m) stores the reference word index associated with the $AccDist(w_{x-1}+1,w_x)$ which results in the smallest value for T(x,m).

T(x,m) and L(x,m) are determined for the best word sequences of K or fewer words (ie. $1 \le x \le K$). K is the predetermined maximum number of words allowed in the input utterance. Once these matrices have been populated, the best sequence of connected words can be found. This is achieved in two steps: by firstly determining the best number of words (V) in the input sequence ($V \le K$) using T(x,N) and then using L(V,N) to backtrack to the previous word boundaries in the optimal connected word sequence. The best word hypotheses associated with each adjacent set of word boundaries (a,b) can then be found in WdHyp(a,b).

The complete two-level algorithm is presented below in a form adapted from [Sak79] so that it can easily be implemented by computer software.

Two-level Algorithm:

Word Level Matching Stage

```
Get input pattern X
Initialise matrix AccDist(ℓ,m) to ∞ and WdHyp(ℓ,m) to 0 ∀ ℓ,m
For refwdno := 1 to W do  { W is the number of reference words in the lexicon }
   GetDistMatrix(Dist( ))  { GetDistMatrix( ) is a procedure to calculate the distance matrix
                             Dist(i,j) := δ(x_i,r_j^(refwdno)), ∀i,j }
   For ℓ := 1 to (N-J_refwdno+h) do        { constant h is the adjustment window constraint }
      For m := (ℓ+J_refwdno-h) to (ℓ+J_refwdno+h) do
         acc_dist := CalcAccDist(ℓ,m)
         If ( acc_dist < AccDist(ℓ,m) ) then
            AccDist(ℓ,m) := acc_dist
            WdHyp(ℓ,m) := refwdno
         EndIf
      EndFor { m }
   EndFor { ℓ }
EndFor { refwdno }
```

Phrase Level Matching Stage

```
T(x,m) := 0, for 1≤m≤N and 1≤x≤K
Determine the values of T(x,m) and L(x,m) by the following:
      For x := 1 to K do   { ie. for every possible number of words in the input sequence }
         For m := 1 to N do
            MinDist := ∞
            ℓargmin := 0
            For ℓ from max{1,(m-J_max-h)} to (m-J_min+h) do
               TempDist := AccDist(ℓ,m) + T(x-1,ℓ-1)
               If (TempDist < MinDist) then
                  MinDist := TempDist
                  ℓargmin := ℓ-1
               EndIf
            EndFor { ℓ }
            T(x,m) := MinDist + T(x-1,ℓargmin)
            L(x,m) := ℓargmin
         EndFor { m }
      EndFor { x }
```

{ where $J_{max} = \max(J_n)$
and $J_{min} = \min(J_n)$, $\forall n\ 1 \le n \le W$ }

Two-level Algorithm (cont'd):

Determine the number of words (V) in the optimum sequence of words:

```
MinDist := ∞
V := 0
For x := 1 to K do
    If ( T(x,N) < MinDist ) then
        MinDist := T(x,N)
        V := x
    EndIf
EndFor { x }
```

The position of the word boundaries WB(0) = 1, WB(1), ..., WB(V-1), WB(V) = N of the optimum sequence are thus determined (in reverse order):

$$WB(V) = N,\ WB(V-1) = L_{V-1}(WB(V)) = L_{V-1}(N),\ WB(V-2) = L_{V-2}(WB(V-1)) = L_{k-2}(L_{k-1}(N)),\ \ldots,\ WB(0) = 0$$

Finally, the best concatenated word hypothesis sequence is:

WdHyp(1,WB(1)), WdHyp(WB(1)+1,WB(2)), ..., WdHyp(WB(V-1)+1,N)

5.3.6 The Level-Building DTW Algorithm [Mye81]

The level-building[1] DTW algorithm is an efficient implementation of the two-level algorithm. The paper by Myers and Rabiner [Mye81] covers in great detail, the implementation of and comparison between this algorithm and the two-level algorithm. The details are not regenerated here, but several of the most important observations are highlighted.

The major difference between the level building and two-level algorithms is the order in which the ideas of the algorithm are carried out. In the two-level algorithm, the first stage (ie. word level matching) involves generating all the best word hypotheses for all possible partial sequences of the input pattern. The second stage (ie. phrase level matching) pieces together word hypotheses in order to determine the best sequence of connected word hypotheses making up the sequence of words in the input pattern.

In the level-building algorithm, on the other hand, the phrase matching stage is incorporated in the word matching stage. The "levels" correspond to the word hypotheses. Optimal paths are generated for every possible number of levels less than some maximum number of levels. That is, all possible first word hypotheses are determined for the first level, then all

[1] This algorithm is a special case of the *stack algorithm* developed by Bahl and Jelinek [Bah75]. 6.

possible second word hypotheses, beginning at the end of the best first level word hypotheses, are generated and so on. Back pointers are used to store the input pattern frame number of the last position of the best path of the previous level from which the word hypothesis of the current level originated. The optimal warping path can therefore be recovered at the end level processing by using the back pointers to trace back along the best path. The reference word number corresponding to the best matching path at the various levels is also stored so that ultimately, the matching connected word sequence can be recovered.

A necessary parameter of this algorithm is the constant determining the maximum number of levels, that is the maximum possible number of words in the connected word utterance. Once all word hypotheses are determined for all possible number of levels ($\leq$ maximum number of levels), the best sequence is decided firstly by determining the best number of levels or the number of words in the test utterance, called L. This is done by choosing the level number whose best path ends at the last frame of the test pattern, and whose accumulated distance at that point is a minimum (ie. smaller than the best hypotheses at any other level number). A backtracking technique is used to uncover the word boundaries and the corresponding reference words associated with the decisions to place the word boundaries in that position. This stage is similar to the backtracking stage described in the following section.

Myers and Rabiner improved the efficiency of the level building algorithm further by introducing certain "range reduction techniques" [Mye81]. The idea behind these techniques is to reduce the number of DTW matches which are the most costly operation in the algorithm. The improved algorithm developed by Myers and Rabiner was called the "reduced level building" algorithm. It was been shown to be approximated 20 times more efficient than the two-level building algorithm while the "reduced level building" algorithm 30 times more efficient [Mye81].

5.3.7 The One-Stage Dynamic Programming (DP) Algorithm [Ney84]

The one-stage DP algorithm was devised by Vintsyuk in 1971 and later developed by Bridle and Brown in 1979 [Ney84]. It was overshadowed by the two-level and level building algorithms until Ney revised it and drew attention to its merits. It has been shown to be both more efficient and less complicated than the previous algorithms. In the following section, an adapted version of Ney's one-stage algorithm is presented.

A set of grid points (i,j,k) is constructed in 2-dimensions as depicted in figure 5.3. The horizontal axis is calibrated by the input frame index i ie. the i^{th} feature vector of the input

pattern. The vertical axis is referenced by two variables, a "base pointer" k determining the first feature vector of the k^{th} reference pattern R_k and an "offset index" j marking the j^{th} feature vector of the base (k^{th}) reference pattern. J_k is the last frame or feature vector of the k^{th} reference pattern and is followed by the first feature vector (j=1) of the $k+1^{th}$ reference pattern ($1 \leq k \leq W$, where W is the number of words in the lexicon).

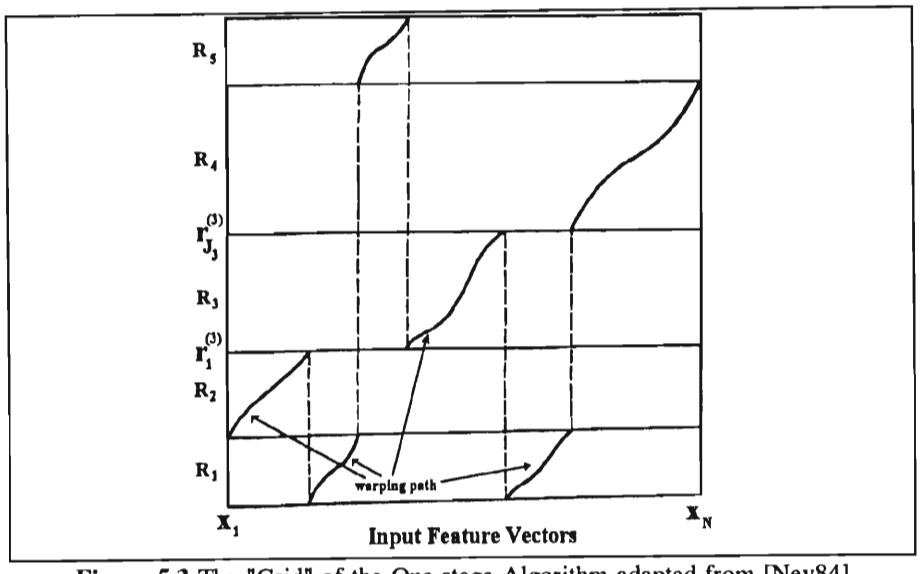

Figure 5.3 The "Grid" of the One-stage Algorithm adapted from [Ney84]

Similar to the case of isolated word recognition (IWR), the local distance $\delta(x_i, r_j^{(k)})$ (ie. the distance between x_i and $r_j^{(k)}$) is stored for each grid position (i,j,k) in a matrix called Dist(i,j,k). The one-stage algorithm is based on finding the best path through the grid points (i,j,k). The distinguishing feature of this CWR algorithm (as opposed to the previous ones) is that the best path is determined by a time warping function parameterised by a *single* index (with suitable continuity rules). The idea is simple since it is based directly on the DTW algorithm used for IWR and has been shown to perform as well as the previous algorithms with savings in computational effort and storage space.

The warping path $\{w(1), w(2), ..., w(\ell), ..., w(L)\}$ through the grid is determined by the warping function:

$w(\ell) = (i(\ell), j(\ell), k(\ell))$, where ℓ is the path index of the ordered set of path elements.

The continuity of the warping path is specified by two continuity rules called *transition rules* in order to describe the more complex nature of connected speech (where only one continuity rule was required for the IWR case). From figure 5.3 it is evident that the warping path *within* a reference pattern obeys the same rules of continuity as those defined for IWR. At the *word boundaries* however, a different rule is needed to cater for the vertical "jumps" to the start of different reference patterns.

The *within-word transition rule* therefore states that the point (i,j,k) for j>1 can be reached from one of only three points:

$$\{ (i-1,j,k), (i-1,j-1,k), (i,j-1,k) \}$$

The *between-word transition rule* deals with the problem of 'joining' the final sound of one reference pattern with the first sound of the next. Formally the rule is written as:

$$\text{if } w(\ell) = (i,1,k) \text{ then}$$
$$w(\ell-1) \in \{ (i-1,1,k); (i-1,J_{k^*},k^*) \} \text{ where } 1 \le k^* \le W$$

Notice that the between-word transition rule does not exclude the possibility of the same word being immediately repeated in the input pattern (ie. the last frame of a reference pattern 'joining up' with its own first frame as the start of the next word). In addition, syntactic information can be introduced at this point, to restrict the choice of reference patterns k^* to those which, according to some finite state syntax, are allowed to follow the k^{th} reference pattern.

The minimum accumulated distance at each grid position (i,j,k) is determined in a similar way that it was in the IWR case but takes into account the new transition rule and the slightly different grid structure. The values in the minimum accumulated distance array AccDist(i,j,k) are governed by the following dynamic programming (DP) relations:

Within-word relation (ie. j>1):

$$\text{AccDist}(i,j,k) = \text{Dist}(i,j,k) + \min \{ \text{AccDist}(i-1,j,k), \text{AccDist}(i-1,j-1,k), \text{AccDist}(i,j-1,k) \} \quad \text{(DP1)}$$

Between-word relation (ie. j=1):

$$\text{AccDist}(i,1,k) = \text{Dist}(i,1,k) + \min \{ \text{AccDist}(i-1,1,k); \text{AccDist}(i-1,J_{k^*},k^*): k^*=1, ..., W \} \quad \text{(DP2)}$$

The AccDist(i,j,k) matrix must be initialised to ∞ for the boundary conditions ie. for all matrix entries where i = 0 or j = 0, except where i = j = 0 when AccDist(0,0,k) = 0.

The algorithm for determining the best warping path follows that of the IWR case but uses the new ideas introduced above:

Algorithm:

1. Calculate the local distances for each point (i,j,k) in the grid:

```
For i := 1 to N do                    { N   = # feature vectors in the input pattern }
    For k := 1 to W do                { W   = # reference words in the lexicon }
        For j := 1 to J_k do          { J_k = # feature vectors in the kth reference pattern }
            Dist(i,j,k) := δ(x_i,r_j^(k))   { δ() = a distance function (eg. Euclidean) }
        EndFor { j }
    EndFor { k }
EndFor { i }
```

2. Initialise the boundary conditions of the accumulated distance matrix:

```
For k := 1 to W do
    AccDist(0,0,k) := 0
    For i := 1 to N do
        AccDist(i,0,k) := ∞
    EndFor { i }
    For j := 1 to J_k do
        AccDist(0,j,k) := ∞
    EndFor { j }
EndFor { k }
```

3. Calculate accumulated distance matrix AccDist(i,j,k) at every grid point (i,j,k) using DP1 and DP2:

```
For i := 1 to N do
    For k := 1 to W do
        AccDist(i,1,k) := Dist(i,1,k) + min [ AccDist(i-1,1,k); AccDist(i-1,J_k*,k*): k*=1,...W ]
        FromCell(i,1,k) := the grid position which resulted in a minimum AccDist() in the above min[] function
        For j := 2 to J_k do
            AccDist(i,j,k) := Dist(i,j,k) + min [ AccDist(i-1,j,k), AccDist(i-1,j-1,k), AccDist(i,j-1,k) ]
            FromCell(i,j,k) := the grid position which resulted in a min. AccDist in the above min[] function
        EndFor { j }
    EndFor { k }
EndFor { i }
```

The min[] functions in the above algorithm determine the course of the warping path. The previous point (i_x,j_y,k_z) on the best path to the point (i,j,k), whose minimum accumulated distance is chosen by the min[] functions, is stored in FromCell(i,j,k) to avoid having to redetermine the previous point in the best path to each point (i,j,k). That is, FromCell(i,j,k) = (i_x,j_y,k_z). Notice however that each element in FromCell() stores three data words of memory for every position in the grid ie. a value to reference each of the indices i, j and k. This is expensive when one considers that only the best path is ultimately required. To avoid this expense, a new system for extracting the ultimate sequence of

words in the input pattern will shortly be devised. First, the above algorithm is completed using the FromCell() array to extract the best path.

In order to determine the best warping path, one must first find the grid point (N,J_k,k)[1] with the smallest minimum accumulated distance - that is, the grid point (N,J_{k_0},k_0) for some reference pattern k_0 that has a minimum accumulated distance smaller than that of any other point (N,J_k,k) for all $k \neq k_0$. This grid point (N,J_{k_0},k_0) is the last frame of the best path which 'unlocks' the entire optimal path sequence through the grid (and therefore the connected word sequence) using the backtracking method similar to that described in the IWR case.

Algorithm to find Warping Path: (continues where the algorithm above left off)

```
Smallest := ∞
k0 := 0
For k := 1 to W do
    If ( AccDist(N,Jk,k) < Smallest ) then
        Smallest := AccDist(N,Jk,k)
        k0 := k
    EndIf
EndFor { k }
```

The warping path is thus the set of points:

$\{ (N,J_{k0},k_0), \text{FromCell}(N,J_{k0},k_0), \text{FromCell}(\text{FromCell}(N,J_{k0},k_0)), ..., (1,J_{kz},k_z) \}$ where $1 \leq z \leq W$

The connected sequence of words can be determined from the reference pattern indices k in the above set of warping path points.

There are several weaknesses with the one-stage algorithm in its form above:

- The matrices Dist( ) and AccDist( ) are large storage structures whose elements (i,j,k) are only accessed or used for a very short duration of the total time of the algorithm. It is evident that in order to determine the minimum accumulated distance AccDist(i,j,k), one needs only the current (i^{th}) and previous ($i\text{-}1^{th}$) columns of AccDist( ) and the current (i^{th}) column of Dist( ).
- The matrix FromCell(i,j,k) has already been shown to be wasteful of space and a new backpointer system should be introduced to overcome this weakness.

Thus, the minimum accumulated distance array AccDist(i,j,k) is replaced by two column arrays CurrAccDist(j,k) and PrevAccDist(j,k) representing the current (i^{th}) and previous (i-

[1] That is, a grid point in the last frame of the input pattern, in the last frame of one of the reference patterns.

1^{th}) input frame columns in the grid. These arrays operate in the following way: once all the minimum accumulated distances of say the i^{th} column have been evaluated and stored in the entries of the CurrAccDist(j,k) array, the entries in the CurrAccDist(j,k) array are copied over to the PrevAccDist(j,k) array and the CurrAccDist( ) array is reinitialised for evaluating the minimum accumulated distances of the $i+1^{th}$ column. In this way, only the necessary minimum accumulated distances are available for evaluating every minimum accumulated distance at any point in the grid.

The second point which needs addressing is the backtracking procedure. The reason for storing the backpointers FromCell(i,j,k) was to avoid the task of reevaluating the path decision during backtracking. This structure however requires a large amount of storage space. In addition, it stores the backpointers of every possible path through the grid where only the backpointers of the best path are necessary to solve the problem of finding the sequence of words in the input pattern. Therefore, backpointers for each position (i,j,k) are set up to store the input frame index of the last frame of the previous word in the best path reaching point (i,j,k). The backpointers are stored in two structures PrevBP(j,k) and CurrBP(j,k) like those for determining the minimum accumulated distances for each point. As a result, two other structures, a "From Template" FT(i) and a "From Frame" FF(i) are needed to remember the previous 'best path' word boundary frames and the reference patterns associated with them to maintain a backtracking mechanism for extracting the best sequence of words.

The "From Template" array thus evaluates and stores for every input frame i, the template or reference pattern k_0 with the smallest minimum accumulated distance in its last frame ie. it stores the template index k_0 which results in CurrAccDist(J_{k_0},k_0) $\leq$ CurrAccDist(J_k,k) for all $k \neq k_0$. At the same time, the backpointer CurrBP(J_{k_0},k_0) (to the last frame in the previous word in the path 'joined to' (J_{k_0},k_0)) is stored in the "From Frame" FF(i).

Once the values FT(N) and FF(N) (where N is the last input frame) have been evaluated, the word boundaries of the unknown sequence of words are determined as:

{ N, FF(N), FF(FF(N)), ..., 0 }

The words associated with these word boundaries (ie. the ultimate sequence of words of the unknown input pattern) can easily be 'looked up' in the "From Template" array and are written (in reverse order) as:

{ FT(N), FT(FF(N)), FT(FF(FF(N))), ... }

The Revised Algorithm:

```
Initialise all the entries in PrevAccDist( ) to ∞ except PrevAccDist(0,k) = 0 for all k
Initialise CurrAccDist(0,k) = ∞ for all k
Initialise all the entries in PrevBP( ) and CurrBP( ) to 0

For i := 1 to N do
    For k := 1 to W do

        CurrAccDist(1,k) := Dist(1,k) + min [ PrevAccDist(1,k); PrevAccDist(J_k',k'): k'=1,...W ]

        If ( argument of the result of the above 'min' function was PrevAccDist(1,k) ) then
            CurrBP(1,k) := PrevBP(1,k)
        Else
            CurrBP(1,k) := i - 1
        EndIf

        For j := 2 to J_k do
            CurrAccDist(j,k) := Dist(j,k) + min [ a:PrevAccDist(j,k), b:PrevAccDist(j-1,k), c:CurrAccDist(j-1,k) ]
            Case ( argument of the result of above 'min' function )
                a: CurrBP(j,k) := PrevBP(j,k)
                b: CurrBP(j,k) := PrevBP(j-1,k)
                c: CurrBP(j,k) := CurrBP(j-1,k)
            EndCase
        EndFor { j }
    EndFor { k }

    smallest := ∞
    bestk := 0
    For k := 1 to W do
        If ( CurrAccDist(J_k,k) < smallest ) then
            smallest := CurrAccDist(J_k,k)
            bestk := k
        EndIf
    EndFor { k }

    FT(i) := bestk
    FF(i) := CurrBP(J_bestk,bestk)

EndFor { i }
```

Then the word boundaries of the ultimate sequence of words can be generated by a recursive procedure eg.

```
Def WordBoundaries( WdBorder )
    If ( WdBorder > 0 ) then
        WordBoundaries( FF(WdBorder) )
    EndIf
    Output(WdBorder)
EndDef
```

which generates the word boundaries in left-right order ie.

{ 0, ..., FF(FF(FF(N))), FF(FF(N)), FF(N), N }

And the sequence of words in the input pattern can also be generated by a recursive procedure:

```
Def GenerateWords( WdBorder )
    If ( WdBorder > 0 ) then
        GenerateWords( FF(WdBorder) )
        Output FT(WdBorder)
    EndIf
EndDef
```

which generates the sequence of words in left-right order ie.

{ ..., FT(FF(FF(FF(N)))), FT(FF(FF(N))), FT(FF(N)), FT(N) }

5.3.8 Comparison of the CWR DTW Algorithms

In the second half of this chapter, three connected word recognition (CWR) dynamic time warping (DTW) algorithms were studied namely:

- the two-level algorithm
- the level building algorithm
- the one-stage algorithm

The recognition accuracy of the three algorithms is comparable since they are in many ways similar to each other. Their computational efficiency, storage requirements and specific problem-strategies however differ considerably.

The two-level algorithm solves the problem in two stages: firstly, finding all the best word hypotheses for every partial sequence of the input pattern (the word level) and secondly,

joining adjacent word hypotheses (the phrase level) in order to determine the best overall concatenation of word hypotheses across the entire input pattern.

The level building algorithm reorders the steps in the two-level algorithm to produce a substantially more efficient solution. This algorithm incorporates some of the 'phrase level' processing (ie. concatenation of word hypotheses) into the word level stage with the result that the number of possible word hypotheses and the number of ways of combining them is greatly reduced.

Finally, the one-stage algorithm sets up the reference patterns and the input pattern in such a way that the time alignment problem can be treated in a manner very similar to that used for the isolated word recognition DTW algorithm. This solution is considerably less complicated than, and results in increased efficiency and storage reduction over the previous two. The results of a comparison between the computational efficiency and storage requirements in the above algorithms (and the improved version of the level building algorithm, the reduced level building algorithm) based on Ney's detailed comparisons [Ney84] are presented to support these claims.

	Two-Level Algorithm	Level Building Algorithm	Reduced Level Building Algorithm	One-Stage Algorithm
Computational Cost	100	16	3	4
Storage Cost	100	72	72	8

Table 5.1 Relative Computational and Storage Costs of the CWR DTW algorithms

5.4 Conclusion of Template Matching Techniques

In summary, template matching is the process of matching an unknown input pattern against a set of reference patterns (or templates) in the system's lexicon or dictionary. Three broad techniques for template matching were examined: absolute matching, linear time alignment and non-linear time alignment (DTW).

The first two template matching techniques fail to address the effects of coarticulation and variability in speech utterances and as a result, do not perform effectively. The non-linear time alignment template matching technique on the other hand, achieves good results and was discussed in further detail with respect to: isolated word recognition (IWR), connected word recognition (CWR).

Template matching techniques appear to be well suited for recognition systems with small vocabularies because:

- the training stage involves building one reference word for each word in the dictionary and thus, the smaller the vocabulary, the less tedious the training stage.

- response times are linked to the number of words in the lexicon since template matching involves matching the input with each reference pattern

Finally, two weaknesses in the template matching approach are mentioned:

- As has already been mentioned, template matching systems are limited to small-sized vocabularies.

- Secondly, and perhaps more importantly, template matching uses a *word* unit of recognition which, although it results in a relatively easy training stage, can be limiting for the recogniser. For example, a person can hear the (sound) parts of in word, say "be..." in "be*cause*" or "be*fore*", and use them to predict and verify other sounds in their proximity. Although the warping path in the DTW technique can give an indication of the sounds within the input pattern, extracting this information is difficult. It is also difficult to determine sections of the warping path without first generating the entire minimum accumulated distance array of every possible point in the grid, by which stage the within-pattern information may be of no use anyway. In short, template matching matches at the word level and therefore lacks the human-like ability to recognise sub-word segments of speech and use them to improve the search for and verification of sounds in their surrounding context.

Chapter 6 - Hidden Markov Modelling

6.1 Introduction

6.1.1 Non-Parametric vs. Parametric

The *non-parametric or template matching approach* discussed in the previous chapter matches the actual speech feature vectors (made up of LPC coefficients for example) of the input pattern with those of the reference patterns.

In this chapter, a *parametric stochastic approach*[1] to word recognition is examined as implemented by *hidden Markov models*. Stochastic models characterise the statistical properties of the system. Markov (introduced in chapter 4) and hidden Markov processes are examples of statistical models.

An approach is said to be *parametric* if the parameters of the model, determined during a training stage, are used to measure the probability that some set of events would be generated by or match those of the model during testing.

The underlying assumption of parametric stochastic modelling of speech is that:

> "*the speech signal can be represented as a random process and that the parameters of this process can be estimated in a well-defined and precise manner*" [Rab89].

6.1.2 Markov Processes

In speech processing, contextual knowledge can often be used (and is certainly used by humans) to recognise, predict and verify speech events. For example, if a recognition system has recognised the first word of a sentence as "the ...", the next word will most probably be a noun. As a result, the recogniser can focus on the nouns in the dictionary in order to 'narrow' the search for matching the next part of the signal. Consider another example, this time concerning a within-word context. If a recogniser (identifying sub-word units) has correctly recognised the sounds "th i ng ..." (with more sounds to come), the

[1] The terms statistic, stochastic and random are synonymous.

probability that the next sound is a "b" is very small though high for sounds like "s" (for *things*) and "k" (for *think*).

6.1.3 Chapter Overview

In chapter 4, the Markov process was shown to be able to model the context of events. In this chapter, Markov processes are extended to overcome the problems presented by time dependent systems (eg. speech), using hidden Markov models. The chapter is divided into two sections: the first focusing on isolated word recognition (IWR) and the second on connected word recognition, as was done in the previous chapter. Several different types or forms of hidden Markov models are studied though much of the detail is omitted from this study. References to the (omitted) details are given where necessary. The chapter is concluded with a brief summary of the differences between the parametric and non-parametric approaches and a comparison of their performances.

6.1.4 Brief Literature Review

The best introductory texts on hidden Markov modelling are the 'tutorial' by Rabiner [Rab89], the chapter in Speech Processing by Moore [Moo92] and the contents of a lecture on the comparison of parametric and non-parametric techniques by Levinson [Lev85a]. An integrated, more detailed analysis is found in [Lev85b]. Liporace [Lip82] and Juang [Jua86] discuss continuous density (as opposed to discrete) HMMs in rigorous detail. Furthermore, new methods for estimating the HMM parameters are described in [Gao90a]. Several papers covering diverse applications of HMMs and their analyses are to be found eg. Rabiner et al [Rab85b] present an isolated digit recogniser using continuous mixture densities while Wilpon et al [Wil90] have developed a word spotting, continuous speech system. Rabiner and Levinson [Rab85a] improve the performance of a level-building algorithm for connected word recognition using HMMs. De Mori et al [DeM90] implements and compares a HMM system to classify vowels with the results obtained from a neural network built for the same purpose. Lee et al [Lee90a] discuss their SPHINX continuous speech recognition system which uses HMM methodologies. Lee [Lee90b] covers the details on the technique using 'context-dependent phons' (HMM-like) to implement a stochastic, continuous speech recognition system.

6.2 Discrete Hidden Markov Models

6.2.1 Concepts

In discrete Markov processes described in chapter 4, each state is related to a physically observable event. It was shown that Markov processes have difficulty expressing systems that are time dependent.

Hidden Markov models (HMMs) solve the time dependency problem of Markov processes by introducing a probability function assigned to every state to determine the probability of any physically observable event (observation) being generated by or chosen for that state. A consequence of this probability function is that the observable events are no longer related to a particular state in the model. In order to explain the nature of HMMs, the classical analogy of a HMM[1] is presented in the following game of choosing coloured balls from urns, reported in both [Rab89] and [Lev85a].

'Genie Game' Analogy:

> Suppose there are N large urns, each filled with a fixed but unknown number of coloured balls. The total number of different colours is K. The game is controlled by a genie who hides the urns from view using a screen. The genie then chooses a coloured ball and shows it to the player. The player observes the colour of the ball and keeps a record of it. The genie then places that ball back in the urn from which he chose it and repeatedly chooses a coloured ball and shows it to the player in the manner described above. The point of the game becomes apparent when the genie asks the player to determine the probabilities of drawing any particular coloured ball (of K colours) from any of the N urns. The player must also determine which ball came from which urn. The task as described is obviously impossible. However the genie gives the player his rules for selecting from which urn the ball must be chosen. There is one further precondition for selecting the urn by which the genie must comply: the rule for selecting the next urn depends probabilistically on only the previous urn selected. This precondition is just the Markov property described in chapter 4. The game as it is now described, is a hidden Markov process.

Why are HMM 'hidden'? Consider the "game" analogy. The probabilistic selection of the urns does not directly correspond with an observed event (as it does in a Markov process).

[1] Ascribed to Jack Ferguson and his colleagues in lectures on HMM theory [Rab89].

Instead the observation is determined by a second (the first determining which state to belong to) probabilistic function. In the 'genie game', this function determines the colour of the ball to be extracted. Therefore a hidden Markov model can be thought of as:

> "a doubly embedded stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observations." [Rab89]

6.2.2 Notation

A HMM 'looks' very like a discrete Markov process. It has the following similar features:

- a set of N states $\{ S_1, S_2, ..., S_N \}$
- q_t represents the state in which the system is, at time t[1]. Thus $q_t = S_i$ means that the system is in state S_i at time t.
- a state transition probability matrix A such that a_{ij} represents the probability of transitting from states S_i to S_j.

In addition to the above, it contains:

- a column vector π which stores the initial probability of being in each of the states; *i* is thus the probability of being in state S_i at time t=1.
- a probability distribution matrix B (of size N×K) stores the probabilities of observing each event at each state, ie. the result of the *observation symbol probability distribution function* which for state S_i is $b_i(k)$ $1 \le i \le N$ and $1 \le k \le K$ where K is the total number of distinct observation symbols (or the alphabet size). The observation sequence X is the list of T observation symbols: $x_1, x_2, ..., x_T$.

[1] The t variable is the discrete time index.

6.2.3 Parameters of a HMM

A HMM is specified by its parameters namely: the probability distributions A, B and π. The primary issue of constructing a HMM is that of determining the optimal values of the parameters. The parameters of the model are usually denoted by $M = (A,B,\pi)$.

The configuration of the model, the number of states N and the size of the alphabet K must be decided before building HMMs.

6.3 HMMs in Isolated Word Recognition

In isolated word recognition (IWR), each word *w* is represented by a HMM M_w $1 \leq w \leq W$ where W is the number of words in the lexicon. Estimating the optimal parameters for each HMM M_w is called the *training or learning procedure*. The *testing procedure* determines the probability that an unknown word input pattern (ie. a sequence of speech feature vectors ≡ the observations) is generated by or is the outcome of each of the HMM M_w. The "winning" word is thus the one whose HMM results in the highest probability of all the other M_ws for that sequence of observations. The training and testing procedures for discrete HMMs are presented in the next sections followed by those for continuous HMMs. Before doing so, several implementational notes are made to give insight into the nature of HMMs and their use in isolated word recognition.

6.3.1 Configuration and Number of the States

Many configurations of HMMs have been investigated for different applications. In word recognition systems however a *left-right* or *Bakis* configuration is the standard. Left-right HMMs characterise the time variant nature of speech by only allowing the system to traverse to a state greater than the one its in (or to remain in the current state). That is, every state S_i in the model is linked to itself and the subsequent states ie. S_i, S_{i+1}, S_{i+2}, ... The configuration is practically determined by setting the transition state probabilities a_{ij} where $j<i$ to 0. In this way, the successive states in the HMMs representing words can be thought of as describing and characterising the statistical probabilities of the different 'sections' of the word. The number of states will determine the number of 'sections' in the word.

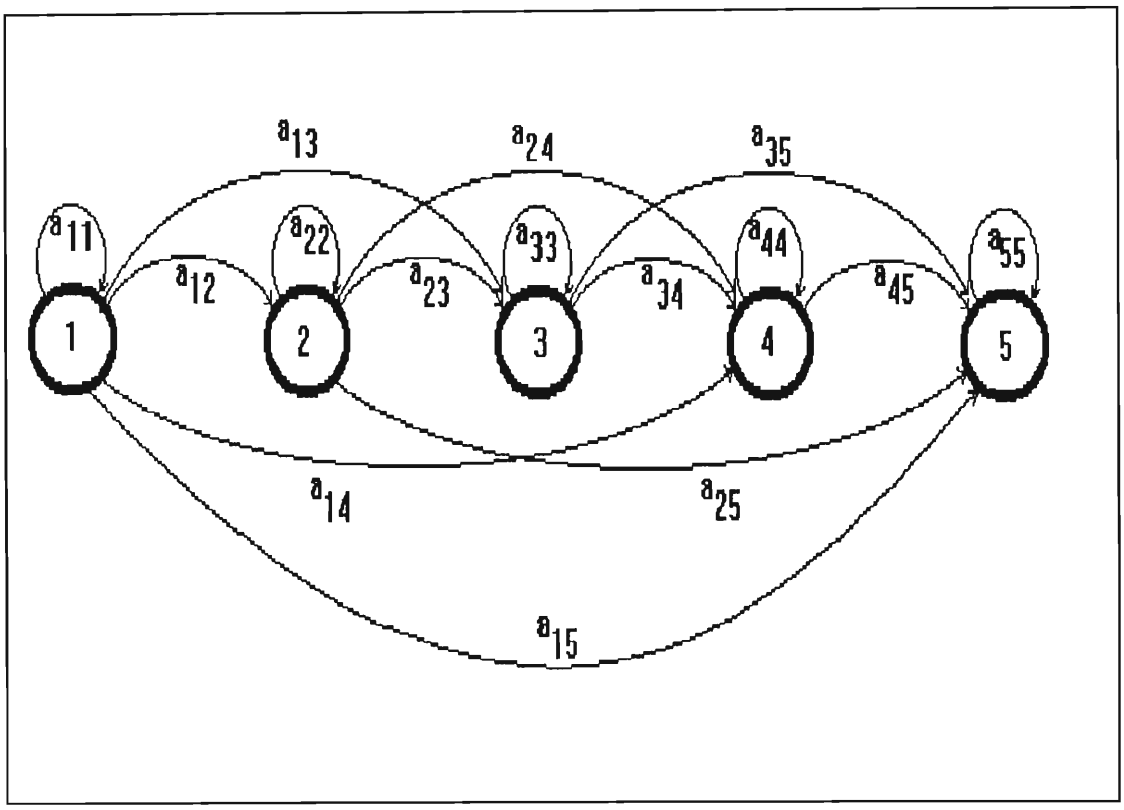

Figure 6.1 The Left-Right Markov Model Configuration

Bakis chose to set the number of states roughly equal to the number of frames (speech feature vectors) in the training pattern of the word being modelled. This has several side-effects, for example catering for training patterns of different length and building routines that can handle a variable number of states for different word HMMs. Another option which avoids these problems is to fix the number of states for all the word models in the lexicon to be equal to the average number of speech sounds in the words in the lexicon. The reasoning for this choice is to try and match the 'sections' of the word characterising the states in the model with the speech sounds in the word. It would be ideal if each state would be responsible for one of the observed sounds in the word. Notice that the speech sounds are not precisely defined since an exact alignment of states to specific sound classes is practically impossible for an entire vocabulary. In practice, the number of states tends to be between 5 and 9 [Rab89].

6.3.2 Discrete vs. Continuous HMMs

In the formal definition (under notation) of the HMM, an alphabet of K observation symbols was mentioned. With respect to speech recognition, these K observation symbols are representative of 'speech sounds' and $b_i(k)$ $1 \leq k \leq K$ is the probability of the k^{th} 'speech sound' resulting in the i^{th} state. Obviously these 'speech sounds' are not phonetic classes eg. phonemes (or else the hardest part of the recognition problem would be accomplished). In fact the K 'speech sounds' are the codewords determined by the *vector quantisation*

technique presented in chapter 4. That is, K is the size of the *codebook* of pretrained *codevectors* which are encoded as *codewords*. The codevectors can be thought of as single speech feature vectors describing a particular 'speech sound'.

Therefore a speech feature vector of an input word pattern is first encoded by the codeword of its nearest matching codevector k in the codebook; and then this codeword represents it in the various HMM algorithms. The codeword is thus a discrete representation of the speech feature vector and as a result, HMMs using this approach are called *discrete* and $b_i(k)$ is a discrete probability density function. There is obviously a danger in this technique for greater error in matching due to the loss in 'resolution' of the speech feature vector (observed input) being modelled. The way to avoid these problems is using *continuous* HMMs ie. that have continuous observation densities. Such systems are not investigated further but the interested reader should begin with [Rab89] or [Rab85b] and follow with [Jua85a], [Lip82] and [Jua86][1] .

For completeness, there is another class of HMMs called *autoregressive* models but they are not studied further. See [Rab89] or [Jua85b] for an introductory investigation of these types of models.

In the next sections, the algorithms for testing and training HMMs in an isolated word recognition system are presented.

6.3.3 Testing Discrete HMM

Assume that an input pattern $X = x_1\ x_2\ ...\ x_T$ is the observation sequence. The probability of observing X from one of the hidden Markov models M_w in the lexicon (assume training has already taken place) can be written as:

$$P(X \mid M_w) = P(x_1\ x_2\ ...\ x_T \mid M_w)^2$$

Consequently, the best matching word w_0 can be determined by:

$$w_0 = \text{argmax}\ P(X \mid M_w)$$

where argmax determines the value w_0 of argument w which results in $P(X \mid M_w)$ being a maximum.

---

[1] [Lip82] and [Jua86] are complex articles and are thus difficult to read though they are historically important as a pioneering papers in this area.

[2] $P(X \mid Y)$ 'reads' as it does traditionally in statistics; ie. the probability of X given Y.

The problem of finding $P(X \mid M)$ for a hidden Markov model M simplistically reduces to the problem of finding the sum (over all state paths) of the joint probabilities that every state path through the HMM exhibits the observation sequence $X = x_1 \ldots x_T$. The solution of this problem is of exponential order (N^T) since it is based on the problem of generating every state path through the model which can be achieved recursively by traversing (until $t = T$) from the current state to all N other states. The details of this are covered in [Rab89].

6.3.4 Forward-Backward Procedure

The so-called *forward-backward* procedure[1] offers a more efficient method of evaluating $P(X \mid M)$ than the previous. Only the 'forward' part of the procedure is needed to determine $P(X \mid M)$. Because the 'backward' part is used in subsequent algorithms, it is also presented in this section. The forward and backward parts of the forward-backward procedure are represented by the variables $\alpha_t(i)$ and $\beta_t(i)$ respectively.

The forward variable $\alpha_t(i)$ is written as:

$$\alpha_t(i) = P(x_1\, x_2 \ldots x_t,\, q_t = S_i \mid M)$$

which measures the probability that the system is in state S_i at time *t* and generates the feature vector x_t after the model M has previously generated the partial sequence of feature vectors $x_1\, x_2 \ldots x_t$ (regardless of the sequence of states).

The backward variable $\beta_t(i)$ is similarly defined as:

$$\beta_t(i) = P(x_{t+1} \ldots x_T,\, q_t = S_i \mid M)$$

THIS IS WRONG. (SHOULD CONDITION ON $q_t = S_i$)

which determines the probability that the system is in state S_i at time *t* and generates the feature vector x_t and a partial sequence of feature vectors $x_{t+1} \ldots x_T$ in the future (regardless of the sequence of states).

6.3.4.1 Forward Variable

The forward part of the forward-backward procedure is given by the following inductive definition:

---

[1] Developed by Baum and his colleagues of statistical background [Rab89].

$$
\begin{aligned}
&\textit{Initialisation:} \\
&\alpha_1(i) = \pi_i \, b_i(x_1) \qquad\qquad 1 \le i \le N \\
&\textit{Induction:} \\
&\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \, a_{ij} \right] b_j(x_{t+1}) \qquad 1 \le t \le T-1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1 \le j \le N
\end{aligned}
\tag{6.1}
$$

Finding the probability that an observation sequence $x_1 ... x_T$ is described by a model M can thus easily be determined by:

$$
\begin{aligned}
P(X \mid M) &= P(x_1 \, x_2 \, \ldots \, x_T \mid M) \\
&= \sum_{i=1}^{N} \alpha_T(i)
\end{aligned}
\tag{6.2}
$$

6.3.4.2 Backward Algorithm

Calculating the backward variable is similar to that of the forward variable given above. The backward variable is the probability that the sequence $x_{t+1} ... x_T$ follows from being in state S_i at time t given model M. This can be written as:

$$
\begin{aligned}
&\textit{Initialisation:} \\
&\beta_T(i) = 1 \qquad\qquad 1 \le i \le N \\
&\textit{Recursion:} \\
&\beta_t(i) = \sum_{j=1}^{N} a_{ij} \, b_j(x_{t+1}) \, \beta_{t+1}(j) \qquad T-1, \, T-2, \, \ldots \, 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1 \le i \le N
\end{aligned}
\tag{6.3}
$$

6.3.5 Viterbi Algorithm

The Viterbi algorithm is a dynamic programming procedure that determines the *best state sequence* for a given observation sequence of feature vectors. It is not possible to know *a priori* the best number of states N or the correct sequence of states $q_1 \, q_2 \ldots q_T$ since states are not linked to physically observable events (eg. speech classes) in HMMs as they are in Markov processes. Moreover, the correct state sequence cannot be determined because of this. A *best state sequence* can be found as the state sequence which results in the highest probability of producing the required observed output. The probability associated with traversing the best state sequence is evaluated in the Viterbi algorithm. It can also be

used to determine the best matching word HMM of the speech input pattern X. That is, the word w_0 whose HMM M_{w0} results in a maximum probability is chosen as the 'winner':

$P(Q_{w0},X \mid M_{w0}) \geq P(Q_w,X \mid M_w)\ \forall\ w \neq w_0$ and Q_w is the best sequence of states $\{q_1 \ldots q_T\}$ through M_w.

Knowing the best state sequence is useful for analysing the nature of the states and later will be used to uncover the word boundaries in continuous speech recognition. An interesting application is being undertaken by Van der Merwe and Du Preez [Van91]. They are attempting to align the phonetic labels of a phonetic transcription (known *a priori*) of an utterance with the phonetic event in the utterance using state sequence information. The idea of this project is to automate the tedious task of segmenting the speech signal.

A delta variable $\delta_t(i)$ is defined for the Viterbi algorithm by the following expression:

$$\delta_t(i) = \max_{q_1 q_2 \cdots q_{t-1}} P(q_1 q_2 \cdots q_{t-1},\ q_t = S_i,\ x_1 \ldots x_t \mid M) \qquad (6.4)$$

The delta variable $\delta_t(i)$ measures the probability of having traversed the best state sequence to time t-1 observing the feature vectors $x_1 \ldots x_{t-1}$ and at time t choosing state S_i to be the best state. The delta variable is very similar to the alpha (forward) variable except that the 'sum' operator is replaced by a 'maximum' operator to determine the best path sequence up to the previous time slot. The delta variable $\delta_t(i)$ is determined by the following inductive expression:

$$\delta_{t+1}(j) = \max_i [\ \delta_t(i)\ a_{ij}\ ]\ b_j(x_{t+1}) \qquad (6.5)$$

The object of the Viterbi algorithm is to determine the best state sequence which is achieved by storing the i indices which maximise the $[\delta_t(i)a_{ij}]$ term in equation (6.5) for all $1 \leq t \leq T$ and $1 \leq j \leq N$. A two-dimensional array $\psi_t(j)$ is used to store these previous values of i. The best state sequence can then be extracted by a dynamic programming procedure using the information stored in the $\psi_t(j)$ array.

The complete Viterbi algorithm is given below as:

$$
\begin{aligned}
&\textit{Initialisation:} \\
&\delta_1(i) = \pi_i \, b_i(x_1) && 1 \le i \le N \\
&\psi_1(i) = 0 \\
&\textit{Induction:} \\
&\delta_t(j) = \max_{1 \le i \le N} [\, \delta_{t-1}(i)\, a_{ij} \,]\, b_j(x_t) && 2 \le t \le T \\
& && 1 \le j \le N \qquad (6.6) \\
&\psi_t(j) = \underset{1 \le i \le N}{\textit{argmax}} [\, \delta_{t-1}(i)\, a_{ij} \,] \\
&\textit{Termination:} \\
&P(q_1 q_2 \ldots q_T, X = x_1 \ldots x_T \mid M) = \max_{1 \le i \le N} [\, \delta_T(i) \,] \\
&q_T^* = \underset{1 \le i \le N}{\textit{argmax}} [\, \delta_T(i) \,]
\end{aligned}
$$

The final state sequence can be determined starting with state q^*, by recursively determining the previous best state using the following algorithm:

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T\text{-}1,\ T\text{-}2,\ \ldots,\ 1 \qquad (6.7)$$

That is, the best state sequence in reverse order is given as:

$$\{\, q_T^*,\ q_{T-1}^* = \psi_T(q_T^*),\ \psi_{T-1}(q_{T-1}^*) = \psi_{T-1}(\psi_T(q_T^*)),\ \ldots \,\}$$

6.3.6 Training (Learning) Stage

The training stage in template matching systems (chapter 5) is a simple procedure that involves constructing a single reference pattern for each word in the system's vocabulary for a new speaker. The reference pattern is therefore a representative sequence of speech feature vectors of that word.

On the other hand, a HMM is constructed to represent each word in the lexicon in the stochastic approach. In fact, because the values for N (ie. the number of states) and K (the number of observations symbols) are usually fixed for a particular IWR system, the lexicon need only store the parameters (A,B,π) of each particular word. The training stage of a system using HMMs is therefore concerned with determining the best parameters of each HMM representing a particular word. *More than one* training pattern is used to train the reference HMMs in the lexicon. This is because in the left-right model, the number of observations that occur in any state in the HMM before traversing to the next state is statistically too small to result in reliable estimates of the HMM parameters. By using several training patterns, this problem can be averted. As a result of the ability to use more

than one training pattern, training patterns from more than one speaker can be used to produce a speaker independent set of HMM reference parameters.

Training then determines the optimal set of HMM parameters for each word in the lexicon. This is achieved by maximising the probability of generating (or recognising) the training patterns in each HMM. The problem of determining whether a probability $P(X \mid M_w)$ is optimal is very difficult and in fact, the problem of finding the optimal model parameters of a HMM is not analytically possible. Nor is it even possible to estimate the *optimal* parameters given a finite set of training observations [Rab89]. It is however possible to build an algorithm which iterates towards a *local optimum* set of model parameters. This optimising process is achieved by re-estimating the model parameters (A,B,π) using the *Baum-Welsh re-estimation algorithm*[1].

6.3.6.1 Baum-Welsh Re-estimation

The re-estimation algorithm iteratively updates randomly initialised values for the model parameters (A,B,π) - ie. π_i, a_{ij} and $b_j(k)$ for $1 \le i,j \le N$ and $1 \le k \le K$ - of some word model M. After each iteration, the forward and backward variables must be recalculated since they are dependent on the re-estimated parameters. In addition, because the training procedure requires several training patterns in left-right models, it is necessary to consider when to introduce a new training pattern in the re-estimation cycle and when to terminate the entire procedure (ie. when the local optimum set of model parameters is found).

The re-estimation algorithm is dependent on the gamma variable defined as:

$$\gamma_t(i) = P(q_t = S_i \mid X,M)$$

or the probability of being in state S_i at time t given the observation sequence $X = x_1 x_2 \ldots x_T$ and the hidden Markov model M. The gamma variable can then be expressed in terms of the forward and backward variables in the following way:

[1] Other techniques include the expectation-modification method [Den92] and gradient techniques [Rab89].

$$\gamma_t(i) = P(q_t=S_i \mid X, M)$$

$$= \frac{P(x_1 \ldots x_t, q_t=S_i \mid M)\ P(x_{t+1} \ldots x_T \mid q_t=S_i, M)}{P(X \mid M)} \qquad (6.8)$$

$$= \frac{\alpha_t(i)\,\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\,\beta_t(i)}$$

The forward variable $\alpha_t(i)$ accounts for the 'first part' of the partial observation sequence $x_1 \ldots x_t$ while $\beta_t(i)$ for the remainder $x_{t+1} \ldots x_T$. The denominator $P(X \mid M)$ ensures that the gamma variable is normalised to a probability value ie.

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \qquad (6.9)$$

The second variable needed for the re-estimation algorithm is $\xi_t(i,j)$. This variable stores the probability of being in state S_i at time t and state S_j in the next time slot $t+1$ given the observation sequence $x_1 \ldots x_T$ and the hidden Markov model M. The variable $\xi_t(i,j)$ can also be written in terms of the alpha and beta variables and it turns out that $\gamma_t(i)$ can be written in terms of $\xi_t(i,j)$. To see the link with the gamma variable, $\xi_t(i,j)$ must cater for the transition from S_i to S_j as well as the probability of being in state S_i at time t and state S_j at time t+1, given the observation sequence $x_1 \ldots x_T$. Thus, $\xi_t(i)$ is defined as:

$$\xi_t(i,j) = P(q_t=S_i,\ q_{t+1}=S_j \mid X, M)$$

$$= \frac{\alpha_t(i)\ a_{ij}\ b_j(x_{t+1})\ \beta_{t+1}(j)}{P(X \mid M)} \qquad (6.10)$$

$$= \frac{\alpha_t(i)\ a_{ij}\ b_j(x_{t+1})\ \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i)\ a_{ij}\ b_j(x_{t+1})\ \beta_{t+1}(j)}$$

The denominator in the equation (6.10) again acts to ensure the result is a normalised probability. In addition, the gamma variable $\gamma_t(i)$ can be written as the sum of the $\xi_t(i,j)$ terms over all j for $1 \le j \le N$. That is,

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \qquad (6.11)$$

Having determined its variables, the re-estimation procedure is now presented.

6.3.6.2 Parameter Re-estimation Formulae

The Baum-Welsh re-estimation procedure incrementally updates the values of the parameters of the hidden Markov model M. The initial values for π, A and B are discussed later. The updated values of specified below:

$$\begin{aligned}\bar{\pi}_i &= \textit{expected number of times in state } S_i \textit{ at time } t=1 \\ &= \gamma_1(i)\end{aligned} \tag{6.12}$$

$$\begin{aligned}\bar{a}_{ij} &= \frac{\textit{expected number of transitions from } S_i \textit{ to } S_j}{\textit{expected number of transitions from } S_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}\end{aligned} \tag{6.13}$$

$$\begin{aligned}\bar{b}_j(k) &= \frac{\textit{expected number of times in } S_j, \textit{ observing } x_k}{\textit{expected number of times in } S_j} \\ &= \frac{\sum_{t=1}^{T} \gamma_t(j)}{\textit{observing } x_t}\end{aligned} \tag{6.14}$$

6.3.6.3 The Algorithm (Baum-Welsh Re-estimation)

The following algorithm performs the re-estimation of the parameters of a set of HMM $\{M_w: 1 \le w \le W\}$ representing the W words in the system's lexicon.

Algorithm:

```
For w := 1 to W do                                   { W is the number of words in the lexicon }

    Initialise M_w {=(A,B,π)}
    Initialise M̄_w {=M_w=(A,B,π)}                    { To cater for the first time in Repeat loop }

    For t_w := 1 to NumTrainingPatterns do
        Repeat
            M_w ← M̄_w                                { Copy the new set of parameters M̄ over old M }
            Calculate α_t(i) and β_t(i) for all t, 1≤t≤T and all i, 1≤i≤N
            Calculate ξ_t(i,j) and γ_t(i) for all t, i and j

            π̄ ← re-estimated initial state distribution π
            Ā ← re-estimated transition matrix A
            B̄ ← re-estimated observation probability distribution B
            M̄_w ← (π̄, Ā, B̄)

        Until ( P(X | M̄_w) <= P(X | M_w) )
    EndFor { t_w }
EndFor { w }
```

Scaling Requirements

Since probability values are in the range 0 to 1, it is necessary to be aware of the possibility of *underflow*. Underflow occurs when the values in the probability variables become too small for the precision of the variable type (eg. real or double) due to excessive multiplication. One way to solve the problem is to increase the precision range of the probability variables. This however increases the size of the machine word needed to store them and will require addition and multiplication routines to process them. These overheads will seriously degrade the execution times of the HMM algorithms. By scaling, the problem of underflow and machine constraints can be circumvented. The details of scaling are contained in [Rab89].

6.4 HMMs in Connected Word Recognition

A level building algorithm similar to the one mentioned in chapter 5, is one approach for solving connected word recognition using HMMs. The algorithm is very similar to the level building algorithm used by the template matching approach. The only difference is that maximum statistical probabilities are used in place of minimum accumulated distances, and the reference words in the lexicon are word HMMs with parameters describing each word instead of the reference word patterns found in template matching.

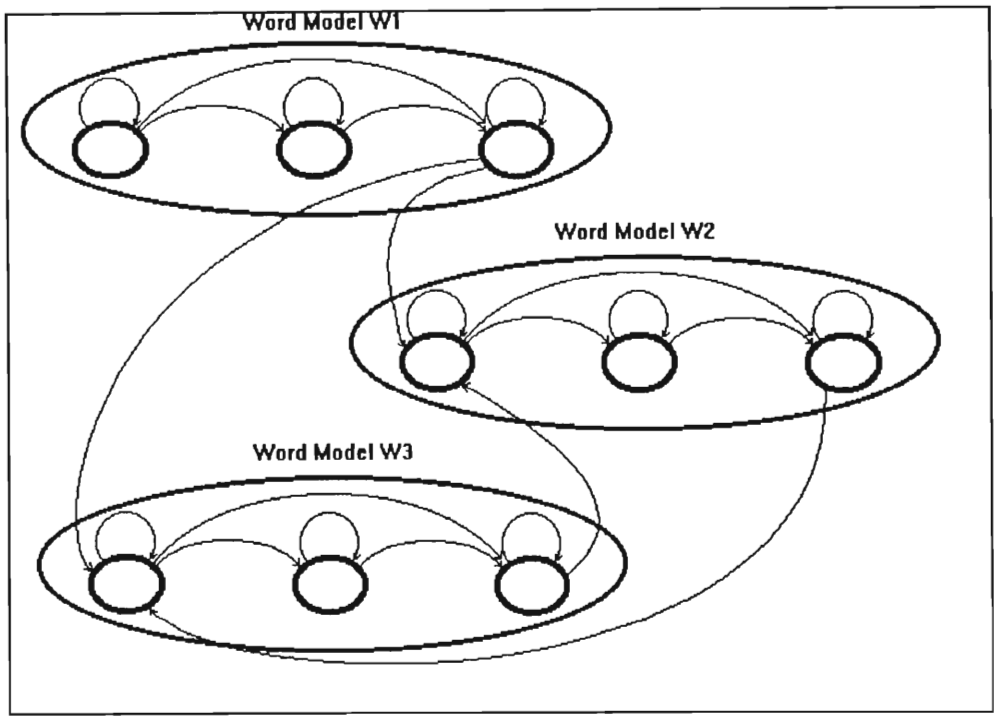

Figure 6.2 Connected Word Recognition using HMMs

6.4.1 Level Building using HMMs

The CWR problem using HMMs is that of determining the state path (where the next state in the path can be the current state, a subsequent state in the current word model or another state in another model) which maximises the probability of observing the sequence of feature vectors of the input pattern.

The level building algorithm incrementally (starting from level 1) constructs 'levels' which determine the number of words considered to be in the input pattern at that level. At each level, the best state path in every state and at every time slot is determined by starting new paths from the best state at every time of the previous level in order to determine all subsequent best paths using the Viterbi search with the reference models in the lexicon. The level building algorithm is presented below in a form easy to implement in software.

Algorithm

```
For ℓ := 1 to MNL do                    { MNL is the maximum number of levels - ie. for all other levels}
    For w := 1 to MNW do                { MNW is the maximum number of words in the lexicon }
        If (ℓ = 1) then
            δ1(1) := b1^w(x1)           { Initialisation for ℓ = 1; the w superscript denotes word model }
            For i := 2 to N do          { N is the constant number of states in the HMMs)
                δ1(i) := 0
            EndFor { i }
        Else
            δ1(1) := 0
            For t := 2 to T do
                If ( LevelProb(ℓ-1,t-1) > (δt-1(1)*a11^w) ) then
                    δt(1) := LevelProb(ℓ-1,t-1)*bj^w(xt)
                    αt(1) := t-1        { α here is a temporary backpointer and not the
                                          forward variable }
                Else
                    δt(1) := δt-1(1)*a11^w
                    αt(1) := αt-1(1)
                EndIf
            EndFor { i }
        EndIf
        For t := 2 to T do              { T is the # feature vectors in the input pattern }
            For j := 1 to N do
                maxtemp := δt-1(1)*aij^w
                argmaxtemp := 1

                For i := 2 to N do
                    If ( (δt-1(i)*aij^w) > maxtemp) then
                        maxtemp := δt-1(i)*aij^w
                        argmaxtemp := i
                    EndIf
                EndFor { i }

                δt(j) := maxtemp * bj^w(xt)
                αt(j) := αt-1(argmaxtemp)
            EndFor { j }
        EndFor { t }

        For t := 1 to T do
            Prob(ℓ,t,w) := δt(N)
            BP(ℓ,t,w) := 0              { BP array stores the backpointers }
        EndFor { t }
    EndFor { w }

    For t := 1 to T do
        tempmaxprob := Prob(ℓ,t,1)
        argmaxprob := 1
        For w := 2 to MNW do
            If ( Prob(ℓ,t,w) > tempmaxprob ) then
                tempmaxprob := Prob(ℓ,t,w)
                argmaxprob := w
            EndIf
        EndFor { w }
        LevelProb(ℓ,t) := tempmaxprob
        LevelBP(ℓ,t) := BP(ℓ,t,argmaxprob)
        LevelWd(ℓ,t) := argmaxprob
    EndFor { t }

EndFor { ℓ }
```

```
{ Determine the optimum number of levels and the best ending word's state }

bestlevel  := 0
tempmax := -0.1
For ℓ := 1 to MNL do
        If (LevelProb(ℓ,T) > tempmax) then
                tempmax := LevelProb(ℓ,T)
                bestlevel := ℓ
        EndIf
EndFor { ℓ }

{ To generate best sequence of words, call function BestWords() as follows: }
BestWords(T,BestLevel)

{ Dynamic Programming BackTracking Routine }

Def BestWords(time, level)
        If (level > 0) then
                BestWords(LevelBP(level,time),level-1)
                Output(LevelWd(level,time))
        EndIf
EndDef
```

6.5 Comparison Between Template Matching and HMM in WR Systems

Chapter 5 and 6 are similar in that they study *word recognition* techniques which use the *word* as the smallest distinguishable unit of speech. The techniques are very different in other respects.

Firstly, the template matching techniques match the input pattern (ie. its feature vectors) *directly* against reference patterns in the lexicon. The match is on real values representing the patterns in time. HMMs on the other hand, use statistical information about the input pattern. This information is embedded in the parameters of the HMMs of each word in the lexicon. Thus the parameters of each word model can be thought of as the reference "patterns" in this approach. The reference set of parameters which result in the highest probability of the input pattern being generated by it, is the "winning" word in this system.

Secondly, a further point of departure of the two systems is succinctly expressed by De Mori et al (following Moore's argument):

> "However, from a computational point of view the Markov models require an order of magnitude less storage and execution time; where the DTW based techniques have a very simple training phase (only data collection) and a very complicated recognition phase, Markov models are just the reverse. It has been overwhelmingly agreed that Markov models provide the correct balance for any practical system (Moore, 1984)." [DeM90]

Chapter 7 - Implementation

7.1 Chapter Outline

Several approaches to speech recognition described in the previous chapters were implemented in this project. In this chapter, the practical aspects and results of the following topics are presented:

1. The hardware used to sample, amplify and filter the speech signal.
2. The preprocessing functions used to extract features from the speech signal (see chapter 3) are analysed using graphical representations of the functions. The graphs help one to familiarise oneself with the acoustic nature of the speech sounds.
3. A ZAPDASH-type segmenter (see chapter 4).
4. A speaker-dependent, isolated word recognition system using the DTW algorithm (see chapter 5).
5. A connected word recognition system using the one-stage DTW algorithm (see chapter 5).
6. A speaker-independent, isolated word recognition system using left-to-right discrete HMM for the words in the lexicon (see chapter 6).
7. The front-end of a continuous speech recognition system using the self-ordering map to classify frame-sized segments of the speech signal (see chapter 4).

7.2 Hardware Considerations

The hardware can be divided into three main parts:

- the front-end preprocessor
- the digital computer controlling the digital speech processing applications
- the postprocessor providing audio-feedback.

7.2.1 Front-End Preprocessor

The front-end preprocessor consists of the following hardware components (see chapter 3):

- a dynamic microphone
- a preamplifier
- a low- and high-pass filter
- an I/O card which performs analogue-to-digital (AD) conversion and interfaces with the digital computer

A microphone should be mounted on a stand to avoid noise generated when it is shaken. The best signals are generated when the speaker talks into the microphone at close range. This removes much of the background (office) noise. It will however make the system vulnerable to 'noise' generated from the opening and closing of the lips and 'puffs' of air at the end of words by being so close to the microphone. This must be monitored and can often be eliminated moving slightly further away from the microphone.

The pre-amplifier and filters are connected in series to 'correct' the input signals coming from the dynamic microphone. The pre-amplifier amplifies the input signal (of the order of 10 millivolt) to a value typically in the range -10 to +10 volts. The range is determine by the range accepted by the AD converter. If the voltage is over-amplified so that it exceeds the range accepted by the AD converter, the result is amplitude clipping. On the other hand, if the voltages received by the AD converter are small, the effective resolution of the AD converter is reduced. For example, if the maximum voltage received by an AD converter with 12-bit resolution and -10 to +10V range were 3V, then the effective resolution is reduced by 70%. To cater for this, a variable resistor of discrete steps allows one to select several amplification gains (×100 - ×10000) to ensure maximum usage of the allowable range of voltages.

The amplified signal passes through a low-pass filter (6th order Chebychev with a slight dip in bandpass) with cut-off frequency f_c at 3.4kHz. The attenuation of the power in the signal at 5kHz is about -50dBs which is regarded as 'silence'. A sampling frequency of 10kHz could therefore be used so that the Nyquist rule was adhered to and there was little chance of high-frequency foldback. A discussion of these consideration was undertaken in chapter 3. A high-pass filter was used to eliminate the frequencies less than 75Hz using a 4th order Butterworth filter with f_c at 175Hz. The main aim of the high-pass filter was to eliminate the low-frequency noise especially the "mains' hum".

The preamplified and filtered (analogue) input signal, then feeds into the PC-30 Input/Output interface card which is inserted into an expansion slot of a 12MHz-AT personal computer. The card contains the AD 574 analogue-to-digital conversion (ADC) chip, an 8253 timer chip and an 8255 parallel peripheral interface (PPI) chip for performing the AD conversion of the analogue signal.

There are two methods of sampling using the PC-30 hardware namely, software clock sampling (SCS) and interrupt controlled sampling (ICS). In both cases, the AD 574 chip is interfaced through the PPI.

Software Clock Sampling (SCS)

In SCS, the AD conversion of the signal is controlled by software. Below are the steps taken by the sampling routine and the Pascal code associated with them:

0. Initialise sample_count to 1 to mark the first AD conversion

1. Initialise the PPI to basic I/O mode (see appendix B for other modes)

   Port[$703] := $92

2. Use PPI Port C ($702) for channel selection and to clear bit C_0

   Port[$702] := (channel SHL 4) + 2

3. Use PPI Port C ($702) for channel selection and to set bit C_0

   Port[$702] := (channel SHL 4) + 3

Step 2 followed by 3, ie. the clearing and setting of bit C_0, causes a positive edge trigger to initiate the start of AD conversion controlled by software.

4. Wait $\pm 40\mu s$ till the end of AD conversion. This is achieved by a delay loop. The results of the AD conversion can be read from port A ($700) and B ($701) of the PPI at the end of conversion.

5. Combine the converted digital values stored in port A (least significant bits or LSBs) and port B (most significant bits or MSBs) of the PPI in the following way:
   i. mask off the four most significant bits (MSB) of port B because the resolution of the ADC is only accurate to 12-bits.
   ii. Shift the result computed in i. to the left by 8 bits, thus positioning the most significant 4 bits of the 12-bit word.
   iii. Add the 8 LSBs stored in port A to the result of ii. and store the entire sample in an array of consecutive samples.

   That is,

   Sample[sample_count] := ((Port[$701] and $0F) SHL 8)+Port[$700]

6. Increment sample_count by 1 and goto step 2 until the required number of samples have been taken.

The delay loop determines the frequency at which sampling takes place. An obligatory minimum delay of 40μs is needed for the AD conversion to be accomplished. This implies that the maximum sampling frequency (using this ADC chip) is $f_{max} \approx 1/40\ \mu s = 25kHz$. Sampling any faster will result in starting the conversion of a new sample before the old sample has been converted to its digital value.

Determining the sampling frequency requires much effort using the SCS method. Both the number of machine instructions and the time ('clock ticks') taken to execute each machine instruction needs to be accurately determined in order to evaluate the precise sampling frequency. The ICS method allows one to set and control the sampling frequency and was therefore adopted in this project.

Interrupt Controlled Sampling (ICS)

The ICS triggers AD conversions at precisely determined time intervals using the 8253 programmable timer (on the PC-30 board. The sampling rate can be selected by the user and remains constant throughout sampling unlike SCS.

A 'start conversion' is triggered by the timer clock counter terminating and issuing a pulse which the PPI uses to trigger the AD 574. When the AD conversion is finished, the AD 574 generates an interrupt request (IRQ 5) to the processor. An interrupt service routine was developed to stores the converted digital value, increment the sample counter (the sample_count above) and then can be used to perform other tasks. The next 'start conversion' cycle begins with the timer counter reaching its terminal value. This procedure is terminated (when the number of samples required has been met) by disabling the ICS mode.

7.2.2 Postprocessor

A simple 'no frills' postprocessor was developed to play-back the digitised speech samples stored in the memory of the controlling PC. The output of one (of the three) digital-to-analogue chips on the PC-30 board was connected through an amplifer to a speaker. A simple ICS routine was designed to control the output of the digital samples at the same frequency at which they were initially sampled. A graphical module was developed to highlight regions of the (graphical form of the) speech signal, explode the selected region and output the samples of the region as feedback to the postprocessor. It was found very useful when learning how to associate parts of the graphical signals with the acoustic sound it represents.

7.3 Preprocessing Functions

All the preprocessing functions described in chapter 3 were experimented with and studied for possible use in speech recognition systems. A series of graphical representations of these functions is presented in this section highlighting the main observations. The results are divided into the two types of preprocessing functions: temporal or amplitude-time functions and spectral or frequency-time functions.

7.3.1 Temporal Functions

The following 'short-time' temporal functions are discussed in this section:

- Energy (or intensity) function
- Peak-amplitude function (maximum absolute amplitude in each frame)
- Zero-crossing function
- Turning-points function

A diagonistic signal graphing system was constructed to analyse digitised speech signals. It has the following features:

- Graph the output of the preprocessing functions
- Zoom in on any portion of the signal
- Play-back a portion of the signal to the user through the postprocessor

The graphing utilities are useful aids for the person performing the tedious task of manually labelling frame-sized training segments of speech. The user interface of this graphics package is shown in figure 7.1. The following options (see box options at the base of the interface) are available:

- Get File - Select an old sample file from disk
- TakeSamples - Capture a new speech signal using the microphone
- Play Back - Play back the portion of the speech sound in the Explode Signal window
- Sig. Stats - Used to display the results of several temporal preprocessing functions. They are displayed on a full screen as depicted in figure 7.2.

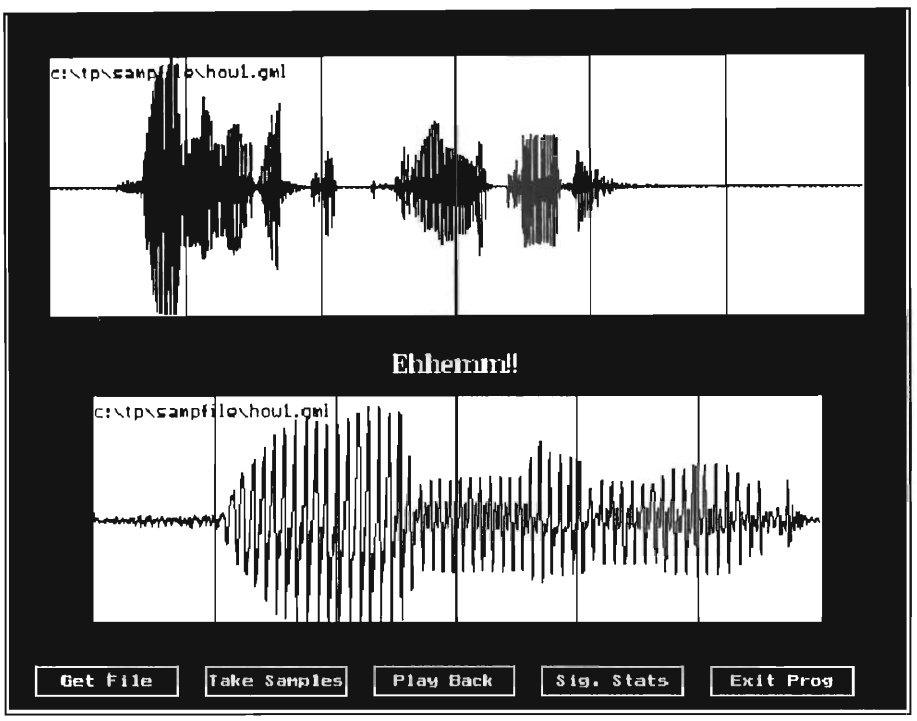

Figure 7.1 Graphics Interface of the Analysis Package

Generally, the amplitude functions are used for broad segmentation into regions of vocalic sounds and to distinguish between sound and silence. The zero-crossing and turning-points functions measure the number of oscillations in the signal and therefore distinguish regions of noisy (high frequency) sounds. See the "h", "j", "ks", "t" sounds in the utterance *how many objects are on the table?* in figure 7.2.

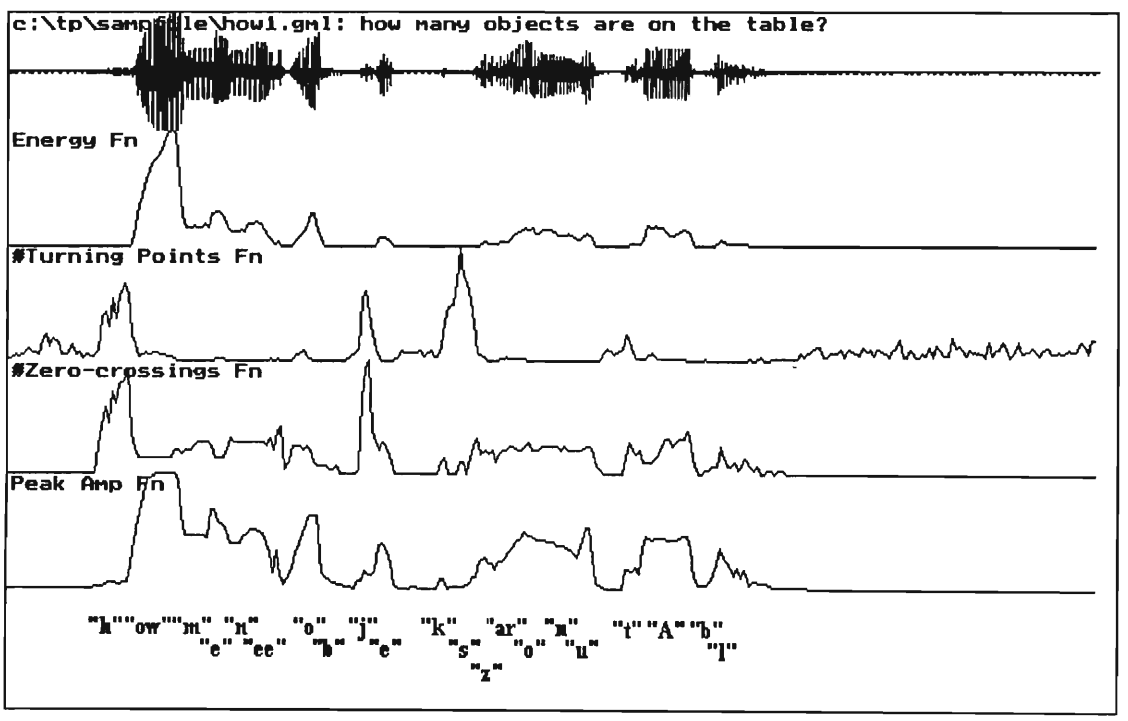

Figure 7.2 Graphs of the Preprocessing Functions

7.4 Segmentation Algorithm

A segmentation algorithm based on the ZAPDASH segmenter discussed in chapter 4 was attempted. The results however were not encouraging. The major difficulty in building a robust segmenter is determining the thresholds on the output of the temporal preprocessing functions. Broad heuristics to achieve this are presented in the decision tree in figure 7.3.

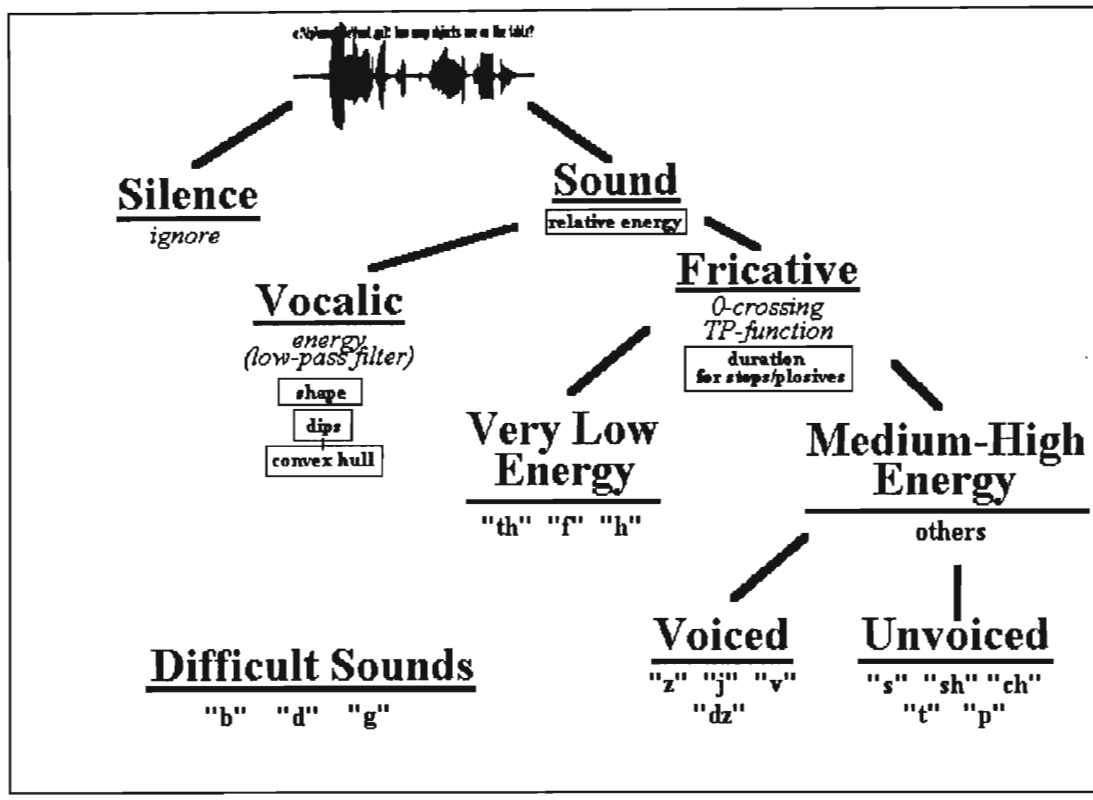

Figure 7.3 Segmenter Decision Tree

The graphical results of the segmenter based on the decisions above are presented in figure 7.4. Examples of undersegmentation and oversegmentation are highlighted by circles in the figure. The circle on the left shows undersegmentation due to too large a choice of threshold value in the convex deficiency procedure. The circle on the right demonstrates oversegmentation because of too small a threshold value on the zero-crossing function.

The convex deficiency procedure is used to distinguish dips in large segments of vowel-like resonant regions of the signal. It is discussed by Waibel [Wai88].

Figure 7.4 Graph of Segmentation of the utterance:
How many objects are on the table?

7.5 IWR System using the DTW Algorithm

An isolated word recognition (IWR) system was implemented using a standard dynamic time warping (DTW) algorithm [DeM90] to align the input patterns of isolated words with reference word patterns. The DTW algorithm used a slope constraint of zero which exhibits marginally poorer results compared with a slope constraint of one [Sak78] but is quicker. The IWR DTW system was tested using several subjects, including: two male adults, two female adults and two young females (±13 years old). The results are presented below in two forms. In the first case, the five best accumulated distances for test word patterns matched with the reference word patterns of a lexicon of spoken digits (with the word "AY" in *say*), is tabled to show the relative scores of the competing reference patterns. In the second case, general statistics are tabled regarding the performance of the system.

The tables 7.1 and 7.2 are interesting in so far as to highlight potential mismatches of the speakers in subsequent testing sessions. For example, speaker KLL's utterance of "three" results in a best match with the reference pattern "three" (accumulated distance = 14.75), closely followed by second best matching "two" (accumulated distance of 14.91). The same, although not as close, may be said of KLL's "nine" with reference patterns "nine" and "five" potentially being confused. Speaker GML's "nine" is potentially confused with "one". The reason for the 'closeness' between the accumulated distances in each of the above instances can be attributed to the 'closeness' in the sounds of the 'confused' words.

Take for example GML's "nine". If the first "n" sound were said quickly, 'running into' the following "I" sound, the result is acoustically quite similar to "w u n" (ie. one) since the "w" is usually a short sound and the "u" sound is quite similar to the "I" sound except in duration. The DTW algorithm has the ability to non-linearly time align sounds, overcoming duration problems and hence the close matching of "one" for "nine".

Input Word	Time taken	1st Word	2nd Word	3rd Word
"zero"	62s	"0":22.78	"2":25.32	"7":34.18
"one"	53s	"1":15.93	"7":24.27	"3":26.26
"two"	52s	"2":18.59	"7":23.41	"1":28.23
"three	52s	"3":17.33	"2":24.24	"7":25.23
"four"	61s	"4":21.87	"2":30.90	"7":33.51
"five"	78s	"5":22.04	"1":28.46	"9":28.72
"six"	39s	"6":14.30	"8":18.93	"7":29.17
"seven"	52s	"7":13.65	"2":21.82	"1":24.74
"eight"	51s	"8":14.39	"6":25.08	"7":31.97
"nine"	66s	"9":**24.98**	"1":**25.38**	"3":28.15

Table 7.1 Male Speaker (GML) Results of Best 3 Matching Reference Words and their Minimum Accumulated Distances

Input Word	Time taken	1st Word	2nd Word	3rd Word
"zero"	45s	"0":16.12	"1":27.19	"7":27.77
"one"	39s	"1":11.38	"7":20.05	"5":20.11
"two"	33s	"2":10.42	"8":14.59	"6":17.28
"three"	35s	"3":**14.75**	"2":**14.91**	"8":17.95
"four"	25s	"4": 6.81	"8":13.06	"6":14.20
"five"	38s	"5":12.65	"1":22.14	"4":24.40
"six"	11s	"6": 3.47	"8": 8.11	"4":13.12
"seven"	43s	"7":11.21	"2":18.78	"1":20.51
"eight"	17s	"8": 5.13	"6": 9.22	"4":15.00
"nine"	59s	"9":**21.45**	"5":**22.84**	"1":25.65

Table 7.2 Female Speaker (KLL) Results of Best 3 Matching Reference Words and their Minimum Accumulated Distances

Speakers	11 word lexicon	28 word lexicon	120 word lexicon
GML (male)	99.4	94.6	91.6
NKA (male)	99.6	95.4	-
AEM (female)	99.6	93.6	-
KLL (female)	99.1	94.3	89.2
CAK (young female)	97.0	87.5	-
LLL (young female)	99.4	92.5	-
Average recognition time	± 20s	± 1 min 20s	± 8 min

Table 7.3 Recognition Accuracy using different sized Lexicons

The above table (7.3) of results shows that the DTW isolated word recognition algorithm is fairly robust both in terms of the size of the vocabulary and for people of different ages and sexes. The results compare favourably with those of Pitchers [Pit90] whose system achieved 'close to 100%' (10 words), 93.6% (30 words) and 87.0% (120 words) recognition accuracy [Pit90]. He also suggests ways of reducing the errors which tend to occur for relatively few, frequently confused words. By determining these confusable words, one can introduce other reference patterns for them in the lexicon in order to reduce the error rates. One important difference between Pitchers' results and these, is the time taken to recognise words. He uses the ordered graph search [Bro82] which has a three to one saving in the number of dynamic time warpings performed by the algorithm over the standard algorithm implemented in this project. His 120 word recogniser achieves recognition times of 27 seconds compared with 8 minutes by this system. Another important reason for the discrepancies in time are that his system was aided by a maths coprocessor (80287) whereas the system used in this project did not have one.

7.6 CWR System using the "One-Stage" DTW Algorithm

A connected word recognition (CWR) "One-stage" dynamic time warping (DTW) algorithm was developed with a view to recognising connected strings of spoken digits. A typical application for such a system is an automatic audio-dial telephone.

The results of this system were generally poor (except for recognising single words which achieved accuracies similar to the isolated word DTW recognition system) for overall sentence recognition. It managed to 'spot' most of the words in their correct positions in the sentence but would often omit one or two words, or include other words which were

not spoken. For example, the author uttered the following digit sequence "842819" and the system matched this with "84626819". Another interesting result of the CWR DTW system was found with the introduction of the reference word pattern "ay" (in *say*) in the lexicon to reduce a class of errors involving recurring 'over-detection' of the digit "eight". The "eight" reference word pattern typically contains an initial vocalic region (the start of the "ay" vowel sound), followed by silence to form the stop and finally the plosive "t". It was found that the recogniser would match an "eight" if there was ever a silence period between two connected digits where the silence period was preceded by a vocalic sound and followed with a fricative sound eg. "one...two...three" was recognised as "1-8-2-8-3". This class of errors was reduced by adding the "ay" sound into the lexicon.

The reason for poor sentence recognition rate is due to the fact that the algorithm uses a slope constraint of zero which allows unrestricted non-linear time alignment. This results in a 'freedom' to align very small parts of the input pattern with entire reference words eg. the recurring "6" in the example described above. By adding further constraint such as a slope constraint of one and an adjustment window, it is thought the results of overall sentence recognition would improve closer to the >99% accuracies reported by [Sak79]. The other algorithms by [Mye81] and [Ney84], report similar results to those of [Sak79].

A major consideration in the connected digit recogniser is the slow response time. To recognise a string of six spoken digits typically required between two and three minutes. Again, it is mentioned that the PC-AT (80286) used to execute the speech algorithms described here, did not have a maths coprocessor which would reduce recognition times considerably. In addition, algorithmic methods of reducing the number of DTW matches can be found in Myers and Rabiner [Mye81] and Rabiner and Schmidt [Rab80].

7.7 IWR System using Left-to-Right Discrete HMMs

An isolated word recognition (IWR) system was implemented using stochastic, discrete hidden Markov models for each word.

Because discrete HMMs are used, a preprocessing stage (performing the k-means segmental algorithm) was required to determine a set of discrete codevectors representing the speech sounds in the language. The representative codevectors closest to each speech feature vector in the observed signal were used instead of the actual vectors of the signal. The observation symbol probability distribution $\{b_j(k)$: $1 \le j \le N$ and $1 \le k \le M$, N is the number of states in the HMM and M is the number of codewords in the codebook$\}$ for each word was determined in terms of these codevectors, x_k.

A left-to-right model configuration was used for the word models. The word models were trained by a procedure based on the Baum-Welsh training algorithm described in chapter 6 but adapted for the left-right model. The Baum-Welsh re-estimation algorithm involves determining the word model parameters (ie. the state transition probabilities a_{ij} and the observation symbol probabilities $b_j(k)$). In the left-right model, the initial probabilities of being in any state but the first are zero. That is, $\pi_1 = 1$ while $\pi_j = 0$ for $j \neq 1$. It is also important to ensure that the observation symbol probabilities $b_j(k)$ are greater than zero at all times in the algorithm (which requires checking after each iteration of the re-estimation algorithm) and that the following statistical conditions are adhered to:

$$\sum_{i=1}^{N} a_{ij} = 1 \qquad 1 \le j \le N$$
$$\sum_{j=1}^{N} b_j(k) = 1 \qquad 1 \le k \le M \tag{7.1}$$

where M is the size of the codebook.

Input Word	Time taken	1st Word	2nd Word	3rd Word
"zero"	101s	"0":-185	"1":-276	"AY":-280
"one"	81s	"1":-87	"7":-130	"9":-138
"two"	91s	"2":-65	"AY":-117	"3":-119
"three	93s	"3":-106	"8":-178	"0":-195
"four"	75s	"4":-167	"0":-209	"5":-221
"five"	103s	"5":-173	"3":-270	"1":-281
"six"	35s	"6":-71	"4":-88	"8":-92
"seven"	61s	"7":-130	"1":-132	"0":-141
"eight"	66s	"8":-89	"4":-158	"AY":-167
"nine"	106s	"9":-124	"1":-196	"7":-205
"AY"	68s	"AY":-80	"8":-103	"1":-118

Table 7.5 Male Speaker (GML) Results of Best 3 Matching Word HMMs and their Accumulated log Probabilities from the Viterbi Search

Table 7.5 shows a set of results determined by the Viterbi search on 11 reference digit HMMs. The negative scores are a result of working with the log of the probabilities of the variables in the algorithm. This was done to avoid having to scale the probability values, which are vulnerable to underflow as a result of excessive multiplication. Any probability with a value of zero was forced to be $1/10^4$ (since log a for $a <= 0$, is undefined).

The overall recognition accuracy of the isolated discrete HMM recognition system for three speakers on the 11 word (the digits "0"-"9" and "AY") dictionary, was fairly poor: 86% (GML), 74% (KLL) and 59% (RPJ). Rabiner [Rab89], for example, reports a recognition error rate of only 3.7% for a similar system (though with additional features). The poor results are attributed to the following reasons:

- 100 discrete codevectors (from vector quantisation using the k-means segmental algorithm - see chapter 4) were constructed from a training set of speech feature vectors made up predominantly from GML's (male) and to a lesser extent KLL's (female) speech signals. This is perhaps the reason for the significantly poorer results of RPJ and, to a lesser extent, KLL.
- only one signal per word was used to train the HMMs. More than one training signal are recommended by Rabiner to ensure enough statistical information is obtained for each word model [Rab89].
- Rabiner's system was trained by 100 utterances for each digit, each utterance by a different speaker. His system also used enhanced speech features and it is not clear (but quite likely) whether he also used duration probabilities to improve the performance of the system.

Considering the above restrictions, the results of this system are as expected. Further results were not obtained due to time constraints.

7.8 A Front-End Process of a CSR System

Two continuous phoneme recognisers were implemented.

The first algorithm used the k-means segmental algorithm to categorise speech sounds as any one of 100 codevectors. These vectors were then mapped onto phoneme-type classes. This is achieved by gathering a set of manually pre-classified feature vectors and labelling the codevectors to which they are closest with these class labels. A Euclidean distance measure was used to determine the 'closeness' between vectors made up of cepstral coefficients. Because there are only about 45 phonemes, several codevectors may be mapped onto a particular phoneme class. One must ensure that the converse is not true, ie. a codevector is not labelled by several phoneme classes. Classification is performed by reading in a string of input feature vectors (of the input word or sentence), then matching each feature vector with its closest codevector and assuming the codevector's class label as the class label of the input feature vector.

The second phoneme recogniser was based on Kohonen's self-organising map algorithm. In addition, the LVQ1 algorithm was used to attune the class boundaries to a better approximation of the Bayesian decision surface.

The first algorithm provides a very rough estimation of the class boundaries as does the self-organising map without the fine-tuning due to one of the LVQ algorithms. The recognition results of the first algorithm are thus expected to be poorer than those of the second using the LVQ1 algorithm.

The k-means segmental algorithm achieved recognition results ranging from 63-72% accuracy in different sentences spoken by the author. The system generates several interesting errors but due to time constraints, ways to avert these were not examined. The self-ordering map with the LVQ1 algorithm achieved phoneme recognition accuracies ranging from 85-93%. Kohonen reports accuracies of between 96-98% for the integrated continuous speech recogniser with a self-ordering map as a front-end preprocessor [Koh88]. This system appears to have excellent potential as a continuous speech recognition system because it is both simple to implement, has excellent (real-time) response and good recognition potential.

Chapter 8 - Natural Language Processing (NLP)

8.1 Introduction

Natural Language Processing (NLP) is the term used to describe the higher-level processes of speech and language in communication. NLP research has historically dealt with *text* rather than *speech* communication because speech has so many problems of its own (eg. noise, coarticulation, uncertainty of speech units) that it tends to distract from the focus of NLP. In this chapter, speech-specific problems are therefore ignored while chapter 9 on expert systems and speech understanding deals specifically with the implementation and integration of NLP in speech recognition systems. In fact, the integration of NLP in a speech recognition system is considered to be the point of departure from a mechanical pattern recognition system to an 'intelligent' speech understanding system.

Much of the research of NLP is inspired by the way humans understand language. Thus, the psycholinguists, cognitive psychologists and more recently the cognitive scientists have been important in the development of NLP theories. At the same time, computer scientists with their 'high-level programming languages' and 'artificial intelligence' have also played an important role in developing this field.

The argument for including NLP in speech recognition systems is supported by empirical evidence. For example, the HARPY system (detailed in chapter 9) improved its results dramatically from 42% sentence recognition accuracy when relying on phonetic knowledge alone to 97% when knowledge about the syntax, semantics and pragmatics of the language (ie. NLP) was included [McT87]. In addition, a number of psychology experiments on human subjects have shown that the results of the human recognition system degrade significantly without the aid of language cues (eg. see [War83]).

8.1.1 Main Components in NLP

The main components in NLP are divided into three categories. These are:

- the syntax ie. the grammar or structure of a language
- the semantics ie. the notion of truth and the meaning of sentences
- the pragmatics ie. the intentions or meaning of the speaker in context

Rich and Knight [Ric91] list five components of NLP, dividing pragmatics into *discourse integration* and *pragmatic analysis*. They introduce *morphological analysis* as a separate component but in this chapter it is discussed under semantics.

The aim of this chapter is to examine the main issues in NLP and see how they might be incorporated or modelled in a NLP system.

8.1.2 History of NLP

The development of high-level computer languages, at almost the same time that psycholinguists like Chomsky introduced revolutionary theories on grammars, represents the real beginnings of NLP.

The landmark in its history was Winograd's SHRDLU[1] program [Gaz89a] developed in 1971. SHRDLU was a one-armed robot that manipulated blocks of different shapes and sizes when commanded to do so by a commander using English sentences (text not speech). The robot's world was restricted to a BLOCKWORLD (the blocks on a table) which reduced the complexity of the NLP domain. Although the robot was never built, its actions were simulated by a program written in LISP. SHRDLU achieved good results in understanding a subset of English commands and queries entered from the keyboard. The most important result of the program was to show that NLP could be 'solved' even though for a restricted language-domain.

SHRDLU's main achievements were:

- interpreting questions, statements and commands
- drawing inferences
- explaining its actions
- learning new words
- handling ellipsis (omissions in text) and pronouns

8.1.3 NLP in Text and Speech

There are several cases where ambiguities are found in the speech but not in the textual form of a sentence and vice versa. Two examples are given below to highlight this.

[1] 'SHRDLU' is derived from the last 6 letters of the 12 most frequently occurring letters in English.

Consider the following example [McT87]:

He gave the boy plants to water. 8.1

He gave the house plants to charity. 8.2

Example	Direct Object	Indirect Object	Other
8.1	plants	boy	to water (infinitive form of verb)
8.2	house plants	charity	-

Table 8.1 Roles of Words in Sentence

The above table shows the role (part of speech) played by the relevant words in sentence. How does one then decide that *boy plants* is not a composite noun or that *house plants* is not an indirect-direct object pair[1]? If the sentence is in text form, all possible readings must be considered and the one which makes the most sense is then accepted. Checking all possible options can be tedious. In spoken English, it rarely seems necessary to test all possible options. Slight timing and rhythmical features (the prosodic features) support one syntactic form of the sentence in favour of others. For instance in the spoken version of 8.1, one naturally inserts a slight but detectable pause between *boy* and *plants* to indicate that they are separate parts of speech. On the contrary, the words *house* and *plants* are spoken together (without a detectable pause) in example 8.2, reinforcing their role as a composite noun.

The importance of the duration of detectable silence and other prosodic features has generally been neglected in speech recognition research. This is probably because of the difficulty of expressing ideas like a 'slight pause' and the sensory effects of various types of rhythms. Waibel [Wai88] has covered some ground investigating these prosodic cues in speech but this is certainly an area for extensive development.

The previous examples show how the spoken form of the sentence can reduce the number of possible structural forms it can assume. The following example (from [Lad85]) demonstrates how the textual form of the words clarifies the meaning of the sentence where it is unclear in its spoken form.

The sun's rays meet. 8.3

The sons raise meat. 8.4

[1] Compound noun interpretation problems are discussed in detail by Jones [Jon85].

Thus the meaning of the similar sounding (differently spelt or not) words must be resolved from the context of the conversation or the structure of the sentence.

8.2 Syntax of a Language

Formal language theory as described by [Coh86] and [Aho73] offers important theoretical grounding in the structure or grammars of formal (ie. mathematically contrived) languages. The details are not included here as, despite being lengthy and rigorously defined, they are not adequate for describing the nuances of natural languages. Rich and Knight explain:

> "Grammar formalisms ... provide the basis for many natural language understanding systems. ... We should point out here that there is general agreement that pure, context-free grammars are not effective for describing natural languages. As a result, natural language processing systems have less in common with computer language processing systems (such as compilers) than you might expect." [Ric91]

Thus, instead of concentrating on the strict formalisms, a more practical approach is adopted using examples to explain the theories.

Informally, *syntax* is the structure of a language (in formal language theory, the syntax is called the *grammar* of a language). It is important for controlling the flow of thoughts and ideas in language communication. Without it, listeners or readers (called addressees[1] henceforth) would have great difficulty understanding each other because there are so many possible combinations of putting the words of a language together.

For example, try and decipher:

dog house man the the kicked the of out

for

the man kicked the dog out of the house

From the above example, it is clear that syntax is a language tool which limits the possible combinations of words by specifying the logical order according to their relationship with the other words in the sentence. The parts of speech and their ordering and context in the sentence therefore carry much meaning themselves. The addressees can use knowledge of the grammar to predict or at worst limit the choice of subsequent words, which helps significantly in speeding up the recognition process.

---

[1] And the speaker or author is the *addresser*.

On the other hand, syntactical well-formed sentences are often not necessary in natural languages. Omitting some of the syntactic details can sometimes enhance the meaning. Consider the following breathless tale of a scout to his general (it does not even have a verb):

thousands of them ... cannons ... horses ... chariots ... millions ...

The wounded and dying hero will certainly not use flowing, syntactically well-formed sentences in his last words but he is understood nonetheless. These examples show that the meaning of the message is ultimately what is important. The addresser uses various language tools to convey the meaning to the addressee in the best way possible. The syntax or structural form of the language is a tool but not all important.

Because the rules of syntax are common to both addresser and addressee, the addresser uses them to order his thoughts while the addressee uses them to verify and predict the addresser's thoughts and words. In speech recognition, predicting from the syntax of the sentence reduces the number of possible sentences. This enhances the recognition problem by limiting the search through the lexicon to the words of particular parts of speech. At first glance, prediction seems redundant (since one still has to verify what was predicted) but its importance is emphasised by the fact that prediction is naturally performed by people when they read or speak. One only has to read or speak a sentence, stopping short of the end, to prove that addressees constantly predict future ideas, words and even sounds. They will be able to complete (fairly accurately) the rest of the sentence and even get the gist of the rest of the paragraph or conversation. It seems as if the predicted options are stored in easy-to-get-at memory which speeds up the verification process and thus the overall recognition. What about the time taken to predict? People have the ability to process many things concurrently (eg. walking and talking) and in addition speech produces new symbols (ideas, words and sounds in increasing rates) at a relatively slow rate. Therefore the brain has time enough to predict future events from the lowest level (sounds) to the highest (themes).

Finally, Atkinson et al [Atk84] in their chapter on natural language acquisition show that the syntax of a language is assimilated naturally (almost unknowingly) by humans in infancy. It is therefore remarkable that students in schools struggle to grasp and conquer 'grammar' taught at school. The reason for this may become apparent later in this chapter after studying the role of semantics and its relationship with syntax. What one should see is that these components (syntax, semantics etc) of a language are intricately related and are often very difficult to separate. For teaching purposes separate analysis is considered

desirable because it results in fewer side-effects, but for people processing language this is not easily done.

8.2.1 Notation and General Concepts

A structured *sentence* in a language is made up of *clauses* ie. sentences, in which complete ideas are expressed, joined together by conjunctions. The clauses are made up of *phrases* which are groups of *words* expressing partial ideas contributing to the main idea of their clauses. Every word is associated with an atomic part of speech called a *terminal* eg. *boy* is a *noun, the* is a *determiner* and *with* is a *preposition.* A structured sentence therefore can be divided hierarchically into sentences, clauses, phrases (ie. *non-terminals*) and terminals. This division (or synthesis if processed bottom-up) process is called *parsing* the sentence. An example of a parsed sentence is shown in figure 8.1. The leaf nodes in the parse tree are the words in the sentence and one up from them are the terminals. The root node in the tree is the start symbol or sentence (S) in natural languages.

The parse tree is determined by the grammar of the language. The rules of the grammar are expressed by *production rules* of the form:

S → S conj S | NP VP | VP
NP → det noun
...

The production rules have a *rule head* (on the left hand side (LHS) of the arrow) and *rule body* (on the RHS of the arrow) and are interpreted as follows (see the production rules above):

Production Rule 1: The sentence (S) consists of either (|) two sentences joined by a conjunction (conj) or a noun phrase (NP) followed by a verb phrase (VP) or simply a VP on its own.

Production Rule 2: An NP consists of a determiner (det) and a noun.

The *terminals* or atomic parts of speech (eg. noun, verb etc) are typically written in small case while the *non-terminals* in capitals and small (eg. PP ≡ Preposition Phrase and AdvP ≡ Adverbial Phrase).

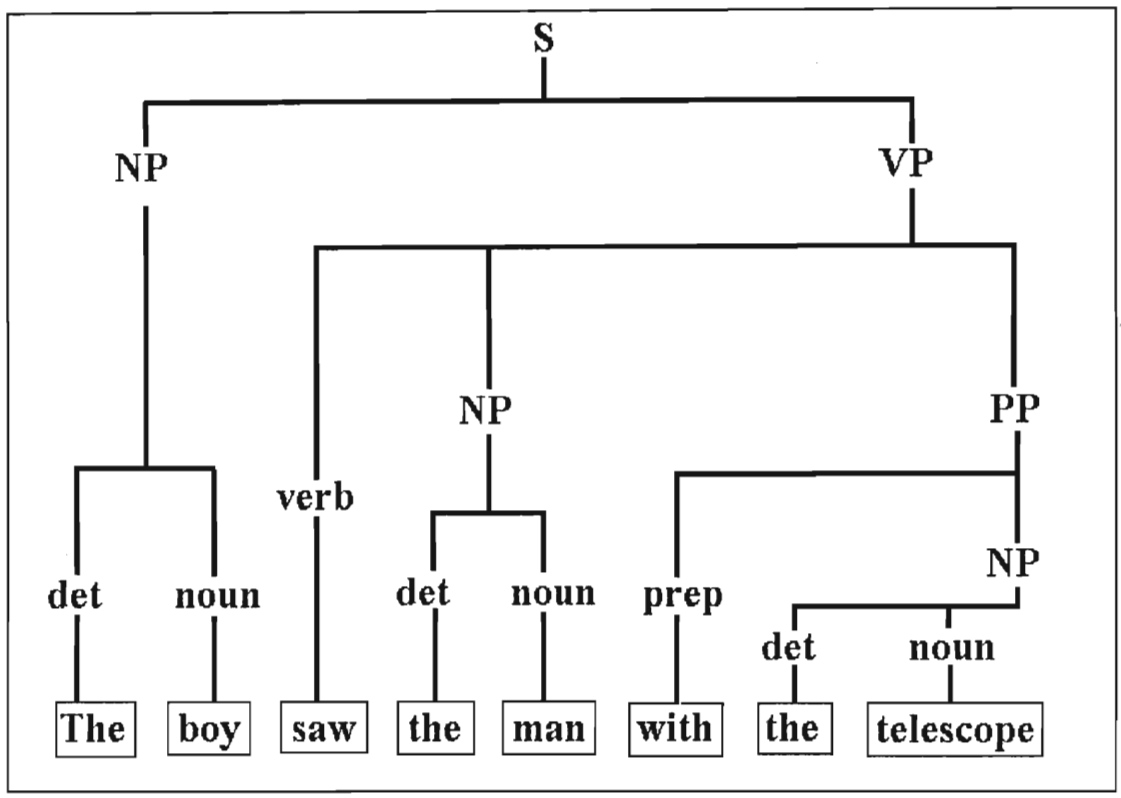

Figure 8.1 Parse Tree Diagram of the sentence: *the boy saw the man with the telescope*

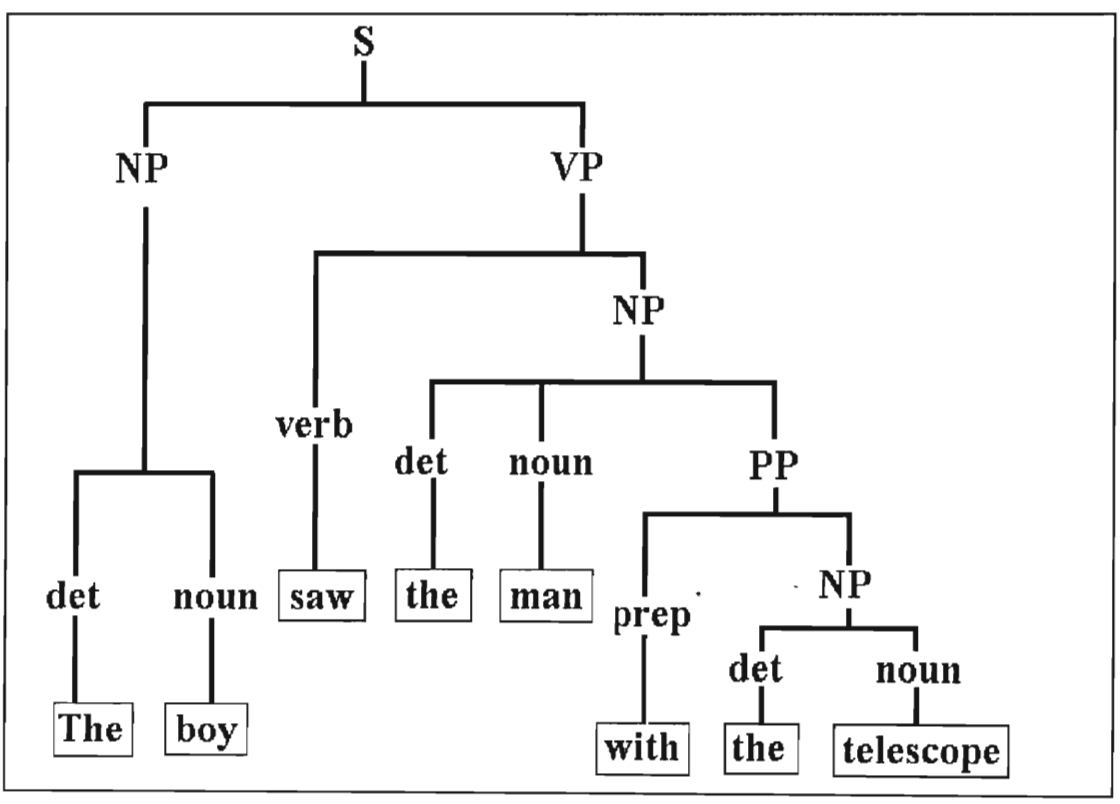

Figure 8.2 An Alternate Parse Tree to that of Fig. 8.1

Several classes of grammars[1] are defined in formal grammar theory which determine the form of the production rules and thus limit the types of sentences that can be found in a language.

There are three important considerations associated with parsing; the parsing strategy, the search strategy and structural ambiguity.

Parsing Strategy

The *parsing strategy* refers to whether a *top-down* or *bottom-up* approach is used. In the top-down approach, the parser starts at the root node and builds the parse tree by successive refinement (divide-and-conquer) of the non-terminals at each level in the tree until the terminal nodes matching the parts of speech of the words in the sentence are reached. The bottom-up approach starts by looking up the parts of speech of the words in the sentence (ie. finding the terminals). The consecutive parts of speech are then merged into the phrases they belong to hierarchically up the tree until the start symbol (S) at the root of the tree is reached.

Searching Strategy

A *searching strategy* is necessary to resolve potential path decision problems that arise when two or more production rules can be applied at some point in the parse. The searching strategy thus determines how the production rules are implemented during parsing. There are two well-known searching strategies, depth- and breadth-first (although several others do exist eg. the beam search and the iterative deepening technique). The main search strategies are explained by the following example.

[1] Ie. the Chomsky classes of grammars - Regular, Context-Free, Context-Sensitive and Unrestricted.

Example: *the cat bit the rat*

Using the following production rules:

S	→	NP VP	(P1)
NP	→	det noun	(P2)
VP	→	verb	(P3)
VP	→	verb NP	(P4)

1. Using top-down, depth-first search:

1.1	S	Start with start symbol
1.2	NP VP	Expand S by (P1)
1.3	det noun VP	Expand NP by (P2)
1.4	det noun verb	Expand VP by (P3). Done? No, more words to parse ∴ redo 1.4
1.5	det noun verb NP	Expand VP by (P4).
1.6	det noun verb det noun	Done? Yes. Stop.

2. Using top-down, breadth-first search:

2.1	S	Start with start symbol
2.2	NP VP	Expand S by (P1)
2.3	det noun verb	Expand NP by (P2) and VP by (P3). Done? No, more words to parse ∴ redo 2.3
2.4	det noun verb NP	Expand NP by (P2) and VP by (P4).
2.5	det noun verb det noun	Expand NP by (P2). Done? Yes. Stop.

3. Using bottom-up, depth-first search:

3.1	det	Look up speech category (LUSC) for *the*
3.2	det noun	LUSC for *cat*
3.3	NP	Synthesise (synth) 3.2 by (P2)
3.4	NP verb	LUSC for *bit*
3.5	NP VP	Synth verb by (P3)
3.6	S	Synth 3.5 by (P1). Done? No, more words to parse ∴ redo 3.5
3.7	NP verb det	LUSC for *the*
3.8	NP verb det noun	LUSC for *rat*
3.9	NP verb NP	Synth NP in 3.8 by (P2)
3.10	NP VP	Synth verb NP in 3.9 by (P4)
3.11	S	Done? Yes. Stop.

4. Using bottom-up, breadth-first search:

4.1	det noun verb det noun	LUSCs for each word in *the cat bit the rat*
4.2	NP VP NP	Synth 1st and 2nd det noun by (P2) and verb by (P3) in 4.1
4.3	S NP	Synth NP VP in 4.2 by (P1). Problem. Redo 4.2.
4.4	NP verb NP	Synth 1st and 2nd det noun in 4.1 by (P2)
4.5	NP VP	Synth verb NP in 4.4 by (P4)
4.6	S	Synth NP VP in 4.5 by (P1). Done? Yes. Stop.

From the above examples of the different combinations of searching and parsing strategies, the top-down depth-first (TDDF) and bottom-up breadth-first (BUBF) strategies seem to be the best performers. The TD parsing strategy decides early on what the structure of the sentence ought to look like. Therefore, in order to reduce the chance of an early incorrect prediction of the structure, the DF searching strategy is used to try and align the 'umbrella' structure with the actual parts of speech of the observed words in the sentence as early as possible.

On the other hand, the BU approach has the attribute that it can correctly determine all the terminals at the leaf nodes and therefore has the attribute of being able to predict the local best phrase matches of any local substring of consecutive terminals in some part of the sentence. Thus, the BU approach prefers the BF searching technique, which works laterally across the sentence. Good analyses of these and other searching techniques (with respect to general artificial intelligence problems) can be found in [Ric91], [Cha85] and [Win84].

Ambiguity

A sentence exhibits *structural ambiguity* if it has two or more possible parse trees (assuming the grammars are the same) ie. if there are several different ways of parsing the same sentence using the same set of grammar rules. Different parses alter the meaning of the sentence because words in different phrases perform different roles in the sentence. Compare the two possible parses of the sentence illustrated in figures 8.1 and 8.2:

the boy saw the man with the telescope[1]

In figure 8.1, the boy, while looking through the telescope saw the man whereas in figure 8.2 the boy saw the man who was holding or who owned a telescope. The meanings are clearly different.

Parsing is thus the process of extracting the structure of the sentence. It relies on the grammar rules, the parse and the search strategy.

Two processes related to parsing are *recognising* and *generating*. A recogniser is a system which determines whether or not a sentence is made up of the correct sequence of words determined by the grammar. A generator is a system which produces a correct sequence of words determined by the grammar. In formal language theory textbooks, recognisers and

[1] A variation of the 'more ambiguous' sentence *the silly robot saw the red pyramid on the hill with a telescope* [Win84].

generators are studied to introduce the concepts of grammars. The classes of systems used to recognise, generate or extract the structure of the sentence are:

- syntax recogniser - returns only whether the sentence was correctly recognised or not. It can therefore be used to verify the structure of a sentence but the "YES" or "NO" type answer it returns is not particularly useful to systems that are trying to uncover the meaning of the sentence.

- syntax generator - generates allowable sequences of words but is prone to combinatorial explosion if there are many options to choose from in the production rules. It is also characteristically a 'mindless' generation based on the production rules and search strategy and thus results in the prediction of 'silly' sentences.

- syntax parser - divides the sentence into its grammatical parts of speech in such a way as to 'understand' the structure of the sentence by building an abstract representation of it in memory.

Of the above three systems, the parser is the most flexible and 'intelligent'. It has both the ability to verify the correctness of a sentence (or phrase) and to predict possible parts of speech from the current state of the system.

There are several other methodologies for analysing the structure of sentences including statistical Markovian systems which use maximum likelihood probabilities to predict and verify the grammatical structure of sentences. They are a natural extension of the work discussed in chapter 4 and 6 and are not pursued further in this section. + references ****

8.2.2 Transition Networks

A transition network (TN, also called a graph [Coh86]) is a set of states joined together in some configuration by transition arcs. It has three components:

- A network name
- A list of declarations about initial states (denoted by *) and final states (denoted by +)

- A set of 3-tuples where the elements are:
    - the "from state"
    - the "to state" and
    - the "action" as a result of traversing the arc from the "from state" to the "to state"

TNs are limited in that they cannot have two arcs with the same "actions" leaving from the same node and are prone to redundancy (eg. the repeated *det* → *noun* subgraphs in figure 8.3).

Consider a very simple English grammar G1 which only allows sentences of the form:

S → det noun verb det noun

Eg. The sentence *the cow jumped the moon* is valid in this grammar.

The TN of G1 looks as follows:

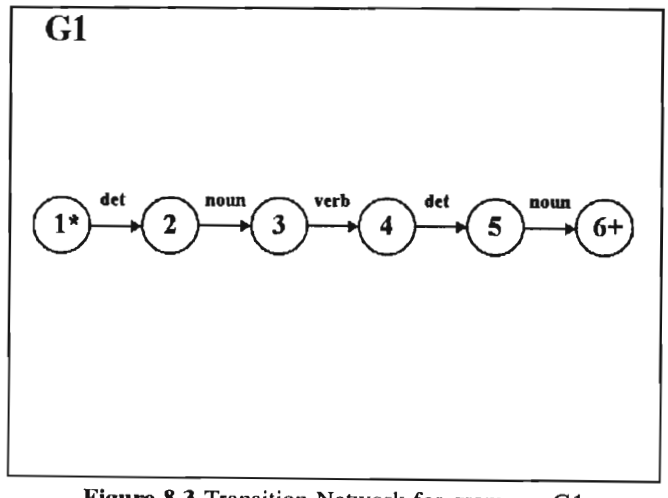

Figure 8.3 Transition Network for grammar G1

8.2.3 Recursive Transition Networks

Recursive transition networks (RTNs) allow recurring subgraphs in the network to be replaced by a single abbreviated node representing the subgraph to reduce redundancy found in transition networks.

For example, in grammar G1 the recurring "det → noun" subgraph can be replaced by a named subnetwork called NP. That is, G1 can be rewritten to form grammar G2:

S	→	NP VP
NP	→	det noun
VP	→	verb NP

RTNs also have the ability (through recursion) to handle non-deterministic networks. Non-determinism is introduced to the network if there exists more than one path with the same "action" from the same state. The decision of which path to choose in these situations is resolved by the searching strategy (eg. depth-first, breadth-first etc) of the recursion algorithm.

For example, consider production rules and diagrams of the subgraphs of grammar G3 below:

S	→	NP VP
NP	→	det noun
VP	→	verb PP | verb NP | verb
PP	→	prep NP

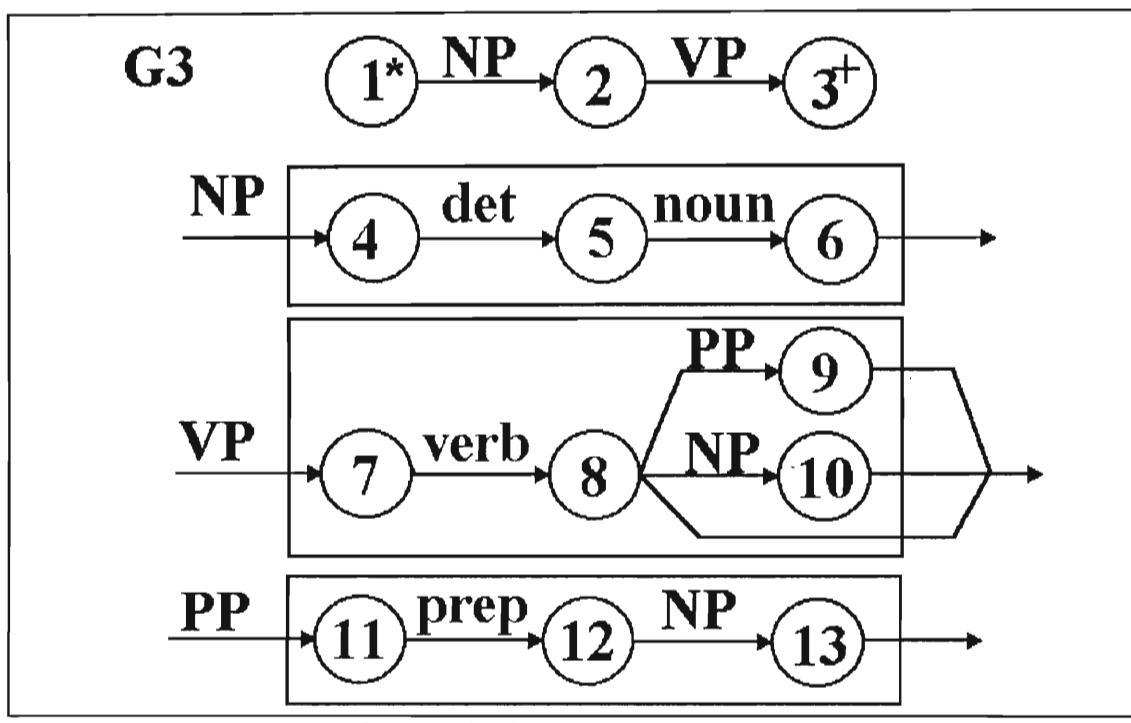

Figure 8.4 Subgraphs of Grammar G3

The VP subgraph has three arcs emanating from its first node, each with a *verb* "action" and thus requires a decision regarding which path to take. The decision strategy is not usually determined locally (it can be by looking ahead and down each path and choosing

the best path) but is globally fixed for the entire recognising process. The paths are often chosen as they appear in the production rules. The paths are executed in turn according to the decision strategy and processed until either a successful parse occurs or it cannot parse any further. In the event of not being able to parse any further, the parser must return to the point of the last decision and select the next path option. This procedure is continued recursively until a successful parse is found or every path has been explored without success. For example, using G3 to parse the sentence *the dog barked* with a top-down depth-first strategy will result in the following order of the production rules being applied:

1.	S	Start with start symbol
2.	NP VP	Expand S
3.	det noun VP	Expand NP
4.	det noun verb PP	Expand VP in 3 with 1st prod. rule option.
5.	det noun verb prep NP	Expand PP in 4. Mismatch of prep. Redo 3.
6.	det noun verb NP	Expand VP in 3 with 2nd prod. rule option.
7.	det noun verb det noun	Expand NP in 6. Mismatch of det. Redo 3.
8.	det noun verb	Expand VP in 3 with 3rd prod. rule option. Done? Yes. Stop.

8.2.4 Augmented Transition Networks

Augmented transition networks (ATNs) are RTNs with procedures for checking the consistency of the sentence with respect to tense, number, person etc. in related phrases in the sentence. For example, in order to verify that the number (singular or plural) of the subject (NP) of the sentence matches the number of the verb associated with it, the number of the NP of the subject must first be determined (in a procedure attached to the NP). The result is then passed to the VP where it is matched in another procedure to the number of the verb of that VP.

Person	Singular	Plural
First	I	We
Second	You (sg)	You (pl)
Third	He, she or it	They

Table 8.1 Person and Number

For example:
- the phrase *a boys* has a number conflict within the NP. It should either read *the boys* or *a boy*.
- In the sentence *Now he kick the ball*, the subject NP (*he*) is 3rd person singular. The verb *kick* must therefore have a matching third person

singular ending. Moreover, the tense of the VP must be present due to the present tense set by temporal adverbial phrase *Now*. Extracting this information and passing it between the relevant nodes in the ATN results in a mismatch - *kick* must either be *kicks* or the person altered to some other than 3rd.

A detailed description of the processing steps of an ATN can be found in [Ric91]. It should also be noted that ATNs are necessarily top-down parsers.

8.2.5 Well-Formed Substring Tables and Charts

One of the major problems with parsing is how to avoid redundant reparsing of phrases in the sentence. Reparsing is a direct result of 'blind' recursive implementations of non-deterministic grammars. Such parsers try all possible parses of a sentence. When one fails, they try another until they either succeed, complete all possible parses or continue searching forever because of a cycle in the production rules (ie. A → B and B → A). In order to eliminate reparsing sections of the sentence, it seems appropriate to store partial parsing solutions. Two mechanism for storing intermediate parsing results are *well-formed substring tables* (WFSTs) and *charts* (which are an extension of WFSTs).

Well-Formed Substring Tables (WFSTs)

A WFST is a data structure which keeps a record of parsing solutions it has already found so that it can avoid looking for them again [Gaz89]. A chart is an extension of a WFST which stores both the phrase structure information of parsed strings and the hypotheses that they parsed successfully.

An expensive way of keeping a record of all parses would be to store a set of partial parse trees. A WFST achieves the desired effect without storing the partial parse trees.

Method:

1. Number the string 0 to *n* (*n* is the number of words in the string) where 0 is placed before the start of the string and *n* is placed after the last word in the string.
2. The gaps between the words in the string are numbered 1 to *n*-1.
3. A WFST stores the starting and final position of a partially parsed string and the part of speech categorising (speech category) it.

A WFST is described as a set of 3-tuples (*Start,Finish,Label*) where

Start is the i^{th} node in the sentence
Finish is the j^{th} node in the sentence, ($0 \leq i < j \leq n$)
Label is some category (e.g. S or VP)

For example:

^{0} the ^{1} cat ^{2} sat ^{3} on ^{4} the ^{5} mat ^{6}

(0,1,det) (1,2,noun) (2,3,sat) (3,4,prep) (4,5,det) (5,6,noun)
(0,2,NP) (4,6,NP)
(3,6,PP)
(2,6,VP)
(0,6,S)

The above 3-tuples are the main ones used to build the parse structure of the example sentence but others do exist and are dealt with in the parsing process (eg. (2,3,VP) and (0,3,S)).

WFSTs inform the parser of correct parses of parts of the sentence. They fail however to inform what phrases and terminals make up those successful parses and as a result, force the parser to reparse those sections of the string. Charts extend the structure of WFSTs to include this information.

Charts

A chart is a WFST which incorporates information about what has been parsed and what is to be parsed. Where a WFST stored the speech category of a substring, a chart stores *hypotheses* about the rules the parser has tried to parse. A "." delimiter called dot in a hypothesis indicates to what extent the parser has completed parsing that rule. For example, S → NP.VP means that the parser has successfully parsed a NP in an attempt to parse the production rule: *a sentence is made up of a NP and a VP* but has not parsed the VP. A chart thus has two additional fields (ie. is a 5-tuple) namely *Found* and *ToFind*:

- *Found* is the list of parts of speech already parsed by the parser (ie. the NP in S → NP.VP
- *ToFind* is the list of parts of speech still to be parsed (ie. the VP in S → NP.VP)

For example: Using the previous WFST example, the initial entry into the chart using a top-down strategy is (0,6,S, ,NP VP) which transcribes as: trying to parse the sentence S, have not started thus have found nothing but hope to find a NP followed by a VP.

The details for implementing a WFST or chart parser (using any combination of the parsing and search strategies) are contained in [Gaz89a] (LISP implementations of these structures are given) and [Gaz89b] (Prolog implementations are given). Charts have great potential in speech recognition systems because of their flexibility and capacity to embody all the information about partial parses of the sentence. Speech understanding systems, discussed in the next chapter, tend to produce better results using the 'island-generation' technique which pieces together 'islands' of well-recognised parts of the speech signal. The chart accommodates this kind of approach. They are also efficient in that they eliminate redundancy from the parsing process. They can act both as a predictor and verifier of the structure of a sentence. The predictor can use the information from the *ToFind* fields along with a suitable parsing and search strategy to predict the structure of the rest of the sentence.

8.3 Semantics

8.3.1 Difference between Semantics and Pragmatics

The *meaning* of sentences can be divided into two parts: *semantics* and *pragmatics*. Semantics has been studied since the days of Plato and Aristotle whereas pragmatics, originally part of semantics, has only recently been made into a separate field. Semantics deals with the 'on-the-surface' truth value of the sentence. Pragmatics, on the other hand, deals with contextual issues and the speaker's intentions.

The following example illustrates the difference between these types of meaning.

I love ice-cream

Said listing "a few of your favourite things" to a friend, the meaning is clearly a truth statement about your world. This is an example of semantics.

On the other hand, if you loudly exclaimed the statement "I love ice-cream" as your friend walked towards you licking an ice-cream, the meaning is much more than the statement of

truth about your fondness of ice-cream. Your intention (and therefore your meaning) is to be offered some of the ice-cream. This type of meaning is called pragmatics.

Some introductory points and definitions are made about semantics [Hur88]:

- It is important to distinguish between the literal meaning of words and sentences and the meaning intended by the addresser.
- The meaning of words and sentences are best understood by the people who speak that language.
- A *proposition* is a syntactically well-formed sentence which makes a factual assertion and thus can be TRUE or FALSE
  eg. *the house is yellow* or *all cats are cleverer than dogs.*
- To be understood, speakers continually use references to things in the world to indicate what is being talked about eg. *My dog chased your cat up the old oak tree in Dairy Lane.* All the underlined parts of the sentence are called *referents.*
- The *sense* of a sentence is what is meant by it. It is often very difficult to explain and is sometimes expressed in terms of opposites, synonyms, descriptions or examples.
- A *predicate* is a structure/function for describing the state of a referent or the relationship between referents eg. in *red*(*house*), *red*() is the predicate and the *house* the referent argument while in *chased*(*my dog,your cat,up the tree*), *chased*() is a predicate with three referent arguments.
- Of great importance in semantics is the notion of the 'world' or universe of discourse in which sentences are made. For example, as mentioned at the start of the chapter, SHRDLU's 'world' is a table full of blocks called BLOCKWORLD. If SHRDLU were asked about anything outside its BLOCKWORLD (eg. how many planets are there in the solar system), it would not be able to answer or at best would respond, "I don't know what you mean by 'planets' and 'solar system'". If a NLP or speech recognition system is going to be able to communicate with speakers in the real world, certain boundaries must be set to determine the universe of discourse between it and the people communicating with it.
- There are three levels of context in a conversation: the universe of discourse, the context of a particular conversation and the immediate situation of an utterance in the conversation.
- A *stereotype* is an abstract specification of an object. A *prototype* is a particular object which best matches the stereotype. For example, a stereotype of a bird might be a smallish, non-descript, feathered, beaked, eyed and winged creature that can fly. A prototype may be a dove-like or robin-like bird but it would rarely

be an ostrich- or penguin-like bird (unless you were an eskimo). These form part of the set of all birds but are 'exceptions' rather than 'the rules'. Stereotypes and prototypes are useful starting points from which to express or build meanings.

A NLP system must be able to understand what is being stated, what has been stated previously, what is implied by the addresser and what can possibly enter the conversation through the context. It must therefore have the following ingredients [Ric91] [Gaz89a]:

- a parser which provides a knowledge of the structure of the sentence
- a lexicon of words and their meanings
- a way of representing knowledge as it is used and exists in the system's 'world'. This should include stereotypical knowledge about procedures which are common to the users of the system, associative links between related knowledge and human-like, structured organisation of information. The system should also contain information about itself (*metaknowledge*), especially to restrict implausible facts that may be generated by the inference engine (the mechanism which deduces new facts from the database of facts.
- a mechanism for inferring new facts from the existing knowledge and data inputs to the system and for managing formal logic
- an interface between the system and the users which should be as natural as possible answering questions asked by the users and explaining or giving reasons for the deductions it makes

8.3.2 The Lexicon

The basis of any natural language is the set of meanings of 'things' in the 'world'. Everyone's 'world' is different because not everyone knows the same 'things' and people seem to see 'things' differently. An important question is how are these 'things' to be stored in a NLP system? This question is answered in the artificial intelligence textbooks under the heading of *knowledge representation.*

Consider the simple knowledge base of an infant learning his mother tongue. He begins by labelling objects in his small 'world'. First "mamma" is associated with mother, then "dadda" with father and so on and his 'world' gradually develops. In a simple system such as this, one manages to describe the 'world' by labelling every entity in that 'world'. Simple (mamma=image of mother) relations would suffice. But mothers usually mean more to their babies than just that. Mother carries a bottle which feeds and satisfies baby and mother comforts baby when baby is disturbed by something. So the simple entities in the child's database also contain relations or associations with other entities or ideas. Later still,

the child will be able to distinguish between birds and bees for example, even though both birds and bees have wings and can fly.

Representing knowledge in the lexicon has important implications for the sense of the words in sentences and the meaning of the sentence is represented. The knowledge in the lexicon defines, describes and labels the objects in the 'world' whereas the 'semantic picture' (meaning of the sentence) dynamically evolves with the construction of the ideas of the sentence, representing the way the entities interact with each other in the sentence and in the course of the dialogue. The dynamic knowledge changes considerably during the course of conversation whereas the knowledge in the lexicon usually remains unaltered or alters very gradually.

Knowledge Representation

There are three basic data structures used for storing the meanings of words in the lexicon[1]:

- attribute-value pairs
- records
- objects

An attribute-value pair (AVP) is a 2-tuple which describes an entity and can be represented as: (Attribute,Value). Eg. A *block* may have the AVP (colour,green) and a *man* the AVP (has_a,beard).

A record is an extension of the attribute-value pair. It consists of an entity name and a group or set of attribute-value pairs.

Eg. Record *block*
(colour,green)
(madeof,wood)
(size,small)
End Record

Objects on the other hand are records which can be linked to other objects in a hierarchical manner so that redundant information about entities in the 'world' can be eliminated. Objects are linked to other objects by relation arcs, two special ones being the *is_a*

[1] In fact, these form the basis of most artificial intelligent systems.

(expressing class inclusion) and *instance_of* (expressing class membership) relations. The resulting system of arcs and objects between objects is called a *semantic network* (also called *a set of frames* or a *slot-and-filler* system [Ric91]). Eg. "Tweety" is an instance of a canary which is a bird which is a flying object. Thus objects encode class knowledge. Their ability to reduce redundant information is shown in the following example. It is a waste of effort and space to store the same general information about birds and canaries when a canary is a bird and naturally inherits all the relevant information that makes it a bird (eg. can fly, got 2 legs, whistle etc). Objects use inheritance of information from encumbent classes to eliminate this redundant information.

Semantic networks are depicted as follows:

- network nodes store the object classes and members
- network arcs are labelled with an appropriate object attribute relation

There are several rules for objects:

- No *is_a* arc cycles are allowed. Eg. object A is an object B which is an object A is not allowed.

- If a specific class or member does not have the attributes of the class of which it is a member, that attribute must be redefined by that object. Eg. a penguin is a bird but it cannot inherit the (can_fly, yes) attribute of bird. Therefore the object penguin must redefine this attribute as (can_fly, no).

- Whenever there is a pair of entities, neither being a descendant of the other but with a descendant in common, there is no attribute that both entities specify a value for.

- An entity can only have one value for an attribute-value pair defined locally.

It is worth noting that these objects are similar to the object data structures that have recently become popular. The knowledge that can be represented by attribute-value pairs or records is clearly not as sophisticated as that represented by a network of objects. These networks give the system an opportunity to express the associative links that seem to exist in the human mind. The mind's capacity for associations is far more sophisticated. For example, humans tasting wine will often associate the smell of the wine with the smell of some food or (for the more daring tasters) with an experience eg. walking through the

woods just before sunset. Networks give the system the potential for making associations between entities, be they ideas, feelings or simply other entities.

Several other ways for representing knowledge in the system exist including: frames consisting of predefined slots which must be filled to complete the sense of the sentence and predicates which have already been mentioned. The details of these structures are contained in most artificial intelligence textbooks (eg. [Ric91] [Bra90]). The popularity of predicates has increased because of the complementary structure of the Prolog programming language.

Another way of representing knowledge is defined in the section on pragmatics. This caters for stereotypical information which follows a chronological order and is called a *script.*

Morphemes

At first glance, it seems that by storing all the *words* in the system's 'world', together with their meanings (including relation arcs), the lexicon will be complete. It turns out that semanticians define a unit slightly smaller than the word as the *unit of knowledge* in the lexicon, namely the *morpheme.*

Morpheme literally (from the Greek) means the 'minimal meaningful unit' [Sco1889]. Words in a language usually consist of a root form with different prefixes, suffixes and other slight transformations to alter the base meaning. To illustrate this point, several examples are listed below:

Root Form	Changed Form	Type of Change	Difference in Meaning
monkey	monkeys	suffix	plural
man	men	transformation	plural
happy	unhappy	prefix	negated (opposite)
eat	has been eaten	transformation + suffix	passive past
count	recount	prefix	compound ("re~" ▪ again)
black	blacken	suffix	adj→verb ("~en" ▪ to make ~)
black	blackness	suffix	adj→noun ("~ness" ▪ a ~ state)

Table 8.2 Morphological Transformations

In English, the most important and frequently used of all the transformations are those to determine the plurality of nouns and the person, tense, voice and mood of verbs. English lacks the rich, morphological structure found in Latin or Greek (and many of the romantic languages derived from Latin). It is informative to examine very briefly the structure of nouns and verbs of Latin.

All Latin nouns belong to one of five *declensions*. Each declension has a set of rules defining various suffix endings to alter the meaning of the noun. Take for example (Winston Churchill's favourite [Chu47]) the noun *mensa* (=table) of the first declension:

Latin Form (singular)	Approximate English meaning	Latin Form (plural)	Approximate English Meaning
mensa	the table (subject of the sentence)	mensae	the tables (subject)
mensa	O table (addressing the table)	mensae	O tables
mensam	the table (object of the sentence)	mensas	the tables (object)
mensae	of the table, or table's	mensarum	of the tables or tables'
mensae	to or for the table	mensis	to or for the tables
mensa	by, with or from the table	mensis	by, with or from the tables

Table 8.3 Example of the Form of Latin's First Declension

The root *mens-* is the morpheme while the declension endings serve to guide the addressee both syntactically and semantically.

Latin (or French) verbs have *conjugations* or sets of rules for verb endings which determine the verbs':

- the person (1^{st}, 2^{nd}, 3^{rd})
- number (singular or plural)
- tense (present, future, past etc.)
- voice (active or passive) and
- mood , mood (indicative, subjunctive etc.), and voice variations of the verb. English does have several suffix endings but on the whole is *inflectional* ie. has augment verbs to determine the variations in the form of the verb.

Latin Form	English Equivalent	Description
amo	I love or am loving	1st person singular, present tense, indicative mood, active voice
amat	He, she or it loves	3rd person singular else same as previous
amavi	I loved or have loved	1st person singular past (perfect) tense else same as previous
amem	I may love	present tense, subjunctive mood else same as previous
amor	I am loved	indicative mood, passive voice else same as previous
amamur	We are loved	1st person plural else same as previous

Table 8.4 Examples illustrating Latin's morphologically rich Verb Forms

From the table above, it is clear that Latin-like languages would conserve space in the lexicon. All they need store are the root of the word and the class (conjugation or declension) to which it belongs.

In English, nouns have suffix ending transformations to handle plurality (exceptions like ox↔oxen must be taken care of). Verbs in English are inflectional because auxiliary words are used to alter the meaning to describe different moods, voices and some of the tenses (as is seen from the other table of English equivalents to the Latin forms). With inflectional languages, the verb can be split up across the sentence - a phenomenon which can be hazardous for the parser eg. *Have you been that badly punished*? (verb underlined). Despite English's inflectional nature, rules can still be defined for the various forms of the verbs in English and stored as a code (except for the exceptions to the rules which must be stored in full).

8.3.3 Semantic Representations

Several mechanism for embodying and constructing the meaning of sentences have evolved, these being:

- Semantic grammars
- Case grammars
- Conceptual dependencies
- Compositional semantics

Semantic Grammars

Semantic grammars are very structural in appearance and the grammatical production rules are expressed in terms of both syntax and semantic markers. That is, semantics plays a role in determining how sentences are parsed. Semantic Grammars tend to require many rules because every semantic option for a word must be expressed in a rule. They are useful for reducing the structural ambiguity discussed earlier. For more about these grammars see [Ric91].

Case Grammars

Case grammars determine the semantic roles of the parts of speech related to the central verb of the sentence. Eg. (open (agent Jim)(object door)(instrument hand)) for *John opened the door with his hand* or *the door was opened by John with his hand.* Several roles for the parts of speech have been determined including [Ric91] (examples are the underlined words):

- (A)gent - initiator of the action (the subject of active verbs) eg. *he runs*
- (I)nstrument - cause of the event eg. *he shot him with a .22*
- (D)ative - Entity affected by the action eg. *Jim died*
- (F)actitive - object or being resulting from the event eg. *He made Eve*
- (L)ocative - Place of the event eg. *He hid in the house*
- (S)ource - Place from which something moves eg. *From Cape Town to Cairo ...*
- (G)oal - Place to which something moves eg. *he went to town*
- (B)eneficiary - on whose behalf the event occurred eg. *We did it for him*
- (T)ime - the time at which an event took place eg. *at sunrise, the cock crowed*

The above abbreviations in brackets are used to describe the entities making up the case grammars. The case grammars are expressed as generally as possible with brackets denoting parts of speech which are omitted eg. only the object is necessary for the open case grammar since sentences like *the door opened* are grammatical. This example is thus expressed as *open* [_ _ O (I) (A)] which transcribes as: *open* has an object (obligatory) and also possibly has an instrument and an agent. The values for the (I) and (A) although often omitted from the sentence, are sometimes implied and can and should be determined by a 'smart' system to complete the meaning.

Eg. *The wind was blowing hard ... The door swung open.*
(A) is *the wind* and (I) is *the force of the wind*

Notice that the roles describe the relationships between the parts of speech in the sentences. They can be effectively implemented using frame slots which must be filled with the relevant parts of speech to construct the meaning of the sentence.

Conceptual Parsing

Conceptual parsing is a technique which uses a specific knowledge representation called the *conceptual dependency* representation [Sch82]. This knowledge representation is language independent, expressing knowledge about the events found in sentences of any language. It is made up of conceptual primitives which are atomic verbs describing the basic actions expressed in any language. The set of these primitive actions are defined below as presented in Schank and Abelson [Sch77]:

- ATRANS Transfer of an abstract (eg. give)
- PTRANS Transfer of the physical location of an object (eg. go)
- PROPEL Application of physical force to an object (eg. push)
- MOVE Movement of a body part by its owner (eg. kick)
- GRASP Grasping of an object by an actor (eg. clutch)
- INGEST Ingestion of an object by an animal (eg. eat)
- EXPEL Expulsion of something from the body by an animal (eg. cry)
- MTRANS Transfer of mental information (eg. tell)
- MBUILD Building new information out of old (eg. decide)
- SPEAK Production of sounds (eg. say)
- ATTEND Focusing of a sense organ towards a stimulus (eg. listen)

For example, INGEST is the action of taking anything into the mouth including drinking, eating, breathing, smoking. The atomic verbs are linked to the rest of the words in the sentence by the following symbols:

- Arrows denote the dependency between entities
- Cases are denoted by labels above the arrows (eg. O = object, I = instrument see case grammars for others)
- Double arrows denote a mutual link between the actor and action
- Tense is denoted above the double arrows (p ≡ past tense)
- Primitive actions as defined above are written in capitals

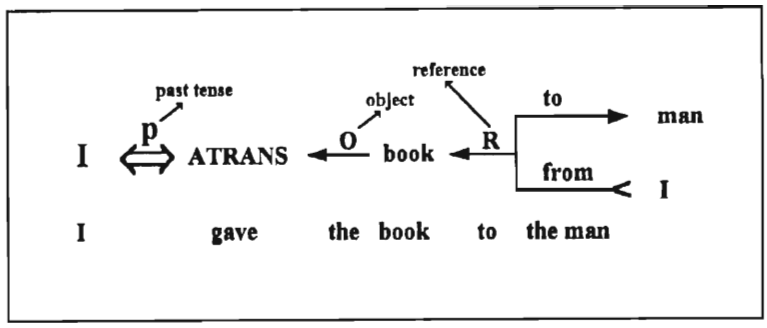

Figure 8.5 Conceptual Dependency Diagram adapted from [Ric91]

Compositional Semantics

> The *principle of compositionality*[1] proposes that the meaning of the whole is a function of the meanings of the parts.

In terms of NLP, the meaning of the sentence can be developed by determining the meaning of the morphemes and words, then combining them to build the meaning of the phrases and clauses until these combine at the top level to form the sentence. One can go even further to describe the meaning of the paragraph from the meaning of the sentences.

The principle of compositionality provides an agenda or framework for constructing a semantic picture. It also requires contemporaneous knowledge of the syntactic structure of the sentence (ie. phrases and clauses which are the parts) since the meaning is built up on increasing levels of these classes. The details of how to implement this type of semantic representation is contained in [Gaz89b].

8.3.4 Using Semantics to Resolve Ambiguity

Earlier in the chapter, it was shown that structurally ambiguous sentences were a result of being able to parse them in more than one way. Ideally one would like to resolve this problem as early as possible since other parts of the sentence or communication may rely on it. Semantics is one of the tools that can be used to resolve the ambiguity in a sentence.

For example consider the following sentences:

rolling balls is fun.
rolling balls are fun.

The former is the contracted form of the sentence *the act of rolling balls is fun.* The number of the verb (*is*) is singular thus compelling the subject NP of the sentence to be

[1] The principle of compositionality is attributed to Frege, a mathematician and philosopher (*circa* 1900).

singular. If *rolling balls* is read as a composite noun, its plurality causes a number conflict with the verb *is*. The system can thus use this semantic clue to read the sentence as (the act of) *rolling balls is fun* (ie. the grammatical identity of rolling is the *gerund*). The second example requires the opposite (ie. reading it as a compound noun). Clearly semantic information can be used to guide the parser to optimise its decisions in finding the best parse of the sentence. It therefore seems obvious to incorporate semantic checking with parsing to reduce the effort of the parser. There are however instances where the work done by the semantic analyser is redundant or unimportant and therefore has the reverse effect.

8.3.5 Inference

Humans continually infer things from conversations. Consider the following story:

> *A shot rang out. The lady slumped to the floor ...*

One would most likely *infer* or *deduce* that the shot which rang out, was a gun-shot and the bullet of which hit the lady causing her to slump to the floor.

Inference is important in NLP for the following reasons:

- to infer answers (or facts) about the 'world' which do not exist in the database in the desired form.
- to make predictions about possible future inputs (ie. to track the addresser's 'train of thought')
- to assimilate new information into the database of existing facts
- to solve problems about how the world does or might behave
- to detect that two statements are semantically equivalent or that a statement is anomalous.

In order to make inferences, the NLP system needs an *inference engine*. The inference engine uses rules and some logic system for inferring new facts or interpretations from the existing ones and perhaps new input to the system. Charniak and McDermott [Cha85] highlight three roles of the inference engine: to assert new fact in the database, to retract or discard facts from the database and to query the database in an attempt to produce new facts.

When new facts are asserted, this may lead to more facts being generated by certain rules that exist in the database. An important issue is when to check for possible new facts being generated from these facts. One obvious moment is when new facts are first generated. This

is sometimes referred to as an event-driven strategy which uses a well-known technique called forward-chaining. For example, if a fact *p* is asserted and the rule (if *p* then *q)* exists, then *p* triggers the assertion of *q* according to the forward-chaining technique. Another strategy is to wait until a query arises from the user or system and then work backwards from the desired query to the axiomatic facts. This strategy is called *backward-chaining*. For example, suppose the system had a single fact *p* in the database and two rules (if *p* then *q*) and (if *q* then *r*). The query "Is *r* true?" can be determined by inspecting the then parts of the if-rules and producing a set of subgoals which will result either in the axioms (basic facts) of the database being reached or not. Therefore, the goal "is r true?" becomes "is *q* true?" which becomes "is *p* true?" and because *p* is true (a fact), the other subgoals are also true and so is the result of the query.

There are advantages and disadvantages to both these strategies. Forward-chaining can lead to an explosion of new facts generating further facts themselves but they respond to new data which is important for example when new parts of speech are found in the sentence. Backward-chaining on the other hand responds well when handling problems that are best solved by a top-down, divide-and-conquer technique eg. an NLP system answering a question posed by the user. Cyclical deductions and pursuing 'red herrings' are potential problems in backward-chaining. Further details of the inference engine can be found in [Cha85] and [Win84].

8.3.6 Primitives and Canonical Forms

One of the greatest problems with knowledge is its granularity or base form. The idea is not to store the same information twice. Consider the following example using a predicate knowledge representation (which was mentioned before but not detailed):

on(ball,block)	ie. the ball is on the block
supports(block,ball)	ie. the block supports the ball

Only one of the above predicates need be stored if the following rules are included in the database.

on(X,Y) if supports(Y,X)
supports(X,Y) if on(Y,X)

Having two predicates *on* and *support* to describe (almost) the same idea is not only wasteful of space but creates potential hazards for maintaining the accuracy of the database.

If, for example, the ball is removed from the block, there are two places where the database must be changed instead of just one.

8.4 Pragmatics

As has already been mentioned, pragmatics is closely related to semantics and seems to include everything which does not quite fit under semantics. In this section, several of the most important issues in pragmatics will be discussed; the contextual information of NPs, given versus new information (from the context), prediction, focusing and planning and methods for handling stereotypical events and information.

8.4.1 Contextual Information in NPs

Noun phrases (NPs) on the whole, deal with objects (and abstracts) in the 'world' ie. they contain the 'things' that are interacting with each other in the sentence. In addition, they play another more subtle and yet equally important role in giving the addressee extra information about these 'things'. This information helps the addressee (better) understand which 'things' the addresser is communicating about since they are typically numerous.

NPs without any modifiers can play one of two roles in the sentence.

For example: *SHRDLU picks up blocks.*
The robot picks up blocks.

There are two NPs in each of the above sentences. In the first, *SHRDLU* refers to a specific 'thing'. Such NPs are called *definite*. Definite noun phrases (when they are not proper nouns) are usually supplied with the definite article (determiner), *the*. Definite NPs are used by the speaker when the addressee is already familiar with the object in the NP. That is, if the addressee did not know who or what SHRDLU was, the first sentence would be meaningless. Proper nouns are obviously definite (since a specific object is being referred to) while common nouns require special attention by the system. For example, in the second sentence, *the robot* is definite (ie. is known to the addressee) because if it has not previously been mentioned, seen or explained to the addressee, the statement carries no purposeful meaning.

The main point about definite NPs is that they imply that the addressee has already been introduced to or can identify the object from the context of the sentence in the rest of the conversation or from the surrounding environment.

The second noun phrases (NPs) ie. *blocks* in both of the example sentences, are examples of indefinite noun phrases. That is, they stand for the generic class of objects and not any in particular. The indefinite article *a* is often an indication of an indefinite (singular) NP.

Given vs. New Information

The primary function of declarative statements (or assertions) is to present the addressee with new information. The new information is based on ideas and facts already known to the addressee[1]. That is, the addressee's knowledge is added to incrementally by new facts, predominantly in the form of declarative sentences.

For example: The block I put on the table is green.

The new information is the colour of the block. The block must be definite in order that the addressee can incorporate the new information into his database. Therefore, in the above example, the referent clause *I put on the table* is included to resolve any possible ambiguity.

For example: I have placed an object on the table.
The yellow block supports this red plastic ball.

The *this* in the second sentence refers to the *new block* of the first sentence. The second sentence tells the addressee about the locality and several attributes of the *new object.*

8.4.2 Given Information Restricting Referents

Given information can be used to restrict the possible number of referents for noun phrases. In the previous example if *the yellow block* does not exist, there is a contradiction. Likewise if there is already something else on *the yellow block* with no space for another object, there is a contradiction. Restricting referents of this sort can be used to resolve the identity of pronouns.

Example

The commander placed a block on the table.
It was green.

---

[1] Poses the interesting psychological/philosophical question about a person's first thought - the inheritance vs. environment issue.

The it refers to a block (a new block). Knowing the colour attributes of the commander and the table, can help to eliminate possible references for it. This is not even necessary, since owing to its recent entry onto the scene, the new block is the centre of focus of the communication at that point in time.

Example:

> There are two objects, a ball and a box.
> The one contains the other.

The lexicon should contain the fact that a box can contain something. This predetermined knowledge is restrictive in the above example since it reduces (and in most senses, eliminates) the possibility that the ball contains the box.

8.4.3 Understanding by Prediction

Till now understanding has been a passive affair, where the addressee compares what is communicated with what already exists in the database. The human addressee however is generally not passive. He is continually predicting the outcome and intentions of conversations, sentences, phrases and even sounds in words. Prediction is based on the understanding of the past and present. Validation is the process of checking the consistency of the predictions with the actual interpretations of the messages.

How does one predict events? Prediction is not prophecy. Prediction involves identifying the initial parts of ideas, types of problems and themes that seem to recur naturally. The recurrence of such events is noted through experience. This does not mean that prediction is simply a matter of remembering and matching a sequence of events. It can be this, or it can mean associating a process or strategy used perhaps in different circumstances to solve a problem. It may also involve a statistical element similar to that described in chapters 4 and 6.

Prediction and validation are costly processes since every prediction must be stored and later validated and removed from memory. This gives rise to potential storage space problems and issues like when to delete predictions from memory. The focus of the communication is vital for maintaining the predictions. Another tool for controlling prediction is the use of stereotypical information.

Stereotypical Information

One important way of controlling the explosion of predictions is to represent stereotypical information in predefined data structures called scripts (or schemata). Scripts store knowledge of recurrent chronological events. The order of the entries in scripts is therefore vital and enables the addressee to predict and follow the flow of chronologically predictable messages. Another important feature of scripts is that they embody the details in a compact way and will often embody more (taken-for-granted) information than what would normally be communicated by the addresser.

A script contains a name and several fields. The ordering of the fields is vital since they represent the sequence of events which determine the script. The most popular example of a script depicts the order of events carried out when dining at a restaurant [Sch82].

Script: Dining at restaurant

- Arrive at restaurant
- Request table
- Sit at table
- Receive menus
- Order from menus
- Receive drink and food
- Drink and eat
- Ask for bill
- Receive bill
- Pay bill
- Leave restaurant

End Script

The importance of the sequential nature of scripts is illustrated by the following example. If a conversation began: *we paid the bill for the dinner at Mike's Kitchen* ..., one would not expect the speaker to continue with the details of the restaurant experience from when they arrived. In this way, scripts are able to focus and order the attention of the addressee.

Implementing Scripts

New ideas, shifts in the focus of the argument and tangents to a story create problems for scripts. There are two implementation issues to consider: one, when to open the script and two, when to close the script. They both lead to the potential problem of a multitude of

open scripts which results in too much detail to follow and confusion between similar events in different scripts.

For example, the restaurant script might be initiated when the addresser mentions something about dining at a restaurant. The addresser might however have mentioned this as a peripheral point in the argument. In such a case, it would be optimal not to open the script and at worst, to close it immediately one is certain that the emphasis of the conversation is elsewhere. On the other hand, it is important to be able to determine when the focus is no longer on the events of the script in which case it must be closed.

Focusing

The focus of the conversation is important not only for prediction of message events but also to resolve potential problems with pronouns and ellipsis of obvious words or ideas. Several tools or mechanisms can aid in focusing the system's attention, for example:

- The way knowledge is represented (eg. scripts and even conceptual dependencies)
- Goal planning - the idea being that by understanding the goals of the addresser, one can follow his train of thought (see [Wil83])

8.5 Conclusions

The importance of this chapter cannot be overemphasised. The incorporation of natural language processing in speech recognition systems is vital for building 'intelligent' speech systems with powers beyond merely pattern matching speech signals.

From this chapter, two important observations can be made.

- Firstly, the contributions of the main NLP components (ie. syntax, semantics and pragmatics) are each important for constructing the sense of the communication.

- Secondly, these components are usually contained in separate units in systems for ease of implementation but will probably need to be integrated into one interdependent system to achieve real success in the field. The problem with integration is that systems becomes very complex and difficult to update or modify as a result.

In the next chapter, a system for integrating NLP with a low-level speech pattern recognition system is investigated.

Chapter 9 - Expert Systems and Speech Understanding

9.1 Chapter Outline

This chapter examines speech understanding systems (SUSs), featuring in particular blackboard expert systems as a means for modelling and ultimately implementing an SUS. Designing and implementing an SUS is a major undertaking which can take a team of researchers several years to complete. Very few have been built and even fewer have been successful. How are SUSs different from the other speech recognition systems discussed in the earlier chapters of the thesis and where do they fit into the field of speech recognition?

SUSs typically utilise a front-end phoneme[1] pattern recognition system to recognise the basic speech sounds in the signal. The accuracy of phoneme pattern recognition techniques was shown in chapter 4 to be vulnerable to misclassification because of the similarity of certain sounds (eg. the vowels). Misclassification at an early stage in word or sentence recognition can ruin the chances of the overall recognition result. To reduce the chance of a misclassification, several of the best possible speech sounds are usually stored for each frame of the speech signal [Red76]. Continuous speech recognisers (CSRs) would then use a set of rules to merge equivalent neighbouring sounds hierarchically to form syllables, words, phrases and even sentences. SUSs differ from CSRs in the following ways:

- they embody large amounts of information about natural language and speech processing
- they make decisions about the most likely sequence of words using many different sources of knowledge
- they exhibit structures modelled on those thought to control the human cognitive processes
- they build an abstract representation of the meaning (semantic and pragmatic) in the sentence and conversation

SUSs therefore differ from CSRs in that they integrate and embody the concepts and mechanisms discussed in chapter 8 on natural language processing (NLP). Ideally they should exhibit their 'understanding' by functioning 'normally' (if in a restricted 'world') as a person might.

---

[1] Or an equivalent subword sound class eg. demi-syllable

The work covered in this chapter draws on natural language processing (NLP), artificial intelligence (AI) and expert systems. Some of the concepts have already been presented in chapter 8 where NLP was investigated from a theoretical perspective. The major part and emphasis of this chapter is centred on expert systems, especially blackboard expert systems which are seen to be well-suited for implementing SUSs. Finally, a case study of the HEARSAY-II system is undertaken as a model SUS.

9.1.1 The Artificial Intelligence Debate

There is a great difference between a speech recognition machine that can recognise verbal commands and one that can hold a meaningful conversation with a person. The latter might be described as being more 'intelligent' than the former. But what is intelligence?

> "Intelligence is far easier to recognise than to define, and so AI researchers often concentrate on trying to produce what they consider to be 'intelligent' behaviour in machines, rather than defining intelligence or knowledge." [Har89]

Hart highlights the problem of defining intelligence and the way AI researchers have side-stepped the issue. The definition of intelligence has long been debated by philosophers and psychologists and has yet to be precisely determined. AI research (influenced by computer people who generally prefer to implement systems rather than discuss their philosophical nature and worth) has continued without over-careful attention to the philosophers' concerns and as a result, philosophers have become very critical of what is being done in the name of 'intelligence' [Bor87]. Instead of focusing on the semantics of this debate, this chapter concentrates on what has been done to date in AI and the related field of expert systems.

Cohen and Feigenbaum define AI in the following way:

> "Artificial intelligence is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behaviour - understanding language, learning, reasoning, solving problems, and so on." [Coh82]

This definition is more forgiving than the well-known Turing test of AI. That is, if there are two rooms, one with a human inside it and the other with the 'intelligent' computer

system, the computer system achieves 'intelligence' when a human questioner cannot distinguish the computer's answers from the human's [Tur50].

AI systems are thus measured according to human behaviour, which has several important implications when considered in the light of the following arguments by Vygotsky:

> "The most significant moment in the course of intellectual development, which gives birth to the purely human forms of practical and abstract intelligence, occurs when speech and practical activity, two previously completely independent lines of development, converge. ... Now speech guides, determines, and dominates the course of action; the planning function of speech comes into being in addition to the already existing function of language to reflect the external world." [Vyg78]

In the above, Vygotsky relates his conclusions to experiments comparing the problem-solving strategies and abilities of children with those of primates. The children[1] achieved far superior results solving the simple tasks because of their ability to 'talk through' their strategies. Primates on the other hand showed no inclination to abstract the problem at hand and tended to utilise whatever was in their field of focus to solve the problem using a 'brute force' (ie. 'try every possibility') approach. From his conclusions, Vygotsky is suggesting that speech is the major ingredient in elevating human behaviour from the ranks of animal behaviour and is what ultimately makes humans intelligent. These ideas add tremendous weight to the importance of the study of NLP and speech recognition.

In terms of speech recognition, the isolated word, connected word and even continuous speech recognisers do not exhibit 'intelligent' qualities. It is only when mechanisms are introduced into these systems to model the cognitive processes found in humans that they can begin to be considered 'intelligent' systems. Among the most important aspects of the cognitive model are [Rey83]:

- the ability to store large amounts of information
- the structures which organise and retain this knowledge
- the ability to deduce things from and about this knowledge
- the ability to learn new facts about the system's 'world'

[1] Only those over a certain age, infants tended to act as the primates did.

9.2 Expert Systems

One of the major characteristics of speech is that it embodies a large amount of redundant information in order to improve the recognition of the difficult-to-distinguish speech sounds. This redundant information is comprised of syntactic, semantic, prosodic and pragmatic knowledge as well as information about the speaker (eg. dialect, voice qualities etc.). Speech recognition systems which attempt to encapsulate and utilise this diverse information are classed as *knowledge-based systems*. Systems which attempt to behave as human experts in a restricted, knowledge-intense problem-domain, are known as *expert systems*[1]. No speech recognition system can theoretically be classed as an expert system because speech is not restricted to certain human experts but is naturally assimilated by all. However, several of the mechanisms and methodologies applied by expert systems are equally applicable to knowledge-based systems and therefore discussed here.

The ubiquity of 'expert systems' in today's society, coupled with some impressive results from these systems has attracted much attention (and some hype). Like their human expert models, they embody a large amount of knowledge with problem-solving strategies to exploit this knowledge in order to find solutions to the domain problems. Examples of expert systems include those handling problems such as: *diagnosis and remedy*, *analysis and prediction*, *game-playing*, *planning*, *design* and *monitoring*. An interesting study of the more important and successful expert systems is undertaken by Johnson and Keravnou [Joh88]. Among these are MYCIN, a medical diagnosis and remedy system and PROSPECTOR, a geological analyser and predictor of locations for mineral deposits.

9.2.1 People involved in building Expert Systems

There are three identifiable roles in building an expert system. The first is that of the *domain expert*. He is the best available person (preferably several people) with training and experience in the problem-domain (subject area) of the expert system. His knowledge typically has a theoretical or textbook foundation accompanied with practical 'know-how'. The 'know-how' is often more useful than textbook knowledge because its form is usually more flexible, less formal and easier to represent in the system. As a result the expert system's representation of the knowledge often assumes a similar character to that of the human expert's. It is often quite difficult to extract the 'know-how' knowledge from the domain expert. For example, consider how difficult it is for humans (who can be thought of as experts in speech recognition) to describe the prosodic cues they use to determine the

[1] Quinlan points out that: "without a doubt, some current activity that goes under the banner of expert systems does not belong there, being just an attempt to cash in on a fashionable buzzword" [Qui87].

tone or mood of the speaker. Even when a person has understood the jargon, he will still struggle to explain what he does so naturally.

The second and perhaps most important role is that of the *knowledge engineer*. The knowledge engineer's task is to extract, unravel and assimilate the knowledge for the system from every possible source (textbooks and the domain experts) into a symbolic form, by which it may be represented in the expert system. This stage in the development of the expert system is referred to as the 'bottle-neck' (eg. in [Hay83] and [For84]) because it involves all the major design issues and the time consuming role of extracting the knowledge. The domain knowledge is extracted from two main sources: published textbooks full of theoretical definitions, facts and details and experiential knowledge acquired by 'picking the brains' of the best domain experts in that field. Experiential knowledge such as this, is usually full of what is termed heuristics, rules-of-thumb or educated guess-work. This type of knowledge is the most difficult to extract because the domain experts often struggle to put into words what is so natural to them. The task of acquiring the domain knowledge is usually delegated to a separate member of the team of designers who is known as the knowledge engineer. A detailed description of the knowledge acquisition process (and a case study) is presented in Hayes-Roth et al [Hay83].

In a speech recognition project, defining knowledge about natural language and speech processes may seem trivial because of the argument that speaking is 'natural to humans' but as is shown by scholars' lack of affinity with classroom grammar, the abstraction of this information is very difficult. The knowledge engineer will therefore have to rely on knowledge from linguists, psychologists and cognitive scientists. It should also be pointed out that because of the diversity of the types of knowledge in speech recognition and natural language processing, the problem of building a speech understanding system is usually divided into modules each with a separate knowledge engineering component.

In addition, it is interesting (if obvious) to note that the knowledge engineer becomes quite an expert in the problem-domain by the end of the project. This is useful for the designing stage of the system when the domain expert is often not always on hand to answer for the design decisions made.

The third role is that of the programmer. It is his task to encode the knowledge assimilated by the knowledge engineer. Another function of his role is to create the user interface which provides feedback of the system's decisions to the user. If an expert system shell (a package or utility program for building expert systems) is used, the role of the programmer may be eliminated.

9.2.2 Components of an Expert System

An expert system is comprised of four basic components [Eng88]:

- a knowledge base
- an inference engine
- a working storage
- a user interface

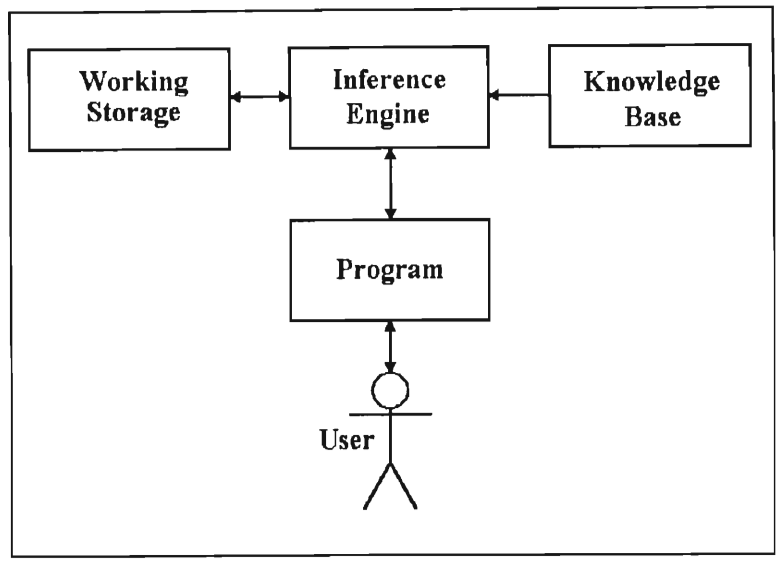

Figure 9.1 The Components of a Typical Expert System

The first two components have been discussed in some detail in the previous chapter with respect to NLP. The knowledge base is the database of facts required by the system to solve the domain problems. The most important issue concerning the knowledge base is how to represent the knowledge. The inference engine, on the other hand, has been called the 'gate-keeper' (by [Cha85]) in that it services the queries from the system (perhaps from the user) by searching through the database for answers and then submits the answers and possible explanations back to the system. The forward- and backward chaining problem-solving strategies relate to the inference engine were also mentioned in Chapter 8. The working storage contains the internal representation of the current state of the solution as determined by the system.

The user interface is that part of the expert system which communicates with the user. It performs the vital roles of providing response, feedback and explanations of the decisions taken by the expert system to the user. Explanations are given to show the user the "thinking steps" taken and inferences made by the expert system to reach its conclusions.

In the case of an expert speech understanding system, explanations about how decisions are made at the lower levels of phoneme recognition say, would not be understood by the user. Such explanations may be useful during the developmental stage of the system to the designers qualified to understand them. In human conversation, audible feedback and non-speech gestures (body language) play an important role in 'getting the message across' and acknowledging that the message has been received. Ultimately, speech recognition systems will need to incorporate these human-like interfacing techniques for effective conversation to result. The ultimate system is a seeing, gesturing computer as prophesied by the quotation from Byte magazine in chapter 1.

9.2.3 An 'Expert System' for Speech Recognition

It has been mentioned on several occasions that speech contains much redundant, encoded information about the speaker and the speech message. Information about the speaker can help recognition for instance by preparing the listener to adjust to a foreign accent or a high pitched voice. Speech sound units have been shown to be difficult to extract from the speech signal without the aid and integration of higher level language tools. This syntactic, semantic, pragmatic and prosodic knowledge available in the speech signal is also used by humans to perfect the otherwise vulnerable pattern recognition performed by the ear. The classical expert system model described above is too limiting for the diversity of this speech and language knowledge and as a result the more powerful blackboard expert system model was developed.

9.3 Blackboard Systems

Blackboard systems are complex integrated expert systems with the ability to handle multiple knowledge sources. The term *blackboard* was coined to describe a system with a global problem-solving area on which the partial solutions and ultimately the final solution can be viewed and updated by any of a number of problem-solving expert systems called *knowledge sources*. A 'puzzle building' analogy (eg. in [Red76]) is often used to describe the blackboard system and how it operates.

9.3.1 The Puzzle Building Analogy

In the analogy, the *blackboard* is the area where the partial solutions of the puzzle are placed. The *knowledge-sources* are the puzzle-builders gathered in a classroom with a blackboard onto which the over-sized pieces of puzzle (with a sticky backing) are placed. The 'most promising' pieces of puzzle are placed by the puzzle-builders on the blackboard.

The other 'intelligent' puzzle-builders match their pieces against those on the blackboard to see if they can contribute to the solution. In order to prevent congestion around the blackboard, a *controller* or *supervisor* is introduced to monitor which and when builders are allowed to update the pieces on the blackboard. The builders indicate to the supervisor whenever they see they have a piece to contribute. The supervisor makes a note of them and then (according to some scheduling plan) commands them to approach and update the blackboard one at a time. The analogy can be extended even further if the builders are given separate parts of the puzzle to work on. For example, one builder may be given all the edge pieces and a high priority for placing his pieces on the blackboard. Another builder may be in charge of dividing all the pieces of puzzle into colour coordinated regions and then sharing out these regions to the other builders. The skills required by the different kinds of builders can thus be very different. This illustrates the need to have separate knowledge sources for specific tasks in a blackboard system. The puzzle solution can be carried out concurrently with several builders approaching the blackboard at once (though physically the blackboard can become congested). With a more elaborate, physically less taxing system, concurrency can be achieved. The key points in the analogy that are applicable to blackboard systems are:

- the central blackboard solution space
- the individual knowledge sources (puzzle-builders) with different skills and functions in working to the solution
- the knowledge sources solving the problem from their perspective of the blackboard. That is, problem-solving is distributed and can thus be implemented concurrently.
- the supervisor or controller which controls the updating of the blackboard by the knowledge sources, scheduling their update requests and thus controlling the global problem-solving strategy.

9.3.2 Blackboard Model

The *blackboard model* describes the problem-solving strategy and the design of the knowledge structures in a blackboard system. The model illustrated in figure 9.2, can be compared with that of the classical expert system depicted in figure 9.1.

The classical expert system was seen to be comprised of: a single *knowledge base*, the database of the system facts and an *inference engine* to handle queries and infer new facts and solutions from the old ones. It also has a *working memory* which stores the status of the partial solutions and an up to date internal representation of the semantics of the

working system. Finally, these systems often have a *user interface* which allows them to communicate with users from the real world, ideally in a natural way.

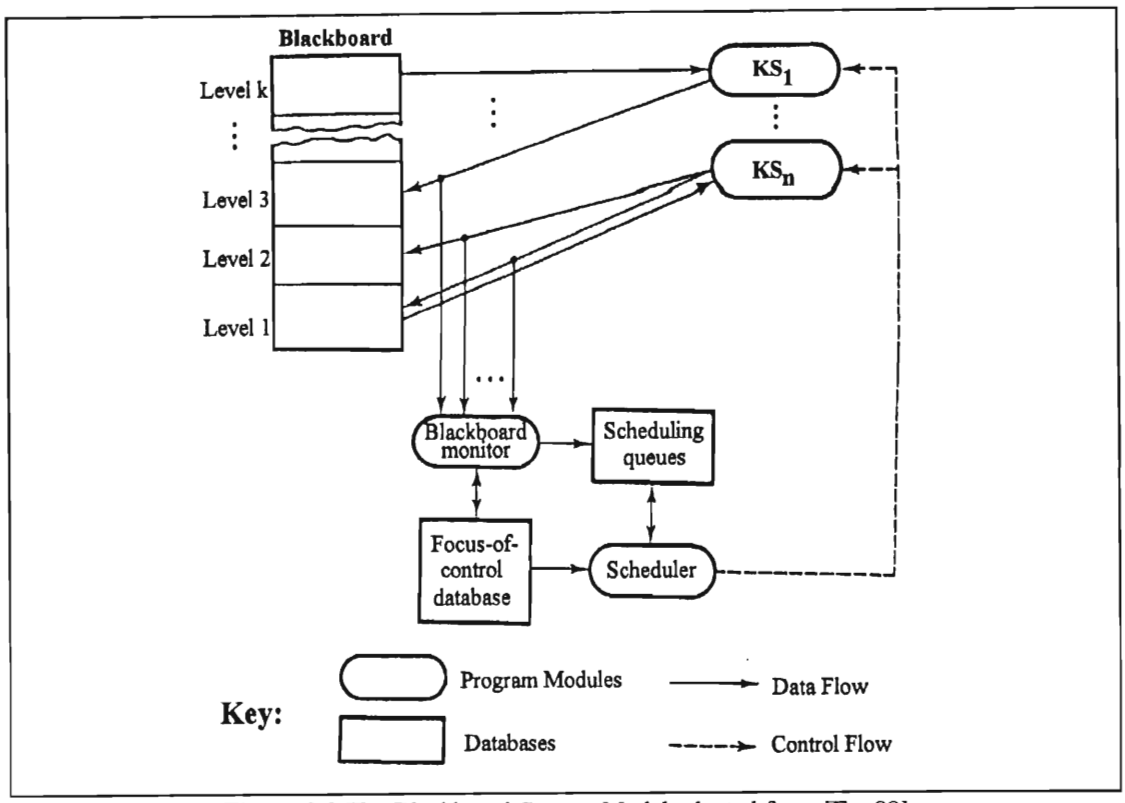

Figure 9.2 The Blackboard System Model adapted from [Eng88]

The blackboard system on the contrary consists of several *knowledge sources*, each comprised of its own knowledge base and inference engine; a global work-space called the *blackboard*, a *blackboard controller* and a *user interface*.

In the classical expert system, the most common problem-solving strategies used by the inference engine are the forward- or backward-chaining techniques. The blackboard system on the other hand, uses a distributed problem-solving approach where several knowledge sources are asynchronously and possibly concurrently involved in solving the problem. Knowledge sources are stimulated (triggered) when they 'see' they have something new to offer the solution and thus the problem-solving strategy is termed *opportunistic*.

In the practical implementation of the blackboard model, some central control is required to keep the solutions on the blackboard consistent for all the knowledge sources at all times. As a result, it is the controller's role to schedule (according to some optimal strategy) the knowledge sources that have indicated they have something to offer the solution. It is important to note that the central controller does not dictate how and when the knowledge sources do their 'viewing' and 'thinking' on the issues on the blackboard.

In this respect, the problem-solving strategy is distributed and can be implemented concurrently.

The knowledge sources are usually separated into independent units, all of which have access to the common solution information on the blackboard. For example, in the speech recognition problem, different knowledge sources might be built for solving the different parts of the recognition problem, like building words from sounds ('word builder') or constructing the internal 'semantic picture' of the sentence ('semantics generator'). In addition, both the 'word builder' and 'semantics generator' knowledge sources would be able to view the word hypotheses generated on the blackboard. The 'word generator' might use the knowledge of the previous words to restrict the probabilities of the next word while the 'semantics generator' would use the recognition of each new word to construct the 'meaning picture' of the sentence. The independence of the knowledge sources is achieved by each having a separate knowledge base specific to its perspective of the overall problem, together with an inference engine designed to query and manipulate its knowledge base. The inference engines can be forward- or backward-chaining according to the nature of its knowledge and the angle from which it approaches the problem.

Additional control of the blackboard is ensured by standardising the routines that read and write data to the blackboard. Consideration must be given to when to lock records in the blackboard data structure so that the different knowledge sources (Kss) do not have access to different results at the same time. (That is, KS1 accesses record A on the blackboard, a moment later KS2 updates record A which is then overwritten by a new value determined by KS1). The standard blackboard maintenance routines (reading and writing) also simplify the communication between the different knowledge sources through the blackboard, thus limiting the possibility of inconsistent information in the system. Some systems have partitioned blackboards to allow knowledge sources access to only certain parts of the solution relevant to them. This is another form of control that can be introduced to ensure consistency of the information.

9.3.3 Early History of Blackboard Expert Systems

The early development of blackboard systems is centred on one (the HEARSAY-II system) of the three projects commissioned by DARPA[1] to build a speech understanding system in 1971. The beginnings of the concepts of the blackboard had been thought out by Newell several years earlier:

---

[1] Defense Advanced Research Projects Agency

> "Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and to judge when he has something worthwhile to add to it. This conception is just that of Selfridge's Pandemonium (Selfridge, 1959): a set of demons, each independently looking at the total situation and shrieking in proportion to what they see that fits their natures ..." [New62] quoted in [Eng88]

Practical work using blackboard ideas was begun before the DARPA project by Reddy with his HEARSAY-I system [Red76], the fore-runner of HEARSAY-II (see [Erm76] and [Eng88]). Since then the scope of problems tackled by the blackboard system model has diversified. A complete history of the development of the field is contained in a comprehensive book on blackboard systems by Engelmore et al [Eng88].

DARPA Speech Understanding Project

DARPA commissioned three organisations: Carnegie-Mellon University (CMU)[1]; Bolt, Beranek and Newman (BBN)[2] and System Development Corporation with Stanford Research Institute (SDC/SRI) in 1971 to build speech understanding systems with the following requirements:

- vocabulary of >1000 words
- 95% accuracy rate in sentence translation
- speaker-independent but cooperative
- within 5 years
- the system understands in real-time

A comprehensive review of the DARPA (sometimes referred to as ARPA ie. without the D for Defense) project is covered by [Kla77]. Little is mentioned of the SDC/SRI system in the main discussions on speech understanding (eg. see [Whi76b]) while the HWIM and especially the HEARSAY systems have had much written about them. The best sources for the HWIM system are penned by Woods in various articles including [Woo75] [Woo85] and (with Wolf) in [Wol79]. The HWIM system which attempted to control "travel budget management" [Wol79] (ie. to act as a travel agent) was never fully completed and thus the results were fairly poor (44-49%) sentence recognition. Another reason for these results (in comparison with those of the HEARSAY-II system) was the greater scope of allowable

---

1 Actually two systems were produced by CMU, the HARPY and HEARSAY-II systems.

2 The BBN system was called HWIM, an acronym for **H**ear **W**hat **I** **M**ean.

sentences in order to function in the real world as a travel agent. The details of the HWIM system are beyond the scope of this chapter which focuses on blackboard systems. The HEARSAY-II speech understanding system was the first practical system to employ the blackboard system model. Much has been written about this system because it performed the best of the three in the DARPA project and also because of the interest in its blackboard structure. Its authors Erman and Lesser under the supervision of Reddy (and a team of other researchers) have written much about their system. A good overview is found in [Les75] while [Red73] contains the details of the predecessor HEARSAY-I system used to understand spoken chess moves. The details of HEARSAY-II's blackboard system design are admirably dealt with in [Eng88].

The HEARSAY-II speech understanding system was completed by 1976 and came close to reaching the goals specified by ARPA. A compiled version of HEARSAY-II called HARPY (see [Low77]) achieved the best results of all the systems, actually accomplishing the requirements set by DARPA. Unfortunately, the implementation of the HARPY system was 'hard-wired' for predefined speech knowledge and was not easily updated or extended. For example if new words were included in the lexicon, the entire program had to be recompiled, a 13 hour (!) process [Eng88]. HEARSAY-II on the other hand, achieved only 90% accuracy with response times several orders slower than real-time. This system was based on the blackboard model which, owing to its modularity, allowed for extension to the knowledge bases without recompiling the entire system. The concluding sections of this chapter examine the HEARSAY-II system as described by its designers in [Eng88].

9.3.4 HEARSAY-II

The various processes in the HEARSAY-II system are summarised to begin with.

Firstly, the speech signal is sampled and preprocessed to obtain characteristic speech feature vectors. The ZAPDASH segmenter (see chapter 4) is used to segment the signal into broad phonetic categories distinguishing regions of silence, frication, sonorance (vowel-like), voicing etc. These segments are then labelled by a vector quantization procedure (see chapter 4) with the codewords associated with the codevectors which best match the speech feature vectors extracted for each time frame. In fact, the best few codewords (and a confidence factor associated with each) are stored for each time frame so that the possibility of missing a phoneme at this early stage of processing through poor pattern matching is negligible.

Next, combinations of labelled phoneme segments are synthesised hierarchically into syllable, word, phrase and finally sentence hypotheses. At various stages of processing,

predictions are made about possible strings of syllables, words, phrases occurring in the future (based on what has already been processed). These predictions are later verified by post-processing once the actual hypotheses have been generated from the bottom-up synthesis process. At the same time, a 'semantic picture' of the sentence[1] is developed to eliminate certain possibilities and focus the recogniser on the subject-matter of the sentence. In the next section, the basic features of the HEARSAY-II system are discussed including the blackboard structure, the knowledge sources and the scheduler.

The Blackboard

The basic strategy behind any speech understanding system is to reduce the uncertainty of the low-level phoneme-based (or equivalent subword sound unit) pattern recognition 'ear' using higher-level speech and language knowledge. This can be achieved by storing on the blackboard the best few phoneme matches for each time frame and then predicting strings of phonemes as syllables, words, phrases and finally sentences in a hierarchical manner (ie. in levels). The system assigns to every hypothesis at every level a confidence factor measuring its credibility. Hypotheses in the same (or overlapping) time frames at the same level are considered to be *competitive* while those in the same time frame at different levels are said to be *cooperative*. The best options are thus determined by the confidence factors of the hypotheses at any time frame of the signal. *Links* are kept between levels (eg. word → phrase) describing the cooperative relationships between hypotheses at different levels. In addition, competitive links are kept between hypotheses competing for the same time frame at a particular level.

The blackboard is structured hierarchically with phoneme hypotheses at the lowest level and sentence hypotheses and semantic abstractions at the top levels. This allows the solution to be tackled in a top-down and bottom-up manner. The hypotheses corresponding to recognised sound events at the lowest level are used to synthesise information up the hierarchical structure. At the same time, decisions about the sentence structure and meaning made globally are refined (using the divide-and-conquer strategy) until the hypotheses lower down are met and matched.

Also, for every new hypothesis on every level, there are usually repercussions for the hypotheses on the same level and the ones above. For example, if a new phoneme hypothesis is determined, it will trigger interest from the syllable and word levels which will need to use the new phoneme to update their evolving syllables and words. The levels which predict speech events in advance will require verification of their predictions when

[1] Including pragmatic information

the actual events are later determined. The system uses a prediction and verification strategy to optimise the search for the global solution. This approach was shown to model the way humans listen or read in chapter 8.

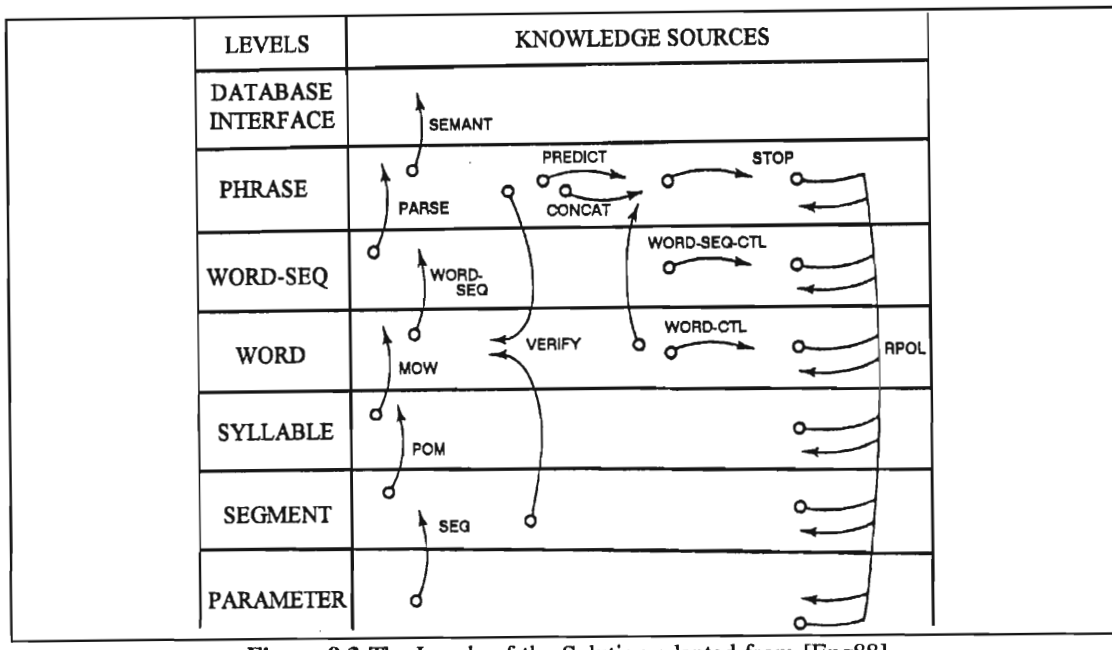

Figure 9.3 The Levels of the Solution adapted from [Eng88]

Figure 9.3 shows the various levels of solution and the interlevel processes associated with each level.

The Knowledge Sources

The knowledge sources have two main components: a *precondition* and an *action*.
The precondition of a particular knowledge source is the set of events or actions that must take place on the blackboard in order for that knowledge source to be able to contribute to the (partial) solution of the problem. This set of prerequisite events or actions matching the knowledge source's precondition is called the *stimulus frame*. Once the precondition has been met, the knowledge source sends the stimulus frame and a *standard report*[1] to the scheduler of what the knowledge source hopes to achieve and the time taken to perform this if it were allowed to update the blackboard. This message to the scheduler is known as the *response frame*.

The action component of a knowledge source is a set of procedures which the knowledge source carries out when the scheduler grants it permission to update the blackboard according to its request (submitted by the response frame). The steps carried out by the

[1] It is standard because the format of the report is set for all knowledge sources.

action of some knowledge source typically generates stimulus frames triggering the preconditions of other knowledge sources. Thus changes in the blackboard create requests for more changes to the blackboard in an opportunistic manner.

The HEARSAY-I system used a polling system where the controller would poll each knowledge source to check the blackboard to see whether its precondition was met. This technique was rejected due to its inefficiency. Instead the primitive changes in the blackboard associated with each precondition, is given to the blackboard controller. When these primitive changes to the blackboard take place, the controller triggers the relevant knowledge sources in an interrupt-driven manner.

A summary of the various knowledge sources of the HEARSAY-II system is presented below [Eng88]:

Signal acquisition, parameter

- **SEG**: digitizes the signal, measures parameters and produces a labelled segmentation
- **MOW**: creates word hypotheses from syllable classes
- **WORD-CTL**: controls the number of word hypotheses that MOW creates

Phrase-island generation

- **WORD-SEQ**: creates word-sequence hypotheses that represent potential phrases from word hypotheses and weak grammatical knowledge
- **WORD-SEQ-CTL**: controls the number of hypotheses that WORD-SEQ creates
- **PARSE**: attempts to parse a word sequence and, if successful creates a phrase hypothesis from it

Phrase extending

- **PREDICT**: predicts all possible words that might syntactically precede or follow a given phrase
- **VERIFY**: rates the consistency between segmented hypotheses and a contiguous word-phrase pair
- **CONCAT**: creates a phrase hypothesis from a verified contiguous word-pair

Rating, halting and interpretation

- **RPOL**: rates the credibility of each new or modified hypothesis using information placed on the hypothesis by other Kss
- **STOP**: decides to halt processing (detects a complete sentence with a sufficiently high rating, or notes the system has exhausted its available resources) and selects the best phrase hypothesis or set of complementary phrase hypotheses as the output
- **SEMANT**: generates an unambiguous interpretation for the information-retrieval system which the user has queried

The Scheduler

The task of the scheduler is to determine when to execute the pending tasks described by the response frames. The priority of the various pending tasks is predominantly determined by the usefulness of their execution to the overall solution. This is measured by checking the stimulus and response frames of the pending tasks and other general blackboard information eg. time since the hypothesis was generated. There are several potential hazards with scheduling, the most important being the possibility of an explosion in the number of pending knowledge sources waiting to update the blackboard. This problem can be averted by increasing the power of the scheduler to remove requests by various knowledge sources to update the blackboard if it deems that their contributions are ineffectual.

9.4 Conclusion

This chapter showed the need for an integrated knowledge-based system to improve the relatively poor pattern recognition results of phoneme-based speech recognition systems. The blackboard expert system based on problem-solving strategies of expert systems was studied as the most successful system to date for solving this task.

Building a speech understanding system is a sizeable ask. The HEARSAY-II speech project, for example, needed 40 man-years to implement [Eng88]. Hayes-Roth implies the same thing when he rather pessimistically reviews the speech understanding problem, saying that it will take a small Apollo project team three years to build a speech recogniser with a 1000-word vocabulary with a very restricted grammar and universe of discourse [Hay83].

The blackboard appears to have the structure and design flexibility to undertake this enormous task. What is lacking at the moment, is hardware technology and algorithms to solve the speech recognition problem using concurrent and distributed processing.

Finally, it is interesting to note that since the DARPA project, there seems to have been no further systems attempting to solve speech understanding. Researchers have possibly grown despondent with the small rewards gained from such undertakings and prefer to take on and solve smaller parts of the bigger picture.