

Evaluation of modern large-vocabulary speech
recognition techniques and their implementation

by

Renier Adriaan Swart

*Thesis presented in partial fulfilment of the requirements for
the degree of Master of Science in Electronic Engineering at
the University of Stellenbosch*



Department of Electrical and Electronic Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Supervisor: Prof J.A. du Preez

March 2009

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature:

R.A. Swart

Date:

Copyright © 2009 University of Stellenbosch
All rights reserved.

Abstract

Evaluation of modern large-vocabulary speech recognition techniques and their implementation

R.A. Swart

Department of Electrical and Electronic Engineering

University of Stellenbosch

Private Bag X1, 7602 Matieland, South Africa

Thesis: MScEng

March 2009

In this thesis we studied large-vocabulary continuous speech recognition. We considered the components necessary to realise a large-vocabulary speech recogniser and how systems such as Sphinx and HTK solved the problems facing such a system.

Hidden Markov Models (HMMs) have been a common approach to acoustic modelling in speech recognition in the past. HMMs are well suited to modelling speech, since they are able to model both its stationary nature and temporal effects. We studied HMMs and the algorithms associated with them. Since incorporating all knowledge sources as efficiently as possible is of the utmost importance, the N-Best paradigm was explored along with some more advanced HMM algorithms.

The way in which sounds and words are constructed has been studied extensively in the past. Context dependency on the acoustic level and on the linguistic level can be exploited to improve the performance of a speech

recogniser. We considered some of the techniques used in the past to solve the associated problems.

We implemented and combined some chosen algorithms to form our system and reported the recognition results. Our final system performed reasonably well and will form an ideal framework for future studies on large-vocabulary speech recognition at the University of Stellenbosch. Many avenues of research for future versions of the system were considered.

Uittreksel

Evaluation of modern large-vocabulary speech recognition techniques and their implementation

R.A. Swart

Departement Elektries en Elektroniese Ingenieurswese

Universiteit van Stellenbosch

Privaatsak X1, 7602 Matieland, Suid Afrika

Tesis: MScIng

Maart 2009

In hierdie tesis het ons kontinue spraakherkenning in die konteks van groot woordeskatte bestudeer. Ons het gekyk na verskeie komponente wat benodig word om so 'n stelsel te realiseer en hoe bestaande stelsels soos Sphinx en HTK die probleme opgelos het.

Verskuilde Markov-Modelle (VMM's) is in die verlede gereeld gebruik vir akoestiese modellering van spraak. VMM's is ideaal vir spraak toepassings aangesien hulle daartoe in staat is om stasionêre sowel as temporale eienskappe te modelleer. Ons het VMM's en die algoritmes wat benodig word om VMM's te realiseer, bestudeer. Verskeie kennisbronne moet effektief benut word vir 'n bruikbare stelsel en daar is maniere ondersoek om dit te bewerkstellig.

Die manier waarop klanke en woorde in spraak gebou word is voorheen al deeglik bestudeer. Konteks-afhanklike klank- en taalmodelle en die tegnieke wat al toegepas is om hulle te benut is bestudeer. Konteks is 'n uiters

belangrike bron van kennis en moet in ag geneem word vir enige effektiewe spraakherkenner.

Nadat ons gekyk het na verskeie gewilde benaderings tot die spraakherkennings probleem het ons 'n volledige stelsel van ons eie ontwerp. Ons het verskeie algoritmes gekies en geïmplementeer en die herkennings resultate gerapporteer. Ons stelsel was redelik akkuraat en sal 'n ideale raamwerk vorm vir toekomstige studies in groot woordeskat spraakherkenning by die Universiteit van Stellenbosch.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations who have contributed to making this work possible:

- Krygkor (Pty) Ltd for making funds available and also for permission to publish the research results,
- Prof Johan du Preez at the University of Stellenbosch for leading me through the entire study process,
- Dr Herman Engelbrecht at the University of Stellenbosch for all his guidance in difficult times,
- Marisa Crous for always standing by me and listening to my technical rants.

Contents

Declaration	i
Abstract	ii
Uittreksel	iv
Acknowledgements	vi
Contents	vii
List of Figures	viii
Nomenclature	ix
1 Introduction	1
1.1 Motivation	1
1.2 Background	2
1.2.1 Audio processing	2
1.2.2 Acoustic modelling	2
1.2.3 Language modelling	3
1.2.4 Search techniques	3
1.3 Literature Study	4
1.3.1 History of speech recognition	4
1.3.2 Speech Recognition in recent years	8
1.3.3 Summary	12

1.4	Objectives	13
1.5	Overview of this work	13
1.5.1	Hidden Markov Models	13
1.5.2	Context Dependency	15
1.5.3	Implementation of test system	15
1.5.4	Experimental investigation	16
1.6	Contributions	18
2	Hidden Markov Models	19
2.1	HMM Definition	19
2.2	HMM assumptions	20
2.2.1	Markov assumption	21
2.2.2	Output-independence assumption	21
2.3	The HMM problem	21
2.3.1	The evaluation problem	22
2.3.2	The decoding problem	24
2.3.3	The learning problem	27
2.4	HMM Types	27
2.4.1	HMM Topology	28
2.4.2	State output probability distributions	28
2.5	HMM Applications	30
2.5.1	Whole word models	31
2.5.2	Phoneme models	33
2.6	Evaluating HMMs	35
2.7	More HMM algorithms and optimisations	36
2.7.1	Beam segmentation	36
2.7.2	Multi-level HMM segmentation	38
2.7.3	N-Best paradigm	39
2.8	Summary	44
3	Context dependency	46
3.1	Context types	46

3.2	Context-dependent phoneme modelling	47
3.2.1	Trainability	47
3.2.2	Decision Trees	49
3.3	Language Modelling	52
3.3.1	Definition	52
3.3.2	Techniques	54
3.4	Summary	59
4	Implementation	60
4.1	Multi-level Beam HMM segmenter	61
4.2	N-Best segmenter	66
4.3	Language model	77
4.4	N-Best grammar segmenter	78
4.5	Silence Detection	81
4.6	Context-independent phoneme modelling	81
4.7	Context-dependent phoneme modelling	85
4.8	Word Spotter	87
4.9	Summary	89
5	Experimental investigation	90
5.1	Continuous phoneme recognition on Hub-4 broadcast speech	91
5.1.1	Flat start monophone recognition on Hub-4 broadcast speech	91
5.1.2	Forced alignment monophone recognition on Hub-4 broadcast speech	92
5.1.3	Forced alignment triphone recognition on Hub-4 broad- cast speech with monophone densities	93
5.1.4	Forced alignment triphone recognition on Hub-4 broad- cast speech	94
5.2	Effect of beam on accuracy and performance	99
5.3	Continuous word recognition on Hub-4 broadcast speech de- velopment set	99

5.3.1	Finding effective parameters for phase one.	100
5.3.2	Finding effective parameters for phase two.	105
5.3.3	Finding effective parameters for phase three.	108
5.4	Determining the effect of a word length penalty	110
5.5	Sphinx 3 Hub-4 Development set evaluation	112
5.6	Continuous word recognition on Hub-4 broadcast speech evaluation set	115
5.7	Summary	116
6	Conclusion	118
6.1	Concluding Perspective	118
6.2	Future work	120
	Bibliography	122
A	General Speech Recognition and Evaluation Techniques	129
A.1	Linear Discriminant Analysis	129
A.2	Mel-frequency cepstral coefficients	130
A.3	Perplexity	132
A.4	A density for the estimated average power of Gaussian noise	133
A.5	The mean and variance of an estimate of the variance of the estimated mean power	135
B	Phoneme set	140
C	Question set	142

List of Figures

2.1	A small HMM example.	21
2.2	HMM used to illustrate use of path matrix. In the multi-level example, null states 2, 4 and 6 are considered word endings (super states).	26
2.3	A possible path matrix $B_t(j)$ produced by the Viterbi algorithm applied to the HMM in Fig. 2.2.	26
2.4	An example of a four state left-to-right HMM.	28
2.5	A general Gaussian distribution.	29
2.6	HMM for an N -word classifier for isolated word recognition. . .	32
2.7	HMM for an N -word spotter for continuous speech recognition. The addition of the transition from the final state to the first state (feedback loop) enables the HMM to recognise a sequence of words instead of just isolated words.	33
2.8	HMM for a word model constructed from three phonemes. . . .	34
2.9	Viterbi expansion without beam, where each possible expansion is made.	37
2.10	Viterbi expansion with beam, where only the higher scores are expanded.	38
2.11	The N-Best Paradigm. Inexpensive knowledge sources (KSs 1) are incorporated early, while the remaining and more expensive knowledge sources (KSs 2) are used to generate the final hypothesis [43].	40
2.12	An example of a 5-Best list represented as a lattice.	41

2.13	The traceback-based N-Best deficiency. Paths with different histories cannot be distinguished [43].	43
2.14	The word-dependent N-Best algorithm. Paths with different preceding words are combined [43].	44
3.1	HMMs not sharing PDFs, where many PDFs need to be trained and a large amount of training data is required.	48
3.2	HMMs sharing PDFs, where fewer PDFs need to be trained resulting in less data being necessary for them to be properly trained.	48
3.3	A simplified example of a decision tree that determines risk for cancer. Males over the age of 40 and female smokers have the highest risk.	49
4.1	The full path matrix from a small example. The arrows indicate how backtracking takes place once the matrix is fully populated.	62
4.2	The super path and time matrix from a small example. The arrows indicate how backtracking takes place once the matrices are fully populated.	63
4.3	Example of HMM used for two-word N-Best spotter. The symbol under each state indicates which phoneme is represented by that particular state. A list of the phonemes used in our implementation can be found in App. B.	68
4.4	N-best relying on 1-best segmentation. Super states such as states 33 and 39 are shown in the same matrix as non-super states for the sake of this illustration. The arrows show the 1-best path.	69
4.5	Illustration of steps 1 and 2 in the N-Best segmentation algorithm. Step 1 forms the marker entry at state 0 and time 0, while step 2 expands that entry to the glue states and in turn the begin states.	71

4.6	Illustration of step 3 in the N-Best segmentation algorithm. The begin state entries in the 1-best matrix are populated with the best scores from their associated N-Best lists.	72
4.7	Illustration of step 4 in the N-Best segmentation algorithm. Normal 1-best segmentation takes place exactly as is done with the Viterbi algorithm.	73
4.8	Illustration of step 6 in the N-Best segmentation algorithm. After emitting states 8 and 11 are expanded to end states 1 and 2, the 1-best end states scores are used to form the end state lists.	74
4.9	Illustration of a later iteration of step 2 in the N-Best segmentation algorithm. The newly formed end state lists are expanded and merged to the glue states and in turn the begin states. . . .	75
4.10	N-best end states and begin states path and score matrices. . . .	77
4.11	1-best path, score and begin state backtrack buffers.	78
4.12	Example of HMM used for 2 word N-Best grammar spotter. Words are not connected and the feedback loop and glue states are removed. The language model is responsible for connecting the words.	79
4.13	Monophone training process.	81
4.14	Example of utterance HMM used in forced alignment. Optional silences separate words.	83
4.15	Triphone training process.	85
4.16	HMM configuration used in first phase of word spotter. Words are placed in parallel, with optional silences separating words. The language model weights are incorporated last in the segmentation process, so that they are incorporated first when backward Viterbi segmentation takes place.	88
4.17	Process used to generate word hypothesis.	89
5.1	An HMM topology commonly used to model phonemes.	92
5.2	Context-dependent phoneme spotter with an alphabet containing two symbols. Only matching contexts can be recognised. . . .	95

5.3	Number of parameters for various MOCs. The number of clusters drops as the maximum occupation count is increased. . . .	97
5.4	Insertion and deletion rates for various parameter counts. . . .	97
5.5	Accuracy for various parameter counts. The most accurate systems are between 1000 and 2000 clusters.	98
5.6	Ticks required for monophone spotting for beam values of e^{Beam} . The processing power required increases as the beam becomes wider.	100
5.7	Accuracies found for monophone spotting for beam values of e^{Beam} . The maximum accuracy is quickly reached and a beam width that is increased further has no effect.	101
5.8	Lattice correctness for various average word beginnings per frame counts. The lattice correctness only increases slightly for systems with an average of more than 20 word beginnings per frame. . .	104
5.9	Linear model used for word length penalty. The more phonemes the word contains, the smaller the penalty becomes.	111
A.1	Triangular filters used in MFCC.	131

Nomenclature

Acronyms:

CART	Classification and Regression Tree
CFG	Context Free Grammar
CHMM	Coupled Hidden Markov Model
DL	Description Length
EM	Expectation-Maximisation
FFT	Fast Fourier transform
GMM	Gaussian Mixture Model
GW	Grammar Weight
HMM	Hidden Markov Model
KS	Knowledge Source
LDA	Linear Discriminant Analysis
LM	Language Model
LPC	Linear Prediction Coefficients
LVCSR	Large-Vocabulary Continuous Speech Recognition
MAPMI	Maximum Active Phone Model Insurance
MDL	Minimum Description Length
MFCC	Mel-frequency cepstral coefficients
MOC	Minimum Occupation Count
PDF	Probability Density Function

PLP	Perceptual Linear Prediction
SAM	Structured Adaptive Mixture
SNR	Signal-to-Noise Ratio
WER	Word-Error Rate
WIP	Word Insertion Penalty
WPF	Words Per Frame

Symbols:

$\alpha_i(t)$	Forward probability
$\beta_i(t)$	Backward probability
$\gamma_i(t)$	Probability of being in state i at time t , given the observation sequence \mathbf{X} and the model Φ
$\boldsymbol{\mu}$	Mean vector
$\boldsymbol{\pi}$	Initial state distribution in HMM
Φ	Hidden Markov Model
$\hat{\Phi}$	Hidden Markov Model with maximised likelihood for observing a training observation sequence \mathbf{X}
Σ	Covariance matrix
$\zeta_t(i, j)$	Probability of taking the transition from state i at time t to state j at time $t + 1$
a_{ij}	Transition probability from state i to state j in an HMM
\mathbf{A}	Probability matrix containing state transition probabilities for an HMM
$b_i(\mathbf{x})$	Probability of observing \mathbf{x} at state i in an HMM
\mathbf{B}	State output probability distributions for an HMM
$B_t(i)$	Best-path state history
c_{jk}	Weight of the k th mixture in the GMM at state j
k	HMM state index
N	The number of states in an HMM

\hat{N}	The number of super states in an HMM
N^B	The number of begin states in an HMM
N^E	The number of end states in an HMM
N_F	The number of terminating states in an HMM
\mathcal{N}	Gaussian probability distribution
$P(\cdot)$	General probability
$Q(\Phi, \hat{\Phi})$	Baum's auxiliary function
$s(n)$	Discrete-time speech signal
s_t	Hidden Markov Model state occupied at time t
\mathbf{S}	Hidden state sequence in an HMM
T	The amount of observations in an observation sequence
$V_t(i)$	Best-path probability
\mathbf{x}_t	Observation at time t
\mathbf{X}	Sequence of observations

Chapter 1

Introduction

1.1 Motivation

Speech is by far the easiest and fastest way to communicate for humans. It requires little effort for us to speak and we can communicate very efficiently in this way. For this reason, it is sensible to develop ways for humans to communicate with computers in the same way. Little or no additional training is required from the user. Traditional interfaces such as the keyboard and the mouse can prove difficult for individuals not used to computers.

Speech recognition systems are developed in an attempt to realise such ideas. Unfortunately speech recognition is an extremely complex task and is infamous for being very difficult to carry out efficiently. On larger vocabularies (greater than 1000 words), confusability and processing requirements grow rapidly and along with it the complexity of the problem.

In this thesis we explored large-vocabulary continuous speech recognition (LVCSR) by looking at some current systems and by implementing a basic LVCSR system of our own. We considered various approaches and developed the fundamental building blocks necessary to realise speech recognition with a large-vocabulary. We compared the results of our system with Sphinx 3 [37] and considered possible future developments.

1.2 Background

A typical large-vocabulary speech recogniser will need all of the following components:

1. Audio processing
2. Acoustic modelling
3. Language modelling
4. Search techniques

1.2.1 Audio processing

Computers have limited storage, which renders them unable to store continuous signals. For this reason, we need to convert raw audio data received from a microphone into a discrete signal $s(n)$. This is done by sampling the continuous signal with some form of analog-to-digital conversion. This discrete signal is then converted into a sequence of feature vectors

$$\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m. \quad (1.2.1)$$

The audio processing component of the speech recogniser is responsible for this task.

1.2.2 Acoustic modelling

In speaker independent speech recognition we have to represent a diverse range of speakers with a single model. For this reason modelling the speech is difficult and models need to be very versatile. This model grows ever more complex as more and more speakers need to be understood, until eventually you have the model that is completely speaker independent. When we need to distinguish between a larger variety of sounds (phonemes) this becomes increasingly difficult. Confusability increases and the accuracy of the model

becomes more important. Statistical Hidden Markov Models (HMMs) represent these speech models well, since they are intrinsically related to time. HMMs are discussed in Chapter 2.

1.2.3 Language modelling

Language models are another valuable knowledge source. They are especially valuable in LVCSR applications, since there are so many word combinations to consider. The language model reduces the search space by incorporating knowledge about the type of speech being recognised. In a true LVCSR system this grammar would be quite vast, but even a very general grammar makes the task more feasible. There are many ways to construct and incorporate this grammar into the search.

A popular technique is to have a training set of common spoken label sequences. These labels can be anything, but phonemes or words are used in most cases. All these label sequences are used to create the language model. This model can now be used for many things, including to determine the probability of a word sequence being found. Since we are only interested in recognising word sequences our grammar will be based on words. Language modelling is discussed in greater detail in Chapter 3.

1.2.4 Search techniques

The search techniques are highly dependent on the modelling techniques, but they play a significant role in any recognition task. Ideally we need to find the highest probability for $P(\mathbf{W})P(\mathbf{X}|\mathbf{W})$, where $P(\mathbf{W})$ is the probability of word sequence \mathbf{W} and $P(\mathbf{X}|\mathbf{W})$ is the probability of the observation sequence \mathbf{X} being generated given word sequence \mathbf{W} . In practice this is not feasible for large vocabularies, since we would need to consider each possible word combination. The search space needs to be narrowed down based on the acoustic observations. We would need to only consider word sequences that sound very similar to the acoustic observations. By further

reducing the search space with a grammar (incorporated through $P(\mathbf{W})$), we can reduce the search space sufficiently for a large-vocabulary search to be feasible. The application of this idea to HMMs is explored further in Chapter 2.

1.3 Literature Study

1.3.1 History of speech recognition

Speech recognition has been studied extensively throughout the world for many years. Initial speech recognition systems had very limited resources and were forced to restrict their capabilities to ensure feasibility. The first systems modelled each word in the vocabulary for a specific speaker for isolated words and recognition accuracies were acceptable. They found decent results by using one or more of the following constraints:

1. Speaker dependence
2. Small vocabulary
3. Isolated word recognition
4. Restricted grammar

One of the primary problems with speech recognisers is the number of speakers it needs to be able to recognise. If we have multiple speakers we not only have confusability between models, but also between speakers. People might pronounce the same word completely differently. On the acoustic modelling side it is clearly very difficult to create models that are accurate for a diverse range of speakers. Vast differences occur in the realisations of speech units related to context, style of speech, dialect and speaker. In 1975, Itakura [19] was one of the first to show that speech recognition for a single speaker was possible. He used dynamic time warp techniques to recognise isolated words in a 200 word vocabulary and achieved a 97.3%

accuracy. One of the main reasons for Itakura's success was the fact that his system was trained and tested on a single male speaker. Words were recognised by calculating a minimum prediction residual and the models were constructed using linear prediction coefficients (LPC). Computational power was very limited at that time, which resulted in the system processing telephone recordings at 22 times slower than real time.

Another big problem with recognition accuracy is the vocabulary size. If the vocabulary becomes larger than about 1000 words we need significantly more processing power and memory. The inherent confusability of many words also add to the difficulty of the problem and lead to reduced recognition accuracy. The 1975 Hearsay II system at Carnegie Mellon University [26] was able to recognise continuous speech for a much larger vocabulary of around 1200 words with an accuracy of 87%. This was only possible due to a simple English-like grammar with a perplexity¹ of 4.5, which reduced the search space significantly. The system was based on the hypothesise-and-test paradigm and used cooperating independent knowledge sources communicating with each other through a shared data structure. A convenient modular structure was designed and could incorporate new knowledge into the system at any level.

Others focused on recognising continuous speech. Itakura only recognised isolated words, which makes recognition easier. Continuous speech has the added dimension of unknown word boundaries. If we can assume that only a single word was uttered in a given test sample we can simply compare the test sample with the model of each word in the vocabulary and find the best match. Baker [3] used uniform stochastic modelling to represent knowledge sources in the same year as Hearsay II and used a probabilistic function of a Markov process to model speech. By introducing the idea of using HMMs in speech recognition he formed the foundation for many studies continuing today. They recognised continuous speech with an accuracy of 84%, but they were still using a small set of speakers and a

¹See Appendix A.3 for more information on perplexity.

limited 200 word vocabulary.

If we have a single speaker and a small vocabulary we can relatively easily create whole-word models for each word in the vocabulary. Data scarcity will not be a problem since we can easily gather examples of each word from the single speaker. In later years some were able to lift this constraint. In 1982 Wilpon *et al.* [51] made use of statistical clustering techniques to create models that were not dependent on a specific speaker. They reported an accuracy of between 80% and 97%, but were still using a very small vocabulary of around 129 words.

It was not long before very large vocabulary recognition was attempted. The speech recognition group at IBM attempted a very-large-vocabulary task with their Tangora System [20] in 1985. They recognised isolated words from a 5110 word vocabulary and used a trigram language model. Such a language model gives the probability of any word within the vocabulary when given the preceding two words. The specific language model used by them had a perplexity of 160. They constructed their vocabulary from the most common words in a massive business memo database, which totalled about 25 million words. This system achieved between 93.3% and 98% accuracy, but was speaker dependent.

Context-dependent phoneme modelling was studied in later years and some success was achieved at BBN [8] in 1987. They developed the BYBLOS system, which was designed for large-vocabulary applications. The system integrated acoustic, phonetic, lexical and linguistic knowledge sources with great success. They modelled co-articulation effects using HMMs and recognised around 97% on a 350 word vocabulary. Technically the system was speaker dependent, but after a short² enrolment time the system could very accurately recognise utterances from a new speaker. They used a language model with a perplexity of 60.

Further progress was made at Bell Labs by Rabiner in 1988 [38] on continuous and speaker-independent recognition by creating a connected

²Five to fifteen minutes.

digit recogniser. By modelling both instantaneous and transitional spectral information with mixed HMMs they improved significantly on previous results by achieving accuracies of 97% and above. This result is particularly impressive since they used no grammar and these accuracies were measured on whole sentences.

One of the first to almost completely lift all of the constraints that were mentioned earlier was Kai-Fu Lee with the 1988 Sphinx system [25]. This would play a major role in speech recognition development for the next decade. Triphone models were used along with word-dependent phone modelling. Deleted interpolation was also used to combine robust models with detailed ones. Using a grammar with a perplexity of 997 and a vocabulary of around 1000 words he was able to recognise 73.6% of the words correctly. This was a great achievement since the recogniser was speaker independent and recognised continuous speech on a large vocabulary with a very general grammar. Kai-Fu Lee was one of the first to prove that large-vocabulary speaker-independent continuous speech recognition was possible.

In later years, Huang *et al.* [16] further improved on Sphinx with the 1993 Sphinx-2 system. Focus was placed on the improvement of speech recognition systems with increased task perplexity, speaker variation and environment variation. This was achieved by making use of semi-continuous HMMs, sub-phonetic modelling, improved language modelling and speaker-normalised features. Long distance bigrams were incorporated along with a special back-off model to create an accurate and efficient language model. They attempted to recognise speaker-independent continuous speech. When using a grammar with perplexity of 60 and a vocabulary of 1000 words they achieved an accuracy of around 97%.

The Cambridge University Speech Group developed large-vocabulary speech recognisers using their 1993 HTK system [53] in the same year. They made use of state tying and Gaussian mixture density HMMs to model triphones. They applied their system to the 5000 word 1993 Wall Street Journal corpus and found an accuracy of around 95%. A trigram language

model was used and applied using a single pass dynamic network decoder. The HTK system is still being developed and is examined further in Section 1.3.2.1.

The next phase for the Sphinx system was the release of Sphinx 3 [37] in 1996. Two language models were used in this system: one to guide the decoder in the actual recognition, and a different one for re-scoring the N-best output hypotheses³. The fact that they generated an N-best list of hypotheses allowed them to optimise the language model weight and insertion penalty with Powell's algorithm. When evaluating a 25000 word vocabulary and a language model with perplexity 170 they obtained an accuracy of 65.1%.

Other new concepts were introduced in later years, such as the use of discriminative training for large-vocabulary HMM-based speech recognition. The maximum mutual information estimation (MMIE) [52] proved to have many advantages. This technique allowed the estimation of triphone HMM parameters which led to significant reduction in word error rate for the transcription of conversational speech relative to the best systems using maximum likelihood estimation (MLE).

1.3.2 Speech Recognition in recent years

In recent years some notable LVCSR systems have been developed. In this section we look at some of the more powerful systems and examine their capabilities. This is by no means an exhaustive study, but the more popular systems are examined.

1.3.2.1 HTK

The HTK toolkit [53] is still under development at the Cambridge University Engineering Department and version 3.4 was released in 2006. It consists of tools for building and manipulating continuous density HMMs.

³A 200-best list was used in the final implementation of Sphinx 3.

The main focus of the system was an extensive structure for training and evaluating HMMs using various techniques in order to advance speech recognition research. HTK also supports a wide selection of acoustic modelling techniques, including diagonal and full-covariance Gaussian mixture HMMs.

All the standard feature extraction techniques such as MFCC and PLP are included in the system. Some other experimental techniques such as Vocal Tract Length Normalisation (VTLN) were also added in later versions.

HMM support on HTK is extensive and all the normal HMM representations are included. State-of-the-art HMM modelling is the core feature of the system, which is capable of parameter tying and decision tree state clustering to create triphones. HTK also supports MLLR and MAP adaptation, which makes it particularly useful as a speaker-independent system.

A silence detection tool is also part of the system and can be used to subdivide longer segments and lighten the load on the decoder. Various optional outputs can be extracted from any decoder in the system, including a word lattice and N-Best sequences. The powerful large-vocabulary decoder was also included in version 3.4 and supports multiple parallel data streams. Bigram and trigram language modelling capabilities with advanced back-off techniques and cross-word triphone support make HTK a very powerful toolkit.

1.3.2.2 AVCSR

An audio-visual continuous speech recognition system was developed at Intel [27], which made use of the Coupled Hidden Markov Model (CHMM). The CHMM can describe the asynchrony of the audio and visual features and at the same time preserve their natural correlation over time. This enables the system to recognise continuous speech more accurately than an equivalent audio-only system.

Each of the possible phoneme-viseme pairs are modelled by an CHMM and they reduced the word error rate of their audio only speech recognition

system at an SNR of 0dB by over 55%. This is definitely a promising avenue for future research, but the addition of visual data complicates the data gathering process.

1.3.2.3 Aurora

The Aurora system [35] was developed in 2002 and is an extension of the baseline system developed in 1999 [10]. The system made use of MFCCs and lexical processing was done by means of HMMs and a lexical tree. A virtual copy of the tree is created for each n -gram word history (a dynamic tree approach), which resulted in two benefits:

1. More efficient lexical processing due to the reduced number of nodes
2. Grammar knowledge is incorporated at an earlier stage

A hierarchical variation of the Viterbi search was implemented and they incorporated two types of pruning:

1. Beam pruning
2. Maximum active phone model insurance

Triphones and a bigram language model were used to generate a lattice. This lattice was then used for rescoring using more expensive cross-word triphone models and they found decent performance.

1.3.2.4 Sphinx-4

Sphinx remains one of the best LVCSR systems with the 2004 Sphinx-4 system [50, 24]. Sphinx-4 was implemented in Java and still focuses on using HMMs to recognise speech. The system has an easy-to-understand modular design and many implementations of the various standard building blocks for speech recognisers. This enables a user to customise Sphinx-4 to meet their own specific application needs.

The Sphinx-4 framework consists of three primary modules, which can be configured individually and interact in various ways. The configuration manager also provides various tools which can be used to measure word error rate, memory usage and runtime speed.

- **FrontEnd:** The FrontEnd takes an input signal and calculates feature vectors using this information. Sphinx-4 is highly configurable and capable of producing parallel sequences of features. The more popular feature extraction techniques are available in the system, including MFCC and PLP [14].

Various data processors can be connected in sequence to produce the specific features required. Normally each block propagates features as they are calculated to the next data processors, but Sphinx-4 has a different approach. Each block requests data from previous blocks as required, enabling it to manage data efficiently and also request data from the history or the future.

Sphinx-4 uses advanced Endpoint detection to separate speech and non-speech segments. Only speech segments are sent to the decoder, preventing unnecessary data processing.

- **Linguist:** The linguist block is a representation of an arbitrarily selected grammar and grammar type. Three main grammar formats are supported:
 - Word-list grammar: Simple unigram from list of words.
 - n -gram: Statistical n -gram models in the ARPA-standard format.
 - Finite state transducers [30].

Separate from this grammar is the sentenceHMM graph, which is a directed state graph where each node represents a unit of speech. Also using a wide variety of building blocks such as phonemes and a lexicon, it constructs the appropriate HMM.

- **Decoder:** The decoder makes use of a search manager to search through a tree of hypotheses constructed with data from the linguist. Each node in the tree has references to a node in the "sentence HMM", which then in turn allows it to get all info about grammar state, word and so forth. The search module has a list of active tokens, which are the best scoring leaves in the tree. When new acoustic data is received, these active tokens are expanded and the weaker points are pruned. After all features are processed, the final active list is given as a result. This list can be converted into an N-best list, or the best scoring path can be extracted.

Decoding can also be customised and the user can choose traditional breadth-first Viterbi search or variations of it, but also more recent techniques such as Bush-Derby [45]. Depth-first search, such conventional stack decoding, can also be performed.

1.3.3 Summary

From studying existing systems some basic universal features appear essential. HMMs are very popular and widely studied, making them an obvious choice for acoustic modelling. A language model should also be incorporated, since previous studies have shown that it is essential to reduce word error rates.

Clearly, most LVCSR systems make use of a multi-pass approach, which enables them to use inexpensive knowledge sources to reduce the search space. Later stages then apply the more detailed and expensive knowledge sources to this reduced search space. This results in a highly dynamic system with various parameters that can be tuned for accuracy and performance.

One of the typical steps in reducing the search space is to perform an N-Best search. This is a relatively cheap way to significantly reduce the search space, while retaining important information.

1.4 Objectives

We wanted to achieve the following objectives:

- Study the chosen components necessary for a large-vocabulary speech recogniser.
- Implement these chosen components as necessary, building a framework for future LVCSR work.
- Combine an appropriate existing language modelling implementation with these components into a functional complete LVCSR system.
- Test our system with the 1996 NIST Hub-4 Broadcast Speech evaluation.

1.5 Overview of this work

This section contains a summary of the work done in this thesis. Further detail and motivations are left for later chapters. Chapter 2 describes the theory and algorithms behind the fundamental building block for our system, namely the Hidden Markov Model (HMM). In Chapter 3 we explore the ways in which neighbouring utterances and words interact as we look at context dependency. Chapter 4 describes all the components of our final speech recognition system and how they were implemented. Our evaluation methods and results are shown in Chapter 5.

1.5.1 Hidden Markov Models

Any speech recogniser needs an acoustic modelling technique. One of the most popular is the HMM, which is a powerful statistical modelling technique. In Chapter 2 we considered HMMs in detail and studied the various algorithms associated with them. First we defined the HMM and some of

the theory necessary to understand them. The problems associated with HMMs were then examined.

We also considered various ways to use the HMM to construct word or phoneme models. For smaller vocabularies we can consider training an HMM for each word. We need enough training data for each of these words to ensure that the parameters for the associated HMMs are properly estimated. This becomes extremely difficult with larger vocabularies and we need to find a different approach to modelling words with HMMs. We considered a commonly used approach, which is to break up words into smaller building blocks (phonemes). This relatively small set of phonemes is used to construct each of the words in the vocabulary. Finding enough training data for each of these phonemes is much simpler than for word models, which is why they are preferred for larger vocabularies. We decided to use this approach for our implementation.

As the number of words in our vocabulary grows we are forced to find more efficient ways to decode our HMMs. Memory and processing requirements quickly become unmanageable for HMMs modelling large vocabularies. We considered some of the more advanced techniques such as beam and multi-level HMM segmentation, which can drastically improve performance. The beam ensures that the most unlikely paths are abandoned early on. This reduces both memory usage and processing requirements. We also considered multi-level HMM segmentation, which enables us to reduce memory usage even further. Since we know exactly at which states words end in our complete HMM we can reduce memory usage by only storing information related to those particular states.

Finally, we considered the N-Best paradigm and how it can be used to extract information more efficiently. Making use of our most expensive knowledge sources (KSs) when performing a search on this massive search space would require an enormous amount of processing power and memory. This is why the search space is often first reduced by making use of less expensive KSs. One way to achieve this is to perform an N-Best search with

simple acoustic and language models. The N-Best list is then rescored with more complex acoustic and language models. This multiple-pass approach was used in our final implementation.

1.5.2 Context Dependency

The way in which neighbouring utterances and words interact is an essential KS in speech recognition. In Chapter 3 we examined various context types that are of importance to a speech recogniser. On the speaker level these include gender, age and dialect. On the utterance level we considered context-dependent phonemes; they are used to model the way neighbouring phonemes affect one another in natural speech. One significant problem with context-dependent phonemes is that the number of phonemes grows rapidly as the context becomes more detailed. This causes the same trainability problems we found with word models. Various ways to address this issue were considered, such as decision trees.

Context between words was considered next as part of our investigation of language modelling. We examined many approaches that have been attempted in the past, such as n -grams, decision tree models and context free grammars. One problem that is common to all language modelling approach is a shortage of training data. We considered various more advanced techniques surrounding n -grams that alleviate this problem, such as Good-Turing discounting and Katz smoothing. Both these techniques result in more accurate n -gram modelling.

1.5.3 Implementation of test system

The implementation of our system was described in Chapter 4. Our first problem was segmentation of our massive HMM containing 800000 states and describing our 18000 word vocabulary. The normal Viterbi algorithm requires too much memory when the number of states becomes this large. However, such strict segmentation is not necessary in speech recognition.

Since we knew which states are important on the word level, we were able to reduce the memory requirements significantly with the implementation of our multi-level beam segmenter.

We decided on a multi-pass approach to our word spotter, since it enables us to incorporate more expensive knowledge sources gradually. We described how we expanded our multi-level beam segmenter to search for the N-Best paths so that they may be rescored later. The next challenge was to incorporate a language model into our segmenter. The SRI Language Modelling toolkit was chosen to create our n -gram language model. We described how we incorporated the ARPA format language model into our system and included it in the N-Best segmenter.

An essential component to any large-vocabulary speech recogniser is an efficient silence detection technique. We used an algorithm developed by Johan du Preez to aid our word spotter and phoneme modelling. Our approach to context-independent phoneme modelling was discussed next and we described how we used these models and a decision tree to create context-dependent models.

Finally, we explained how all the various components in our system were combined to form our complete three-phase large-vocabulary continuous speech recogniser.

1.5.4 Experimental investigation

In Chapter 5 we described our experiments and the associated results. Firstly, we tested our context-independent phoneme modelling approach. We were able to train monophones from the Hub-4 training data and achieve a monophone spotting accuracy of 42.10%. These monophones were used to initialise our context-dependent phoneme models. After we experimented with various minimum occupation counts we achieved a triphone spotting accuracy of 45.90%. This improvement demonstrates the usefulness of modelling context between phonemes.

The aim of our next experiment was to determine the effect of a beam on

phoneme spotting. For monophone spotting we found that with a properly chosen beam width you can drastically improve performance while barely affecting recognition accuracy.

Once we had our triphone models we were able to start optimising the various parameters of our word spotter experimentally. Since optimising the vast number of parameters at the same time would be nearly impossible we decided to optimise each phase individually on the development set. With our final baseline system we achieved a word spotting accuracy of 32% on the development set.

Finding long words made up of a concatenation of several phonemes is more difficult than finding words which consist of one or two phonemes. Longer words depend on a long sequence of well matched phoneme models. To compensate for this we experimented with a word length penalty. We found that a simple word length penalty improved accuracy by almost 2% for phase two and almost 1% for phase three. Our word length penalty model can be improved further, but demonstrates that word recognition accuracy can be increased with a word length penalty.

In order to put our results into context, we attempted the same development set evaluation with the well-known Sphinx 3 system. The accuracies were now no longer calculated by our own system, but by the SCLite software as required by the official Hub-4 specifications. Sphinx 3.2 found a word-error rate (WER) of 56.3%, while our system found a WER of 68.9%. We could see that our system performs significantly worse and various potential reasons for this were discussed. The greatest loss in potential accuracy was found in phase two, where the correctness dropped from nearly 75% to just over 45%.

We found similar results on the evaluation set. Sphinx 3.2 found a WER of 57.1% and our system found a WER of 68.9%. When considering these results we should remember that our system was only developed as a framework for future large-vocabulary speech recognition research and not as an improvement on professional systems.

1.6 Contributions

- During the course of this study we implemented various hypothesis search components necessary for a LVCSR system. These included:
 1. Multi-level forward and backward Viterbi beam segmenters, which drastically reduce memory requirements.
 2. Multi-level exact N-Best segmenters, one of which incorporates knowledge from a language model.
- We combined various components into a LVCSR system. This included:
 1. Audio preprocessing (Mel-frequency cepstral coefficients and linear discriminant analysis).
 2. HMMs for acoustic modelling.
 3. Trigram language modelling with backoff and smoothing.
 4. Various Viterbi searches.
- We implemented a complete system to act as a framework for future LVCSR research at the University of Stellenbosch.
- We were able to show that our newly implemented components reduce memory requirements and improve accuracy with respect to the baseline system at the University of Stellenbosch. We also verified that the multi-pass search paradigm is effective.
- Our final system was evaluated on the 1996 NIST Hub-4 Broadcast Speech evaluation and found moderately accurate recognition rates compared to Sphinx 3 when using the same phoneme set, data, lexicon and language model.

Chapter 2

Hidden Markov Models

Many modern systems make use of a powerful statistical method called the Hidden Markov Model (HMM) to characterise observed data samples of a discrete-time series as described in [5]. HMMs are a fundamental part of pattern recognition in general and has been used effectively in speech recognition by many independent parties [3, 8, 25, 53].

HMMs also form an intuitive framework for the development of concepts such as pruning, multi-level decoding and N-Best decoding.

2.1 HMM Definition

An HMM can be compared to a state machine. It contains states which are connected by transitions. Each transition has a transition weight which is a value between 0.0 and 1.0. The sum of all weights on transitions leaving a state must be 1. Some of these states are called emitting states and contain probability density functions (PDFs) describing some pattern found in the type of data being recognised. In Fig. 2.1 you can see a small example of a first-order HMM. Higher-order HMMs are also feasible, but they will not be considered in this study. For this reason any reference to an HMM in the rest of this work will be to a first-order HMM.

An HMM also has a special parameter ($\boldsymbol{\pi}$) which describes the initial

state distribution. It is a vector containing a probability for each state in the HMM which is the probability that the HMM will start in a given state. In HMMs applied to speech recognition there is mostly a single state in which the HMM state sequence will always begin. In that case all the entries in the $\boldsymbol{\pi}$ vector contains zero except the entry corresponding to the initial state, which contains a 1. The full parameters for an HMM are denoted as

$$\boldsymbol{\Phi} = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi}), \quad (2.1.1)$$

where \mathbf{A} is the matrix of transition weights and \mathbf{B} is a vector of state output probability distributions corresponding to each state, which gives a probability when given an observation.

The HMM is well suited for modelling speech since it models stationary characteristics within state output PDFs, as well as time-varying phenomena within the transition probabilities.

The following notation is used when working with HMMs:

- T - The number of observations in the observation sequence.
- N - The number of states in the HMM.
- \mathbf{S} - The underlying hidden state sequence in the HMM, $\{s_1, s_2, \dots, s_T\}$.
- \mathbf{X} - The sequence of observations, $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$.
- \mathbf{A} - is a matrix of transition weights $[a_{ij}]$, which is the weight of the transition from state i to state j for every state combination $i, j = 1, 2, \dots, T$.
- \mathbf{B} - collection of probability distributions $\{b_i(\mathbf{x}_t)\}$, which is the probability of observation \mathbf{x}_t being generated by the Markov process in state s_i .

2.2 HMM assumptions

The definition of an HMM makes two assumptions.

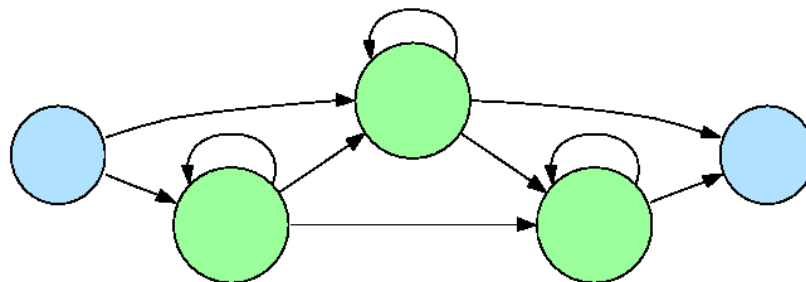


Figure 2.1: A small HMM example.

2.2.1 Markov assumption

The first-order Markov assumption is expressed as

$$P(s_t | \mathbf{x}_1^{t-1}, \mathbf{s}_1^{t-1}) = P(s_t | s_{t-1}) \quad (2.2.1)$$

and states that the probability of entering state s_t given an observation sequence \mathbf{x}_1^{t-1} and a state history \mathbf{s}_1^{t-1} is equal to the probability of entering state s_t when only the previous state is known. This defines the first-order nature of the HMM. Only a history of length 1 affects the next state probabilities.

2.2.2 Output-independence assumption

Mathematically it is expressed as

$$P(\mathbf{x}_t | \mathbf{x}_1^{t-1}, \mathbf{s}_1^t) = P(\mathbf{x}_t | s_t) \quad (2.2.2)$$

and states that the observation vector \mathbf{x}_t is independent of the past and will only be determined by the state that the HMM occupies at time t .

2.3 The HMM problem

There are three fundamental problems that need to be solved for us to use HMMs to their full potential:

1. The evaluation problem: How do we find the probability that a series of observations was generated by a given HMM?

2. The decoding problem: How do we find the most probable state sequence in the HMM that generated the series of observations?
3. The learning problem: Given a model, how do we find the parameters for the HMM so that it has a high probability of generating a given sequence of observations?

2.3.1 The evaluation problem

To solve this we need to find the probability $P(\mathbf{X}|\Phi)$ for our observation sequence \mathbf{X} , given the HMM Φ . This can be done by taking the sum of all probabilities for state sequences that generate observation sequence \mathbf{X} :

$$P(\mathbf{X}|\Phi) = \sum_{\text{each } \mathbf{S}} P(\mathbf{S}|\Phi)P(\mathbf{X}|\mathbf{S}, \Phi) \quad (2.3.1)$$

If we look at the Markov assumption stated in Section 2.2.1, we see that the probability of each state is only dependent on the previous state. This means the state sequence probability term in Eq. 2.3.1 can be rewritten as the joint probability:

$$\begin{aligned} P(\mathbf{S}|\Phi) &= P(s_1|\Phi) \prod_{t=2}^T P(s_t|s_{t-1}, \Phi) \\ &= \pi_{s_1} a_{s_1 s_2} \cdots a_{s_{T-1} s_T} \end{aligned} \quad (2.3.2)$$

If we use this same state sequence \mathbf{S} , we can use the output independence assumption (Section 2.2.2) to write the output probability along the path as:

$$\begin{aligned} P(\mathbf{X}|\mathbf{S}, \Phi) &= P(\mathbf{x}_1^T | \mathbf{s}_1^T, \Phi) \\ &= \prod_{t=1}^T P(\mathbf{x}_t | s_t, \Phi) \\ &= b_{s_1}(\mathbf{x}_1) b_{s_2}(\mathbf{x}_2) \cdots b_{s_T}(\mathbf{x}_T) \end{aligned} \quad (2.3.3)$$

We can now substitute Eq. 2.3.2 and Eq. 2.3.3 into Eq. 2.3.1 to find the likelihood of the observation sequence given the HMM as:

$$\begin{aligned} P(\mathbf{X}|\Phi) &= \sum_{\text{all } \mathbf{S}} P(\mathbf{S}|\Phi)P(\mathbf{X}|\mathbf{S}, \Phi) \\ &= \sum_{\text{all } \mathbf{S}} \pi_{s_1} b_{s_1}(\mathbf{x}_1) a_{s_1 s_2} b_{s_2}(\mathbf{x}_2) \dots a_{s_{T-1} s_T} b_{s_T}(\mathbf{x}_T) \end{aligned} \quad (2.3.4)$$

In practice this equation is equivalent to the following procedure:

1. Start in initial state s_1 with probability π_{s_1} .
2. Generate observation \mathbf{x}_1 with probability $b_{s_1}(\mathbf{x}_1)$ and move to the next state in the state sequence (s_2) with probability a_{12} .
3. Repeat step 2 until the observation in the final state in the sequence is generated.

This is equivalent to solving $\alpha_i(t)$ which is defined as follows:

$$\alpha_i(t) = P(\mathbf{x}_1^t, s_t = i | \Phi) \quad (2.3.5)$$

An efficient algorithm commonly used to find the probability that an HMM generated a sequence of observations is called the forward algorithm [5].

The forward algorithm makes use of a matrix $(\alpha_i(t))$ of size $T \times N$ to store the scores it calculates. The forward algorithm then functions as follows:

1. Initialise first column of $\alpha_i(t)$ matrix (where $t = 1$):

$$\alpha_i(1) = \pi_i b_i(\mathbf{x}_1) \text{ for } 1 \leq i \leq N$$

2. Populate rest of $\alpha_i(t)$ matrix:

$$\alpha_i(t) = \left[\sum_{j=1}^N \alpha_j(t-1) a_{ji} \right] b_i(\mathbf{x}_t) \text{ for } 2 \leq t \leq T \text{ and } 1 \leq i \leq N$$

3. Find probability $P(\mathbf{X}|\Phi)$ of HMM generating observation sequence:

$$P(\mathbf{X}|\Phi) = \sum_{i=1}^N \alpha_T(i)$$

In general when working with models that describe units of speech, we are interested in the results for one specific state. If we know that the HMM state sequence must end in a given state s_F , we can read the probability directly from the α as follows: $P(\mathbf{X}|\Phi) = \alpha_T(s_F)$.

We know that in the forward algorithm there are T observations to process. For each observation t we have to calculate the score for N states, which is found by calculating the score for each transition entering state n . If there is an average of L transitions per state, we have NL calculations for each observation in the observation sequence. Now we can see that the complexity of the forward algorithm is $O(NLT)$.

2.3.2 The decoding problem

When given a sequence of feature vectors representing observations we can find the sequence of states with the highest probability to have generated that observation sequence. This is done by means of HMM segmentation, also called decoding.

A common algorithm used to find the most probable state sequence through an HMM is the Viterbi algorithm [49]. The Viterbi algorithm is based on the forward algorithm mentioned in the previous section. The Viterbi matrix works on the same fundamental principle as the matrix used in the forward algorithm, but also finds the most likely sequence of states given the observation sequence. Instead of summing the probabilities at each time t from all sources to a destination state i , the Viterbi algorithm finds the maximum among these probabilities. The α matrix is replaced by the Viterbi score matrix $V_t(i)$ and an additional path matrix $B_t(i)$ is also required. $V_t(i)$ is defined as

$$V_t(i) = \max_{s_1^{t-1}} P(\mathbf{x}_1^t, s_1^{t-1}, s_t = i | \Phi) \quad (2.3.6)$$

and is of the same dimensions as α in the forward algorithm. The Viterbi algorithm is defined as follows:

1. Initialise first column of V and B matrices (where $t = 1$):

$$\begin{aligned} V_1(i) &= \pi_i b_i(\mathbf{x}_1) \text{ with } 1 \leq i \leq N \\ B_1(i) &= 0 \end{aligned}$$

2. Populate rest of V and B :

$$\begin{aligned} V_t(j) &= \max_{1 \leq i \leq N} [V_i(t-1) a_{ij}] b_j(\mathbf{x}_t) \text{ with } 2 \leq t \leq T \text{ and } 1 \leq j \leq N \\ B_t(j) &= \arg \max_{1 \leq i \leq N} [V_i(t-1) a_{ij}] \text{ with } 2 \leq t \leq T \text{ and } 1 \leq j \leq N \end{aligned}$$

3. Find best score V^* , and state s^* in which best path ends:

$$\begin{aligned} V^* &= \max_{1 \leq i \leq N} [V_T(i)] \\ s_T^* &= \arg \max_{1 \leq i \leq N} [V_T(i)] \end{aligned}$$

4. Backtrack to find best sequence of states:

$$\begin{aligned} s_t^* &= B_{t+1}(s_{t+1}^*) \text{ with } t = T-1, T-2, \dots, 1 \\ \mathbf{S}^* &= (s_1^*, s_2^*, \dots, s_T^*) \text{ is the best sequence} \end{aligned}$$

In practice, the Viterbi algorithm is slightly altered to solve some HMM topology issues. We introduce the concept of the null state, which does not contain a PDF. Null states are extremely useful in separating building blocks in HMMs and are handled somewhat differently to emitting states. Null states are backtracked within the same time step and included in the maximum taken in step 2 above. The introduction of null states allows the $\boldsymbol{\pi}$ vector to be incorporated in the transition weights matrix.

An example HMM is shown in Fig. 2.2. The shaded states are emitting states, while the rest are null states. Fig. 2.3 shows a possible path matrix $B_t(j)$ resulting from the Viterbi algorithm applied to this HMM and eight observations. Note how the paths from null states remain within the same time step.

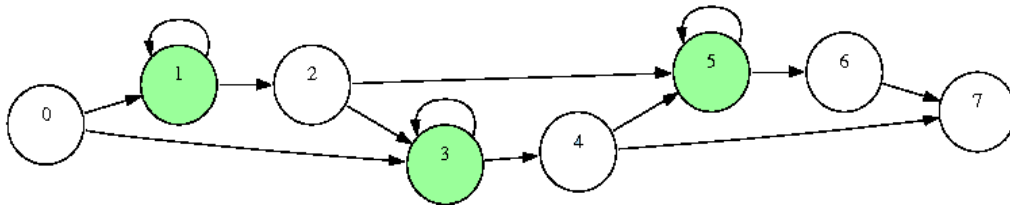


Figure 2.2: HMM used to illustrate use of path matrix. In the multi-level example, null states 2, 4 and 6 are considered word endings (super states).

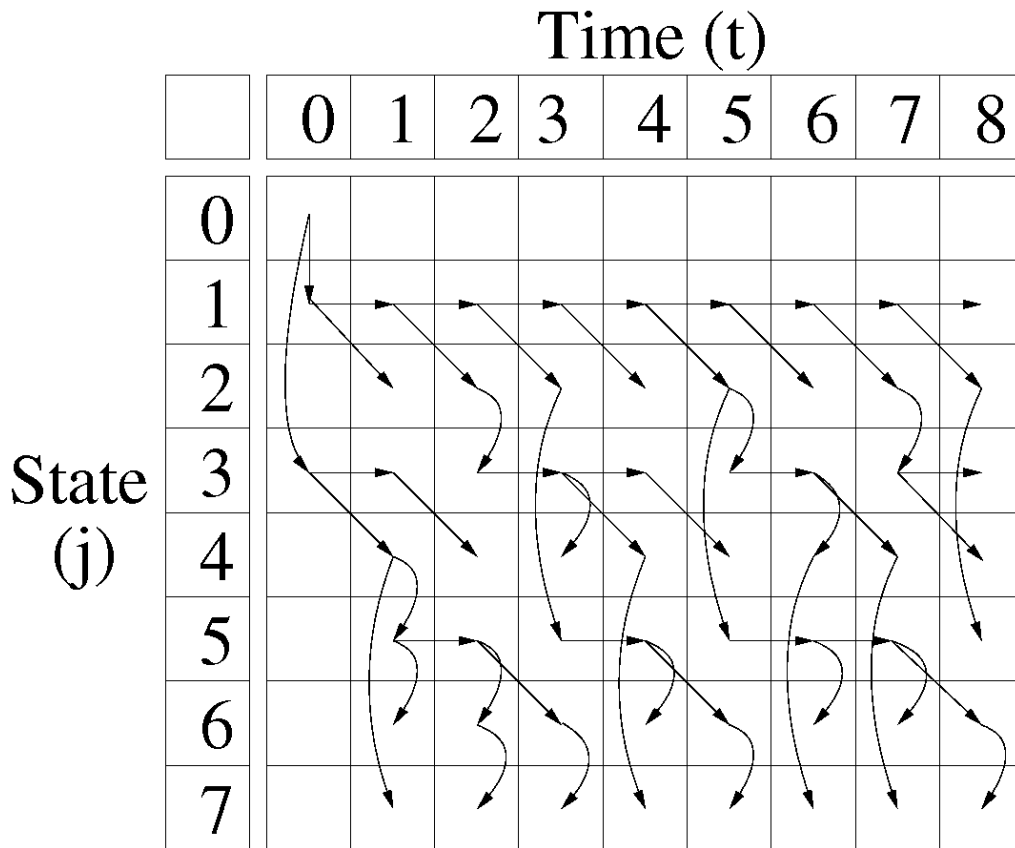


Figure 2.3: A possible path matrix $B_t(j)$ produced by the Viterbi algorithm applied to the HMM in Fig. 2.2.

According to this algorithm description, backtracking is done from the highest scoring state at the final time $t = T$ in the Viterbi path matrix. In our speech recognition applications this is not the case. We are only interested in the best path from the final state in the HMM, which intuitively is

also where the unit of speech ends. Similarly to the forward algorithm, the Viterbi algorithm does all computation in a time-synchronous fashion from $t = 1$ to $t = T$. Thus it can also be shown to have a complexity of $O(N^2T)$.

The HMM in Fig. 2.1 contains 5 states. When HMMs contain many states, this segmentation process can become very expensive in memory. This is especially true when the full state sequence is required from the segmenter. In the case of our Hub-4 HMM, the HMM contained more than 800000 states.

Logarithms are generally used to represent scores, which changes products to sums and prevents numerical underflow when working with small numbers. Scores typically range from $-\infty$ to ∞ .

There are many useful variations of the Viterbi algorithm, such as the backward Viterbi algorithm. It functions exactly like the forward Viterbi algorithm, except for a few differences:

1. Segmentation starts at the final time in the final state.
2. Transitions are reversed. For example, a transition from state 3 to state 5 with a probability of 0.5 would become a transition from state 5 to state 3 with the same probability.
3. The input audio is processed in reverse.

This algorithm is functionally equivalent to the forward Viterbi. The proof and formal derivation of the algorithm can be found in [12].

2.3.3 The learning problem

To solve this problem, we need to find the best parameters for an HMM given a collection of training data. This training data consists of observations and their corresponding symbol from the output alphabet. The method we used for all our training is the iterative forward-backward algorithm, also known as the Baum-Welch algorithm. More information on this algorithm can be found in [5].

2.4 HMM Types

We now have all the tools necessary to train and use HMMs. Now we need to decide on the HMM configuration we will use, which is derived from studying the problem. We need to choose HMM topology and state output probability distributions that meet the requirements of the problem.

2.4.1 HMM Topology

The HMM topology describes the graphical structure of the HMM, which corresponds to the transitions with non-zero weights. Since speech recognition is temporal in nature we need to choose a topology that best describes this. The most popular HMM topology for speech recognition is called the left-to-right topology and was first proposed by R. Baker [4]. An example of a four state left-to-right HMM is shown in Fig. 2.4.

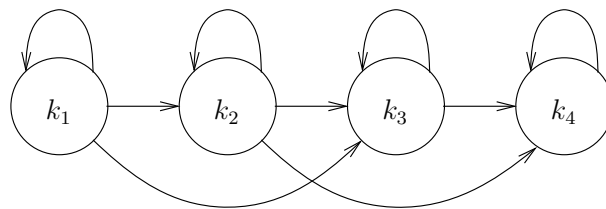


Figure 2.4: An example of a four state left-to-right HMM.

This left-to-right topology effectively models events that follow sequentially over time such as basic speech sounds. Segmentation will always start in the first state and end in the final state. The state index will also grow monotonically since a previous state can never be reached a second time. This simplifies all processing of the HMM since all transition weights to a previous state index will be zero.

2.4.2 State output probability distributions

To model the local characteristics in a speech signal we need to use an appropriate probability distribution. This distribution must be general enough to allow for the variations associated with spontaneous speech. It must also be specific enough to distinguish between the various phonemes.

The most widely used distribution is the Gaussian distribution, since it accurately describes quantities resulting from many small independent random effects that create the quantity of interest. More importantly, it is computationally simple. For these reasons it can be used in a large variety of fields. A general one-dimensional Gaussian distribution with mean a_x and variance σ_x^2 can be seen in Fig. 2.5. In speech recognition we need to model highly detailed information and therefore use high-dimensional multivariate Gaussian distributions. They take the form

$$b_j(\mathbf{x}_t) = (2\pi)^{-\frac{D}{2}} |\boldsymbol{\Sigma}_j|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu}_j)' \boldsymbol{\Sigma}_j^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_j)}, \quad (2.4.1)$$

where D is the dimensionality of the data, $\boldsymbol{\mu}_j$ is the mean vector and $\boldsymbol{\Sigma}_j$ is the covariance matrix of the Gaussian distribution associated with state j . The parameters for such a single Gaussian distribution can be estimated in a single pass using maximum likelihood estimation [22].

Depending on where these Gaussians will be applied we may use one of two common variations. The first is the full-covariance Gaussian distribution, which has a detailed covariance matrix. This model is highly detailed but needs a relatively large amount of data to estimate the large number of parameters. The second is the diagonal-covariance Gaussian distribution. In such a Gaussian distribution all the values in the covariance matrix are zero except the diagonal entries. This results in less parameters to train and less training data necessary to estimate these parameters properly. The problem is that it assumes that individual components in the observation vector are statistically independent.

A single Gaussian distribution represents a region in feature space associated with a specific sound. This does not accurately model speech, since

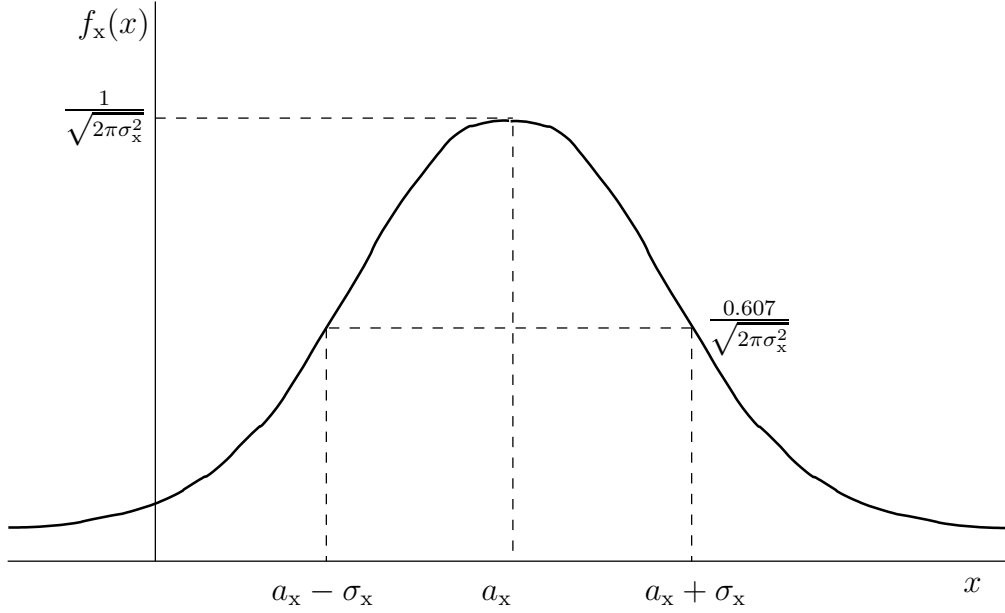


Figure 2.5: A general Gaussian distribution.

there will rarely be only one such a region per sound. The mixture Gaussian distribution attempts to model this collection of regions more accurately by using a weighted sum of Gaussian distributions. For M mixtures the mixture Gaussian distribution for state j has the form

$$b_j(\mathbf{x}_t) = \sum_{m=1}^M c_{jm} b_{jm}(\mathbf{x}_t),$$

where

$$b_{jm}(\mathbf{x}_t) = (2\pi)^{-\frac{D}{2}} |\boldsymbol{\Sigma}_{mj}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu}_{mj})' \boldsymbol{\Sigma}_{mj}^{-1} (\mathbf{x}_t - \boldsymbol{\mu}_{mj})},$$

$$\sum_{m=1}^M c_{jm} = 1$$

and

$$c_{jm} \geq 0 \text{ for all } 1 \leq j \leq N.$$

A common algorithm that determines the parameters of the model based on data is the Expectation-Maximisation (EM) algorithm [15]. This needs multiple iterations as opposed to the single iteration for the single Gaussian distribution.

2.5 HMM Applications

When attempting to model speech with HMMs, there are a few things we need to consider. We already mentioned processing requirements and memory usage, but the problem of modelling the speech remains. Many options are available to us, but we will only consider a few:

1. Create and train an HMM for each word in the vocabulary
2. Break up words into smaller general building blocks

2.5.1 Whole word models

In this approach we need a model for each word in the vocabulary. This works well for small vocabularies since we do not have many models between which to distinguish. It is not very difficult to gather enough training data to estimate parameters for each word and storing these parameters is also feasible. This approach was successfully used on many smaller vocabulary speech recognition tasks, especially isolated word recognition [51]. Isolated word recognition is significantly easier than continuous speech recognition, since we already know that there was only one word spoken per utterance. The only problem is to determine which word was the most likely to have been uttered in the given observation sequence. To determine this we calculate $P(\mathbf{X}|W_x)$ for each word x in the vocabulary. The highest scoring word is the most likely to have been uttered.

An example of the HMM structure used in isolated word recognition is shown in Fig. 2.6. $W_1 \dots W_N$ corresponds to each word model, which itself is an HMM with a chosen topology. $P_{W_1} \dots P_{W_N}$ are corresponding weights, which specify how probable a word occurrence is. If we have no knowledge of how probable words are (no unigram grammar), we can define all these weights to be the same. We find the highest scoring word by finding the most likely state sequence through this model given an observation sequence. Once we know through which of the parallel word models that

state sequence goes, we know the highest scoring model and also which word was most likely spoken. The isolated word recogniser can easily be expanded to be a continuous speech recogniser if we add a feedback loop to the HMM as in Fig. 2.7.

With continuous speech recognition we do not have the luxury of knowing that only one word was spoken. We need to find the correct sequence of words, which makes this task extremely difficult. Similar to isolated word recognition, we find the most likely state sequence through the model in Fig. 2.7. The feedback loop has the effect of concatenating words, because you can enter any word at practically any time from 1 to T . However, this makes the search much more difficult, since we have to check each possible combination of words. The search space for larger vocabularies becomes vast.

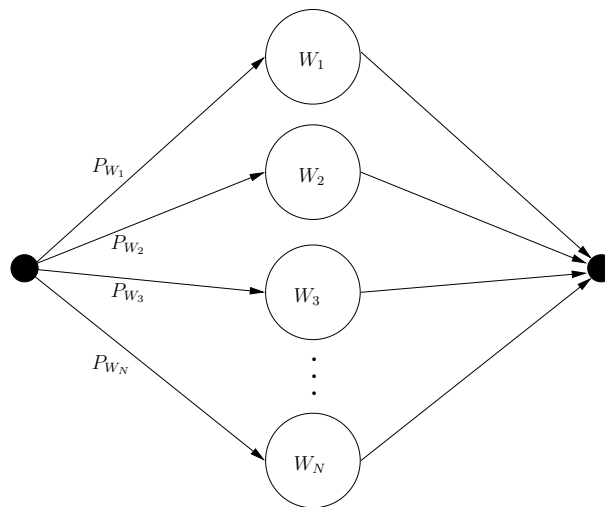


Figure 2.6: HMM for an N -word classifier for isolated word recognition.

Large vocabularies (1000 words or more) make this approach ineffective for various reasons. We typically do not have enough training data to estimate proper parameters for each word model. If another word is added to the vocabulary, we would need to gather more data from speakers, which is an expensive process. For models to work effectively they need as much

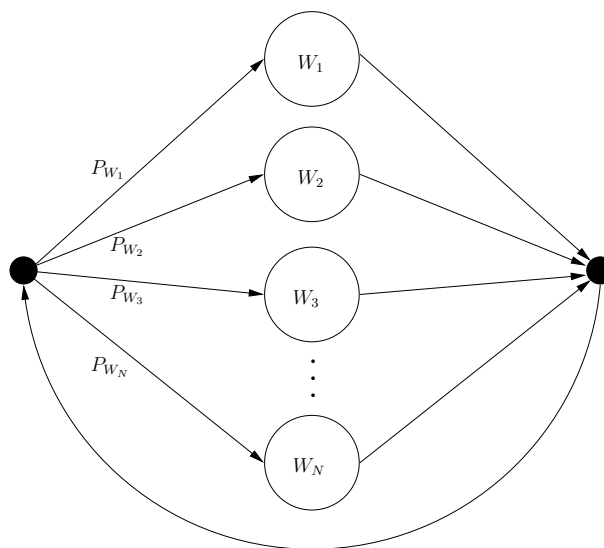


Figure 2.7: HMM for an N -word spotter for continuous speech recognition. The addition of the transition from the final state to the first state (feedback loop) enables the HMM to recognise a sequence of words instead of just isolated words.

training data as possible to avoid poor performance. Another difficulty is the storage of these parameters. When there are so many complex models there can be a vast amount of data, which will negatively affect memory use and processing time.

2.5.2 Phoneme models

For larger vocabularies we need to find a better approach to modelling words than the word model approach. A commonly used technique is to break up words into smaller speech units, called phonemes [25]. In speech recognition applications the most commonly used topology is the left-to-right HMM. This is intuitive due to the sequential nature of speech. An example of this topology can be seen in Fig. 2.1, which is an example of an HMM that represents a phoneme without any context.

A phoneme is a small unit of speech, ideally with a very general pronunciation. We define an alphabet of phonemes and construct each of the words in the vocabulary from a combination of them. By doing this we create a

pronunciation lexicon for our current vocabulary, which is a list of words and their matching pronunciation. We create word models by sequentially linking these phoneme models according to the lexicon. This is illustrated in Fig. 2.8, which is the HMM for a word (W_x) where p_1 , p_2 and p_3 are phoneme models. In this example the word contains three phonemes, but the number of phonemes depends on the length of the word.

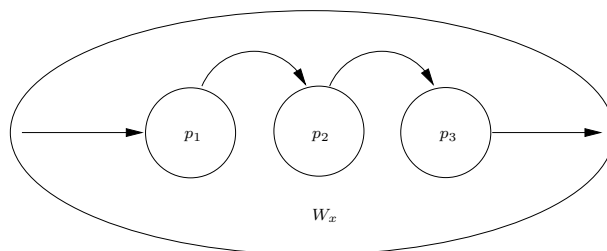


Figure 2.8: HMM for a word model constructed from three phonemes.

Once we have a word model for each of the words in the vocabulary, we can begin to find our initial boundaries. We now iterate through each of the training audio files and use their associated transcription to create a model that describes the entire sentence. This is done by sequentially linking the word models according to each transcription. We then use this large HMM and find the most likely state sequence given the audio file, which also describes the most likely time boundaries for each word and each phoneme. This is called forced alignment [18]. Now we can use these boundaries to train new phoneme models with this abundance of training data.

After such a training iteration we can use the newly trained models to do another forced alignment to find new boundaries that are likely to be more accurate. This accuracy converges to a local maximum within a small number of iterations.

There are various ways to model phonemes, but in this work we will focus mainly on monophones and triphones. Monophones are independent of left and right context. This makes them easy to train since there are very

few instances of them. In the monophone set used in this work there were 42 phonemes. The problem is that monophones do not effectively model effects between consecutive phones, while triphones take both left and right context into consideration. Unfortunately there are many possible combinations of three consecutive monophones and this results in many models. With our monophone set of 42 phonemes there are around 74088 (42^3) triphones. This number can be greatly reduced however, since many triphones do not occur in natural speech. Chapter 3 further explores context dependency and shows how we can use a classification and regression tree (CART) to reduce the number of parameters in the HMMs. Other configurations such as diphones and larger contexts can also be considered, but will not be examined in this work.

2.6 Evaluating HMMs

Determining the accuracy of an isolated word recogniser is relatively easy, since it is just a simple ratio:

$$\text{ACC}_{\text{isolated}} = \frac{k}{n},$$

where n is the number of words tested and k is the number of correctly classified words.

Finding the accuracy of a continuous word recogniser is significantly more difficult, since we cannot simply regard imperfect classifications as incorrect. We use a generally accepted system for measuring the accuracies. Continuous speech recognition accuracy is affected by several factors:

1. Deletion: A correct word is omitted in the recognised word sequence.
2. Insertion: A word is added into the recognised sequence.
3. Substitution: An incorrect word is substituted for a correct word.

In practice we determine these errors by aligning the correct word sequence and the recognition result with dynamic programming. In essence

we match all the correct words possible from both word sequences and then determine deletions, insertions and substitutions from the difference. Once this is done we can calculate the final word error rate (WER) as follows:

$$\text{WER} = \frac{\text{SUB} + \text{DEL} + \text{INS}}{\text{NUM}} \quad (2.6.1)$$

where NUM is the number of words in the correct sequence. The final accuracy is then determined as:

$$\text{ACC}_{\text{continuous}} = 1 - \text{WER} \quad (2.6.2)$$

It is important to note that the denominator used to determine the final accuracy is not used when calculating insertions and deletions. In other words, the final accuracy is not necessarily equal to the insertion ratio subtracted from the correctness ratio.

2.7 More HMM algorithms and optimisations

When the number of states in an HMM becomes larger we have to consider sacrificing accuracy for performance. The number of transitions that need to be considered is no longer practical. To optimise the processing of such HMMs we can consider leaving out the less probable paths and only focusing on those that are most feasible. This introduces the concept of a beam.

In some large-vocabulary speech recognition applications the full state sequence is not important. We are more interested in the words that were formed than we are in the sequence of phonemes or sub-phones. It is possible to find a state sequence in terms of a specified set of states. In the normal decoding case this set of states contains all the states in the HMM, but we can reduce this set and optimise decoding. This introduces the concept of multi-level segmentation.

2.7.1 Beam segmentation

At each time interval in the observation sequence there is a score greater than $-\infty$ at each state entry that is possible to reach. In order to determine

which transitions to expand next from a given state we check the score at the current time first. If the score is minus infinity there is no reason to calculate the forward probability since it will inevitably also be minus infinity.

We can extend this to only evaluate paths with scores above a certain threshold. This is called a beam and eliminates unlikely paths early. This may have the side-effect of eliminating the correct path when the beam is too narrow. In other words, accuracy is sacrificed to increase processing speed. Memory usage is also reduced by leaving some parts of the path matrix unpopulated which would have been used without the beam.

Let us look at a small example to illustrate the effect of the beam. If we are segmenting a fully connected HMM with 5 states, we might have a range of scores as illustrated in Fig. 2.9. The arrows indicate all the expansions that need to be made. Each state can go to each other state, which means we have 5×5 expansions.

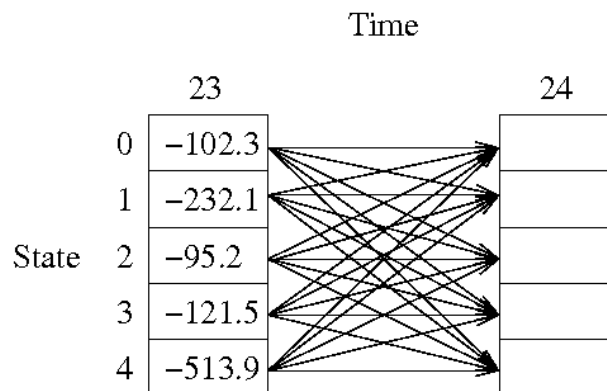


Figure 2.9: Viterbi expansion without beam, where each possible expansion is made.

On the other hand, if we were using a beam on the same example we could have the situation in Fig. 2.10. For this figure we applied a beam of width e^{10} . The best score was found (-95.2) and the beam was subtracted, giving us a threshold of -105.2 . Now instead of expanding all states, we only expand states with scores higher than the threshold. This means we

only expand states 0 and 2, resulting in a mere 10 expansions. The reduction here is significant. This approach would reduce expansions most when the best path is clearly distinguished from other paths. Intuitively this makes sense, since more attention will be given to segments of speech with greater ambiguity.

It can be argued that we might abandon a path that would have become the most likely global path. In practice this does occur, but the loss in accuracy is greatly compensated for by the reduction in computational requirements. An experiment showing the practical implications of the beam is done in Section 5.2.

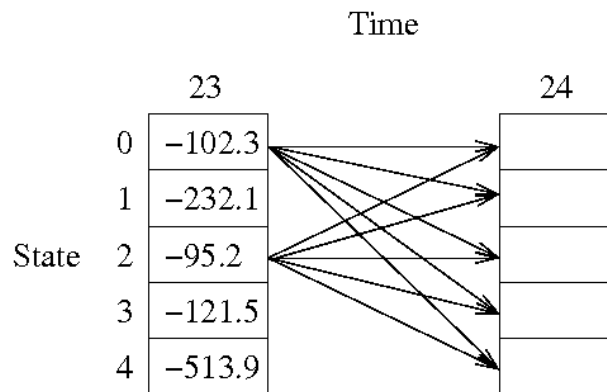


Figure 2.10: Viterbi expansion with beam, where only the higher scores are expanded.

2.7.2 Multi-level HMM segmentation

The basic Viterbi HMM segmentation as described in Section 2.3 gives a path on a state level through the entire HMM. This is a long sequence of states which is converted into a shorter list of phonemes and an even shorter list of words. For a large-vocabulary recogniser, phoneme level transcriptions are not important. We are only interested in the most probable sequence of words given the sequence of acoustic observations. If we

can avoid finding the entire path through the states a great deal of memory can be freed.

Depending on the application there are certain states in a given HMM that are known to be of importance. For example, in speech recognition we are commonly only interested in the states where words end. If we are able to find a state sequence in terms of these end states we can deduce the complete word sequence. Let us call these states “super states”. We reduce memory usage by not storing all the information during segmentation, but only storing information pertaining to these super states.

In the general case where the full sequence of states is found, we have at least one state for each observation vector. It is possible to have more than one state per observation since not all states are emitting. Now if we are able to specify a set of states in the HMM which are important we do not need to store the full path matrix as is done with the normal Viterbi decoding. For example, we only need to know the states in the HMM where word endings are found when the HMM is used to find a word sequence. If this technique is used we can use a small collection of buffers to represent the entire path matrix and only extract the important information from them. This is explained further in Section 4.1.

2.7.3 N-Best paradigm

If we had a computer with limitless processing power and memory, we could create an HMM recogniser that includes all our most expensive knowledge sources at the same time. We could include our most complex language model, acoustic models and search algorithm. This would be the ideal case, but not feasible to implement for a large-vocabulary system.

For this reason the common multi-pass approach might be more reasonable, where our least expensive and most discriminating knowledge sources are incorporated first. Later we can make use of our more expensive knowledge sources, because our search space is being progressively reduced. For the N-Best paradigm as shown in Fig. 2.11, the N-Best list allows us to

make use of expensive rescoring.

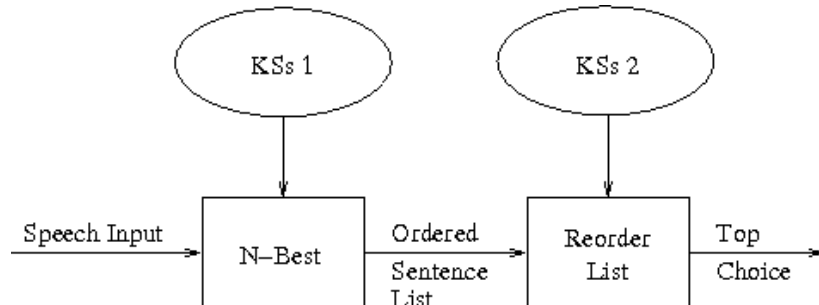


Figure 2.11: The N-Best Paradigm. Inexpensive knowledge sources (KSs 1) are incorporated early, while the remaining and more expensive knowledge sources (KSs 2) are used to generate the final hypothesis [43].

There are however some concerns with this type of multi-pass approach. One concern is that real-time performance can never be achieved, since no matter how fast the first pass is, successive passes cannot begin until the user has finished speaking. Fortunately, this delay can be kept to a minimum with optimised multi-pass algorithms.

Another problem is that the hypotheses generated by this multi-pass search strategy are not strictly correct. We are systematically removing word sequences, which means we might get rid of the optimal word sequence too early. Interestingly, this problem is shared with one-pass systems, where pruning is often necessary to achieve good performance.

The first multi-pass approach that comes to mind is generating more than one hypothesis so that they can be re-evaluated later. This is where N-Best search comes in. It has been shown to be effective in many systems [37, 53, 50].

There are things we need to keep in mind when considering N-Best lists:

- If the N-Best list that is generated does not contain the correct word sequence, later rescoring obviously has no hope of generating the correct sequence.

- In general N-Best lists, the different word sequences will differ by only one word, since similar word sequences using the same acoustic models and language model will have similar scores.
- The number of hypotheses that would be needed to form a list with every possible hypothesis grows exponentially with the utterance length. Longer utterances have more possible word combinations.

There are many different approaches to generating N-Best hypotheses.

2.7.3.1 Word Lattice

The simplest and most intuitive approach to the N-Best algorithm is a simple word lattice [17]. It consists of a collection of words and their associated end times (or begin times). By finding all possible word permutations we can compile an N-Best list, but the lattice is a much more efficient representation. For example, suppose we have an utterance that contains 5 words, but the system is unsure which of 2 words is the best for each word position. A word lattice would then contain 10 words, while an N-Best list would contain $2^5 = 32$ sentences. In Table 2.1 you can see an example of a 5-Best list that was generated by our segmenter. The same word sequences represented in a word lattice is shown in Fig. 2.12.

1. and why is by parts of chip in both
2. and why is by parts the chip in both
3. and why is by parts of chip in told
4. and why is by parts the chip in told
5. and why is by parks the chip in both

Table 2.1: An example of a 5-Best list.

and	why	is	by	parts	of	chip	in	both
				parks	the			told

Figure 2.12: An example of a 5-Best list represented as a lattice.

The word lattice needs to be distinguished from a word graph, in which words are connected explicitly and the temporal constraints are embedded.

2.7.3.2 Exact N-Best

Generating N-Best sentence hypotheses from an HMM can be done relatively efficiently. The efficient N-Best algorithm for Viterbi search was first introduced by Schwartz and Chow [42]. It functions by maintaining a separate record for each separate word history, up to a maximum of N histories. When two or more paths arrive at the same time and state with the same word history, they are merged; otherwise, a new entry is created. Since each word sequence probability is stored separately, an accurate score can be found for any word sequence hypothesis that reaches the end of the sentence.

The algorithm can be inefficient when N becomes large and the search space is large. Normally the complexity of the algorithm is $O(N)$, but when other knowledge sources are incorporated (such as a word lattice) we can improve performance. This is the most accurate and thus preferable algorithm to use when an N-Best search is performed.

2.7.3.3 Traceback-based N-Best

The traceback-based N-Best algorithm [46] is a very simple extension of the normal Viterbi algorithm to find N-Best hypotheses. For all within-word transitions it functions exactly the same way as the Viterbi algorithm. However, for each word ending state at each frame t , all the different preceding words and their corresponding scores are stored in a traceback list. When the search is complete we can search through the stored traceback list to get all the possible permutations of word sequences and the scores corresponding to them.

The advantage of this algorithm is that there is almost no computational overhead above the normal Viterbi algorithm. The disadvantage is that low cost word sequences can easily be lost since there can only be one word

history at a time for each word ending state. This is illustrated in Fig. 2.13, where three words (W_1 , W_2 and W_3) are segmented. Let us assume that the score for word sequence $W_2 - W_3$ is slightly worse than for word sequence $W_1 - W_3$, but still within the allowable beam. Since W_3 has different start times for history W_1 and W_2 , the slightly less likely (but still viable) second best path would be lost and cannot be recovered. If they had coincidentally started at the same time, the post-processing traceback would discover both words W_1 and W_2 . However, this can clearly not be relied on.

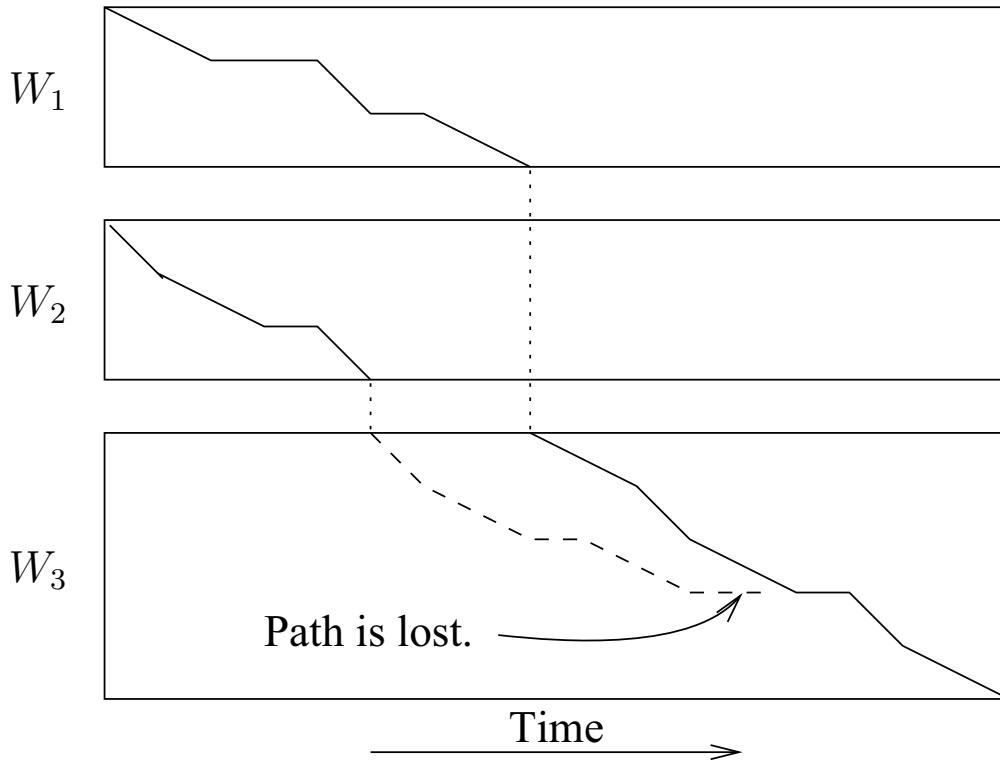


Figure 2.13: The traceback-based N-Best deficiency. Paths with different histories cannot be distinguished [43].

2.7.3.4 Word-dependent N-Best

The word-dependent N-Best algorithm [43] alleviates the deficiency of traceback-based N-Best. It provides a middle ground between maintaining complete

word histories as with the exact N-Best algorithm, and the traceback-based N-Best algorithm, which does not distinguish between histories. The word-dependent N-Best algorithm maintains more than one history per word ending, but only the immediately preceding word is taken into account as shown in Fig. 2.14. This is called the word-dependent assumption and ensures that we can remember both paths when the paths through words W_1 and W_2 begin to overlap within word W_3 .

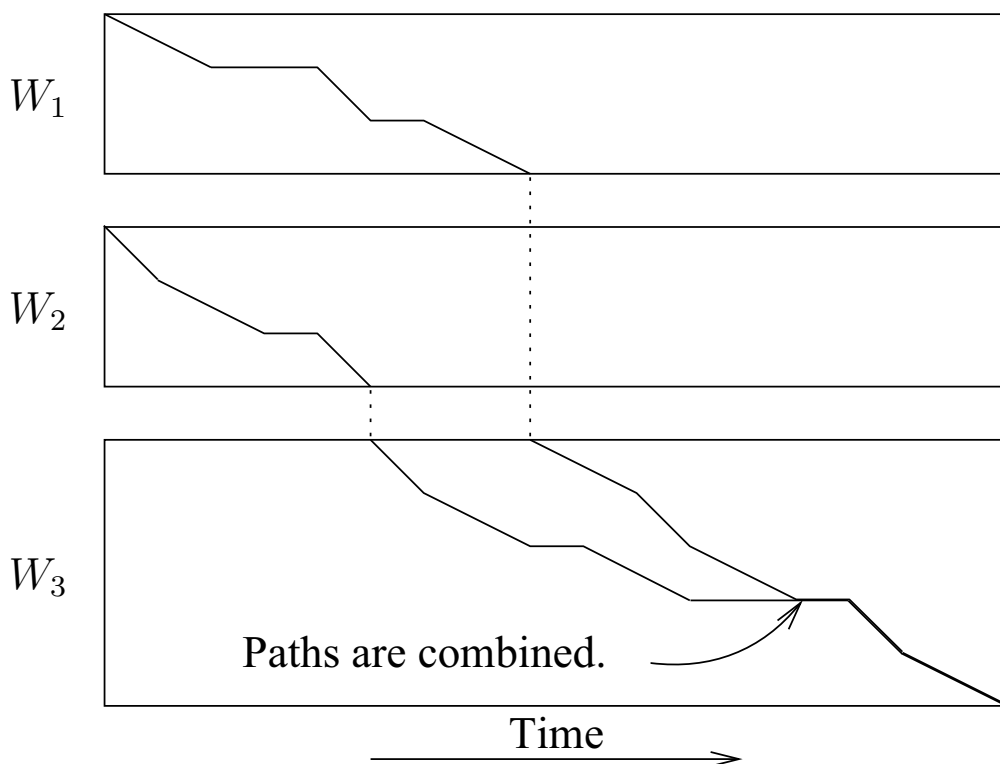


Figure 2.14: The word-dependent N-Best algorithm. Paths with different preceding words are combined [43].

Similar to the traceback N-Best algorithm, a traceback must be performed at the end of the search to find the N-Best hypotheses.

2.8 Summary

To model speech properly, we need a technique that is both accurate and versatile. In this chapter, we defined the HMM and considered their more important aspects. HMMs are ideally suited to temporal modelling, making them a good choice for our speech recognition task. It is also the most commonly used approach. HMMs are used to form all our acoustic models and all our hypothesis search techniques rely on the details in this chapter.

For us to create and use HMMs as acoustic models we need to understand the fundamental HMM problems. The basic Markov assumptions along with the evaluation and decoding problems were considered first. The evaluation problem is solved with the forward algorithm, while the decoding problem is solved with the famous Viterbi algorithm. We used a standard training technique to solve the learning problem, namely the Baum-Welch algorithm. These algorithms created the framework for all our later implementations.

The types of HMMs and their applications were considered next, along with how they are applied to continuous and isolated word recognition. Several fundamental HMM topologies were also studied. For our continuous word recognition system we need to use the continuous word recognition approach. Due to the size of our vocabulary the phoneme modelling approach seems best.

Finally, we considered some of the more advanced HMM algorithms such as beam segmentation and multi-level HMM segmentation. These optimisations are crucial in solving our problem, since the memory and processing requirements would otherwise be immense. The multi-pass approach to speech recognition and the N-Best paradigm are also considered, along with various N-Best algorithms. These techniques reduce the search space even further by permitting us to incorporate our more expensive knowledge sources gradually.

We decided to use a variation of the traceback-based N-Best algorithm, since it is simpler and more efficient than the other algorithms. The word-

dependent N-Best algorithm was also considered, but the experiments performed by Schwartz and Austin [43] show only a slight improvement in accuracy over the traceback-based algorithm.

Chapter 3

Context dependency

The subject of context dependency has been thoroughly studied throughout the years [17, 32]. In order for us to accurately model speech we need to make use of as much knowledge as possible. Context is a great knowledge source and it has been shown that creating context-dependent models will increase accuracy [31].

Language modelling is another valuable knowledge source that takes cross-word effects into account. Many studies have been done in the past on successfully modelling the deep structure that governs natural speech [7, 41].

3.1 Context types

In general we can consider acoustic contexts such as the speaker age, gender, dialect and speaking style. Unfortunately when this is done the resulting system is no longer speaker independent. There are speaker adaptation techniques [29] that can include such information in a speaker independent system, but they will not be considered in this thesis. We will only consider local contexts that occur within an utterance and linguistic phenomena that govern the way in which sentences are constructed.

Cross-word contexts on the acoustic level are very difficult to use since

the number of models will grow rapidly with vocabulary size. Another problem is the fact that the decoding process will become impossible because of the size of the search space. Each word will have a different context depending on the preceding and following words, resulting in a seemingly colossal search space when the vocabulary size grows beyond a few hundred words. For these reasons much of the focus in the past has been on context-dependent phoneme modelling [8, 25], while using language models for cross-word contexts.

3.2 Context-dependent phoneme modelling

3.2.1 Trainability

Modelling co-articulatory effects such as the triphones described in Section 2.5.2 would result in an extensive collection of models. Finding enough examples for each context would prove to be a difficult task.

We would expect to have a more accurate recogniser if we have more complex models, but this is only the case if we can reliably estimate parameters for these models. We need to find a balance between a few general densities and many specific densities. Some contexts are very common and many example utterances can be found in a given training corpus. However, there will often be contexts that do not appear in the training corpus at all. We therefore need to estimate reliable parameters for rare and unseen contexts in some way.

One way to solve this problem is to use a less context-specific model when no properly trained model is available. This is called backing off [39]. For example, we can use a bigram if a properly estimated trigram is not available. However, if we back off from a large number of triphones, we would not be taking full advantage of the context dependency we are trying to model.

The more common alternative is to make use of parameter sharing or state tying [34]. In Fig. 3.1 the ideal configuration is illustrated, where each

state in the system has a unique PDF associated with it. This configuration is ideal, but not realistic since we typically do not have enough training data. If there are many parameters in a system as is the case with triphones, we need an impossible amount of training data to train that many densities.

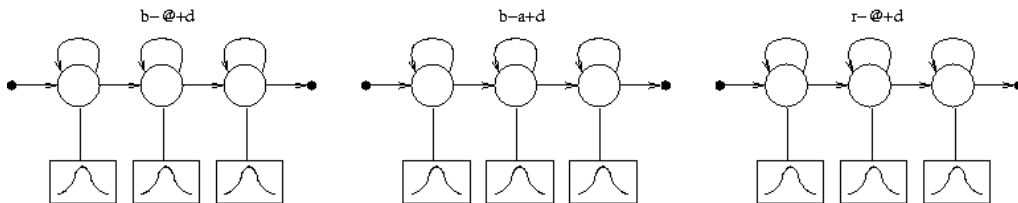


Figure 3.1: HMMs not sharing PDFs, where many PDFs need to be trained and a large amount of training data is required.

A more realistic case is illustrated in Fig. 3.2 where there is a constant number of distributions and all possible states are connected to one of them. This is called parameter sharing or state tying. In effect we are keeping the specific nature of each triphone, but the underlying densities are more general. Each triphone HMM still retains a unique transition weights matrix, but its densities are chosen from a density codebook. A technique commonly used to determine which PDFs to tie is discussed in the next section.

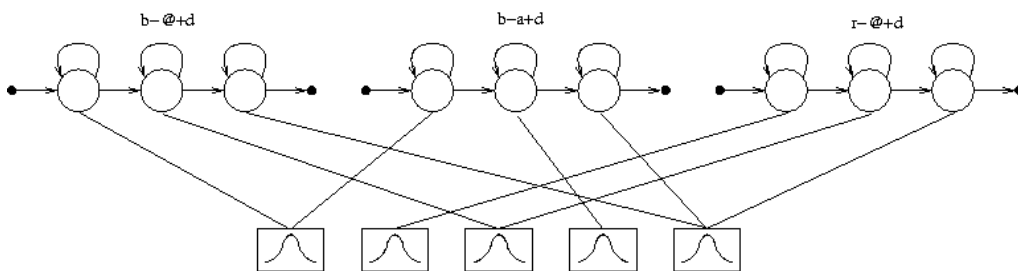


Figure 3.2: HMMs sharing PDFs, where fewer PDFs need to be trained resulting in less data being necessary for them to be properly trained.

3.2.2 Decision Trees

Classification and regression trees (CART) [6] are representations that interpret and predict the structure of a set of data in an intuitive way. By simply attaching a question to each split in the decision tree we can quickly classify an unseen data point into one of the classes found in the training data. In Fig. 3.3 we see a simplified version of a tree that determines the risk for cancer for an individual whose gender, age and smoking status is known. Normally such trees are created by hand after studying a limited number of samples. A CART enables us to automatically construct such trees by making use of a data set and a collection of questions.

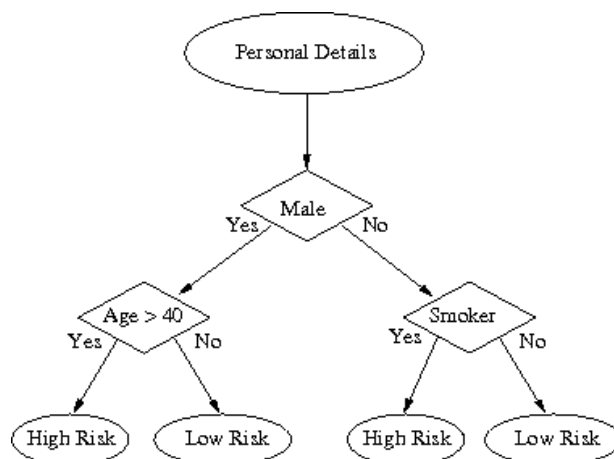


Figure 3.3: A simplified example of a decision tree that determines risk for cancer. Males over the age of 40 and female smokers have the highest risk.

We can apply decision trees to speech recognition by using it to cluster acoustically similar sounds. We construct a tree that groups densities according to their context and state in the HMM. An unseen context and state can then be classified into one of the clusters formed when the decision tree was constructed. We can do this for each context-dependent HMM that occurs and build the required HMM.

The two most important tasks when constructing a CART is choosing a proper question set and determining an appropriate splitting criterion.

The splitting criterion specifies how we decide which question best splits the data.

3.2.2.1 Constructing a decision tree

A top-down sequential optimisation procedure [32] is used to construct the tree. All the training data samples are first placed at the root node of the tree. In the case of a binary decision tree, the root node is then split into two by finding the question that divides the data in the node most effectively. Normally the question that maximises entropy reduction [17] is chosen for the split. However, when we have continuous densities as is the case with continuous HMMs, we need to find other criterium.

The data from the root node is now divided according to the question chosen and placed into leaf nodes. The same process finds the appropriate question for each leaf node until no further splits can be made. A leaf node may not be split if one or more of the following conditions are met:

1. All the data points in a node belong to the same class and no further splits can be made.
2. A minimum occupation count (MOC) is normally maintained throughout all the nodes of the tree to ensure that the tree does not become too specific. If a split results in a node containing fewer data points than the MOC, the split would not be made.
3. The best split among the questions in the question set for the node according to the chosen splitting criterion falls below a chosen threshold η . In other words, $\max_q[\Delta_{DL}^{(q)}(S)] < \eta$, where $\Delta_{DL}^{(q)}(S)$ is the splitting criterion of node S given question q .

3.2.2.2 Question set

The first problem is to construct a question set that distinguishes between the basic pronunciation types. This will allow us to logically cluster contexts and make use of state tying.

Let us assume we have data in the following format:

$$\mathbf{x} = (x_1, x_2, \dots, x_d), \quad (3.2.1)$$

where x_i is a discrete or continuous variable. It is possible to construct a set of questions that would ask all the necessary questions. This question set is also known as the *standard set* of questions Q . We compile it as follows:

1. Each question concerns the value of only one of the variables in the data. These questions are called *singleton* or *simple* questions.
2. If x_i is discrete and can assume values from the set $\{c_1, c_2, \dots, c_k\}$, Q contains all questions in the form: $\{\text{Is } x_i \in S?\}$ where S can be any subset of $\{c_1, c_2, \dots, c_k\}$
3. If x_i is a continuous variable, Q includes all the questions in the form: $\{\text{Is } x_i \leq c?\}$ for $c \in (-\infty, \infty)$

The questions generated by condition 3 above appear to be an infinite set. Fortunately this is not the case, since there is only a finite set of training data. We only use the distinct splits which appear in the training data.

In practice we can reduce the number of questions even further. Since we have an understanding of the problem, we can select a question set which would create easily classifiable clusters. For example, in speech recognition we can select subsets such as stop sounds and fricatives for the question set. It does not matter how large our question set is, because of the data-driven nature of the tree growing process. Each question is asked at each decision point and the question that best splits the data is selected.

3.2.2.3 Splitting criterium

Typically we would evaluate the entropy reduction of each question and select the question with the highest entropy reduction for the question.

Unfortunately there is no simple entropy measurement for continuous distributions such as Gaussians so we need to use a likelihood gain splitting criterion.

There are various splitting criterium that can be used for continuous distributions, but we used the Minimum Description Length (MDL) criterion as described in [44]. When splitting node S with question q into nodes S_{q+} and S_{q-} , the change in model description length (DL) is

$$\Delta_{\text{DL}}^{(q)}(S) = \frac{1}{2} \left\{ \Gamma(S_{q+}) \log |\Sigma(S_{q+})| + \Gamma(S_{q-}) \log |\Sigma(S_{q-})| - \Gamma(S) \log |\Sigma(S)| \right\} + K \log \Gamma(S_0), \quad (3.2.2)$$

where the accumulated state occupancy of each node is $\Gamma(*)$, the dimensionality of the feature vector is K , $\Sigma(*)$ is the covariance matrix of each node and S_0 denotes the root node of the decision tree.

We can rewrite Eq. 3.2.2 as

$$\Delta_{\text{DL}}^{(q)}(S) = \Delta_{\text{Likelihood}}^{(q)}(S) + \text{Threshold}, \quad (3.2.3)$$

where

$$\Delta_{\text{Likelihood}}^{(q)}(S) = \frac{1}{2} \left\{ \Gamma(S_{q+}) \log |\Sigma(S_{q+})| + \Gamma(S_{q-}) \log |\Sigma(S_{q-})| - \Gamma(S) \log |\Sigma(S)| \right\} \quad (3.2.4)$$

$$\text{Threshold} = K \log \Gamma(S_0). \quad (3.2.5)$$

3.3 Language Modelling

3.3.1 Definition

Statistical Language Models are an attempt to model the underlying linguistic phenomena found in speech and the way in which sentences are constructed.

Let \mathbf{W} be a sequence of words from a given vocabulary of a fixed size, given by

$$\mathbf{W} = w_1, w_2, \dots, w_n. \quad (3.3.1)$$

$P(\mathbf{W}|\mathbf{X})$ is the probability that the word sequence W was spoken if the sequence of acoustic features \mathbf{X} was observed. If all the words were equally important, we would find the most likely word string $\hat{\mathbf{W}}$ by finding the highest probability for $P(\mathbf{W}|\mathbf{X})$. This would mean that we would need to consider every possible word sequence:

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) \quad (3.3.2)$$

This would prove to be an almost impossible task, but luckily we can use a language model to reduce the search space. If we rewrite Eq. 3.3.2 using Bayes' formula:

$$P(\mathbf{W}|\mathbf{X}) = \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(\mathbf{X})} \quad (3.3.3)$$

Now $P(\mathbf{W})$ is the probability that the word sequence \mathbf{W} is spoken and $P(\mathbf{X}|\mathbf{W})$ is the probability that we will observe the acoustic data X when \mathbf{W} is spoken. $P(\mathbf{X})$ is the average probability that \mathbf{X} will be observed. This probability remains constant throughout the search, so we only need to find the maximum value for $P(\mathbf{W})P(\mathbf{X}|\mathbf{W})$. This means we can rewrite Eq. 3.3.2 as

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{X}|\mathbf{W}). \quad (3.3.4)$$

The $P(\mathbf{W})$ term is where the language model becomes useful. Without a language model this probability is equal for each \mathbf{W} , since all word sequences are equally probable. This is not the case in real life applications, because we know that certain sentences are very likely to be uttered and some are not possible or extremely improbable. Knowledge about possible sentences helps us to eliminate improbable sentences and reduce the number of hypotheses that need to be considered.

A language model can be used to reduce word-error rates by reordering an N-best list that was generated by a recogniser. This will not improve recognition speed, but only improve accuracy. For the language model to have a significant effect we would need a large value for N (typically greater than 100) to increase the probability that the list contains the correct answer.

Another approach is to have a language model that constantly guides the recogniser and reduces the search space. It has been shown that this improves recognition accuracy and speed when a well trained language model is used [16, 25, 26]. At each decision point (at the end of each word, for example) the recogniser uses the current history and the language model to find $P(w_{t+1}|w_1 \dots w_t)$. In effect the recogniser uses the language model knowledge source to constantly reduce the search space.

Most commonly both these approaches are employed and the recogniser uses a simple grammar during recognition and a more complex grammar to reorder the N-best list that was generated [48].

3.3.2 Techniques

Many ways to solve the $P(\mathbf{W})$ term have been explored, but we will only mention the most noteworthy among them. All approaches share the need for a large training database of word sequences from which the model is constructed.

3.3.2.1 n -grams

n -grams are by far the most commonly used and intuitive approach. n -grams model language as a Markov source of order $n-1$ ($P(w_{t+1}|w_{t-(n-1)}, \dots, w_t)$). Most commonly $n = 3$ is used when a large training corpus (millions of words) is available, but otherwise $n = 2$ is normal. Even with a large training corpora there is still a shortage of data. It is not enough to simply use maximum likelihood estimation from counts for the n -gram probabilities. This was proven to be the case when Rosenfeld [41] took an extremely large training corpora from newspaper articles (38 million words) and observed all trigrams. A third of all the new articles from the same source revealed unseen trigrams and the vast majority of the observed trigrams occurred only once.

There are many ways to combat data scarcity that have already been explored. Smoothing techniques such as Good-Turing discounting [13] are

used to assign some non-zero probability to any n -gram, even if it was never seen in the training data. Backing off to lower order n -grams in various ways such as Katz smoothing [21] also performs well.

- **Good-Turing discounting:**

In order to get a more accurate picture of the probabilities associated with infrequent n -grams, we require a smoothing technique. In essence the n -grams are divided into groups depending on their frequency of occurrence in the training data in order to smooth the parameter according to n -gram frequency.

The Good-Turing estimate tells us that any n -gram that occurs c times, should be treated as an n -gram that occurs c^* times:

$$c^* = (c + 1) \frac{N_{c+1}}{N_c} \quad (3.3.5)$$

where N_c is the number of n -grams that occur c times in the data. The probability for an n -gram a with c counts is now:

$$P(a) = \frac{c^*}{N} \quad (3.3.6)$$

It is important to notice that $N = \sum_{c=0}^{\infty} c^* N_c$, which is equal to the original number of counts in the distribution.

- **Katz Smoothing:**

Katz smoothing is an extension of the idea behind Good-Turing discounting and combines the higher-order models with lower-order models. Katz smoothing extends the Good-Turing estimate for nonzero bigram counts in the following way:

$$R^*(w_{i-1}w_i) = \begin{cases} d_c c & \text{if } c > 0 \\ \alpha(w_{i-1})P(w_i) & \text{if } c = 0 \end{cases}$$

where R^* is the estimated probability for word sequence $w_{i-1}w_i$, c is the number of times that n -gram $w_{i-1}w_i$ occurred and $d_c \approx \frac{c^*}{c}$.

Nonzero bigram counts are discounted with the ratio d_c and zero bigram counts are assigned a value which is a combination of an equalisation factor and the lower-order distribution (the unigram in this case). The factor $\alpha(w_{i-1})$ ensures that the total number of counts in the distribution remains the same. In other words $\sum_{w_i} R^*(w_{i-1}w_i) = \sum_{w_i} R(w_{i-1}w_i)$ for each n -gram order. $\alpha(w_{i-1})$ is calculated so that:

$$\begin{aligned} \alpha(w_{i-1}) &= \frac{1 - \sum_{w_i: R(w_{i-1}w_i) > 0} P^*(w_i|w_{i-1})}{\sum_{w_i: R(w_{i-1}w_i) = 0} P(w_i)} \\ &= \frac{1 - \sum_{w_i: R(w_{i-1}w_i) > 0} P^*(w_i|w_{i-1})}{1 - \sum_{w_i: R(w_{i-1}w_i) > 0} P(w_i)} \end{aligned} \quad (3.3.7)$$

where $P^*(w_i|w_{i-1})$ calculated from the corrected count:

$$P^*(w_i|w_{i-1}) = \frac{R^*(w_{i-1}w_i)}{\sum_{w_k} R^*(w_{i-1}w_k)} \quad (3.3.8)$$

Typically we assume that large counts are reliable and therefore they are not discounted. In other words $d_c = 1$ where $c > k$ for some k . The discount ratio for the lower counts is derived from the Good-Turing estimate as applied to the global bigram distribution. In other words N_c in Eq. 3.3.5 is equal to the total number of bigrams occurring c times in the training corpus. The d_c for $c \leq k$ are chosen to ensure that:

1. The discounts are proportional to the Good-Turing discounts.
2. The total number of counts from the global bigram distribution that are discounted is equal to the total number of counts assigned to bigrams with zero Good-Turing counts.

The first constraint is equivalent to:

$$d_c = \mu \frac{c^*}{c} \quad (3.3.9)$$

for $c \in \{1, \dots, k\}$ with some constant μ . According to the Good-Turing estimate the total mass that will be assigned to bigrams with zero counts is $N_0 \frac{N_1}{N_0} = N_1$, which brings us to the second constraint:

$$\sum_{c=1}^k N_c(1 - d_c)c = N_1 \quad (3.3.10)$$

From this we can conclude:

$$d_c = \frac{\frac{c^*}{c} - \frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \quad (3.3.11)$$

The higher-order n -gram models are derived recursively. The Katz n -gram backoff model is defined in terms of the $(n-1)$ -gram model and so forth. In summary, the Katz probabilities for bigrams are defined as follows:

$$P_{\text{Katz}}(w_i|w_{i-1}) = \begin{cases} C(w_{i-1}w_i)/C(w_{i-1}) & \text{if } c > k \\ d_c C(w_{i-1}w_i)/C(w_{i-1}) & \text{if } k \geq c > 0 \\ \alpha(w_{i-1})P(w_i) & \text{if } c = 0 \end{cases}$$

In essence this system results in maximum knowledge usage throughout the LM. The higher-order n -grams make use of information from the lower-order n -grams wherever insufficient data is available.

3.3.2.2 Decision tree models

A decision-tree-based model breaks up the different possible word histories from the training data according to arbitrarily chosen binary questions about the history at each decision point. The history space is however extremely large and applying decision trees to language modelling is extremely difficult for that reason.

Trees are grown by selecting the most informative question at each node depending on the one with the highest reduction in entropy. At each node there is a yes/no question relating to the words already spoken and at each leaf there is a probability distribution over the allowed vocabulary. There

were a few attempts at constructing such a decision-tree-based language model, such as the one by Bahl *et al.* [2]. The language model they constructed took many months to train and predicted the next word spoken from the preceding 20 words. They only found a minor reduction in perplexity of 4% when using the tree-based language model compared to the baseline trigram. Interestingly, they found a greater reduction in perplexity with a combined model than with either model on its own. From this they concluded that the most effective use of such a tree-based model is in conjunction with a trigram and not as a replacement for it.

3.3.2.3 Context-free grammar

All language models are based on a human understanding of speech to some degree, but the linguistic content in most resulting models is minor. There are some language models that are directly based on language as we understand it, such as the context free grammar.

The context free grammar (CFG) is defined by a vocabulary, a set of non-terminal symbols and a set of production rules. Sentences are generated by starting with an initial non-terminal and the production rules are repeatedly applied until a sentence of terminal symbols from the vocabulary remains. There have been attempts at creating a CFG based on parsed and annotated corpora, such as the CFG created by Marcus *et al.* [28]. This model resulted in good coverage of unseen data.

The normal CFG can be expanded by adding probability distributions to the transitions from each non-terminal. These probabilities can be estimated using algorithms based on the EM algorithm. It was found that these probability surfaces contain many local maxima which are inferior to the global maxima. This makes it ineffective to use the EM algorithm and there is no known efficient algorithm to find the global maximum.

3.4 Summary

In this chapter we considered the importance of context dependency in speech. We explored how modelling context dependency can improve accuracy and considered several context types such as cross-word and co-articulatory contexts.

Next we explored the importance of isolating the most important acoustic contexts and merging the less important contexts with them, because there are many more theoretical contexts than those that appear in practice. For this reason we have insufficient training data and must find an intelligent way to generalise some contexts.

This problem was addressed by describing decision trees and their algorithms. Decision trees are an intuitive representation of a dataset. All the data points in the set are distributed between the leaves of the tree with questions attached to each decision point. We considered how they are constructed and ideally suited for speech recognition applications.

For any large-vocabulary recogniser to be efficient we need to take all knowledge sources into account. It is essential to consider the way in which sentences are constructed in spontaneous speech. We defined the language modelling problem and considered some advances in this field. We then considered some of the most popular language modelling techniques.

We examined n -gram modelling and some of the more advanced expansions associated with it, such as Good-Turing discounting. n -grams are a very intuitive way to model language, but require an immense amount of training data. Little or no configuration needs to be done by hand, which is what prevents us from using CFGs and various other techniques for LVCSR. More exotic techniques such as decision tree models and CFGs are also considered. After comparing various techniques we decided to use n -grams, since they are an effective way to create an accurate language model without supervision. An efficient implementation for training n -grams also already existed, namely the SRI Language modelling toolkit [47].

Chapter 4

Implementation

The entire implementation was done in C++ and each component was fully coded and tested individually. Several pieces of software that were used in our experiments were already implemented and tested in our baseline system:

1. HMM training and representation
2. Signal processing
3. CART training
4. Language model training

In order for us to construct our system we needed to implement various new components including:

1. Multi-level HMM segmenters (forward and backward Viterbi).
2. N-Best segmenters (with and without language model).
3. Integration of the SRI language model with our system.
4. Three-phase word spotter combining all the necessary components.

All components were implemented with reusability in mind. Each component can be replaced or improved individually with minimal effort.

4.1 Multi-level Beam HMM segmenter

In the pattern recognition software system of our DSP department (PatrecII) the Viterbi beam segmenter was already implemented and reduced calculations by eliminating transitions when the score drops below a certain threshold, as described in Section 2.7.1. This proves very effective even with a relatively wide beam, because many unlikely paths are eliminated early.

The idea behind our multi-level implementation is to replace the full path matrix from the normal Viterbi segmenter with reduced path and time matrices.

Both the reduced path matrix and the reduced time matrix have a size of $T \times \hat{N}$, where T is the length of the observation sequence and \hat{N} is the number of super states, usually the states where words end in the HMM. A list of super states is given to the segmenter before segmentation starts. By doing so it only records the super states in the solution path. We will illustrate this by means of a small example.

Say we are given the HMM in Fig. 2.2. In each state the first number denotes the state number and for the sake of simplicity we assume that each state in the HMM is emitting. If we are also given a sequence of arbitrary acoustic observations resulting in the most probable state sequence $\{0, 1, 1, 1, 2, 5, 5, 6, 7\}$, we would have the path matrix in Fig. 4.1. The arrows indicate how backtracking takes place.

The blank entries in the table represent values that are unimportant to us. These entries will contain values, but since we are only interested in the path that ends in the final state (state 7 at time 8) we can ignore entries that are not on the backtracking route. The backtracking is done as follows:

1. Look at the entry in the last state at the last time. The entry is 7, so we know the final entry in the sequence; state 7. (Path = {7})
2. Follow entry to time $T - 1$, which places us in line 6. (Path = {6, 7})
3. Continue following back to the start, state 0, adding the lines recorded

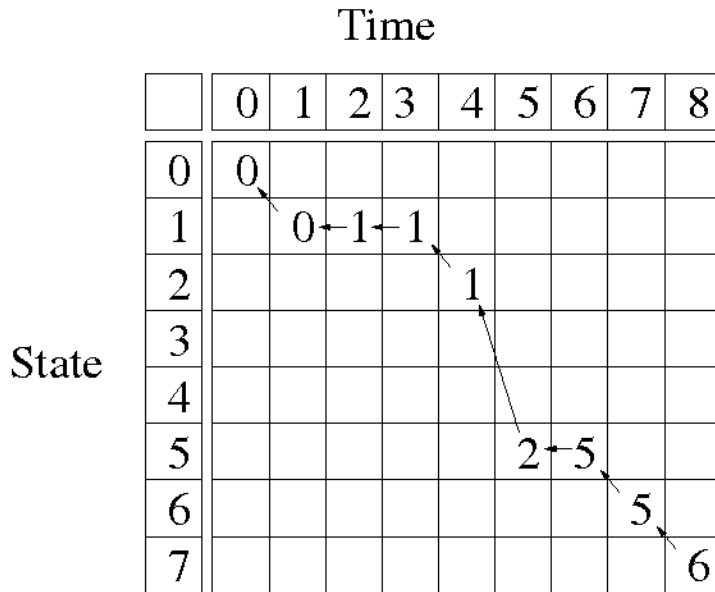


Figure 4.1: The full path matrix from a small example. The arrows indicate how backtracking takes place once the matrix is fully populated.

to the front of the path sequence.

After segmentation, the most probable route through the HMM with the given acoustic observations would have been determined to be $\{0, 1, 1, 1, 2, 5, 5, 6, 7\}$. To find this complete path we need a large matrix to contain the various paths that were found such as the one seen in Fig. 4.1. This matrix is of size $T \times N$, where T is the length of the observation sequence and N is the number of states in the HMM. The matrix is populated using the Viterbi algorithm as described in Section 2.3.2.

Now what if we didn't need the full route? Let's say that state 2, 4 and 6 were super states. This means that we only need to know when the route passes through state 2, 4 or 6. We can immediately see from the full state sequence that the super state sequence would be $\{2, 6\}$. As mentioned earlier we reduced this full path matrix to two, much smaller matrices by processing the HMM exactly as before, but only storing super state information. To illustrate, we show the reduced matrices after segmenting the HMM in Fig. 2.2 with the same observation sequence. The results are shown

in Fig. 4.2, which represents the super path and super time matrices. The first entry in each pair is stored in the super path matrix and the second entry in the super time matrix. The arrows illustrate how the sequence of super states is extracted from these reduced matrices. Once again the blank entries are not shown for the sake of simplicity.

		Time								
		0	1	2	3	4	5	6	7	8
State	2							-1		
	4							-1		
	6									2 6

Figure 4.2: The super path and time matrix from a small example. The arrows indicate how backtracking takes place once the matrices are fully populated.

These two matrices may not seem to be much smaller than the original path matrix, but on larger examples the memory usage reduces significantly. For the original Viterbi algorithm, we need to store a full path matrix of size $T \times N$, where T is the length of the observation sequence and N is the number of states in the HMM. We do not need to store the entire score matrix, but can replace it with 2 buffers:

1. Buffer to keep scores for current time t
2. Buffer to keep scores for next time $t + 1$

The score information is only important when deciding which states to expand for each frame. When expansion of all necessary states at a certain frame is completed we will have no further need for these scores. This is why we can reuse the score buffers. For the multi-level segmentation we only need to store the reduced matrices as described earlier and replace the

full matrices with buffers. The reduced matrix in Fig. 4.2 was formed by using the following buffers:

1. Score buffer of size N to keep scores for current time t
2. Score buffer of size N to keep scores for next time $t + 1$
3. Path buffer of size N to keep paths for current time t
4. Path buffer of size N to keep paths for next time $t + 1$
5. Time buffer of size N to keep time references for current time t
6. Time buffer of size N to keep time references for next time $t + 1$

The score buffers function the same way as with the normal Viterbi segmentation. The path buffers simulate the complete path matrix that would have been stored normally. References to the previous time at which a super state was encountered are propagated by means of the time buffers in parallel with the path entries. Expansions occur slightly differently to the normal Viterbi segmentation. If we expand from one of the super states, it is recorded and passed forward along with the time at which it was found. This will enable us to backtrack to it directly. If we expand from one of the other states, we simply continue passing the previously found super state and time. In both cases the score is calculated and propagated exactly as in the Viterbi algorithm.

After the buffers are expanded fully for a certain observation, the necessary information is extracted from them and inserted into the super path and super time matrices. The buffers are then swapped and used again fully without any extra memory being allocated. This reusability is possible because all the necessary information can be passed along as the buffers are swapped. The advantage is that the buffers are used repeatedly and a full-size matrix is not necessary. We only need to store the greatly reduced path and time matrices.

Segmentation takes place normally until the first super state is encountered, leaving the reduced path matrix with invalid entries. In our example, the first super state is encountered at time 4 where state 2 is found. From there on, the time where it was found and the super state number is passed along between the buffers, until time 7 where another super state (state 6) is found. The new time where this super state was found along with its state number is then passed along until the next super state is found and so forth. Note that this distinction between super states and normal states also enables us to apply different beam widths and maximise efficiency.

The highest scoring super state at the final time is recorded when segmentation is complete, so that it may be used in the backtracking. In our example this is state 6, so the backtracking starts there. Immediately, we recognise the final super state in the sequence, state 6, so it is added to the super path that will be returned. We also know which entry we will be backtracking to next from the path matrix at that entry (state 6, time 8). This is illustrated in Fig. 4.2. The time value is then extracted from the same entry in the time matrix. In this case the time value would be 6, so the next entry to be inspected is state 2, time 6. Now we know the next super state (state 2) and add it to the front of the backtracking path, giving us the path $\{2, 6\}$. In the entry at state 2, time 6, we read a negative state from the reduced path matrix, telling us that there are no more super states to which to backtrack. Note that we can use the frame number to calculate the time at which a super state was found.

It is clear that the result for the super backtracking is exactly the same as finding the super states in the original backtracking. The only difference is that instead of allocating memory for a $T \times N$ path matrix we now only allocate memory for two smaller matrices. Memory is only allocated for two $T \times \hat{N}$ matrices, where T is length of the observation sequence and \hat{N} is the number of super states.

If we choose super states to only be at the end of words we would have an extreme reduction in memory usage. For example, if we have an HMM

with 1000 states and we are processing an acoustic observation sequence of length 20, we would need to allocate memory for $20 \times 1000 = 20000$ entries. If this model contains 50 phoneme spellings (words), there would be 50 super states. For the multi-level HMM segmentation you would only require $20 \times 50 \times 2 = 2000$ entries. Given these improvements we would expect to see a reduction in memory usage by a factor of about $\beta = 2\frac{s}{n}$, where s is the number of super states and n is the total number of states in the HMM. In the case of our Hub-4 evaluation there were about 18000 words and 800000 states in the HMM that were segmented. This results in $\beta = 0.045$, which equates to a major reduction in memory usage.

4.2 N-Best segmenter

We implemented our own N-Best Viterbi beam segmenter and attempted a variation of the traceback-based N-Best approach described in Section 2.7.3.3, where N-Best paths are found during Viterbi segmentation. This is an expansion of the normal Viterbi segmenter with some alterations to the Viterbi matrix and the algorithm used. The entire structure is also based on the multi-level segmentation technique described in Section 4.1, which enables us to reduce memory requirements significantly since we are not interested in exact state sequences. Similarly to the multi-level segmentation technique, our N-Best segmenter also requires knowledge about important states in the HMM being segmented. Two state lists are required:

1. A list of begin states which contains all the states of the HMM in which a word can begin.
2. A list of end states which contains all the states of the HMM in which a word can end.

These states are used to tell the segmenter which states in the HMM are important. The begin states are simply used as place holders to connect

end states to each other. The end state information is the only outcome that is really meaningful to the user.

Instead of only determining the best end state sequence as was the case with the normal multi-level segmenter, we need to find the N-Best end state sequences, where N is an arbitrary number. To do this we need to expand the entries in the reduced path matrix to contain N backtracking sources instead of just one. When backtracking occurs, we would also need to know more than just the super state number and the time to backtrack to, since each of the entries now contains more than one entry. For this reason we add the index to each backtracking source to indicate the offset in the N-List we are interested in. Now each entry in the reduced path matrix contains a list of size N of the following backtracking information:

- State number
- Time
- Route index
- Current score

Fig. 4.3 contains an example of the typical setup required for the N-Best segmenter. The super states contain N-Best lists, while the other states contain only 1-best list as with normal segmentation. This is because for the same sequence of features, the score difference will be the same for the same sequence of models. For this reason we can do a normal 1-best segmentation within words. When an end state is encountered in the segmentation process the current within-word score difference is added to each of the scores contained in the begin state list associated with the end state found.

This newly updated list forms the entry in the end state. This is illustrated in Fig. 4.4, which shows an arbitrary example with $N = 3$. The N-Best entries are shown as (n, t, i, d) , where n is the state number, t is the time, i is the index in the backtracked N-Best list and d is the score.

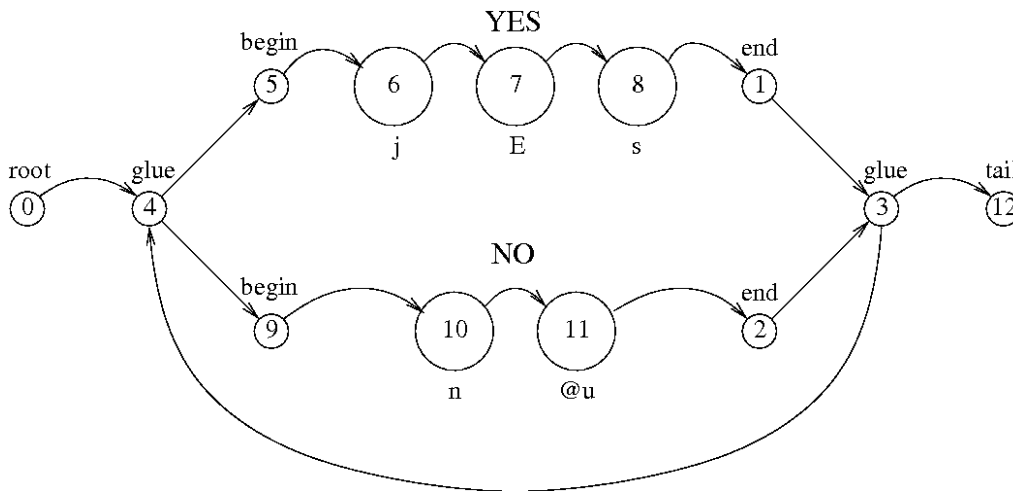


Figure 4.3: Example of HMM used for two-word N-Best spotter. The symbol under each state indicates which phoneme is represented by that particular state. A list of the phonemes used in our implementation can be found in App. B.

States 33 and 39 are end states. The best score from end state 33 is used to initiate the 1-best segmentation. Once a 1-best state is propagated to an end state we use the score difference (16 in this case) to form the new N-best list. Note that begin states are omitted in this figure, since they are only needed when forming the end state lists. Let us consider the 3-Best list contained in state 33 at time 40. Each entry in the list backtracks to a different word at a different time. Since each entry contains a list, we need the index to indicate the exact backtracking source.

To ensure that the end state information reaches the begin states we define a new type of super state called the “glue state”. They are states that connect end states to begin states. Each glue state contains an N-Best list similar to the other super states. States 3 and 4 in Fig. 4.3 are examples of glue states.

To facilitate the final backtracking, we also need to remember the last begin state time and state number so that we may extract the previous end state information from it. This begin state list contains the same information as the preceding end state, which was projected through the glue states into the begin state. Begin states contain the exact same information

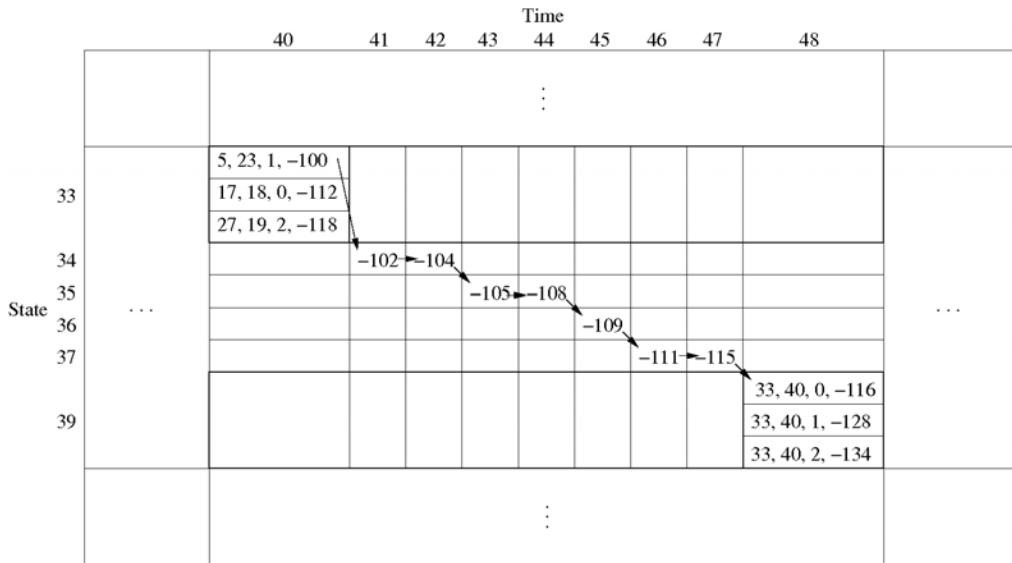


Figure 4.4: N-best relying on 1-best segmentation. Super states such as states 33 and 39 are shown in the same matrix as non-super states for the sake of this illustration. The arrows show the 1-best path.

as the associated end state, except for the scores which differ as mentioned above. End state list entries are formed by combining two things:

1. The state number, time and index of the previous end state, which was propagated to subsequent begin states and in turn the end state concerned.
2. The score found by adding the 1-best score difference to each entry in the begin state associated with the end state concerned.

The group of super states now includes glue states, begin states and end states and all contain N-best lists. In an HMM configuration such as seen in Fig. 4.3 the glue states can be found recursively if the begin and end states are known. The complete algorithm is defined as follows:

1. Initialise root state (also an end state) with one entry (state 0, time 0, index 0, score 0) at time $t = 0$, which will be used as an end point for the final backtracking.

2. Expand end and glue states at time t by merging lists contained in them with those in their destination and applying the HMM transition weights as is normally done.
3. Form begin state 1-bests at time t by extracting only the highest scoring entry from the associated N-Best list.
4. Expand non-super states at time t as with normal 1-best segmentation by first expanding the null states to time t and then adding the PDF score to the emitting states and expanding them to time $t + 1$.
5. Expand null non-super states at time $t + 1$.
6. Form end state N-Best lists at time $t + 1$ by copying the N-Best list from the begin state associated with the end state and adding the score difference found within the word to each entry.

Repeat steps 2 to 6 for time $t = 1 \dots T$ where T is the number of observations.

Once segmentation is complete, we find the final N-best list by merging all the N-best lists contained in end states at time $t = T$ with each other. Each entry in this list represents one of the N-best paths.

The root state is the first state in the HMM and the tail state is the final state. For example, in Fig. 4.3, state 0 is the root state and state 12 is the tail state. There can only be one of each in this algorithm.

We will now illustrate the use of this algorithm by applying it to the example in Fig. 4.3. Please note that states 6, 7, 8, 10 and 11 are simplified emitting states. In practice they would themselves contain the HMM that models the phoneme represented by that state. They would also need a number of non-emitting states to connect them into a whole, which is why we need step 5. It ensures that all the results from emitting states reach the appropriate end states, from where the higher-level N-best lists can take over. For this example we will express the entries in N-best lists as 4-tuples (state, time, index, score). Also, transition weights are 1.0 between

all states for the sake of simplicity. Blank entries in the lattices do not necessarily mean the absence of a value, but that they do not assist in the illustration of the example.

Step 1: This creates an N-Best list containing 1 entry $(0, 0, 0, 0.0)$, which is used as a marker to end the final backtracking. This can be seen in Fig. 4.5 at super state entry 0 (root) and time 0.

		Time	
		0	1
Super state	0 (root)	$(0, 0, 0, 0.0)$ —	—
	1 (end_yes)	—	—
	2 (end_no)	—	—
	3 (glue)	—	—
	4 (glue)	$(0, 0, 0, 0.0)$ —	—
	5 (begin_yes)	$(0, 0, 0, 0.0)$ —	—
	9 (begin_no)	$(0, 0, 0, 0.0)$ —	—
	12 (tail)	—	—

Figure 4.5: Illustration of steps 1 and 2 in the N-Best segmentation algorithm. Step 1 forms the marker entry at state 0 and time 0, while step 2 expands that entry to the glue states and in turn the begin states.

Step 2: We can see the expansion of the list at root state 0 to glue state 4 and from there to begin states 5 and 9 at time $t = 0$ in Fig. 4.5. At each time in the segmentation we will form end state lists for time $t + 1$ in step 6. These lists are expanded in this step to form begin state lists.

Step 3: The best score from each begin state (state 5 and state 9), which is 0.0 in both cases, is used to populate the 1-best lattice at the associated states. This can be seen in Fig. 4.6.

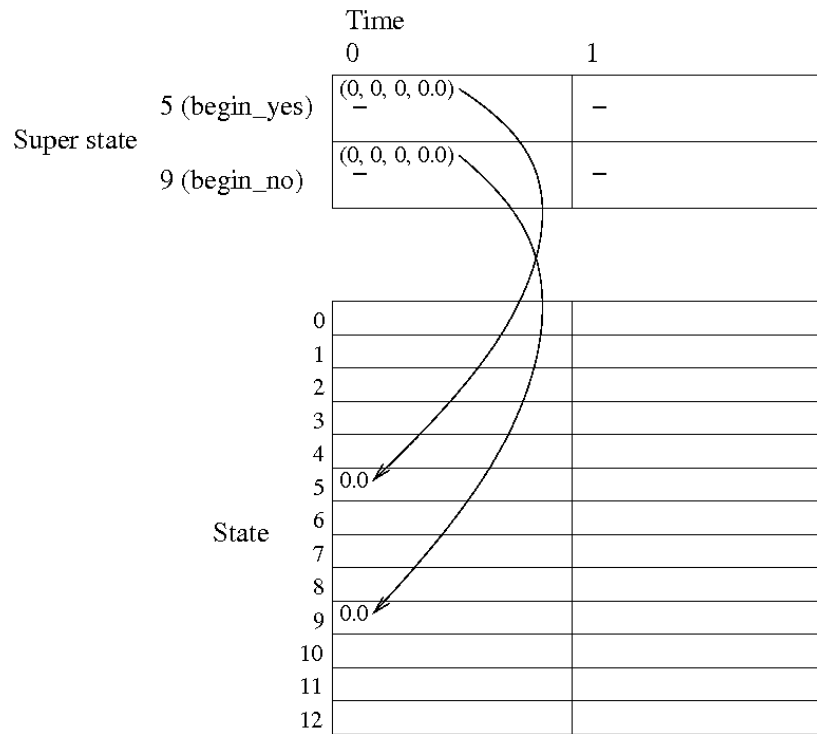


Figure 4.6: Illustration of step 3 in the N-Best segmentation algorithm. The begin state entries in the 1-best matrix are populated with the best scores from their associated N-Best lists.

Step 4: In Fig. 4.7, we can see how HMM segmentation takes place as with the normal Viterbi algorithm. Only the single best route through all the phonemes is calculated for the current time.

Step 5 and 6: Step 5 cannot be applied to this example since there are no non-emitting non-super states in the HMM. Normally the phoneme models would be more complex than a single emitting state and they would contain non-emitting states themselves. In Fig. 4.8 we see the lattices at a later time. We assume that the 1-best Viterbi segmentation has resulted in the scores in the figure. Now in step 6 we form the N-best lists contained in the end states as illustrated. This is done by backtracking within the word from the end state until we find the associated begin state. The N-Best list associated with the begin state is copied to the end state N-best list and the score difference between the begin and end state is added to all the scores

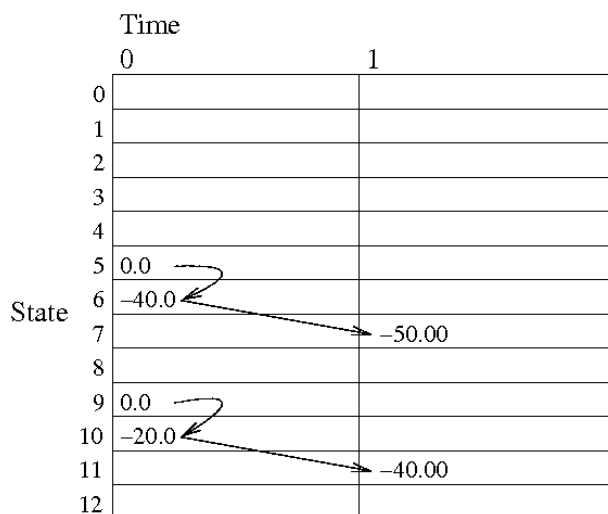


Figure 4.7: Illustration of step 4 in the N-Best segmentation algorithm. Normal 1-best segmentation takes place exactly as is done with the Viterbi algorithm.

in the list. In this case it is just the scores found, since the associated begin states contain 0.0.

Step 2 (later): In Fig. 4.9 we can see how step two later expands the newly formed entries and merges them into glue state 3. They are expanded to glue state 4 and then to begin states 5 and 9. The 1-best lattice is then populated in Step 3 and the cycle continues.

The merging that is done in step 2 eliminates entries that have the same source state, time and index as an entry already in the destination vector, since they represent the same path. N-Best vectors are sorted with the highest scoring entry at the top and the worst scoring entry at the bottom with the rest of the entries unsorted in the middle. This allows us to rapidly determine whether a new entry should be entered into the vector and also eliminates costly sorting. We assumed that the HMM does not contain any emitting end or glue states, since there is no reason for them to exist in speech recognition applications and they greatly complicate segmentation. Two beam widths are used during segmentation. One is used to limit the number of end states expanded in step 2, which is also called a “between-word” or “word-level” beam. The other is used to limit the number of

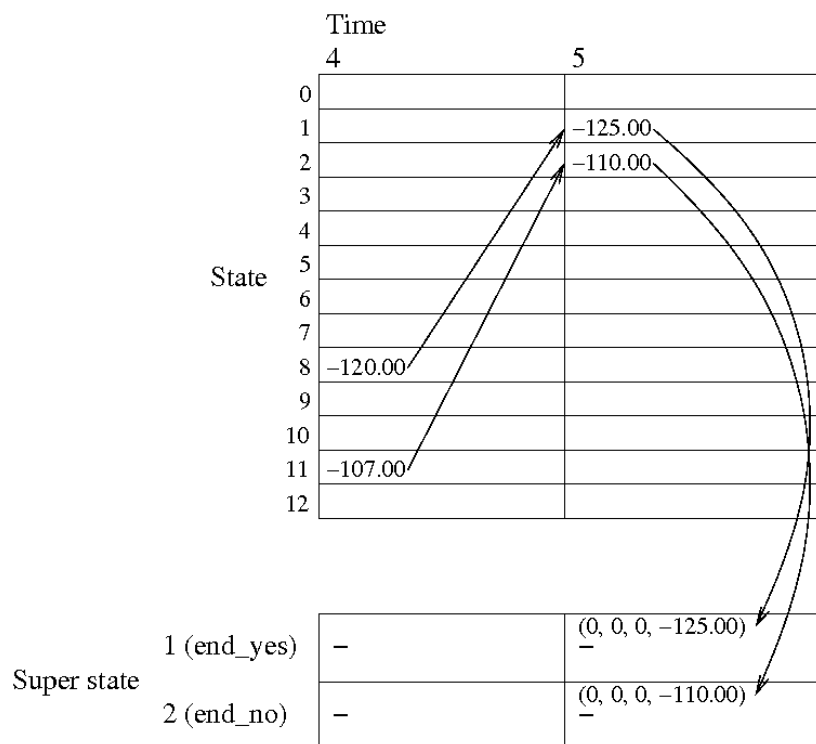


Figure 4.8: Illustration of step 6 in the N-Best segmentation algorithm. After emitting states 8 and 11 are expanded to end states 1 and 2, the 1-best end states scores are used to form the end state lists.

expansions within a word in step 4 and is known as the “within-word” or “phoneme-level” beam.

It is important to note that in step 2 we need to transform the entries in an end state list when we expand them to a glue state. This is to enable the next found end state to backtrack to the correct end state. If we simply projected an end state entry forward we would backtrack directly from the tail to the root. The first N-best list at the root would be projected through the entire HMM. For this reason we transform the end state list in step 2 when they are projected. The projected entry contains the same scores, but the state, time and index point to the entry from which the projection takes place.

The same score buffers described in Section 4.1 are used here. We do not need time buffers, but several additional matrices are required in deter-

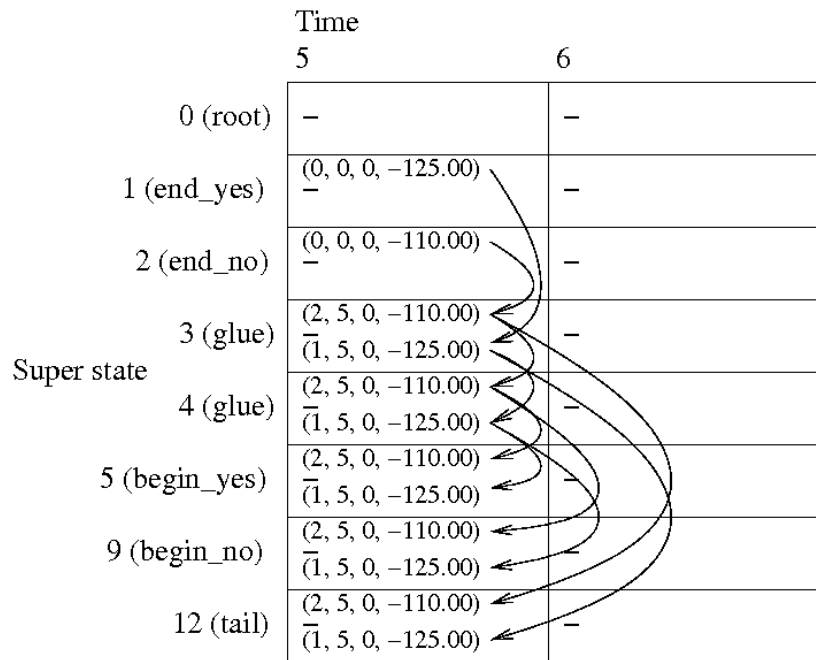


Figure 4.9: Illustration of a later iteration of step 2 in the N-Best segmentation algorithm. The newly formed end state lists are expanded and merged to the glue states and in turn the begin states.

mining these routes:

1. N-Best path matrix for end states, which contains all the paths between end states. This is the most important segmentation result, since it contains all the final word sequences.
2. N-Best score matrix, which contains the scores related the entries in the N-Best path matrix for end states. We can reduce the size of this matrix drastically if specific scores in the word sequences are unimportant.
3. N-Best path matrix for begin states, which contains information used to construct end state lists. This is only used during segmentation.
4. N-Best begin states score matrix, which contains the scores related to the entries in the N-Best path matrix for begin states. This is only used during segmentation.

5. 1-Best path buffers used by the Viterbi algorithm to determine the 1-best path between word begin- and end states. Used in the exact same way as the paths buffer in the Viterbi segmentation.
6. 1-Best score buffers used by the Viterbi algorithm to determine the 1-best path between word begin- and end states. Used in the exact same way as the score buffer in the Viterbi segmentation.
7. 1-Best begin state and time buffers, which contain the backtrack to the last begin state from any entry in the 1-Best matrix. These are equivalent to the paths buffers used in normal Viterbi segmentation, but in this case a pair (state and time) is projected instead of only the state.

The N-Best path and score matrices can be seen as one. They are separated in the implementation to enable further optimisations later. The end state and begin state path and score matrices are illustrated in Fig. 4.10. N^B and N^E are the number of begin states and end states, respectively. The buffers used in the 1-best segmentation are shown in Fig. 4.11.

The only matrix that needs to store a column for each observation vector is the N-Best path matrix, since it contains all the information related to the paths finally found. Each entry in this matrix contains a N-sized list of previous state, time and index. The N-Best score matrix is used during decoding, but the matrix does not need to be stored for the entire length of the observation sequence. Recall that we only use the N-Best score matrix to calculate the new score list for the current end state. There is a limit to the score history length required, since there is a limit to possible word length in practice. We do not need N-Best lists for begin or end states earlier than this, because no words will be that long. Silences are an exception, but they can be represented by a sequence of silences and do not affect accuracy since they are removed in the final word sequence. Similarly the N-Best begin states path and score matrices do not need to be stored in full.

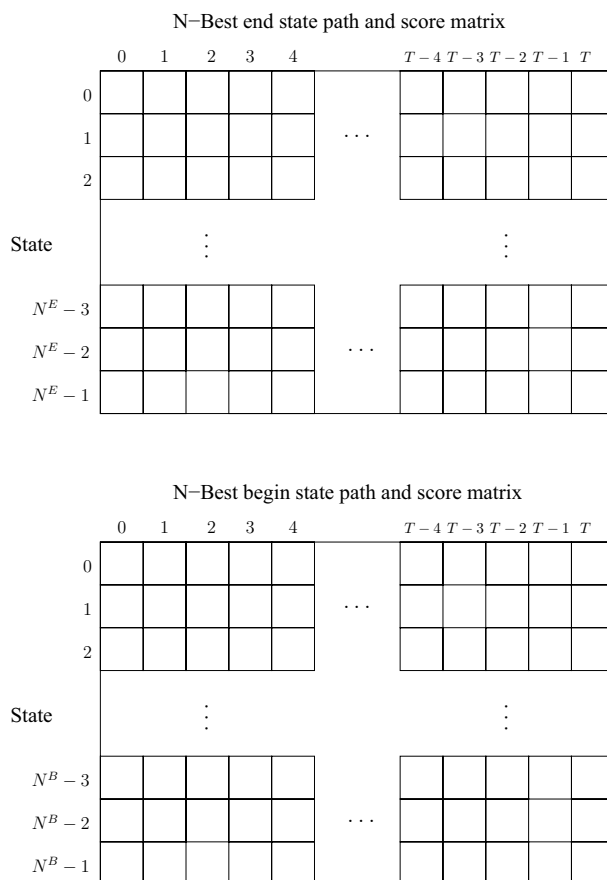


Figure 4.10: N-best end states and begin states path and score matrices.

All the 1-Best matrices can be reduced to two buffers each, which simulate a matrix. This can be done since for any given time t we do not need to remember any information earlier than time $t - 1$. After processing a frame we simply swap these two buffers and clear the buffer at time $t + 1$.

4.3 Language model

For the language model (LM) we decided to use the SRILM toolkit [47], which is a complete package for training n -grams with various advanced techniques.

We represented the trained grammar as an HMM with each of the nodes

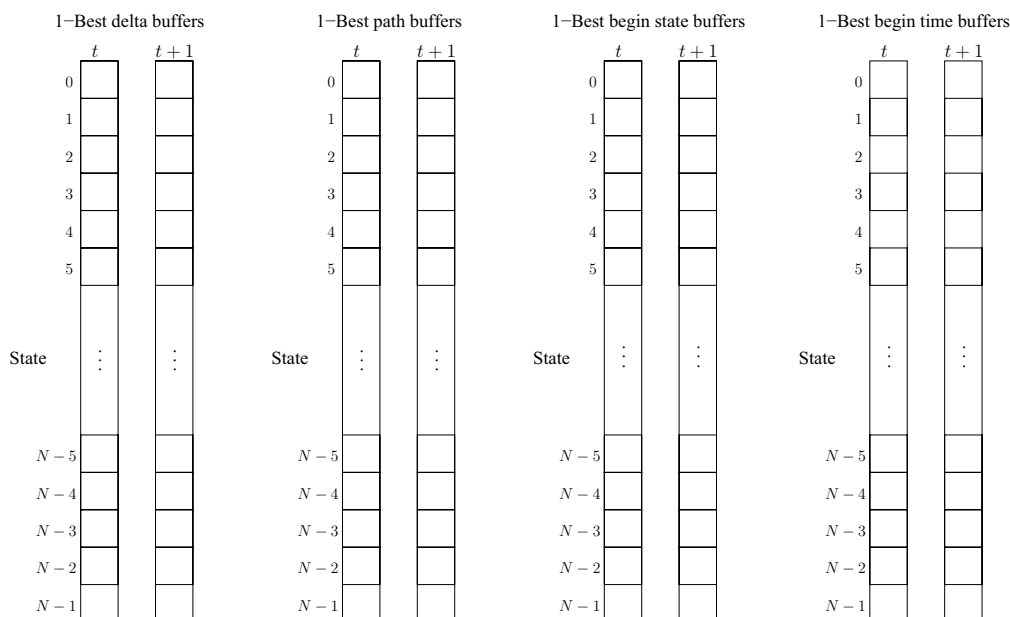


Figure 4.11: 1-best path, score and begin state backtrack buffers.

representing a word in a tree topology. The major problem with this grammar is the same as with any n -gram grammar: insufficient training data. As mentioned in Section 3.3 it is not possible to train an effective n -gram model, even with an extremely large training corpus. For this reason we decided on the standard Good-Turing discounting and Katz back-off for smoothing.

4.4 N-Best grammar segmenter

To incorporate our LM into the N-best segmenter, we need to make some changes to the original algorithm. We want to evaluate only the word sequences that are found in the LM. We can easily find word sequences at any time by backtracking the entries found in a given end state. We now need to add entries to valid destinations given a word history. Efficiency is important here, so we need to find an efficient way to find the appropriate begin state given a destination word.

The N-Best grammar segmenter is an expansion of the normal N-Best

segmenter and can use a language model during segmentation, such as those described in Section 3.3. This grammar takes a word history and calculates a probability associated with that word sequence. The grammar is integrated into the segmenter by changing the HMM topology slightly and adding another step to the segmentation process. The feedback loop and the glue states are removed, along with the transitions from glue states to begin states and from end states to glue states. This is illustrated in Fig. 4.12, which is what the previous example would look like when used with the N-Best grammar segmenter. The segmentation process can no longer rely on the HMM to designate potential next words and a new technique to determine candidate words needs to be found.

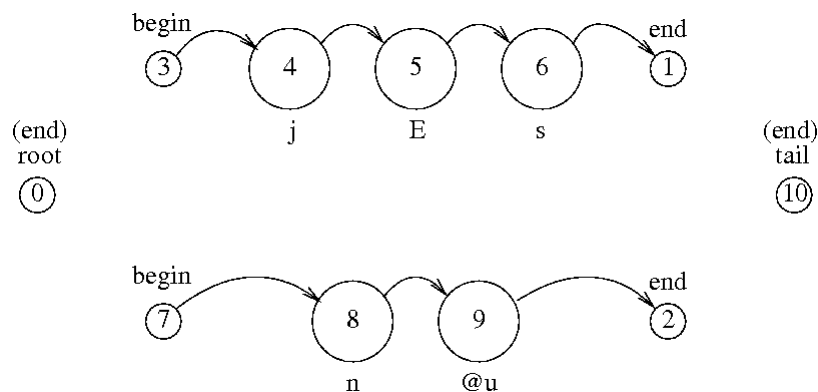


Figure 4.12: Example of HMM used for 2 word N-Best grammar spotter. Words are not connected and the feedback loop and glue states are removed. The language model is responsible for connecting the words.

Evaluating each possible word sequence is extremely expensive and unnecessary. For this reason our N-Best grammar segmenter needs a lattice associated with the begin states at each time. This lattice contains a value for each begin state at each time and indicates whether the particular word should be considered. If we have no extra knowledge to give the segmenter, a fully populated lattice can be given. However, for a large-vocabulary the processing requirements would be prohibitive. For this reason we need to somehow do an initial pass, which gives the N-Best grammar segmenter a

greatly reduced matrix of possible word beginnings. This would enable us to find a detailed N-Best list efficiently. The correctness of the lattice is the maximum accuracy that can be achieved by the N-Best segmenter. In other words, the N-Best segmenter can only produce word sequences in terms of words in the lattice. If the lattice does not contain the correct word at the correct time, it will not be evaluated.

The words are now disconnected in the HMM and can only be connected through a combination of the lattice and the grammar. For each time t and each end state, we backtrack each entry in the N-Best list to find the N-sized list of word histories. Each of these word histories is then augmented with each possible word beginning according to the lattice, and given to the grammar to find the probability. For each of these possibilities, we add a new entry to the list in the N-Best lattice at the begin state of the word. This entry simply contains the source word end state number, current time and appropriate index from which this word was activated. The new entry is abandoned if the score that is found after the grammar score has been added fails to improve on the worst score already in the destination list. This new algorithm eliminates the need for glue states.

We calculate the grammar score G'_s with the following formula:

$$G'_s = G_s^{G_w} + P_{WI} \quad (4.4.1)$$

where G_s is the N-gram score found, G_w is a constant grammar weight and P_{WI} is a word insertion penalty (WIP). For the sake of computational simplicity we keep all the scores in \log_e form. Now all the multiplications caused by transition weights become additions. We add the grammar weight so that we can optimise the scale of the grammar by evaluating the development set.

In order to recognise silences between words, we append an optional <sil> phoneme to each word model in the vocabulary. This results in more accurate time boundaries in the generated transcriptions.

4.5 Silence Detection

In any speech recognition task a fast and accurate silence detection technique is an important asset. Johan du Preez developed a silence detection technique based on the use of densities to model power and power variance of Gaussian noise in a signal. The full derivation as made available by Johan du Preez can be found in App. A.4.

4.6 Context-independent phoneme modelling

The high level process behind our context-independent (monophone) model training is illustrated in Fig. 4.13.

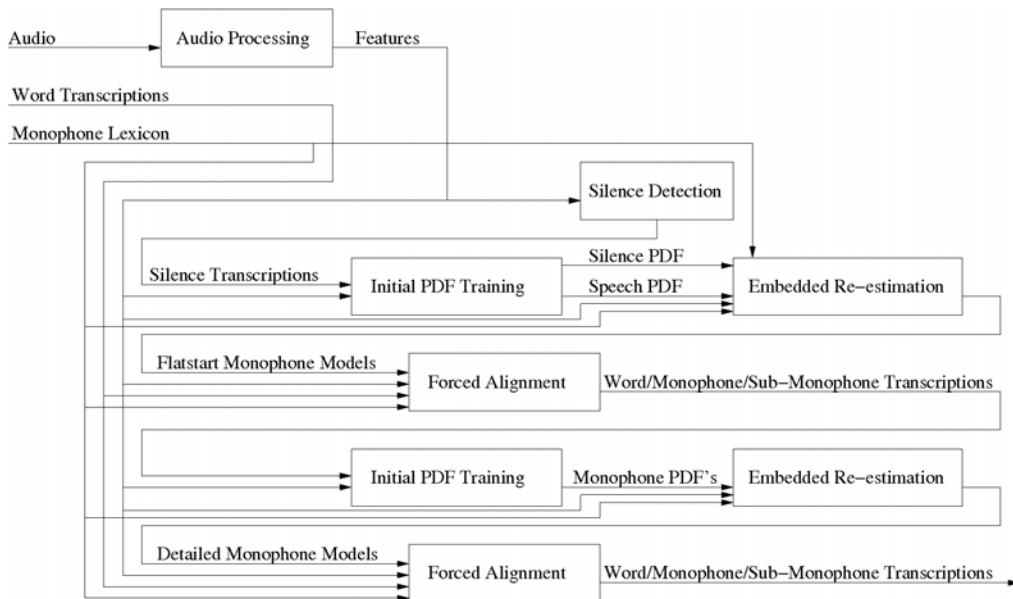


Figure 4.13: Monophone training process.

We used the XML transcriptions that were given with the original HUB-4 training corpus. These transcriptions are on the word level with a few arbitrarily fixed boundaries specified by the transcriber. However, these fixed boundaries are not always present, so it is mostly up to the training to guess the best phoneme-level transcription given the word sequence and

the lexicon. The lexicon was constructed from a subset of the Pronlex lexicon [23].

Often a Gaussian mixture model (GMM) would be used to model speech, but we opted for a more efficient Structured Adaptive Mixture (SAM) [9] Gaussian. SAMs represent GMMs in a tree form with the most significant mixture component at the root node. The densities near the root of the tree have the highest contribution and we can quickly evaluate a reduced number of mixtures to get an approximation of the similarity between a particular density and an observation. This can be seen as a form of fast match.

To construct the initial context-independent monophone models we use a simple initial PDF describing all the speech in the database to initialise each model. This is also known as the “flat start” training strategy [33]. First we create a single PDF from all the speech features and a PDF from all the silence features as found with our silence detection technique. For these two initial models we used a 24-dimensional SAM with 64 mixtures and a diagonal covariance matrix. The speech PDF is then assigned to all the states in the collection of speech HMMs we desire and the silence PDF is assigned to the silence HMM. This initial configuration is put through a sequence of parameter estimation iterations (embedded re-estimation) to improve the parameters for each model. After we find no significant further improvement in performance or we reach a maximum specified number of iterations, training is concluded.

We now proceeded to train these initial phoneme HMMs with a sequence of embedded re-estimation iterations to find our first phoneme HMMs. These HMMs are used to create forced alignments for each utterance in the corpus. This is done by concatenating the phoneme models in the sequence specified by the lexicon for each word in order to create an utterance model. Forced alignment then finds the most likely state sequence given the utterance model and the observation sequence. This gives us the best boundaries for each phoneme and sub-phoneme.

For example, let us assume we have to find the boundaries for an audio file 0.876 seconds long and we know the word sequence “come here” was said. We don’t know exactly at what time the “come” ends and the “here” begins and it is extremely expensive to create transcriptions on such a phoneme or even word level by hand. For this reason it is necessary to use this estimation approach. To create the utterance model we need for the forced alignment, we concatenate the word models as specified in the word sequence. The super model for the utterance HMM in this example is illustrated in Fig. 4.14.

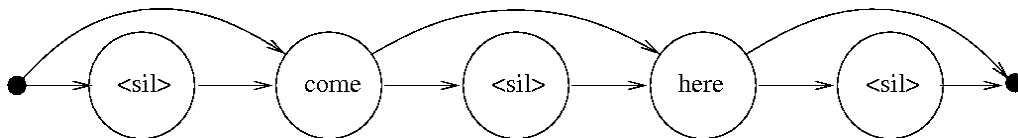


Figure 4.14: Example of utterance HMM used in forced alignment. Optional silences separate words.

Note the optional silence model that was added between each word since there may be silences between words in spontaneous speech. We now create word models from the initial models trained from the flat start iteration. This is done by simply concatenating the phoneme models as specified by the lexicon. In this case these pronunciations are:

come: k a m here: h i r

If more than one pronunciation exists for a word, both phoneme sequences are added in parallel. Now we are left with the final utterance model, which is simply a long concatenation of the phoneme models found during the flat start iteration. We now use the features and this utterance model and find the best possible route using the Viterbi algorithm. After we know the best route we can calculate the times for each transition on a sub-phoneme level. These transcriptions can now be used to create more detailed models. An example of a state-level monophone transcription can be seen in Table 4.1.

End time	Word	Monophone	Sub-Monophone
0.045			k0
0.090			k1
0.120		k	k2
0.150			a0
0.180			a1
0.225		a	a2
0.270			m0
0.305			m1
0.350	come	m	m2
0.395			h0
0.455			h1
0.505		h	h2
0.580			i1
0.640		i	i2
0.730			r0
0.805			r1
0.880	here	r	r2

Table 4.1: Example of a monophone level transcription.

Depending on the topology of the HMMs the sub-phone level paths do not need to go through each state in the HMM. If we have manually placed boundaries in the transcription the forced alignment will not create one big utterance model for the entire sentence, but separate models for the sub-sentences as split by the manual boundary. Also noteworthy is the fact that the times are in multiples of 0.015, which is the length in seconds of each speech frame for our implementation. These estimated times are calculated by multiplying the number of features spent in each state by the frame length of 15 ms.

These forced alignment transcriptions generated by the initial flat start models are then used to train more detailed models and a similar sequence of events is used. We now know the features that relate to each PDF for the utterances more accurately and can initialise each PDF accordingly. More detailed densities were used for these forced alignment models. Here we used a 256 mixtures instead of 64. After a further embedded re-estimation

cycle we have context-independent monophone HMMs.

4.7 Context-dependent phoneme modelling

The high level process behind our context-dependent (triphone) model training is illustrated in Fig. 4.15.

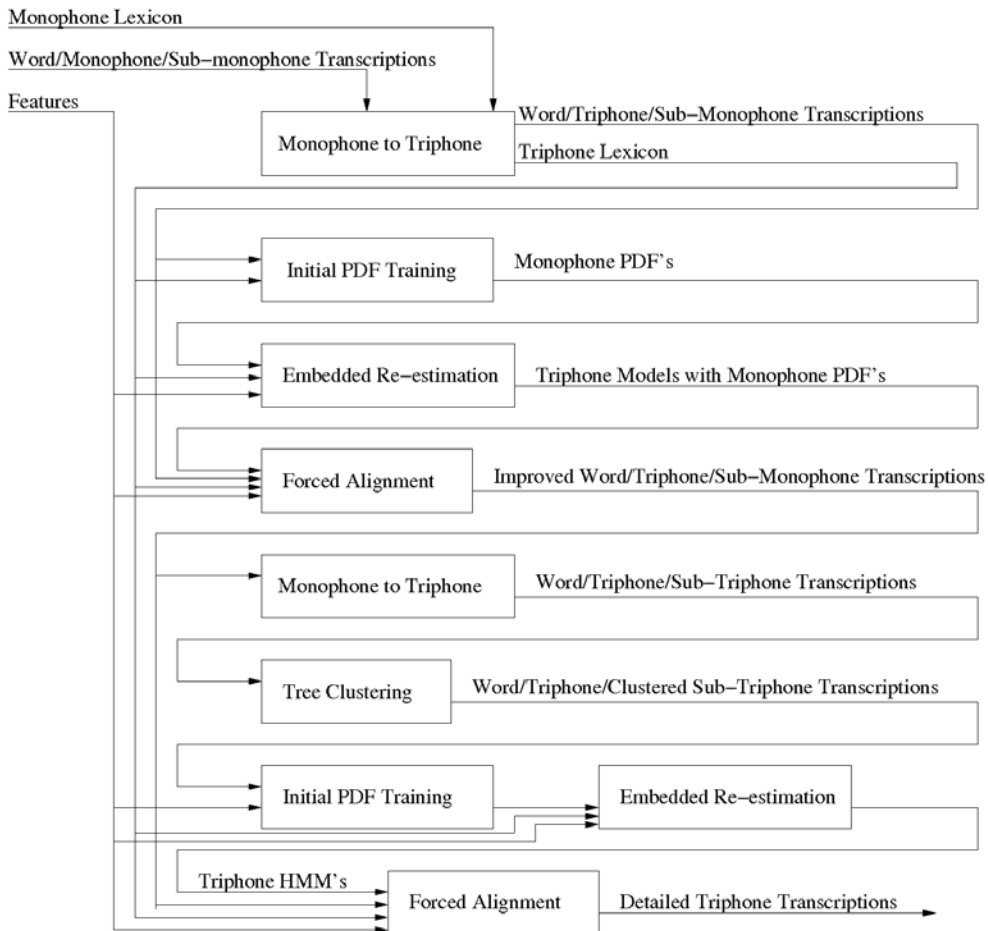


Figure 4.15: Triphone training process.

To construct the context-dependent triphones, we first construct the new triphone transcriptions from the initial monophone transcriptions. This can easily be done by mapping each monophone to a triphone depending on

the surrounding monophones. Each word starts and ends in a biphone to enable us to construct a lexicon and separate words and simplify the use of a grammar. These biphones are estimated in the same way as the rest of the triphones, but a placeholder label is added to the monophone set.

Now we have a triphone set with sub phone labels identical to those of the monophones. This equates to more transition matrices, but the same number of density functions. We now train these triphone models to find our initial triphone models. These models are used to find new triphone transcriptions with forced alignment. In turn, we map each sub-monophone label to a sub-triphone label. This is done by mapping the monophone label contained in the sub-monophone label to the associated triphone label. An example of a detailed triphone transcription can be seen in Table 4.2.

End time	Word	Triphone	Sub-Triphone
0.045			*-k+a0
0.090			*-k+a1
0.120		*-k+a	*-k+a2
0.150			k-a+m0
0.180			k-a+m1
0.225		k-a+m	k-a+m2
0.270			a-m+*0
0.305			a-m+*1
0.350	come	a-m+*	a-m+*2
0.395			*-h+i0
0.455			*-h+i1
0.505		*-h+i	*-h+i2
0.580			h-i+r1
0.640		h-i+r	h-i+r2
0.730			i-r+*0
0.805			i-r+*1
0.880	here	i-r+*	i-r+*2

Table 4.2: Example of a triphone level transcription.

These initial triphone transcriptions are now used to build a decision tree for state clustering. We use this decision tree to create triphones for

all the contexts (seen and unseen) in the closed lexicon that consists of all the words in the training, test and development sets.

4.8 Word Spotter

We decided to break our recogniser up into separate recognition phases. This allows us to gradually reduce the search space as described Section 2.7.3. In addition, this enables us to separately measure and optimise each phase and also to evaluate the effect on the accuracy of each. Since our N-Best segmenter depends on a lattice to perform efficiently, we decided to let our first pass create this lattice.

Initially, a normal forward Viterbi beam search on a parallel word HMM such as the one shown in Fig. 2.7 seemed like a good idea. However, all words would be activated at each time due to the transition to each word. The only distinguishing factor between words would be the grammar score for the hypothesis word at the current time. This is not enough information to make an informed decision on which word beginnings are probable at which time. We need to incorporate acoustic knowledge as well, but a combination of LM score and acoustic score can only be found at the word ending. This is the case if we use the forward Viterbi algorithm.

If we need a more detailed word score at the word beginnings, we could do a backward Viterbi beam search such as described in Section 2.3.2. This would result in Viterbi scores at probable begin states for each frame, which include both the grammar and acoustic information. After we find the lattice, we backtrack and store the most likely word sequence for later evaluation.

Initially we attempted to incorporate the SRI LM toolkit unigram weights as $P_{W_0} \dots P_{W_N}$ in Fig. 2.7. However, this did not produce good results, since backward segmentation is taking place. In this configuration, the unigram weights would only be incorporated upon exiting the begin states. We require them to be included by the time we reach the begin states. For this

reason, we inserted the unigram weights on the end state side of each word. This is illustrated in Fig. 4.16 and produced the desired result.

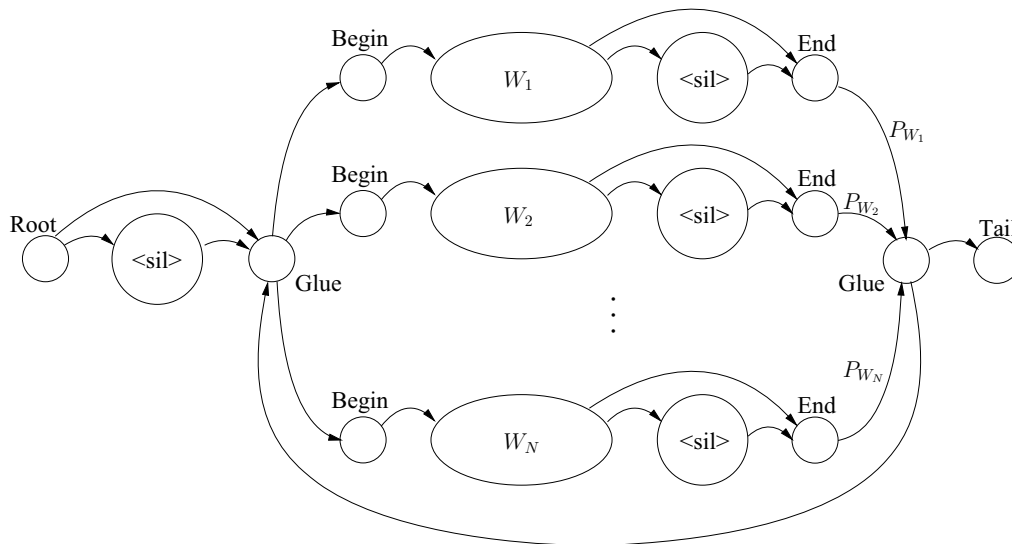


Figure 4.16: HMM configuration used in first phase of word spotter. Words are placed in parallel, with optional silences separating words. The language model weights are incorporated last in the segmentation process, so that they are incorporated first when backward Viterbi segmentation takes place.

Now that we have a lattice, we can perform an efficient N-Best search. An HMM is constructed such as the one shown in Fig. 4.12 and used along with the lattice from the backwards Viterbi search. This results in a list of N word sequences, which can now be rescored to find the final hypothesis. The entire process is illustrated in Fig. 4.17.

The general idea behind the three phases is the systematic reduction of the search space efficiently with minimal loss in potential accuracy. The first phase results in a relatively general answer. Most options are completely eliminated with this fast search. Normally here we would have the most simple grammar or no grammar at all. The second phase does a more detailed search on a greatly reduced search space and a more complicated grammar can be incorporated. The final phase has the smallest search space and we perform our most detailed search and include a powerful grammar.

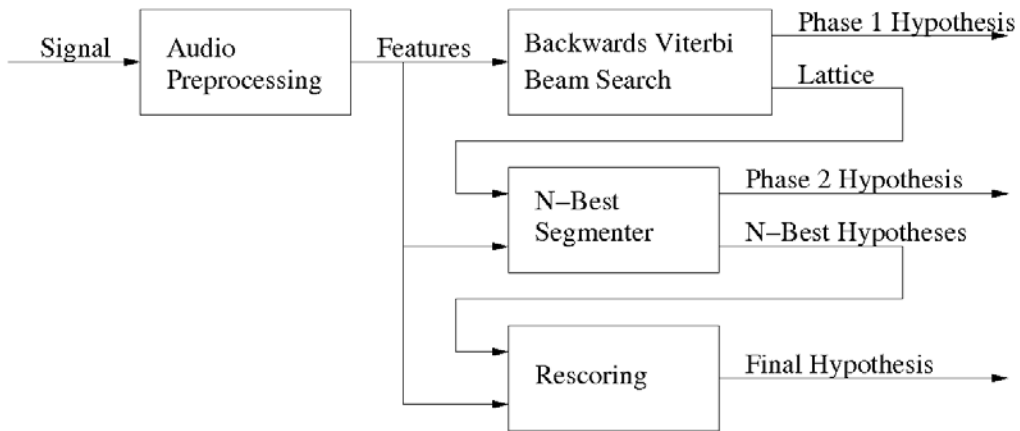


Figure 4.17: Process used to generate word hypothesis.

4.9 Summary

In this chapter we explained how we implemented the theory considered in the previous chapters. First we showed how we incorporated the multi-level segmentation techniques into the normal Viterbi segmentation software to reduce memory usage. This formed the foundation of all the segmenters implemented at a later stage. We showed how we expanded the multi-level segmenter to create our N-Best segmenter and explained with the help of a detailed example how it works.

Next we discussed the LM we used, which was created by a commonly used toolkit implemented by SRI. We went on to explain how this LM was incorporated into our N-Best segmenter. Our silence detection technique was also considered.

We later explained how our context-independent monophones were trained and in turn used to create our final context-dependent triphones. Finally, we explained how our three phase word spotter was constructed. The first phase of the spotter creates a high level lattice using a simple unigram language model. The second phase of the spotter performs an N-Best search using a bigram language model on the lattice created in the first phase. Finally, the third phase rescoring the N-Best list from the second phase with a trigram language model to find the final hypothesis.

Chapter 5

Experimental investigation

In our practical investigation we made use of all the components necessary for a complete large-vocabulary speech recogniser. We tested our system on the 1996 DARPA/NIST continuous speech recognition broadcast news Hub-4 partitioned evaluation. All the audio recordings are excerpts from television and radio news. The audio consisted of approximately 30 hours of training, two hours of development and two hours of testing data. With the partitioned evaluation we conducted, the transcriptions consisted of equally divided word level transcriptions for all the audio with some additional manually placed boundaries.

We constructed a lexicon from all the words in the training, development and evaluation sets, resulting in a vocabulary of around 18000 words. The SRI trigram language model that was used had a perplexity of 350.

In this chapter we experimentally determined appropriate parameters for our phoneme models and word recogniser. We evaluated our hypotheses by making use of our own implementation of the techniques described in Section 2.5.1, except where specified otherwise.

We also performed the same word recognition experiments on Sphinx 3.2 with the same lexicon, phoneme set and language model so that the performance of our system might be compared to it.

5.1 Continuous phoneme recognition on Hub-4 broadcast speech

5.1.1 Flat start monophone recognition on Hub-4 broadcast speech

Motivation: For us to build a word recogniser, we first need to construct proper acoustic building blocks that we will use to represent the wide range of words that will be recognised by the system. The first step in constructing these models is constructing our initial phone level transcriptions from our word level transcriptions.

Experimental setup: We made use of the Callhome Pronlex lexicon as a base pronunciation lexicon and also used the phoneme set proposed by them [23]. The final phoneme set consisted of 40 monophones, 1 silence model and 1 model for other utterances (junk). The 24 dimensional features were calculated using MFCCs with 12 cepstral coefficients and 12 Δ [17] coefficients to represent each 15 ms speech frame. To find our initial monophone level transcriptions we constructed flat start HMMs. We first used silence detection as derived in App. A.4 to find initial silence/speech transcriptions. We then used these transcriptions to train a silence and a speech PDF, which we used in all the states as an initial condition for the HMMs used in the embedded re-estimation. All states in all HMMs are assigned the speech PDF, except the silence HMM and the first state in all stop sounds. This included the phonemes *b*, *d*, *g*, *k*, *p* and *t*.

The 40 monophones were modelled by three-state HMMs with a single skip link, as illustrated in Fig. 5.1. The density on each emitting state was a 256 mixture SAM with diagonal covariance Gaussians. The sil and junk HMMs each contained a single emitting state with the same type of density as the rest of the HMMs, but only 64 mixtures. We created all hypothesis transcriptions using a normal Viterbi segmenter.

Results: Our experiments were done on the development set. After the

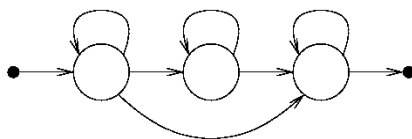


Figure 5.1: An HMM topology commonly used to model phonemes.

flat start training we found the results in Table 5.1 for phoneme spotting:

Substitutions	45.34%
Deletions	15.81%
Correct	38.83%
Insertions	5.52%
Accuracy	33.91%

Table 5.1: Phoneme spotting results for flat start monophone HMMs.

Interpretation: A relatively poor result, but these models are only used to initialise more accurate models.

5.1.2 Forced alignment monophone recognition on Hub-4 broadcast speech

Motivation: We now had our initial monophone transcriptions and could use them to create more accurate models. We can now make use of Linear Discriminant Analysis (LDA), which is a class-based dimension reduction technique and is described in App. A.1. It was unavailable to us in the flat start phase since we did not have any class-level segmentation.

Experimental setup: To simulate using Δ and $\Delta\Delta$ coefficients, we formed 108-dimensional features by concatenating sets of nine neighbouring 12-dimensional MFCCs from the previous experiment. These new features were reduced to 24 dimensional features using LDA. We used the same HMM topology and density functions as before and could train new models using forced alignments formed with the flat start models. We created all hypothesis transcriptions using a normal Viterbi segmenter.

Results: After adding a large measure of context in the calculation of the feature vectors, we found the results in Table 5.2.

Substitutions	35.59%
Deletions	19.75%
Correct	44.64%
Insertions	3.08%
Accuracy	42.10%

Table 5.2: Phoneme spotting results for forced alignment monophone HMMs. A significant improvement on the flat start monophones.

Interpretation: After we trained the monophone models from the flat start monophones we have significantly more accurate monophone models and state-level transcriptions. More intelligent features also contributed to a 8.19% improvement on the flat start models in phoneme spotting accuracy.

5.1.3 Forced alignment triphone recognition on Hub-4 broadcast speech with monophone densities

Motivation: In order for our decision tree clustering to be most effective we need the monophone transcriptions to be as accurate as possible. For this reason we boost our monophone transcription accuracy by creating triphones with monophone densities. This way we do not need to construct a decision tree and can make use of a larger number of transition probabilities.

Experimental setup: We built a set of triphones consisting of all the triphones that occur in the training, testing and development set. All the monophone state-level forced alignment transcriptions were converted to triphone state-level transcriptions. Normally a decision tree would be constructed immediately, but we first created triphone models which have identical state distributions to the monophones. We have the exact number of PDFs as before (122) but now we have 9898 triphone HMMs instead of the 42 monophone HMMs. We created all hypothesis transcriptions using a normal Viterbi segmenter.

Results: After training the triphone models with the same number of densities as the monophones we found the results in Table 5.3.

Substitutions	34.47%
Deletions	18.35%
Correct	47.17%
Insertions	3.63%
Accuracy	44.0892%

Table 5.3: Phoneme spotting results for forced alignment monophone HMMs, which also make use of triphone transition matrices. A 2% improvement on the forced alignment monophones using only monophone transition matrices was found.

Interpretation: As we suspected, we were able to gain some extra accuracy from the training of the triphone transition probabilities. This 2% gain in phoneme spotting accuracy from monophones will help us to create a more accurate decision tree.

5.1.4 Forced alignment triphone recognition on Hub-4 broadcast speech

Motivation: We now had more detailed monophone transcriptions. These were converted to triphone transcriptions, which could be used to build a decision tree. Since most of the triphones do not occur often enough, we need to cluster the rare triphone states with the more common triphone states using decision trees. The number of clusters formed is controlled by the minimum occupation count (MOC). From this decision tree we relabel the triphone state-level transcriptions to clustered triphone state-level transcriptions. These new transcriptions are then used to train the triphones using the same procedure as before.

Experimental setup: The decision tree was constructed for a wide range of MOCs. We also constructed a triphone lexicon from the monophone lexicon which covered all the words in the training, testing and development sets. Using this new lexicon we knew all the contexts that we would need

to recognise and could assign them to the closest known context by using the decision tree. We transformed the state-level transcriptions to clustered state-level transcriptions and trained the 9898 triphone models using embedded re-estimation.

To evaluate the triphone spotting accuracy we started off by constructing reference triphone transcriptions with forced alignments using the newly trained triphone models. Next, we used a triphone spotter to construct triphone level hypothesis transcriptions. The context-dependent phoneme spotter makes use of a grammar where the context-independent phoneme spotter did not. We are only interested in phoneme sequences with contexts that match. This idea is illustrated with the two-alphabet context-dependent spotter illustrated in Fig. 5.2. After finding the most likely phoneme sequence by using this spotter we compare it with the forced aligned transcriptions to find the phoneme spotting accuracy for the utterance. We created all hypothesis transcriptions using a normal Viterbi segmenter.

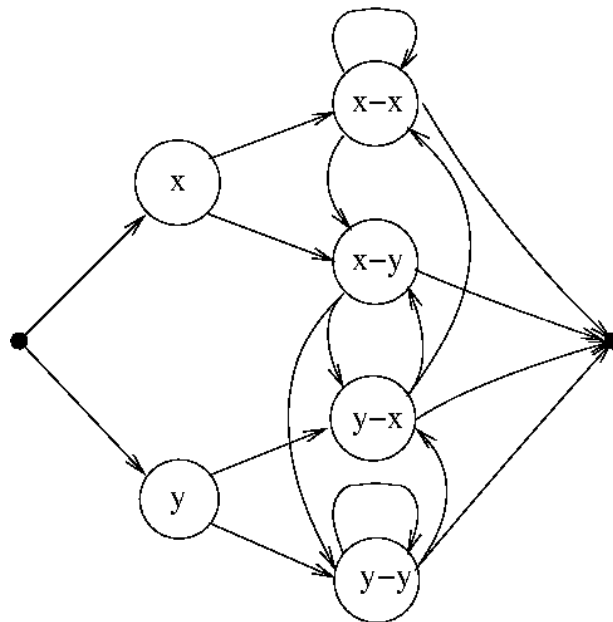


Figure 5.2: Context-dependent phoneme spotter with an alphabet containing two symbols. Only matching contexts can be recognised.

Results: For our range of MOC we found the results in Table 5.4.

MOC	Clusters	Subs	Del	Corr	Ins	Acc
200	6581	36.70%	09.08%	54.21%	9.89%	44.23%
500	4142	36.24%	09.46%	54.29%	9.29%	45.01%
1000	2691	35.98%	09.83%	54.17%	8.97%	45.28%
1500	2020	36.00%	10.19%	53.80%	8.49%	45.46%
2000	1619	35.80%	10.48%	53.71%	8.33%	45.58%
2250	1472	35.56%	10.66%	53.76%	8.14%	45.84%
2500	1364	35.63%	10.67%	53.68%	8.01%	45.90%
2750	1264	35.68%	10.81%	53.50%	7.88%	45.86%
3000	1178	35.74%	10.97%	53.27%	7.71%	45.83%
3250	1096	35.81%	10.91%	53.26%	7.60%	45.93%
3500	1025	35.63%	11.15%	53.21%	7.58%	45.91%
3750	973	35.85%	11.15%	52.99%	7.63%	45.65%
4000	918	36.00%	11.21%	52.77%	7.53%	45.54%
5000	765	36.01%	11.65%	52.33%	7.14%	45.53%
10000	410	37.55%	12.56%	49.87%	6.34%	43.95%

Table 5.4: Triphone spotting results for various MOCs. An appropriate MOC seems to be 2500.

Interpretation: In Fig. 5.3 we see the effect of MOC on the number of clusters. We see the most volatile range of occupation counts are between 1 and 4000, which yields between 1000 and 6500 clusters. Greater occupation counts have little effect on the number of clusters. For this reason we tend to choose our MOC between 1 and 4000. Another consideration is that we would like to keep the number of clusters low, since more clusters would require more training data.

In Fig. 5.4 we see that as the number of clusters drops we have an increase in insertions and a decrease in deletions. The closest we could get them to each other is within the range of 3000 to 5000 clusters, indicating a balanced system.

In Fig. 5.5 we see the accuracy for the various parameter counts. The greatest accuracies are for cluster counts between 1000 and 1500.

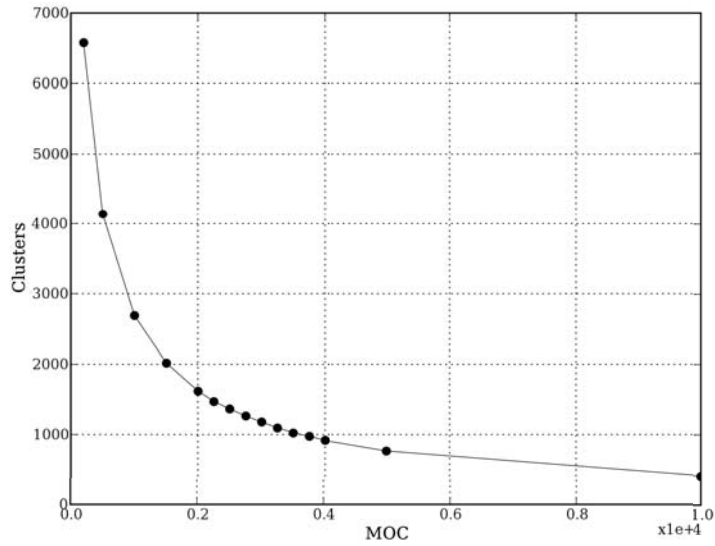


Figure 5.3: Number of parameters for various MOCs. The number of clusters drops as the maximum occupation count is increased.

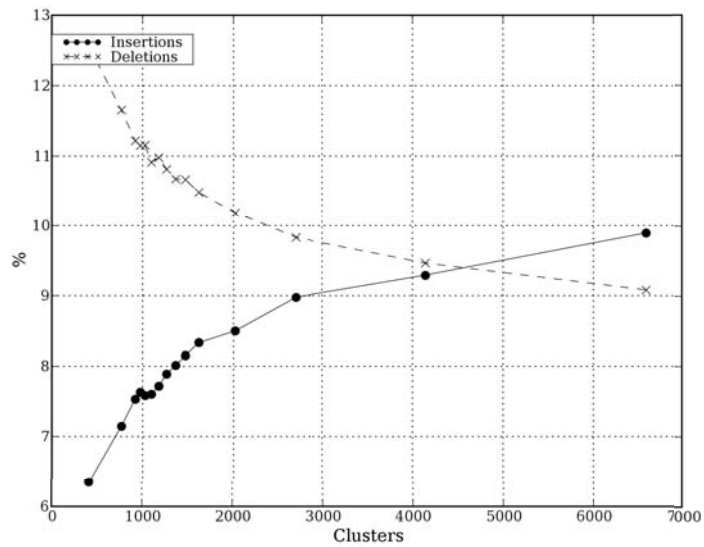


Figure 5.4: Insertion and deletion rates for various parameter counts.

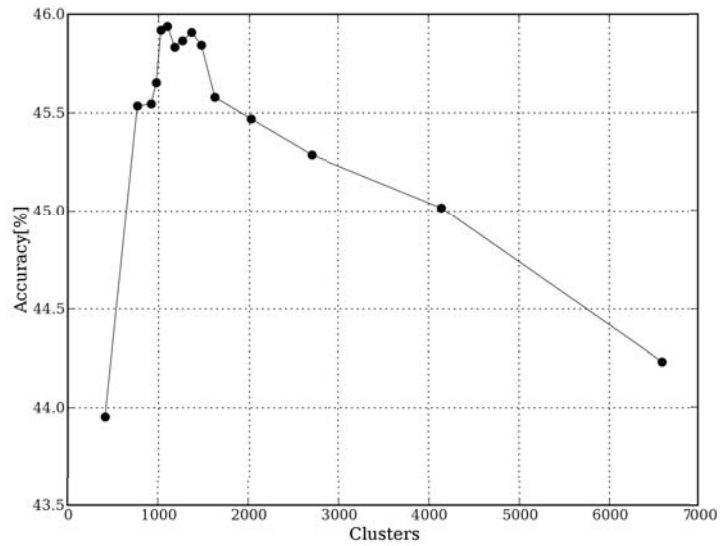


Figure 5.5: Accuracy for various parameter counts. The most accurate systems are between 1000 and 2000 clusters.

Considering all these factors, we concluded that a reasonable number of clusters is 1364 found with a MOC of 2500. These were the triphones that were used in our further experimentation.

5.2 Effect of beam on accuracy and performance

Motivation: To measure the effect of a beam on the performance and accuracy of a recogniser, we need to have some measurement of computational requirements. Normally the number of clock cycles used on a process (or ticks) is a good indication of computational requirements. We want to know what the effect of beam width is on accuracy and performance.

Experimental setup: We performed monophone spotting on the complete development set and measured ticks and phoneme error rate as before on a variety of beam widths. Hypothesis transcriptions were created using a Viterbi segmenter which makes use of a beam as described in Section 2.7.1. We used our monophones as trained in Section 5.1.2.

Results: For beam widths of e^5 , e^{10} , e^{15} , e^{20} and e^{25} we see the required ticks in Fig. 5.6 and the monophone spotting accuracies in Fig. 5.7.

Interpretation: We can see from these results that the accuracy drops as the beam width is reduced. This is to be expected, since the number of paths considered is reduced.

It should also be noted that even with a beam of e^{15} the accuracy has already nearly reached its maximum. This is despite the fact that significantly fewer ticks are required to find the same result. From this we can conclude that a well chosen beam width significantly improves performance while barely affecting accuracy.

5.3 Continuous word recognition on Hub-4 broadcast speech development set

If we assume that good results in earlier phases of our word spotter lead to equal or better results in later phases, we could find parameters for each phase individually. In other words, we assume that later phases in the recogniser are fully dependent on earlier phases.

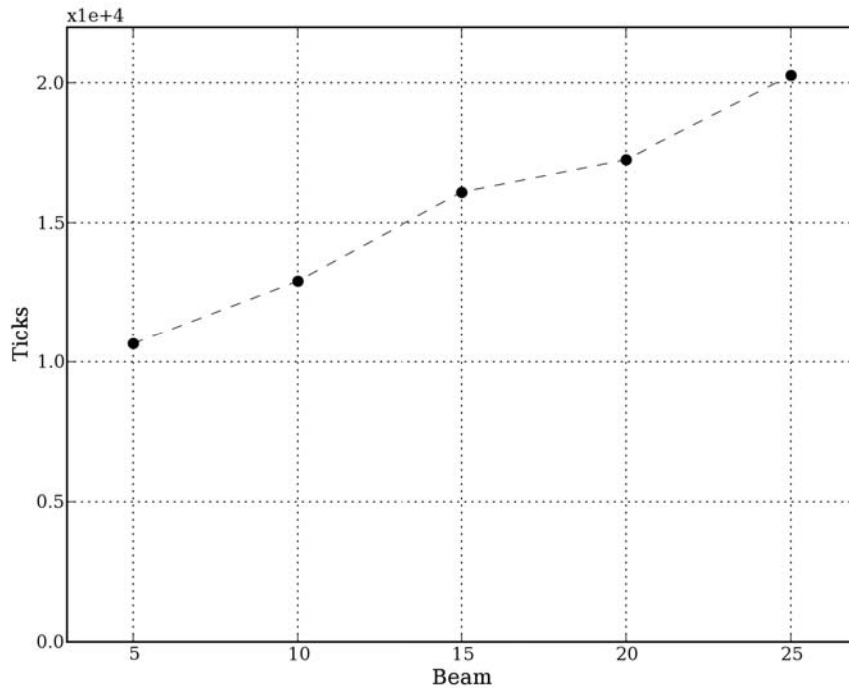


Figure 5.6: Ticks required for monophone spotting for beam values of e^{Beam} . The processing power required increases as the beam becomes wider.

We do not aim to find the ideal parameters for the system, since this could take a very long time. We are simply searching for local optima. This will give us an idea of the performance we could hope to achieve with our system.

5.3.1 Finding effective parameters for phase one.

Motivation: In the first phase the lattice is created, which forms the basis for phase two and consequently phase three. A sparse and accurate lattice from phase one will result in efficient and accurate processing of later phases. If our lattice is overpopulated, we could not hope to efficiently perform the N-Best search in phase two. On the other hand, if our lattice is too sparse and does not contain the correct word sequence, our later accuracies will

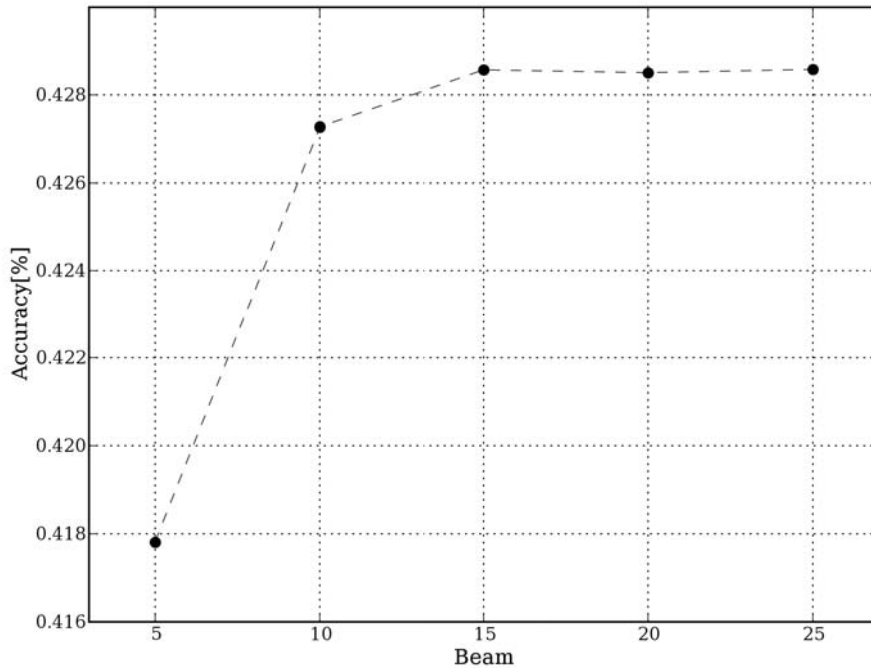


Figure 5.7: Accuracies found for monophone spotting for beam values of e^{Beam} . The maximum accuracy is quickly reached and a beam width that is increased further has no effect.

suffer.

In the worst-case scenario, the normal Viterbi algorithm would have required 22 times more memory than our multi-level Viterbi algorithm for the first phase in our Hub-4 implementation. With smaller vocabularies we might have found an acceptable ratio, but in our experiments we used a large-vocabulary. For this reason, the normal Viterbi segmenter would not have been feasible at all for our experiments.

Experimental setup: We assumed that our word level beam should not be wider than our state-level beam. This is due to the fact that the state-level beam determines the lower bound for the scores that enter word level states and having a wider beam at this stage would not result in more information.

The word level beam was also chosen to be narrower than the state-level beam, since once we reach word level states we have acoustic and linguistic knowledge already included in the score. A beam of e^{50} is considered wide and was chosen as our state-level beam. A beam of e^{30} performs moderate pruning and was chosen as our word level beam.

For various combinations of grammar weights (GWs) and word insertion penalties (WIPs) as defined in Equation 4.4.1, we created a lattice for each sub-segment as described in Section 4.8. A unigram language model making use of Good-Turing discounting was used, which was trained from the Hub-4 training data. The final grammar scores were calculated as described in Equation 4.4.1. We tried GWs 1.0, 2.0, 3.0, 4.0 and 5.0 with WIPs 2, 0, -2, -4 and -6. For each of these configurations we evaluated the two hours of speech in the development set and measured:

1. The average words per frame (WPF).
2. The percentage of words appearing in the correct word sequence and also in the lattice.
3. The accuracy, insertions and correctness for the best scoring path in the lattice.

Results: In Table 5.5 we see the accuracies of the highest scoring paths from phase one for each segment. The associated insertion and correctness rates is seen in Table 5.6. The correctness of the lattice and average word beginnings per frame is seen in Tables 5.7 and 5.8 respectively. In Fig. 5.8 we see the lattice correctness for various average word beginnings per frame.

Interpretation: From the accuracies found in Table 5.5 we see that the most accurate best path is found with GW 4.0 and WIP -6. If the system was forced to give a word hypothesis after the first phase unigram segmentation, we would find this WER of 72.72%.

Since this phase only forms the foundation for later phases in the system, we had to consider the results in more detail. We noted that with GW 4.0 and WIP -6 the correctness is significantly lower than with lower WIPs.

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	2.56%	15.47%	22.01%	25.43%	24.75%
0	7.35%	18.13%	23.85%	26.54%	24.95%
-2	11.21%	20.36%	25.45%	26.83%	24.81%
-4	14.51%	22.32%	26.51%	27.21%	24.25%
-6	16.89%	23.83%	26.92%	27.28%	23.61%

Table 5.5: Accuracies for various WIPs and GWs on most likely path from first phase of word spotter.

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	21.9%, 30.2%	15.1%, 32.6%	11.1%, 33.8%	8.7%, 34.2%	8.3%, 33.0%
0	18.7%, 29.9%	12.9%, 32.3%	9.4%, 33.5%	7.5%, 33.9%	7.5%, 32.4%
-2	16.0%, 29.7%	11.0%, 32.0%	7.9%, 33.4%	6.7%, 33.3%	7.2%, 31.8%
-4	13.5%, 29.5%	9.3%, 31.9%	6.8%, 33.1%	6.0%, 32.9%	6.9%, 30.9%
-6	11.4%, 29.2%	7.8%, 31.6%	6.0%, 32.7%	5.5%, 32.4%	6.4%, 29.7%

Table 5.6: Insertions and correctness respectively for various WIPs and GWs on most likely path from first phase of word spotter.

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	78.19%	78.06%	77.17%	74.83%	69.50%
0	78.31%	77.96%	76.80%	74.10%	68.25%
-2	78.19%	77.81%	76.33%	73.08%	66.51%
-4	78.13%	77.49%	75.74%	72.09%	64.36%
-6	77.95%	77.13%	75.06%	70.65%	62.26%

Table 5.7: Correctness of lattice for various WIPs and GWs on most likely path from first phase of word spotter.

Considering GW 4.0 in Table 5.6, we see that from WIP 2 to WIP 0 we have a significant reduction in insertions, but only a slight reduction in correctness. We can also see in Table 5.7 that when using GW 4.0, with WIPs of 2 and 0 the lattice contains significantly more of the correct words compared to WIP -6. The correctness rate of the lattice was very high (almost 80%), showing us that with proper search techniques in later phases we should be able to find very accurate hypotheses. We can see that phase one worked properly and had been effective in reducing the search space

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	86.63	52.01	31.03	19.81	13.12
0	81.58	48.67	29.20	18.51	12.21
-2	76.74	45.46	27.55	17.28	11.49
-4	72.09	42.44	25.93	16.10	10.71
-6	67.60	39.67	24.39	14.91	10.01

Table 5.8: Average word beginnings per frame in the lattice for various WIPs and GWs.

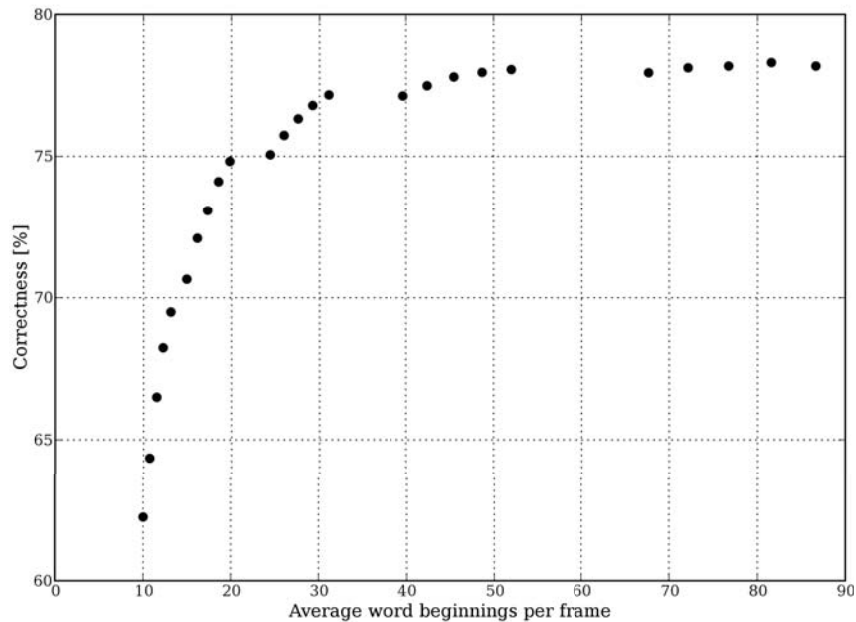


Figure 5.8: Lattice correctness for various average word beginnings per frame counts. The lattice correctness only increases slightly for systems with an average of more than 20 word beginnings per frame.

significantly while retaining as many correct hypotheses as possible.

We assumed that it is better to have too many insertions in our N-Best lists than to have too many deletions, since deletions can never be recovered. If the lattice correctness was our most important consideration, we would tend to choose GW 1.0 and WIP 0. However, the load on later phases would be prohibitive, since we see in Table 5.8 that on average there

are more than four times more word beginnings per frame to consider. In Fig. 5.8 we see that the lattice correctness increases only slightly for systems with an average of more than 20 word beginnings per frame.

In summary, GW 4.0 and WIP 0 seem to find a good balance between best path accuracy, correctness and insertions, average words per frame and lattice correctness for phase one.

These results are seen as the baseline results of the original system at the University of Stellenbosch. If we had performed these experiments with the original system we would have found this WER of 73.46%. Note that these experiments would not have been possible if we were forced to use a normal Viterbi segmenter as we would have done in the past. Our multi-level segmenter enables us to perform evaluations with a large-vocabulary, since memory requirements are vastly reduced.

5.3.2 Finding effective parameters for phase two.

Motivation: Once the lattice from phase one was in place, we could proceed to find the proper parameters for phase two. We had to ensure that our N-Best list contained as many correct words as possible, which would enable our later rescoring to increase accuracy further.

Experimental setup: We chose a state-level beam of e^{50} and a word level beam of e^{30} .

For various combinations of grammar weights (GWs) and word insertion penalties (WIPs), we found our N-Best paths for each sub-segment as described in Section 4.8 with $N = 200$. A bigram language model making use of Good-Turing discounting and Katz smoothing was used, which was trained from the Hub-4 training data. The final grammar scores were calculated as described in Equation 4.4.1. We tried GWs 1.0, 2.0, 3.0, 4.0 and 5.0 with WIPs 2, 0, -2, -4 and -6. For each of these configurations we evaluated the two hours of speech in the development set and measured:

1. The percentage of words appearing in the correct word sequence and

also in one of the N-Best paths.

2. The accuracy, insertions and deletions for the most likely path in the N-Best list (path 1 out of N).

Note that this correctness is not the ideal measurement but only an indicator. To truly measure potential gain from rescoring, we would need to evaluate each of the N-Best paths and use the best scoring path as our maximum potential accuracy. Our measurement is similar to the lattice correctness measure for phase one, and only shows us how many of the correct words survived in the N-Best lists. This is much simpler to measure, yet still gives us a good indication of the quality of the N-Best lists.

Results: The accuracies of the highest scoring paths from phase two for each segment is seen in Table 5.9. The associated insertion and deletion rates appear in Table 5.10. We see the correctness of the N-Best lists in Table 5.11.

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	7.20%	19.10%	25.33%	28.79%	28.40%
0	11.10%	21.27%	26.93%	29.78%	28.70%
-2	14.53%	23.27%	28.22%	30.25%	28.43%
-4	16.91%	24.87%	29.00%	30.73%	27.97%
-6	18.80%	26.09%	29.76%	30.92%	27.31%

Table 5.9: Accuracies for various WIPs and GWs for most likely path in the N-Best list generated from the second phase of the word spotter.

Interpretation: From the accuracies found in Table 5.9 we see that the most accurate best path is found once again with GW 4.0 and WIP -6 . If this was a two-phase system, we would find the WER of 69.08%.

However, we are not finding the parameter to maximise accuracy, but to maximise potential gain from the rescoring in the final phase. It is important to note that the correctness of the best scoring hypothesis drops as the WIP is increased, which is seen in Table 5.10. We again assumed that it is better to have too many insertions than to have too many deletions.

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	20.6%, 32.4%	15.1%, 36.1%	12.0%, 38.1%	10.2%, 39.3%	10.3%, 39.0%
0	17.9%, 32.1%	13.3%, 35.8%	10.5%, 37.9%	9.1%, 39.0%	9.5%, 38.4%
-2	15.5%, 32.0%	11.5%, 35.4%	9.2%, 37.6%	8.4%, 38.6%	9.2%, 37.6%
-4	13.5%, 31.7%	10.1%, 35.2%	8.1%, 37.0%	7.6%, 38.2%	8.8%, 36.7%
-6	11.9%, 31.5%	8.7%, 34.8%	7.3%, 36.8%	5.4%, 37.7%	8.5%, 35.7%

Table 5.10: Respective insertions and correctness for various WIPs and GWs on most likely path in the N-Best list generated from the second phase of the word spotter.

WIP	GW 1.0	GW 2.0	GW 3.0	GW 4.0	GW 5.0
2	36.65%	41.73%	44.52%	45.70%	45.39%
0	36.80%	41.47%	44.04%	45.40%	44.46%
-2	36.69%	41.22%	43.63%	44.92%	43.56%
-4	36.51%	40.92%	43.20%	44.48%	42.40%
-6	36.29%	40.59%	42.68%	43.82%	41.15%

Table 5.11: The percentage of words appearing in the correct word sequence and also in one of the N-Best paths from the second phase of the word spotter for various WIPs and GWs.

For this reason we chose a lower WIP in phase two to maximise correctness. The percentage of correct words in the N-Best lists is seen in Table 5.11. It shows us that when using GW 4.0 with WIP 0 the lattice contains 1.6% more of the correct words compared to WIP -6. If the lattice correctness was our most important consideration, we would be inclined to choose GW 4.0 and WIP 2. We noticed that the correctness of the N-Best lists were almost 30% lower than the correctness of the lattice from the first phase. From this we can conclude that a significant amount of information was lost and another approach for phase two might be considered.

In summary, GW 4.0 and WIP 0 seems to find the best balance between best path accuracy, correctness and deletions and N-Best list correctness. We would expect to see a significant improvement on the best path accuracy from the first phase to the second phase, since we are now making use of a significantly more complex bigram LM. This was indeed the case and we saw

an improvement from 26.54% to 29.78%, which supports the theory that more information can be extracted from the unigram word lattice created in the first phase. We found a significant improvement on the baseline system by performing an N-Best search and simply using the best scoring hypothesis.

Incorporating these more complex LMs into the extensive search space of phase one would call for a different approach. The calculation of the higher-order grammar scores in our implementation was not computationally feasible.

5.3.3 Finding effective parameters for phase three.

Motivation: Now that the lattice from phase one and the N-Best lists from phase two are in place, we can find proper parameters for phase three. Since this is the final step in our recogniser, we only need to maximise accuracy in this phase.

Experimental setup: For various combinations of grammar weights (GWs) and word insertion penalties (WIPs), we rescored our N-Best paths for each sub-segment as described in Section 4.8. An utterance HMM is created for each of the N hypotheses which is evaluated using the same triphones, but using a trigram language model making use of Good-Turing discounting and Katz smoothing. This LM was trained from the Hub-4 training data. The best scoring path, after a re-evaluation of each of the N hypotheses, is chosen as the final hypothesis.

The final grammar scores were calculated as described in Equation 4.4.1. We tried GWs 5.0, 6.0, 7.0, 8.0 and 9.0 with various insertion penalties. GWs lower than 5.0 and higher than 9.0 were attempted, but the results were extremely poor. For each of these configurations we evaluated the two hours of speech in the development set and measured the accuracy, insertions and deletions for the path with the best score after rescoring.

Results: In Table 5.12 we see the accuracies of the highest scoring paths after rescoring from phase three for each segment. The associated insertion

and deletion rates is seen in Table 5.13.

WIP	GW 5.0	GW 6.0	GW 7.0	GW 8.0	GW 9.0
2	30.57%	31.14%	31.20%	31.18%	31.24%
0	31.04%	31.42%	31.42%	31.41%	31.40%
-2	31.37%	31.56%	31.60%	31.57%	31.53%
-4	31.66%	31.66%	31.60%	31.52%	31.53%
-6	31.78%	31.83%	31.69%	31.57%	31.52%
-8	31.76%	31.96%	31.78%	31.71%	31.65%
-10	31.82%	31.98%	31.95%	31.81%	31.70%
-12	31.80%	31.94%	31.96%	31.83%	31.78%
-14	31.87%	31.95%	32.00%	31.92%	31.78%
-16	31.84%	31.94%	31.90%	31.95%	31.88%

Table 5.12: Accuracies for various WIPs and GWs for the hypotheses from the third phase of the word spotter.

WIP	GW 5.0	GW 6.0	GW 7.0	GW 8.0	GW 9.0
2	9.0%, 39.7%	8.6%, 39.7%	8.4%, 39.6%	8.3%, 39.5%	8.3%, 39.4%
0	8.5%, 39.6%	8.2%, 39.6%	8.2%, 39.5%	8.1%, 39.4%	8.1%, 39.4%
-2	8.1%, 39.3%	8.0%, 39.4%	8.0%, 39.5%	8.0%, 39.4%	7.9%, 39.4%
-4	7.7%, 39.2%	7.8%, 39.3%	7.8%, 39.3%	7.9%, 39.2%	7.9%, 39.4%
-6	7.5%, 39.1%	7.6%, 39.2%	7.6%, 39.2%	7.7%, 39.1%	7.8%, 39.1%
-8	7.3%, 38.8%	7.4%, 39.1%	7.5%, 39.1%	7.6%, 39.1%	7.6%, 39.1%
-10	7.2%, 38.8%	7.3%, 39.1%	7.3%, 39.1%	7.4%, 39.1%	7.5%, 39.1%
-12	7.1%, 38.7%	7.2%, 38.9%	7.3%, 39.0%	7.4%, 39.0%	7.4%, 39.0%
-14	7.0%, 38.6%	7.1%, 38.8%	7.2%, 39.0%	7.3%, 39.0%	7.4%, 39.0%
-16	7.0%, 38.6%	7.0%, 38.7%	7.1%, 38.8%	7.2%, 38.9%	7.3%, 39.0%

Table 5.13: Insertions and correctness respectively for various WIPs and GWs for the hypotheses from the third phase of the word spotter.

Interpretation: From the accuracies found in Table 5.12 we see that the most accurate system is found with a GW 7.0 and WIP -14 . This results in a WER of 68%. From Table 5.13 it would seem that higher insertion penalties result in a significant drop in insertions with a slight drop in correctness, which accounts for the more accurate results.

Since we are now making use of a trigram LM, we would expect to see an improvement on the bigram LM result from the second phase. The accuracy was improved from 29.78% in the second phase to 32.00% in the third phase, showing that we were able to extract more information by rescoreing the N-Best list.

5.4 Determining the effect of a word length penalty

Motivation: We now have a simple measure for our WIP, but a more complex form should be considered. Finding long words which are a concatenation of several phonemes is more difficult than finding words which consist of just one or two phonemes. This is simply because the odds of matching a short segment of speech with the phoneme for a poorly articulated word is easier than matching a long segment of speech with a long sequence of phonemes.

For this reason we attempted to make use of a simple word length penalty and measured its effect.

Experimental setup: A penalty model of any complexity can be used, but we decided to simply see if incorporating this knowledge source improves the performance of our system.

A simple linear penalty model as illustrated in Figure 5.9 was used. x is the fundamental penalty and was chosen to be 8 in our experiment. The penalty reduces linearly with the number of phonemes in the word.

We now repeated our complete development set evaluation with the length penalty incorporated in each phase. We used the same GW and WIP for all phases, namely GW 4.0 and WIP 0 for phase one and phase two and GW 7.0 and WIP -14 for phase three.

Results: We found the results for phase one, phase two and phase three in Tables 5.14, 5.15 and 5.16 respectively.

Interpretation: The word length penalty had almost no effect on the first

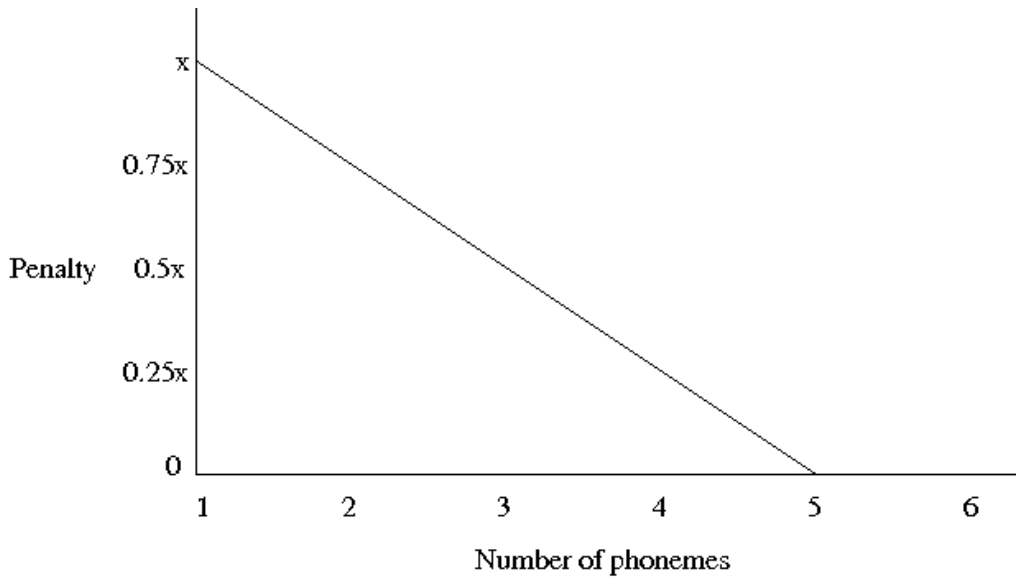


Figure 5.9: Linear model used for word length penalty. The more phonemes the word contains, the smaller the penalty becomes.

	No penalty	Penalty
Accuracy	26.54%	26.54%
Insertions	7.5%	7.5%
Correctness	33.9%	33.9%
Average WPF	18.51%	18.53%
Lattice correctness	74.10%	74.10%

Table 5.14: Phase one results comparing the normal system with the system using a word length penalty.

	No penalty	Penalty
Accuracy	29.78%	31.74%
Insertions	9.1%	6.08%
Correctness	38.98%	37.43%
N-Best correctness	45.40%	43.73%

Table 5.15: Phase two results comparing the normal system with the system using a word length penalty.

	No penalty	Penalty
Accuracy	32.00%	32.90%
Insertions	7.2%	4.7%
Correctness	39.0%	37.2%

Table 5.16: Phase three results comparing the normal system with the system using a word length penalty.

phase of the word spotter, but this is not surprising since the unigram LM plays a relatively small role.

In phases two and three the effect of the word length penalty becomes apparent. Insertions are reduced, while the accuracy of the best path is increased. It should be noted that the correctness of the best path and also the N-Best lists are reduced. This can be attributed to the reduction in hypothesis words that accompanies the drop in insertions. Fewer hypothesis words reduce insertions, but also result in fewer correct words.

From these results we can conclude that we can improve accuracy by adding a penalty to shorter words. This also shows us that our system prefers shorter words and this needs to be accounted for.

5.5 Sphinx 3 Hub-4 Development set evaluation

Motivation: In order to compare our system with Sphinx 3, we performed the Hub 4 evaluation with a very similar configuration to our own system. This allowed us to compare the performance of the two systems as effectively as possible.

Experimental setup: We used the 1999 Sphinx 3.2 system, which was compiled from source code that we downloaded from the Sphinx site. We used a lexicon, a phoneme set and a language model that were identical to those used in our system and all the default parameters for Sphinx 3.2.

Our Sphinx 3.2 setup made use of 3-state no-skip topology HMMs in a lexical tree search structure. A lexical tree is an effective compact way

to represent a lexicon, but there are some problems associated with them. More information on lexical trees can be found in [39].

Triphones (including cross-word triphones) making use of a total of 1000 tied states were trained. The density at each state was a Gaussian mixture model with 8 mixtures. Sphinx 3.2 performed a single pass search with the the SRI trigram language model we trained on the Hub-4 training data.

The hypotheses were evaluated exactly as described by the Hub 4 evaluation specifications, by making use of all the provided word filters and SCLite software.

Results: The results for the development set evaluation of the various Hub 4 focus conditions are shown in Table 5.17. Our development set hypotheses were evaluated in the exact same way and the results are also shown. The focus conditions are described as follows:

- **F0 - Baseline broadcast speech:** (10.15%) Speech that is directed to the general broadcast audience is recorded in a quiet studio environment with a signal-to-noise ratio (SNR) of greater than 20 dB and mostly read from prepared text.
- **F1 - Spontaneous broadcast speech:** (20.46%) Unprepared speech that is directed to one or more conversational partners that is recorded in a quiet studio environment and is presumed to have an SNR of greater than 20 dB.
- **F2 - Speech over telephone channels:** (12.79%) Speech that is collected over reduced-bandwidth conditions, such as telephony and similar media.
- **F3 - Speech in the presence of background music:** (5.00%) Equivalent to F0 or F1, but includes background music such that the speech is still intelligible to the normal listener.
- **F4 - Speech under degraded acoustic conditions:** (10.68%) Speech that is acoustically degraded for reasons other than those in

condition F2 and include reasons such as additive noise, environmental noise or nonlinear distortions.

- **F5 - Speech from non-native speakers:** (4.46%) Speech that satisfies the attributes of condition F0, but is spoken by non-native fluent speakers of American English with a foreign accent.
- **FX - All other speech:** (36.47%) All speech that does not fall in any of the other categories.

The percentage in brackets following each description shows the portion of the total data that falls in that particular category.

	Overall	F0	F1	F2	F3	F4	F5	FX
Sphinx 3	56.3%	37.2%	59.7%	65.3%	65.1%	50.8%	50.3%	57.6
Our system	68.9%	51.5%	68.3%	76.0%	78.3%	65.9%	62.7%	71.9
Difference	-12.6%	-14.3%	-8.6%	-10.7%	-13.2%	-15.1%	-12.4%	-14.3%

Table 5.17: Word-Error rates for Sphinx 3 and our system on the development set.

Our system performed the development set evaluation in about 41.5 hours, while Sphinx 3.2 performed the same evaluation in about 2.25 hours. Thus our system was around 18 times slower than Sphinx 3.2 on a 1.8 GHz dual-core Pentium 4 with 2 GB RAM. We found that our system spent around 45% of the total processing time on phase one and another 45% on phase two. The remaining 10% was spent on the final phase.

Interpretation: The performance of our system was significantly worse than that of Sphinx 3. There are various potential reasons for this:

1. Sphinx had been in development for over 11 years by various teams of scientists.
2. Sphinx 3 made use of lexical tree search and had various copies of different lexical trees in memory at a given time depending on LM context.

3. Sphinx 3 incorporated left-context cross-word triphones.
4. Sphinx 3 made use of various absolute pruning boundaries to control worst-case performance. (Maximum active HMMs, maximum word exits and maximum LM histories per frame)

These techniques should be considered for future versions of our system. Since the vast majority of processing time was spent on the first two phases, we should consider the lexical tree search in particular if performance is to be improved.

5.6 Continuous word recognition on Hub-4 broadcast speech evaluation set

Motivation: We have performed our experiments on the development set and we would now like to see the performance of Sphinx 3.2 and our system on the evaluation set.

Experimental setup: All the parameters for our system as they were determined by earlier experiments were used. We also used the same lexicon, phoneme set and language model.

Results: The results for the evaluation set of the various Hub 4 focus conditions are shown in Table 5.18. Our development set hypotheses were evaluated in the same way and the results are also shown. The focus conditions are the same as in the development set and the proportions the make up of the evaluation set are:

- **F0 - Baseline broadcast speech:** 13.70%
- **F1 - Spontaneous broadcast speech:** 0.01%
- **F3 - Speech in the presence of background music:** 3.35%
- **F4 - Speech under degraded acoustic conditions:** 0.27%
- **FX - All other speech:** 81.65%

	Overall	F0	F1	F3	F4	FX
Sphinx 3	57.1%	46.4%	55.9%	56.4%	83.3%	58.8
Our system	68.9%	61.4%	63.7%	78.1%	94.4%	69.7
Difference	-11.8%	-15.0%	-7.8%	-21.7%	-11.1%	-10.9%

Table 5.18: Word-Error rates for Sphinx 3 and our system on the evaluation set.

Our system performed the evaluation in about 38.0 hours, while Sphinx 3.2 performed the same evaluation in about 2.0 hours. Thus our system was around 19 times slower than Sphinx 3.2 on a 1.8 GHz dual-core Pentium 4 with 2 GB RAM.

Interpretation: These results are consistent with the experiments on our development set and the final difference in accuracy between our system and Sphinx 3 was 11.8% for the evaluation set.

As mentioned earlier, there are many differences between Sphinx and our system that account for the difference in WER. There are many avenues of research that should be considered for future versions of the system at the University of Stellenbosch that would result in more competitive WERs.

5.7 Summary

In this chapter we found appropriate parameters for all the components in our word recogniser. First, we explained our phoneme modelling approach and how we trained monophone models. The accuracy of the baseline monophone models was reported and found to be relatively poor. We later incorporated Δ and $\Delta\Delta$ coefficients and LDA into our features. This expansion of the original flat start monophones resulted in a significant increase in accuracy (8.19%) for continuous monophone spotting.

After finding our initial monophone transcriptions, we could move on to training triphones. We experimented with various MOCs for decision tree-based triphones and were able to improve our baseline 42.10% continuous

monophone spotting accuracy to 45.90% for triphones making use of 1364 tied states.

After finding our triphones, we were interested in measuring the effect of beam width on recognition accuracy and performance. We were able to conclude that a properly chosen beam width improves performance significantly, while barely reducing phoneme recognition accuracy.

Our next challenge was to find appropriate parameters for the vast parameter space for our three-phase word spotter. We decided to determine parameters one phase at a time. After extensive experimentation on the development set, we found proper parameters for all three phases and could perform our final evaluation. However, the lattice correctness of phase one dropped by almost 30% compared to the N-Best lists of phase two. From this we can conclude that other approaches to the search in phase two should be considered, such as the word-dependent N-Best algorithm. We were also able to show that the inclusion of a word length penalty improved performance significantly.

Experiments comparing the performance of our system to Sphinx 3.2 were performed and showed that our system performed significantly worse. Various possible reasons for this were mentioned, including some of the features present in Sphinx 3.2 that were absent in our system.

Chapter 6

Conclusion

6.1 Concluding Perspective

In this thesis we studied some of the components necessary for a large-vocabulary speech recognition (LVCSR) application. We also decided on specific algorithms for each component and implemented them. These various implementations were then combined into a single LVCSR system.

Firstly, we considered the acoustic modelling required for any speech recogniser. Hidden Markov Models (HMMs) were chosen, since they have been widely studied and applied to speech recognition. We also considered some of the more advanced HMM segmentation techniques, such as N-Best HMM decoding, which was shown to be very useful in LVCSR applications.

Next, we studied context dependency and the ways in which it can be incorporated into speech recognition. We considered some of the standard context-dependent phoneme modelling techniques and opted to use the classic decision tree approach. Context between words and the modelling thereof was also considered and we chose to use normal n -gram models as implemented by SRI in our implementation. Our system was developed to have a purely modular approach and any of the components can be replaced so that new and different approaches can be compared in future work.

We described in detail how the various components were integrated to

form a complete, three-phase, LVCSR system. We attempted to find effective values for all the parameters in the system experimentally. Finally, we performed the 1996 NIST Hub-4 Evaluation on our system so that it might be compared with the performance of Sphinx 3.

Several significant new pieces of code are now included in the recognition system at the DSP department of the University of Stellenbosch:

- Multi-Level and backward Viterbi HMM decoders
- N-Best HMM decoders
- Language modelling integration
- Three phase combination of various components

Experimentally we found that our system did not perform as well as we had hoped. We found our system to be 12.6% less accurate than Sphinx 3.2 on the development set and 11.9% less accurate on the evaluation set. However, when considering these results we should remember that Sphinx had been in development for more than 11 years when Sphinx 3.2 was released. Many of the top scientists in the field (including Kai-Fu Lee and Raj Reddy) were intimately involved in the development of the system. Advanced techniques such as lexical tree search and cross-word triphones play a part in its accurate recognition. Various major optimisations such as limiting the maximum number of active HMMs, maximum word exits and maximum LM histories per frame enable them to perform the search more efficiently. These optimisations also allow them to perform a more detailed search in the same amount of time.

When we take these differences into account along with the complexity of the problem, we can justify the relatively poor performance of our system. We did not develop this system to be an improvement on Sphinx, but simply to be a complete and functional LVCSR system. We considered modern speech recognisers and the necessary components for them to function. After deciding on some basic components and implementing them, we

were able to combine them to produce a basic system that will be valuable in the future study of large-vocabulary speech recognition at the University of Stellenbosch. We were able to effectively bring the system closer to a competitive large-vocabulary continuous speech recogniser.

6.2 Future work

Many aspects of our system should be further explored if performance more comparable to Sphinx 3.2 is desired:

- The three phases in our implementation were chosen to illustrate the gains from each phase. By incorporating some more intelligent algorithms and optimisations, the three phases can be combined into one phase as was done in Sphinx 3.2. This approach should be studied further.
- The significant drop in correctness from the lattice in phase one to the N-Best lists in phase two should be addressed. Variations of our phase two search should be considered as it appears that this is where we found the greatest loss in potential accuracy; the traceback-based N-Best algorithm deficiency as described in Section 2.7.3.3 could account for this.
- Finding appropriate parameters for the system and testing them is an extremely extensive and time consuming task. We were only able to find initial guess parameters and these are by no means guaranteed to deliver the best results. Further experimentation with beam widths and word insertion penalties could deliver positive improvements.
- Lexical tree search should be included in the system, since it has been shown that they are essential for an efficient system. Ravishankar [39] was able to improve performance by a factor of 5, while barely increasing the lattice WER. This is accompanied by a much smaller

HMM, resulting in greatly reduced memory requirements. Since the majority of the computational load was on the first two phases, a lexical tree model should be considered to replace our parallel word model.

- Our simple word length penalty reduced the WER by 1.38%. Further experimentation with our word length penalty and other more complex word length penalty models should yield further improvements.
- Information can be shared to a greater extent between the three phases of the spotter, which would yield significant performance improvements. For instance, the acoustic scores found in phase two are recalculated during the rescoring in phase three. Such redundancies should be minimised or eliminated.

Other advanced techniques can also be considered for future versions of our recogniser and include:

- The word-dependent N-Best algorithm, which has been shown to perform well and deliver good results [43]. It should be considered as an alternative to the traceback-based N-Best algorithm used in our implementation. The N-Best algorithm we used did not distinguish N-Best paths on their history, while the word-dependent algorithm takes the single previous word into account.
- Adaptation techniques such as speaker adaptation [29] and acoustic adaptation [37]. These have been used in the past with great success and should be considered.

Bibliography

- [1] Afify, M., Liu, F., Jiang, H., Siohan, O., “A New Verification-Based Fast-Match for Large Vocabulary Continuous Speech Recognition,” in *IEEE Transactions on Speech and Audio Processing*, July 2005, Vol. 13, pp. 546-553.
- [2] Bahl, L. R., Brown, P. F., de Souza, P. V., Mercer, R. L., “A tree-based statistical language model for natural language speech recognition,” in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, July 1989, pp. 1001-1008.
- [3] Baker, J. K., “The DRAGON System – An Overview,” in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, February 1975, pp. 24-29.
- [4] Baker, R., “Continuous speech word recognition via centisecond acoustic states,” in *Proc. ASA Meeting* (Washington, DC), April 1976. Referenced in [11].
- [5] Baum, L. E., “An Inequality and Associated Maximisation Technique in Statistical Estimation of Probabilistic Functions of Markov Processes,” in *Inequalities 3*, 1972, pp.1-8. Referenced in [25].
- [6] Breiman, L., *et al.*, *Classification and Regression Trees*. Pacific Grove, CA: Wadsworth, 1984. Referenced in [17].

- [7] Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J., D., Mercer, R., L., Roossin, P., S., "A statistical approach to machine translation," in *Computational Linguistics*, June 1990, Vol. 16, pp. 79-85.
- [8] Chow, Y. L., Dunham, M. O., Kimball, O. A., Krasner, M. A., Kubala, G. F., Makhoul, J., Roucos, S., Schwartz, R. M., "BYBLOS: The BBN Continuous Speech Recognition System," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 1987, pp. 89-92.
- [9] Cilliers, F. D., *Tree-based Gaussian Mixture Models for Speaker Verification*. MScEng Thesis, University of Stellenbosch, 2005.
- [10] Deshmukh, N., Ganapathiraju, A., Hamaker, J., Picone, J., Ordowski, M., "A public domain speech-to-text system," in *Proceedings of the 6th European Conference on Speech Communication and Technology*, September 1999, Vol. 5, pp. 2127-2130.
- [11] Engelbrecht, H. A., *Automatic Phoneme Recognition of South African English*. MScEng Thesis, University of Stellenbosch, 2003.
- [12] Engelbrecht, H. A., *Efficient Decoding of Higher-order Hidden Markov Models*. Phd Thesis, University of Stellenbosch, 2007.
- [13] Good, I., "The population frequencies of species and the estimation of population parameters," in *Biometrika*, 1953, Vol. 40, pp. 237-264. Referenced in [40].
- [14] Hermansky, H., "Perceptual linear predictive (PLP) analysis of speech," in *The Journal of the Acoustical Society of America*, 1990, Vol. 87, no. 4, pp. 1738-1752. Referenced in [50].
- [15] Hogg, R., McKean, J., Craig, A., *Introduction to Mathematical Statistics*. Upper Saddle River, NJ: Pearson Prentice-Hall, 2005.

- [16] Huang, X., Alleva, F., Hon, H., Hwang, M., Rosenfeld, R., “The Sphinx-II Speech Recognition System: An Overview,” in *Computer, Speech and Language*, 1993, Vol. 2, pp. 137-148.
- [17] Huang, X., Acero, A., Hon, H., *Spoken Language Processing*. New Jersey: Prentice-Hall, 2001.
- [18] Hunt, A. J., Black, A. L., “Unit selection in a concatenative speech synthesis system using a large speech database,” in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Atlanta, GA, USA, 1996, Vol. 1, pp. 373-376.
- [19] Itakura, F., “Minimum Prediction Residual Principle Applied to Speech Recognition,” in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, February 1975, pp. 67-72.
- [20] Jelinek, F., “A Real-Time, Isolated-Word, Speech Recognition System for Dictation Transcription,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 1985.
- [21] Katz, S., “Estimation of probabilities from sparse data for the language model component of a speech recogniser.” in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, March 1987, Vol. 35, pp. 400-401. Referenced in [40].
- [22] Kay, S. M., *Fundamentals of Statistical Signal Processing: Estimation Theory*. New Jersey: Prentice-Hall, 1993.
- [23] Kingsbury, P., *et al.*, *CALLHOME American English Lexicon (PRONLEX)*, Linguistic Data Consortium, Philadelphia, 1997.
- [24] Lamere, P., Kwok, P., Gouvêa, E. B., Bhiksha, R., Singh, R., Walker, W., Wolf, P., “The CMU Sphinx-4 Speech Recognition System,” in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 2003.

- [25] Lee, K., *Automatic Speech Recognition*. Kluwer Academic Publishers, 1989.
- [26] Lesser, V. R., Fennell, R. D., Erman, L. D., Reddy, R. D., "The Hearsay II Speech Understanding System," in *IEEE Transactions on Acoustics, Speech, and Signal Processing*, February 1975, pp. 11-24.
- [27] Liu, X., Zhao, Y., Xiaobo, Pi., Liang, L., Nefian, A. V., "Audio-Visual continuous speech recognition using a coupled hidden Markov model," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2002, Section 2.3.8.
- [28] Marcus, M., Santorini, B., Marcinkiewicz, M., "Building a large annotated corpus of English: the Penn Treebank," *Computational Linguistics*, 1993.
- [29] Matsui, T., Furui, S., "N-best-based instantaneous speaker adaptation method for speech recognition," *International Conference on Spoken Language Processing*, 1996, Vol. 2, pp. 973-976.
- [30] Mohri, M., "Finite-state transducers in language and speech processing," *Computational Linguistics*, 1997, Vol. 23, no. 2, pp. 269-311.
- [31] Odell, J. J., *The Use of Context in Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, 1995.
- [32] Odell, J. J., *The Use of Decision Trees With Context Sensitive Phoneme Modelling*. MPhil Thesis, University of Cambridge, 1992. Referenced in [31].
- [33] Odell, J. J., Ollason, D., Woodland, P., Young, S., Jansen, J., *The HTK Book for HTK V2.0.* Cambridge University Press, Cambridge, 1995.

- [34] Odell, J. J., Woodland, P. C., Young, S. J., "Tree-Based State Clustering for Large Vocabulary Speech Recognition," *International Symposium on Speech Image Processing and Neural Networks*, 1994, pp. 690-693.
- [35] Parihar, N., Picone, J., "DSR Front End LVCSR Evaluation," AU/384/02, Aurora Working Group, Dec. 2002.
- [36] Peebles, P. Z., *Probability, Random Variables and Random Signal Principles*. McGraw-Hill, 2001.
- [37] Placeway, P., Chen, S., Eskenazi, M., Jain, U., Parikh, V., Raj, B., Ravishankar, M., Rosenfeld, R., Seymore, K., Siegler, M., Stern, R., Thayer, E., "The 1996 HUB-4 Sphinx-3 system," in *Proceedings of the DARPA Speech Recognition Workshop*, February 1997.
- [38] Rabiner, L. R., Wilpon, J. G., Soong, F. K., "High Performance Connected Digit Recognition Using Hidden Markov Models," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 1988.
- [39] Ravishankar, M., K., *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon University, May 1996.
- [40] Rosenfeld, R., "Two decades of statistical language modelling: Where do we go from here?," in *Proceedings of the IEEE*, 2000, Vol. 88.
- [41] Rosenfeld, R., *Adaptive Statistical Language Modelling: A Maximum Entropy Approach*. PhD thesis, Carnegie Mellon University, April 1994.
- [42] Schwartz, R., Chow, Y. L., "The N-Best Algorithm: an Efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses," in *Proceedings of the IEEE International Conference*

- on Acoustics, Speech, and Signal Processing*, Albuquerque, New Mexico, 1990, pp. 81-84.
- [43] Schwartz, R., Austin, S., "A Comparison of Several Approximate Algorithms for Finding Multiple (N-BEST) Sentence Hypotheses," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toronto, Canada, 1991, pp. 701-704.
- [44] Shinoda, K., Watanabe, T., "DL-based context-dependent sub-word modelling for speech recognition," in *J. Acoust. Soc. Jpn.(E)*, 2000, Vol. 21, No. 2, pp.79-86. Referenced in [54].
- [45] Singh, R., Warmuth, M., Raj, B., Lamere, P., "Classification with free energy at raised temperatures," in *Proceedings of the 8th European Conference on Speech Communication and Technology*, Geneva, Switzerland, September 2003, pp. 1773-1776. Referenced in [50]
- [46] Steinbiss, V., "Sentence-Hypotheses Generation in Continuous Speech Recognition," in *Proceedings of Eurospeech*, Paris, 1989, pp.51-54. Referenced in [17].
- [47] Stolcke, A., "SRILM – An Extensible Language Modelling Toolkit," in *Proceedings of the International Conference on Spoken Language Processing*, Denver, 2002, Vol. 2, pp. 901-904.
- [48] Tran, B., Seide, F., Steinbiss, V., "A word graph based n-best search in continuous speech recognition," in *International conference on spoken language processing*, 1996, pp. 2127-2130.
- [49] Viterbi, A. J., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, " in *IEEE Transactions on Information Theory*, April 1967, Vol. 13, pp. 260-269.

- [50] Walker, W., Lamere, P., Kwok, P., Bhiksha, R., Singh, R., Gouvea, E., Wolf, P., Woelfel, J., "Sphinx-4: A Flexible Open Source Framework for Speech Recognition", Technical Report SMLI TR2004-0811, Sun Microsystems Inc., 2004.
- [51] Wilpon, J. G., Rabiner, L. R., Bergh, A., "Speaker-Independent Isolated Word Recognition Using a 129-Word Airline Vocabulary," in *The Journal of the Acoustical Society of America*, August 1982, pp. 390-396.
- [52] Woodland, P. C., Povey, D., "Large scale discriminative training of hidden Markov models for speech recognition," in *Computer, Speech and Language*, 2002, Vol. 16, pp. 25-47.
- [53] Woodland, P. C., Odell, J. J., Valtchev, V., Young, S. J., "Large Vocabulary Continuous Speech Recognition Using HTK," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, April 1994.
- [54] Zen, H., Tokuda, K., Kitamura, T., "Decision tree-based simultaneous clustering of phonetic contexts, dimensions, and state positions for acoustic modelling," in *Eurospeech 2003*, pp.3189-3192.

Appendix A

General Speech Recognition and Evaluation Techniques

A.1 Linear Discriminant Analysis

In order to calculate the best feature set for our evaluations we need to transform them into a new space in which the important classes can be more easily distinguished. A systematic approach is to use within-class and between-class scatter matrices to formulate the criterion for class separability. This approach is called Linear Discriminant Analysis. The within-class scatter matrix is calculated as:

$$\mathbf{S}_W = \sum_{\mathbf{x} \in \omega_i} P(\omega_i) E\{(\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^t | \omega_i\} = \sum_{\mathbf{x} \in \omega_i} P(\omega_i) \Sigma_i \quad (\text{A.1.1})$$

where the sum is for all the data \mathbf{x} within the class ω_i . This represents the scatter of samples around the mean for each class. The between-class scatter matrix is the scatter of samples around the mixture mean and is calculated as:

$$\mathbf{S}_B = \sum_{\mu_i \in \omega_i} P(\omega_i) (\mu_i - \mathbf{m}_0)(\mu_i - \mathbf{m}_0)^t \quad (\text{A.1.2})$$

where

$$\mathbf{m}_0 = E\{\mathbf{x}\} = \sum_{\omega_i} \mathbf{m}_i, \quad (\text{A.1.3})$$

which is the expected mean vector of the mixture distribution. To derive the linear transformation matrix \mathbf{A} we can use the eigenvectors of $S_W^{-1}S_B$. We can now easily reduce the dimension of the original input feature by discarding the least significant eigenvalues.

A.2 Mel-frequency cepstral coefficients

The Mel-frequency cepstral coefficients (MFCC) is a popular representation of a windowed short-time signal. It is based on the FFT of the signal. This representation incorporates the behaviour of the auditory system by using a non-linear frequency scale to filter the signal. The short-time Fourier spectrum for the discrete-time speech signal $s(n)$ is calculated by using the short-time DFT [17]. The short-time DFT of a windowed speech signal is defined as

$$X_a[k] = \sum_{n=0}^{N-1} xne^{(-j2\pi nk/N)m} \text{ for } k = 0, 1, \dots, N-1. \quad (\text{A.2.1})$$

Next we need a filter bank with M filters, where each filter m is a triangular filter given by:

$$H_m[k] = \begin{cases} 0 & \text{if } k < f[m-1] \\ \frac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & \text{if } f[m-1] \leq k \leq f[m] \\ \frac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m+1]-f[m])} & \text{if } f[m] \leq k \leq f[m+1] \\ 0 & \text{if } k > f[m+1] \end{cases} \quad (\text{A.2.2})$$

These filters find the average spectrum around the various center frequencies with increasing bandwidths and they are displayed in Fig A.1. Now we define f_1 and f_h to be the lowest and highest frequencies in the filter bank in Hz, F_s the sampling frequency in Hz, M the number of

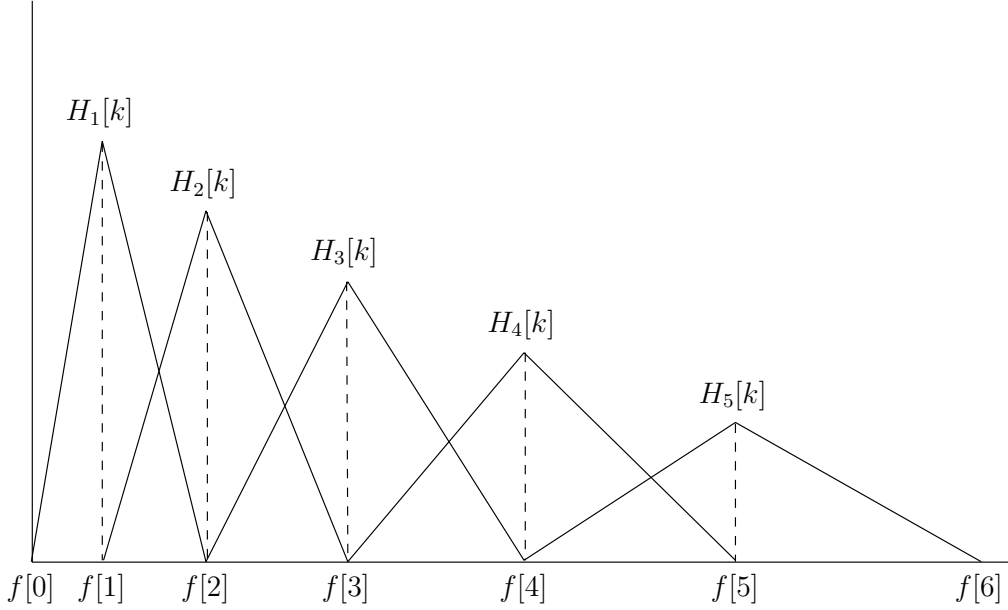


Figure A.1: Triangular filters used in MFCC.

filters and N the size of the FFT. The boundary points $f[m]$ are uniformly spaced in the mel-scale:

$$f[m] = \left(\frac{N}{F_s}\right) B^{-1}\left(B(f_l) + m \frac{B(f_h) - B(f_l)}{M + 1}\right) \quad (\text{A.2.3})$$

where the mel-scale B is given by:

$$B(f) = 1125 \log\left(1 + \frac{f}{700}\right) \quad (\text{A.2.4})$$

and its inverse is given by:

$$B^{-1}(b) = 700(e^{\frac{b}{1125}} - 1) \quad (\text{A.2.5})$$

The log-energy of the output of each filter is then computed as

$$S[m] = \log \left[\sum_{k=0}^{N-1} |X_a[k]|^2 H_m[k] \right] \quad (\text{A.2.6})$$

Finally, the mel-frequency cepstrum is the discrete cosine transform of the output from the M filters:

$$c[n] = \sum_{m=0}^{M-1} S[m] \cos\left(\frac{\pi n(m - \frac{1}{2})}{M}\right) \quad (\text{A.2.7})$$

For speech recognition applications the first 13 cepstrum coefficients are typically used.

A.3 Perplexity

For us to be able to compare different language models we need to define some kind of measure. The purpose of a language model is to reduce uncertainty during recognition. The search space reduces as the grammar becomes more restricting. This is good for processing requirements, but has a negative effect on the variety of sentences that can be recognised. In other words, we do not want to restrict the search to an unnecessary extent or the recogniser will be unable to recognise valid sentences. In contrast, we might put extreme computational strain on the recogniser if the grammar is too general. In the extreme case where there is no grammar the recogniser must evaluate the possibility of each word at each decision point. A typical grammar will not make invalid sentences impossible, but merely improbable.

Such a language model can be constructed by studying the problem you are trying to solve. For example, if we are trying to recognise spoken numbers we will know that numbers in speech are constructed using a simple grammar. By using a grammar we can give the recogniser additional information on the way in which logical sentences are put together. In effect we are limiting the search space to results that are logical. This will have the negative effect of making it more difficult for the recogniser to correctly identify sentences that were actually pronounced incorrectly. However, it will make the results more accurate in general and much easier to obtain.

The measure of constraint at a decision point (j) can be measured by *entropy* (H), which is a measure typically used in information theory.

Entropy can also be seen as the number of bits necessary to specify the next word using an optimal encoding scheme:

$$H(W|j) = - \sum_{w=1}^V P(w|j) \cdot \log_2[P(w|j)] \quad (\text{A.3.1})$$

Now we can define perplexity at decision point j as:

$$Q(w|j) = 2^{H(w|j)} \quad (\text{A.3.2})$$

In most practical cases you will have a finite state grammar with many states, or decision points, and entropy is computed as:

$$H(L) = \sum_j \pi(j) H(W|j) \quad (\text{A.3.3})$$

where $\pi(j)$ is the steady-state probability of being in state j . From this we can calculate the per-word perplexity of this language model as:

$$Q(L) = 2^{H(L)} \quad (\text{A.3.4})$$

A.4 A density for the estimated average power of Gaussian noise

We model a silence signal (with a certain background noise-level) as a sequence of statistically independent Gaussian random variables X_i with $i = 1 \dots N$, each with mean $a_X = 0$ and variance σ_x^2 .

First we will derive a density function for the estimated average power P of X . From this we can determine the mean and variance of P . Let

$$P = \frac{1}{N} \sum_{i=1}^N X_i^2 \quad (\text{A.4.1})$$

be the estimated power. Since the X_i 's are random variables, so is P . We want to find the density $f_P(p)$.

First consider the transformation $Y = X^2$. Then $\frac{dy}{dx} = 2x$ and $x = \pm\sqrt{y}$.

Using basic theory on the transformation of random variables [36] we find that

$$\begin{aligned} f_Y(y) &= \frac{f_X(x)}{\left|\frac{dy}{dx}\right|} \Big|_{x=-\sqrt{y}} + \frac{f_X(x)}{\left|\frac{dy}{dx}\right|} \Big|_{x=+\sqrt{y}} \\ &= \frac{f_X(x)}{2\sqrt{y}} \Big|_{x=-\sqrt{y}} + \frac{f_X(x)}{2\sqrt{y}} \Big|_{x=+\sqrt{y}} \\ &= \frac{1}{\sqrt{2\pi\sigma_X^2}} y^{1/2-1} e^{\frac{-y}{2\sigma_X^2}} u(y). \end{aligned}$$

Therefore Y is Gamma distributed with parameters $b = \frac{1}{2}$ and $a = \frac{1}{2\sigma_X^2}$.

Its characteristic function is given by:

$$\Phi_Y(\omega) = \left(\frac{\frac{1}{2\sigma_X^2}}{\frac{1}{2\sigma_X^2} - j\omega} \right)^{\frac{1}{2}}.$$

Now consider:

$$Z = \sum_{i=1}^N Y_i \text{ with } Y_i \text{ IID.}$$

The characteristic function of Z is now given by:

$$\begin{aligned} \Phi_Z(\omega) &= [\Phi_Y(\omega)]^N \\ &= \left(\frac{\frac{1}{2\sigma_X^2}}{\frac{1}{2\sigma_X^2} - j\omega} \right)^{\frac{N}{2}}. \end{aligned}$$

Therefore Z is also Gamma distributed, but now with parameters $b = \frac{N}{2}$ and $a = \frac{1}{2\sigma_X^2}$.

The corresponding density function of Z is given by:

$$f_z(z) = \frac{\left(\frac{1}{2\sigma_X^2}\right)^{N/2} z^{N/2-1} e^{\frac{-z}{2\sigma_X^2}} u(z)}{\Gamma(N/2)}.$$

The estimated power is calculated as $P = Z/N$. Then $\frac{dp}{dz} = 1/N$ and $z = Np$.

$$\begin{aligned} f_P(p) &= N \frac{\left(\frac{1}{2\sigma_X^2}\right)^{N/2} (Np)^{N/2-1} e^{-\frac{Np}{2\sigma_X^2}} u(p)}{\Gamma(N/2)} \\ &= \frac{\left(\frac{N}{2\sigma_X^2}\right)^{N/2} p^{N/2-1} e^{-\frac{Np}{2\sigma_X^2}} u(p)}{\Gamma(N/2)}. \end{aligned}$$

Therefore, finally, we can conclude that P is also Gamma distributed, but now with parameters

$$b = \frac{N}{2} \tag{A.4.2}$$

and

$$a = \frac{N}{2\sigma_X^2}. \tag{A.4.3}$$

From the properties of a Gamma density this also implies that

$$\bar{P} = b/a = \sigma_X^2 \tag{A.4.4}$$

and

$$\sigma_P^2 = b/a^2 = 2\sigma_X^4/N. \tag{A.4.5}$$

A.5 The mean and variance of an estimate of the variance of the estimated mean power

We now estimate the variance Q of the above estimate of the average power P . This is given by

$$Q = \frac{1}{K} \sum_{k=1}^K (P_k - \bar{P})^2 \tag{A.5.1}$$

with each P_k an independent power estimate such as given by Equation A.4.1. From the results of the previous section we know P_k to be Gamma distributed, and in general we can assume its parameters to be a and b . Its mean is given by $\bar{P} = b/a$ (Equation A.4.4) and its variance by $\sigma_P^2 = b/a^2$ (Equation A.4.5).

The density of Q is somewhat involved, therefore we will rather focus on directly finding its mean \bar{Q} and variance σ_Q^2 . First consider:

$$Y = \sum_{k=1}^K (P_k - \bar{P})^2. \quad (\text{A.5.2})$$

Since the P_k terms are statistically independent we can easily find the mean \bar{Y} :

$$\begin{aligned} E[Y] &= \sum_{k=1}^K E[(P_k - \bar{P})^2] \\ &= K\sigma_P^2 \\ &= Kb/a^2. \end{aligned} \quad (\text{A.5.3})$$

The variance of Y can be determined via its moments around the origin:

$$\begin{aligned} \sigma_Y^2 &= E[Y^2] - \bar{Y}^2 \\ &= E[Y^2] - (Kb/a^2)^2. \end{aligned} \quad (\text{A.5.4})$$

An expression for $E[Y^2]$ is needed:

$$\begin{aligned} E[Y^2] &= E\left[\sum_{i=1}^K (P_i - \bar{P})^2 \sum_{j=1}^K (P_j - \bar{P})^2\right] \\ &= E\left[\sum_{i=1}^K (P_i - \bar{P})^4 + \sum_{i=1}^K (P_i - \bar{P})^2 \sum_{j=1, j \neq i}^K (P_j - \bar{P})^2\right]. \end{aligned}$$

Taking into account that P_i is independent of P_j with $i \neq j$ it now follows that:

$$\begin{aligned} E[Y^2] &= \sum_{i=1}^K E[(P_i - \bar{P})^4] + \sum_{i=1}^K E[(P_i - \bar{P})^2] \sum_{j=1, j \neq i}^K E[(P_j - \bar{P})^2] \\ &= K\mu_4 + \sum_{i=1}^K b/a^2 \sum_{j=1, j \neq i}^K b/a^2 \\ &= K\mu_4 + (K^2 - K)(b/a^2)^2 \end{aligned}$$

with $\mu_4 = E[(P - \bar{P})^4]$ the 4'th central moment of P . Substitute this back into Equation A.5.4 to find

$$\begin{aligned} \sigma_Y^2 &= K\mu_4 + (K^2 - K)(b/a^2)^2 - K^2(b/a^2)^2 \\ &= K[\mu_4 - (b/a^2)^2]. \end{aligned} \tag{A.5.5}$$

Using the binomial expression for the power of a sum the required central moment can be determined as:

$$\begin{aligned} \mu_4 &= E[(P - \bar{P})^4] \\ &= E\left[\sum_{n=0}^4 \binom{4}{n} P^n (-\bar{P})^{4-n}\right] \\ &= E[(-\bar{P})^4 + 4P(-\bar{P})^3 + 6P^2(-\bar{P})^2 + 4P^3(-\bar{P}) + P^4] \\ &= m_4 - 4m_3\bar{P} + 6m_2(\bar{P})^2 - 3\bar{P}^4 \end{aligned} \tag{A.5.6}$$

with m_n the n 'th moment of P around the origin and $m_1 = \bar{P}$. We therefore still need to find expressions for the 2nd up to 4th moments of P around the origin. This can be done via the characteristic function of P .

In general:

$$m_n = E[P^n] = \frac{d^n \Phi_P(\omega)}{j^n d\omega^n} \Big|_{\omega=0}$$

Therefore:

$$\begin{aligned}
 \Phi_P(\omega) &= \left(\frac{a}{a - j\omega}\right)^b = (1 - j\omega/a)^{-b} \\
 \frac{d\Phi_P(\omega)}{d\omega} &= \frac{jb}{a}(1 - j\omega/a)^{-b-1} \\
 \frac{d^2\Phi_P(\omega)}{d\omega^2} &= \frac{j^2b(b+1)}{a^2}(1 - j\omega/a)^{-b-2} \\
 \frac{d^3\Phi_P(\omega)}{d\omega^3} &= \frac{j^3b(b+1)(b+2)}{a^3}(1 - j\omega/a)^{-b-3} \\
 \frac{d^4\Phi_P(\omega)}{d\omega^4} &= \frac{j^4b(b+1)(b+2)(b+3)}{a^4}(1 - j\omega/a)^{-b-4}.
 \end{aligned}$$

After simplification we therefore find the required moments around the origin:

$$\begin{aligned}
 m_2 &= \frac{b(b+1)}{a^2} = \frac{b^2 + b}{a^2} \\
 m_3 &= \frac{b(b+1)(b+2)}{a^3} = \frac{b^3 + 3b^2 + 2b}{a^3} \\
 m_4 &= \frac{b(b+1)(b+2)(b+3)}{a^4} = \frac{b^4 + 6b^3 + 11b^2 + 6b}{a^4}.
 \end{aligned}$$

Substitute this into Equation A.5.6 and simplify to find that

$$\mu_4 = \frac{3b^2 + 6b}{a^4}.$$

Substitute this into Equation A.5.5 and simplify:

$$\sigma_Y^2 = K \frac{2b^2 + 6b}{a^4}. \quad (\text{A.5.7})$$

We now know \bar{Y} and σ_Y^2 via Equations A.5.3 and A.5.7, and from Equations A.4.2 and A.4.3 we know the appropriate values of b and a . From Equations A.5.1 and A.5.2 it follows that $Q = Y/K$. After further simplification (standard expectation manipulations) we finally get that:

$$\bar{Q} = \frac{\bar{Y}}{K} = b/a^2 = \frac{2\sigma_X^4}{N} \quad (\text{A.5.8})$$

and

$$\sigma_Q^2 = \frac{\sigma_Y^2}{K^2} = \frac{2b^2 + 6b}{Ka^4} = \frac{8(N+6)\sigma_X^8}{KN^3}. \quad (\text{A.5.9})$$

Appendix B

Phoneme set

No.	Name	Example
1	@	(a)brupt
2	@r	abs(ur)d
3	@u	alth(ough)
4	D	ano(th)er
5	E	acad(e)mic
6	I	accept(e)d
7	N	bri(ng)
8	O:	acc(o)rd
9	Oi	ann(oy)
10	Q	an(o)maly
11	S	applica(t)ion
12	T	ari(th)metic
13	Z	a(s)ia
14	a	bankr(u)pt
15	ai	basel(i)ne
16	au	black(ou)t
17	b	(b)lame
18	d	blackwoo(d)
19	dZ	bonda(ge)
20	ei	br(a)ces

Table B.1: Phoneme set used in our implementation (part 1).

No.	Name	Example
21	f	break(f)ast
22	g	bu(g)
23	h	a(h)ead
24	i:	abilit(ie)s
25	j	a(bu)se
26	k	a(cc)use
27	l	active(l)y
28	m	ada(m)
29	n	abdome(n)
30	other	(noise)
31	p	abru(p)t
32	r	abso(r)b
33	s	ab(s)tract
34	sil	(silence)
35	t	accen(t)
36	tS	a(ch)ieve
37	u	amb(u)sh
38	u:	am(u)sed
39	v	an(v)il
40	w	any(w)ay
41	z	apologie(s)
42	{	(a)pathy

Table B.2: Phoneme set used in our implementation (part 2).

Appendix C

Question set

High Vowel	I @ i: u u:
Medium Vowel	{ a @ E @r ei @u
Low Vowel	Q { a O: au ai Oi
Rounded Vowel	O: @u Oi u u: w
Unrounded Vowel	Q { a au @ ai E @r ei h I @ i: l r j
Reduced Vowel	@ @
I Vowel	I @ i:
E Vowel	E ei
A Vowel	Q { au ai @r
O Vowel	O: @u Oi
U Vowel	a @ u u:
Unvoiced Consonant	tS f h k p s S t T
Voiced Consonant	b d D g dZ l m n N r v w j
Front Consonant	b f m p v w
Central Consonant	d D d l n r s t T z Z
Back Consonant	tS g h dZ k N S j
Fortis Consonant	tS f k p s S t T
Lenis Consonant	b d D g dZ v z Z
Neither F or L	h l m n N r w j
Coronal Consonant	tS d D dZ l n r s S t T z Z
Non Coronal	b f g h k m N p v w j
Anterior Consonant	b d D f l m n p s t T v w z

Table C.1: Question set used in construction of decision tree (part 1).

Non Anterior	tS g h dZ k N r S j Z
Continuant	D f h l m n N r s S T v w j z Z
No Continuant	b tS d g dZ k p t
Positive Strident	tS dZ s S z Z
Negative Strident	D f h T v
Neutral Strident	b d g k l m n N p r t w j
Syllabic Consonant	@r
Voiced Stop	b d g
Unvoiced Stop	p t k
Front Stop	b p
Central Stop	d t
Back Stop	g k
Voiced Fricative	tS D v z Z
Unvoiced Fricative	tS f s S T
Front Fricative	f v
Central Fricative	D s T z
Back Fricative	tS dZ S Z
Affricate Consonant	tS dZ
Not Affricate	D f s S T v z Z
Silence	sil

Table C.2: Question set used in construction of decision tree (part 2).